UNIVERSIDAD DE CHILE
FACULTAD DE CIENCIAS FISICAS Y MATEMATICAS
DEPARTAMENTO DE INGENIERIA ELECTRICA

DISEÑO Y EVALUACIÓN DE ALGORITMOS EVOLUTIVOS PARA ESTRATEGIAS DE CONTROL PREDICTIVO HÍBRIDO NO LINEAL

TESIS PARA OPTAR AL GRADO DE MAGISTER EN CIENCIAS DE LA INGENIERÍA, MENCION ELECTRICA

MEMORIA PARA OPTAR AL TITULO DE INGENIERO CIVIL ELECTRICISTA

DIEGO ALEJANDRO MUÑOZ CARPINTERO

PROFESOR GUÍA:
DORIS SÁEZ HUEICHAPAN

PROFESOR CO-GUÍA:
CRISTIAN CORTÉS CARRILLO.

MIEMBRO DE LA COMISIÓN:
MARCOS ORCHARD CONCHA

SANTIAGO DE CHILE
ENERO 2010

RESUMEN DE LA TESIS PARA OPTAR AL TÍTULO DE INGENIERO CIVIL ELECTRICISTA Y AL GRADO DE MAGÍSTER EN CIENCIAS DE LA INGENIERÍA, MENCIÓN ELÉCTRICA.
POR: DIEGO MUÑOZ CARPINTERO
FECHA: 25/01/2010.
PROFESOR GUÍA: SRA. DORIS SÁEZ HUEICHAPAN

Diseño y Evaluación de Algoritmos Evolutivos para Estrategias de Control Predictivo Híbrido No Lineal

Las estrategias de control predictivo han cobrado gran interés en los últimos años por parte de la comunidad científica, dado que permiten incorporar índices de funcionamiento de los procesos, tales como de la calidad de su respuesta y del esfuerzo de control, y restricciones operacionales, que incluyen por ejemplo los rangos admisibles de operación de los componentes del sistema. Los índices de funcionamento son muy útiles pues permiten privilegiar distintos aspectos referentes a la dinámica del sistema y además, el uso de restricciones permite tener una descripción mucho más realista del proceso. Para encontrar las variables manipuladas con un controlador predictivo, se debe resolver un problema de optimización que consiste en la minimización del índice de funcionamiento sujeto a las restricciones del proceso. Para sistemas con un cierto grado de complejidad, la resolución del problema de optimización puede ser difícil debido a su alto grado de no linealidad, alta dimensionalidad o a la presencia de variables enteras y continuas (sistemas híbridos). Como una alternativa a los metodos clásicos, para aquellos casos en los que con estos métodos no se puede manejar satisfactoriamente estas dificultades, en este trabajo se propone utilizar algoritmos evolutivos para resolver los problemas de optimización entera mixta que aparecen en el control predictivo de sistemas híbridos no lineales.

Basado en las cualidades de los algoritmos evolutivos, en este trabajo se propone métodos genéricos para el diseño y evaluación de estrategias de control predictivo para sistemas híbridos no lineales basado en algoritmos evolutivos para la optimización de la función objetivo en tiempo real. Las estrategias propuestas se evalúan en tres sistemas híbridos: un reactor *batch* con entradas discretas; el mismo reactor *batch*, pero con entradas discretas y continuas (mixtas); y un sistema de ruteo dinámico de vehículos.

Los estudios presentados justifican la aplicación de algoritmos evolutivos en términos de tiempo computacional y calidad de las soluciones en la mayoría de los casos, consiguiendo obtener controladores más eficientes que otros ya existentes para estos procesos. Además, se estudia diversos aspectos relevantes para la aplicación de algoritmos evolutivos en la resolución del problema de optimización en control predictivo híbrido no lineal, tales como representación de soluciones, manejo de restricciones, y selección de los parámetros del algoritmo. Para este último aspecto, se utiliza un análisis multi-objetivo que permite escoger el tamaño óptimo de población y el número óptimo de iteraciones de acuerdo a criterios basados en el tiempo computacional y la calidad de las soluciones. Este análisis se puede interpretar además como un indicador gráfico de la bondad de una estrategia basada en algoritmos evolutivos, y por lo tanto se utiliza para comparar distintas estrategias dentro de los aspectos mencionados.

Finalmente, se propone algunas líneas de investigación para la aplicación de algoritmos evolutivos en control predictivo híbrido no lineal. Dentro de estos temas se encuentran aspectos que debido a la amplitud del tema no han sido analizados, tales como argumentos de estabilidad o el uso de mezclas entre métodos convencionales y algoritmos evolutivos, y otros que si bien han sido analizados, aún pueden ser mejorados, tales como el análisis multi-objetivo para la sintonía de parámetros y comparación de estrategias.

Agradecimientos

En este apartado quiero expresar mi más sincera gratitud a todos quienes han ayudado de algún modo a cumplir este gran objetivo.

A mis guías, Doris Sáez y Cristián Cortés. Por brindarme la oportunidad de trabajar en su grupo, mis primeros pasos en el mundo de la investigación.,Por las bases y ayuda brindada, para que estos pasos sólo sean los primeros. Además, por aceptar guiar mi Tesis, y por supuesto, su constante buena onda, consejos y apoyo. Mención especial a Alfredo, por su compañerismo y ayuda constante en las labores de investigación.

A Marcos Orchard, por su dedicación, y valiosos comentarios que permitiron elevar la calidad de mi Tesis. Además, por su simpatía y confianza a todo terreno.

Por su puesto a mi familia, en especial a mis padres, Teresa y Marcelo. Les agradezco desde lo más profundo de mi alma por todo el apoyo incondicional, en las buenas y en las malas, y por saber guiarme a encontrar mi propio camino cuando este no se veía claro. A pesar de la importancia de lo anterior, no habría servido de nada de no ser por todo el amor y los valores, que junto con mi hermana Natalia, me han entregado a lo largo de mi vida, gracias a los cuales soy la persona quien soy.

A todos los amigos que me han acompañado durante la estancia en la universidad. Belén, mi amiga de siempre. Por los buenos momentos, y el constante apoyo. Mis amigos mechones: Fabián, Ignacio, Jose, por ser exceletes amigos y todos los grandes momentos que vivimos. Mis amigos eléctricos: Isaias y Rubén, por toda la buena onda, ser grandes compañeros y mejores personas; Leonardo, Gabriel, Asenjo, Erbetta, Gonzalo y Felipe, por toda la buena onda, y hacer que los últimos años de la carrera, a pesar de todas las complicaciones, hayan sido siempre alegres y una constante tira cómica. Mención especial a Felipe, quien constantemente me ha motivado y apoyado en las decisiones ligadas con los estudios venideros. Tampoco quiero olvidar a todos aquellos que no he mencionado, mechones, de plan común, civiles, eléctricos o de cualquier parte de este mundo, que han estado conmigo y han hecho que mi vida sea mejor.

Finalmente a CONICYT, por la beca para mis estudios de magíster.

A todos ustedes, de todo corazón, muchas gracias.

Índice

Índice de Figuras

Índice de tablas

1 Introducción

1.1 Control predictivo híbrido

El control predictivo basado en modelos (*Model Predictive Control*, MPC) es una técnica de control de sistemas en tiempo discreto que pertenece a la familia del control óptimo. La acción de control se calcula en cada instante minimizando una función objetivo que en general considera dos componentes: la desviación respecto de la referencia de la salida estimada del sistema (que se obtienen mediante un modelo) y de los cambios en la acción de control, a lo largo de un horizonte de predicción. De acuerdo a las características del proceso, se puede trabajar con: modelos lineales o no lineales; con o sin restricciones; y con variables continuas, discretas, o incluso híbridas. Dependiendo de la combinación de estos factores, el problema de optimización puede ser muy simple para problemas lineales y sin restricciones, o altamente demandante para sistemas no lineales, con muchas dimensiones, con restricciones (que también pueden ser no lineales), y con variables continuas y discretas.

Las estrategias de control predictivo basado en modelos han cobrado gran interés en los últimos años pues permiten incorporar índices de funcionamiento de los procesos y restricciones en su diseño (Camacho y Bordons, 2004). Los índices son muy útiles ya que permiten privilegiar distintos aspectos referentes a la dinámica del sistema o a las variables manipuladas y, ccon el uso de restricciones, la formulación del problema es mucho más completa pues se puede caracterizar de mejor manera los sistemas reales que se están controlando.

Hasta el momento ha sido posible resolver el problema de optimización de modo eficiente para sistemas lineales y de baja dimensión; sin embargo, para problemas no lineales o de muchas dimensiones, esto no siempre ha sido posible. Más aún, cuando los problemas son híbridos, es decir cuentan con variables continuas y discretas, se obtienen problemas de optimización enteros mixtos, que son NP *hard* y muy difíciles de resolver en tiempo real, aumentando su complejidad exponencialmente con el número de variables discretas.

Una característica especial de los problemas de optimización para el control predictivo, en particular para sistemas no lineales, es que el problema de optimización es variante en el tiempo. Si cambian los puntos de operación (por ejemplo en la partida de un proceso), debido a las no linealidades y a las restricciones, el problema de optimización a resolver puede ser muy distinto de otros ya resueltos en instantes previos. Por lo tanto, se debe asumir que en general el problema de optimización que se resuelve en cada instante es variante en el tiempo, y entonces es altamente recomendable el uso de algoritmos genéricos que presenten un buen desempeño en un rango relativamente alto de problemas.

Es necesario entonces diseñar técnicas de optimización que puedan lidiar con todos los problemas ya mencionados, y lo más importante, que puedan ser aplicados en tiempo real. Los algoritmos evolutivos poseen características que los convierten en una opción altamente llamativa para resolver los problemas de optimización de controladores predictivos complejos, como se explica a continuación.

1.2 Algoritmos evolutivos

La mayor parte de los problemas de optimización prácticos tienen diversas propiedades desafiantes: prácticamente todos tienen un número significativo de óptimos locales, y el espacio de búsqueda puede ser tan grande que el óptimo global exacto puede no ser encontrado en un tiempo razonable; pueden tener múltiples objetivos conflictivos que pueden ser considerados simultáneamente (por ejemplo, costo versus calidad); puede haber un conjunto elevado de restricciones no lineales que debe satisfacer la solución final; puede tener componentes dinámicos que alteran la ubicación del óptimo; y por último, los problemas pueden ser de tal complejidad que si bien existen algoritmos que permiten encontrar soluciones exactas, estas pueden tomar un tiempo excesivamente mayor que el aceptado por la aplicación.

En algunos problemas, las metodologías de solución basadas en búsquedas locales han probado ser muy eficientes, por ejemplo, el algoritmo de Lin-Kernighan (Lin y Kernighan, 1973) para el Problema del Vendedor Viajero. Sin embargo, los algoritmos de búsqueda local, tales como el método del gradiente (Snyman, 2005), no permiten una disminución en la calidad de la solución durante el proceso de búsqueda. Por esta razón, estos algoritmos suelen quedarse estancados en óptimos locales, por lo que no son convenientes para muchos problemas prácticos. Los algoritmos de búsqueda estocástica tales como s*imulated annealing* (Orosz y Jacobson, 2002) búsqueda tabú (Glover y Laguna, 1997), algoritmos evolutivos (Back *et al*., 1997), y optimización por enjambre de partículas (Kennedy y Eberhart, 2001) son valiosas alternativas para enfrentar este tipo de problemas.

Dentro de las alternativas de algoritmos de búsqueda estocástica, los algoritmos evolutivos (*evolutionary algorithms*, EA) y optimización por enjambre de partículas (*particle swarm optimization*, PSO), poseen ciertas características que los hacen muy llamativos para su aplicación en problemas prácticos. Primero, éstos son muy generales y pueden ser aplicado a una gran gama de problemas (continuos, discretos, híbridos, combinatoriales, etc.). Segundo, estos algoritmos pueden ser fácilmente combinados con otras técnicas existentes incluyendo estrategias de búsqueda local y otros métodos exactos. Tercero, a menudo es directo incorporar algún conocimiento del problema en los operadores evolutivos o en la población inicial. Finalmente, estas técnicas pueden manejar problemas con cualquiera de los desafíos del mundo real que ya han sido mencionados (óptimos locales, múltiples objetivos, restricciones, y componentes dinámicos). En este sentido, la mayor ventaja de EAs y PSO yace en su enfoque basado en poblaciones.

Para los óptimos locales, la diversidad de la población permite que los algoritmos exploren diversas áreas del espacio de búsqueda simultáneamente. Por supuesto que esto no garantiza evitar convergencia prematura a un óptimo local, pero la población mejora la robustez de estos algoritmos en este tipo de problemas. En problemas multi-objetivo, EAs y PSO entregan un conjunto de soluciones con distintos grados de compromiso entre los distintos objetivos conflictivos en una sola ejecución del algoritmo, mientras que los algoritmos tradicionales sólo entregan un punto por ejecución. Respecto a los problemas con restricciones, estos algoritmos pueden admitir una mezcla de soluciones factibles e infactibles en sus poblaciones, lo que mejora sus capacidades de explorar las fronteras entre regiones factibles en infactibles. Por otra parte, las poblaciones brindan a estas estrategias una ventaja en los problemas dinámicos, pues la población puede ya contener una buena solución después que el problema cambie. Finalmente, para los problemas de alta complejidad computacional, estas familias de algoritmos permiten encontrar

soluciones sub-óptimas en un tiempo finito (limitado por ejemplo con un máximo número de iteraciones).

Naturalmente, estos algoritmos también poseen desventajas. Desafortunadamente, pueden llegar a ser relativamente costosos en términos computacionales, dado que muchas soluciones deben ser evaluadas en el proceso de optimización. Debido a esto, hubo un reciente aumento en el interés de enfrentar este problema, y algunas estrategias han sido sugeridas. Por ejemplo, se puede almacenar los valores de la función objetivo (y las distintas componentes de ella) de las soluciones candidatas que ya han sido evaluadas, lo que puede ayudar a evitar repetir el cálculo para otras soluciones candidatas que coinciden en alguna componente (Ursem, 2003). Debido al esfuerzo computacional que requiere su ejecución, es entonces importante notar que estos algoritmos no deben ser aplicados sobre cualquier problema, pues puede ser que existan técnicas más simples y rápidas para resolverlo más eficientemente. Por ejemplo, existen sistemas lineales donde un controlador proporcional integral (PI) es altamente efectivo, o bien, la solución del problema de optimización lineal sin restricciones del control predictivo es analítica y no requiere resolver un problema *online* (en línea). En este contexto, EAs y PSO ofrecen la posibilidad de mejorar soluciones encontradas por otras técnicas más simples (por ejemplo utilizando las soluciones que entregan estas técnicas en la población del algoritmo evolutivo), dar un buen punto de partida para las otras técnicas, o bien de entregar soluciones aproximadas en un tiempo finito cuando la solución exacta requiere de un altísimo tiempo de cómputos.

Otra desventaja de EAs y PSO es que poseen más parámetros de sintonía que otras técnicas más simples. Desafortunadamente el valor óptimo de estos parámetros depende del problema. Si bien los parámetros óptimos también dependen del problema en las técnicas más simples, es mucho más sencillo volver a sintonizarlos debido a que son menos parámetros que en EAs y PSO. Debido a lo anterior, la sintonía de parámetros es un aspecto relevante en algoritmos evolutivos.

Gracias a su naturaleza heurística, los algoritmos evolutivos y optimización por enjambre de partículas, pueden ser mucho más rápidos que otras estrategias convencionales, tales como *branch and bound* o planos cortantes, para encontrar soluciones razonables. Además, poseen buenas cualidades para encontrar óptimos globales, por lo tanto es recomendable su aplicación en problemas en tiempo real. Sin embargo, éstos no poseen prueba de convergencia a la solución optima global, salvo en condiciones muy particulares que suelen ser poco realistas. Teniendo en cuenta estas y otras características, en este trabajo se busca proponer metodologías genéricas para el análisis y diseño estrategias de control predictivo híbrido basado en algoritmos evolutivos.

Como se verá más adelante (Sección 4.4), PSO tiene muchas similitudes con los algoritmos evolutivos y ha sido considerado como uno de sus exponentes (Kennedy y Eberhart, 2001). Debido a esto, y por economía del lenguaje, cada vez que se mencione a los algoritmos evolutivos para su aplicación en control predictivo se debe entender que se hace mención tanto a los algoritmos evolutivos clásicos como a PSO.

1.3 Objetivos

El objetivo general de esta tesis es establecer un metodo genérico para el análisis y diseño del método de optimización en el control predictivo híbrido basado en algoritmos evolutivos y optimización por enjambre de partículas.

El desarrollo y evaluación de este metodo genérico se realiza en base a modelos computacionales de procesos reales, que poseen variables continuas y discretas, de mayor y menor escala, en los cuales se aplica control predictivo híbrido. En este contexto, se busca diseñar controladores predictivos híbridos que alcancen rendimientos superiores o al menos comparables respecto de otras técnicas existentes para dichos procesos.

Considerando lo anterior, los objetivos específicos de este trabajo son:

- Definir las representaciones y estrategias de manejo de restricciones más adecuadas de acuerdo al proceso y/o requerimientos del sistema de control a utilizar en los algoritmos evolutivos u optimización por enjambre de partículas para resolver el problema de optimización en el control predictivo híbrido.
- Encontrar una metodología eficiente para sintonizar los parámetros más relevantes de los algoritmos evolutivos u optimización por enjambre de partículas utilizados en la formulación del control predictivo híbrido.
- Proponer indicadores que permitan evaluar la calidad del sistema de control y así poder comparar efectivamente las estrategias propuestas. Estos indicadores se encontrarán basados tanto en la calidad de las soluciones como en la rapidez con que sea posible encontrarlas, debido a las restricciones de tiempo para poder aplicar las estrategias en tiempo real.
- Diseñar un controlador predictivo híbrido basado en algoritmos evolutivos para el proceso de ruteo dinámico de vehículos.
- Diseñar un controlador predictivo híbrido basado en algoritmos evolutivos para el control de temperatura de un reactor batch con entradas discretas.
- Diseñar un controlador predictivo híbrido basado en algoritmos evolutivos para el control de temperatura de un reactor batch con entradas mixtas.

1.4 Estructura de la tesis

Este trabajo se encuentra estructurado del siguiente modo. En primera instancia se presenta los aspectos teóricos más relevantes acerca de las temáticas tratadas en esta tesis: los sistemas híbridos, las estrategias de control predictivo híbrido, y los métodos de optimización estocásticos que incluyen a los algoritmos evolutivos y optimización por enjambre de partículas. Luego, se presenta los aspectos relevantes para la aplicación de algoritmos evolutivos para control predictivo, con sus respectivas propuestas, lo que constituye el principal aporte metodológico de la tesis. Posteriormente se presentan los ejemplos de aplicación donde se ponen a prueba los diseños y análisis propuestos, que permiten finalmente presentar las conclusiones acerca del trabajo realizado.

En base a lo anterior, la estructura por capítulos es la siguiente:

- En el capítulo 2 se presenta una revisión de las clases de modelos para representar sistemas híbridos;
- En el capítulo 3 se presenta un revisión de las técnicas de control predictivo híbrido;
- En el capítulo 4 se presenta una revisión de los algoritmos evolutivos, que incluyendo sus aspectos teóricos y aplicaciones a problemas híbridos;
- En el capítulo 5 se presentan los aspectos a estudiar y sus respectivas propuestas respecto del diseño de controladores predictivos híbridos basados en algoritmos evolutivos;

- En los capítulos 6 y 7 se presentan los casos de aplicación para las estrategias estudiadas con los resultados y análisis correspondientes;
- En el capítulo 8 se presentan las discusiones finales y conclusiones del estudio realizado;
- Finalmente, en el capítulo 9, se presentan los anexos.

2 Sistemas Híbridos

En este capítulo se presenta una revisión general de los modelos para sistemas híbridos que se utilizan en el contexto del control predictivo híbrido. Los sistemas híbridos dinámicos son aquellos sistemas que contienen componentes continuas y discretas, que pueden encontrarse tanto en los estados, como en las entradas y/o en las salidas (Heemels *et al*., 2001). Este tipo de sistemas ha sido modelados de distintas formas: como un grafo de transición de estados (con una dinámica continua en cada estado), o bien como un conjunto de ecuaciones diferenciales o de diferencias, que incluyen variables que pueden tomar valores continuos o discretos.

En la primera categoría de modelos de sistemas híbridos se encuentran las máquinas de estados finitos, los sistemas conmutados y las redes de Petri. Las máquinas de estados finitos son modelos utilizados para sistemas con un número máximo de estados finitos. En estos sistemas, para cada combinación de estados y de entradas, se conoce explícitamente la salida y el nuevo estado (que puede ser el mismo anterior o uno nuevo). Estos sistemas suelen ser representados con diagramas o tablas de estado. Sin embargo, para sistemas híbridos se prefiere la representación mediante autómatas híbridos (Alur *et al.*, 1995). Cada estado del autómata representa un comportamiento continuo del sistema, y las transiciones entre estados pueden ser de forma continua o discreta. En el segundo caso se representan mediante eventos o condiciones. Los sistemas conmutados son sistemas dinámicos híbridos que constan de una familia de subsistemas continuos o discretos en el tiempo y una regla que determina la conmutación entre ellos. Las redes de Petri son herramientas para el estudio de sistemas que están compuestos de partes que interactúan entre ellas, pero su comportamiento puede ser descrito independientemente de las otras partes del sistema. Dichos componentes presentan concurrencia o paralelismo, es decir, sus actividades se realizan de manera simultánea con las actividades de otros componentes del sistema. Las redes de Petri híbridas, que modelan sistemas híbridos, están compuestas de una red de Petri discreta y una red de Petri continua, que interactúan cada una de ellas a través de señales de condición-evento. Así, se toman ventajas de la estructura de redes de Petri y las características modulares de los sistemas condición-evento (David y Alla, 1992).

Dentro de la segunda categoría de modelos para sistemas híbridos, Bemporad y Morari (1999) proponen los modelos dinámicos lógicos mixtos (*Mixed logical dynamical systems,* MLD), donde se consideran entradas, estados y salidas continuas y/o discretas. Otra forma de modelar sistemas híbridos es mediante sistemas afines a tramos (PWA, *Piecewise Afine Systems*) (Sontag, 1981), que definen una función lineal más una constante en cada elemento de una partición poliédrica del espacio de estado-entrada. Una ventaja de estos sistemas es que pueden aproximar sistemas no lineales con un grado de precisión arbitrario. Luego Heemels *et al*. (2001) establece equivalencias entre 5 clases de modelos dinámicos híbridos, incluidos los dos ya mencionados: MLD, PWA, sistemas lineales complementarios (van der Schaft y Schumager, 1998), sistemas lineales complementarios extendidos (De Schutter y De Moor, 1999), y los sistemas Max - Min - Plus – Scaling (De Schutter y van den Boom, 2000). Cada subclase posee sus propias ventajas sobre las otras. Por ejemplo, la implementación de técnicas de control para los modelos MLD, la existencia de criterios de estabilidad en los sistemas PWA y la presencia de condiciones de existencia y unicidad de trayectorias de solución para los sistemas lineales complementarios (Heemels, 2001).

Este tipo de modelos que consisten en un conjunto de ecuaciones de diferencia que incluyen variables que pueden tomar valores discretos, son los más importantes en el contexto del control predictivo híbrido, y dentro de ellos, destacan los modelos MLD y PWA.

A continuación se presentan cada uno de estos modelos. Posteriormente se muestra que todos estos modelos son equivalentes, lo que permite una mayor flexibilidad para el estudio de los sistemas híbridos, basándose en las ventajas que posen cada tipo de modelos, como ha sido mencionado anteriormente.

2.1 Modelos híbridos

Los sistemas híbridos pueden ser modelados como un conjunto de ecuaciones diferenciales, o en diferencias, con variables continuas y/o discretas. Las principales representaciones de este tipo de modelos híbridos se presentan a continuación, con particular énfasis en las representaciones MLD y PWA.

2.1.1 Sistemas dinámicos lógicos mixtos (Mixed Logical Dynamical Systems, MLD)

Una forma de modelar sistemas dinámicos híbridos es mediante un sistema con dinámicas lógicas mixtas (sistemas MLD). Este tipo de sistemas propuesto en Bemporad y Morari (1999) corresponde al siguiente planteamiento:

$$\begin{aligned} x_{k+1} &= Ax_k + B_1u_k + B_2\delta_k + B_3z_k \\ y_k &= Cx_k + D_1u_k + D_2\delta_k + D_3z_k \\ E_1x_k &+ E_2u_k + E_3\delta_k + E_4\delta_k \le g \end{aligned} \tag{2.1}$$

donde $x_k = \begin{bmatrix} x_k^r & x_k^b \end{bmatrix}$, $x_k^r \in \mathbb{R}^n$ es la parte continua del estado y $x_k^b \in \{0,1\}^{n_b}$ es la parte discreta. De forma similar $y_k = \begin{bmatrix} y_k^r & y_k^b \end{bmatrix}$, $y_k^r \in \mathbb{R}^m$ es la parte continua de la salida, y $y_k^b \in \{0,1\}^{n_b}$ es la parte discreta. Además $u_k = \begin{bmatrix} u_k^r & u_k^b \end{bmatrix}$, donde $u_k^r \in \mathbb{R}^l$ es la parte continua de la entrada y $u_k^b \in \{0,1\}^{l_b}$ es la parte discreta. Finalmente, $z_k \in \mathbb{R}^{r_r}$ y $\delta_k \in \{0,1\}^{r_b}$ son variables auxiliares, y $A, B_1, B_2, B_3, C, D_1, D_2, D_3, E_1, E_2, E_3, E_4, g$ son parámetros del modelo.

En los sistemas MLD las expresiones lógicas de la parte discreta de un sistema híbrido son expresadas como restricciones de igualdad o desigualdad. Es decir, por medio de variables lógicas δ_1 y δ_2 que pueden tomar valores 0 o 1, cotas superiores *M*, cotas inferiores *m*, y/o tolerancias positivas ε, se puede convertir cualquier expresión lógica en una igualdad o desigualdad. Por ejemplo:

$$\begin{aligned} &L_1 \wedge L_2 \text{ es equivalente a } \delta_1 + \delta_2 \ge 1 \\ &L_1 \vee L_2 \text{ es equivalente a } \delta_1 = 1, \delta_2 = 1 \end{aligned} \tag{2.2}$$

$$\delta_3 = \delta_1\delta_2 \text{ es equivalente a } \begin{cases} -\delta_1 + \delta_3 \le 0 \\ -\delta_2 + \delta_3 \le 0 \\ \delta_1 + \delta_2 - \delta_3 \le 1 \end{cases} \tag{2.3}$$

$$y = \delta f(x) \text{ es equivalente a } \begin{cases} y \le M\delta \\ y \ge m\delta \\ y \le f(x) - m(1-\delta) \\ y \ge f(x) - M(1-\delta) \end{cases} \tag{2.4}$$

En el anexo se puede encontrar más detalle acerca de cómo construir estas restricciones.

2.1.2 Sistemas afines a tramos (*Piecewise Affine*, PWA)

Un sistema lineal afín a tramos (Sontag, 1981) se puede describir por:

$$\begin{aligned} x_{k+1} &= A^i x_k + B^i u_k + f^i \\ y_k &= C^i x_k + D^i u_k + g^i \end{aligned} \text{ para } \begin{bmatrix} x_k \\ u_k \end{bmatrix} \in X_i \tag{2.5}$$

donde $\{X_i\}_{i=1}^{s}$ es una partición poliédrica del espacio de estado-entrada. Cada X_i está dado por:

$$X_i \cong \left\{ \begin{bmatrix} x_k \\ u_k \end{bmatrix} \text{ tal que } Q_i \begin{bmatrix} x_k \\ u_k \end{bmatrix} \le q^i \right\} \tag{2.6}$$

donde x_k, u_k, y_k denotan los vectores de estados, de las entradas y salida, respectivamente. En (2.6) $A_i \in \mathbb{R}^{n\times n}$, $B_i \in \mathbb{R}^{n\times m}$ y (A_i, B_i) es un par controlable. $C_i \in \mathbb{R}^{r\times n}$, $Q_i \in \mathbb{R}^{p_i\times(n+m)}$, f_i, g_i, q_i son vectores constantes adecuados. Notar que *n* es un número de estados, *m* es el número de entradas, *r* es el número de salidas y p_i es el número de híper-planos que definen el poliedro *i*.

Los sistemas afines a tramos tienen la ventaja de permitir aproximar sistemas no lineales con un grado de precisión arbitrario. Su identificación se logra definiendo una grilla y definiendo el mejor sub-modelo afín que se adapta a cada una de ellas (Johansson, 2002). El principal problema de esta identificación es que su mejora tiene que ser lograda introduciendo un alto número de pequeñas regiones. También se ha logrado su identificación por la combinación de técnicas con utilización de clúster, identificación lineal y reconocimiento de patrones (Ferrari, 2000).

2.1.3 Sistemas lineales complementarios (*Linear Complementary systems*, LC)

La formulación de esta clase de sistemas es la siguiente (van der Schaft y Schumager, 1998):

$$
\begin{aligned}
x_{k+1} &= Ax_k + B_1 u_k + B_2 w_k \\
y_k &= Cx_k + D_1 u_k + D_2 w_k \\
v_k &= E_1 x_k + E_2 u_k + E_3 w_k + g \\
0 &\le v_k \perp w_k \ge 0
\end{aligned}
\tag{2.7}
$$

con $v_k, w_k \in \mathbb{R}^s$ y donde $\perp$ denota la ortogonalidad de vectores ($x \perp y \Leftrightarrow x^T y = 0$). v_k y w_k son las variables complementarias. Además, $A, B_1, B_2, C, D_1, D_2, E_1, E_2, E_3, g$ son parámetros del modelo. En este tipo de modelos las variables complementarias son utilizadas para manejar las variables lógicas y las restricciones de desigualdad con restricciones de igualdad, y para transmitir de un modo directo el efecto de las variables lógicas hacia las ecuaciones de salida y estado.

2.1.4 Sistemas lineales complementarios extendidos (Extended Linear Complementary systems, ELC)

De Schutter y De Moor (1999), muestran que variados tipos de sistema híbridos pueden ser modelados como sistemas ELC. La formulación de esta clase de sistemas es la siguiente**:**

$$
\begin{aligned}
x_{k+1} &= Ax_k + B_1 u_k + B_2 d_k \\
y_k &= C_k x_k + D_1 u_k + D_2 d_k \\
E_1 x_k &+ E_2 u_k + E_3 d_k \le g \\
\sum_{i=1}^{p} \prod_{j \in \phi_i} &\left(g_4 - E_1 x_k - E_2 u_k - E_3 d_k \right)_j = 0
\end{aligned}
\tag{2.8}
$$

donde $d_k \in \mathbb{R}^r$ es una variable auxiliar y $A, B_1, B_2, C, D_1, D_2, E_1, E_2, E_3, g$ son parámetros del modelo. La última expresión de (2.8), debido a las condiciones de desigualdad, es equivalente a:

$$
\prod_{j \in \phi_i} \left(g_4 - E_1 x_k - E_2 u_k - E_3 d_k \right)_j = 0, \quad \text{para cada } i \in \{1, 2, \ldots p\} \tag{2.9}
$$

Así las 2 últimas expresiones de (2.8) son un sistema de desigualdades lineales, donde hay p grupos de desigualdades lineales (un grupo para cada conjunto ϕ_i) tal que en cada grupo al menos una de las desigualdades se cumple como igualdad.

2.1.5 Sistemas Max – Min –plus –Scaling (MMPS)

Los modelos MMPS son introducidos por De Schutter y van den Boom (2000). El sistema híbrido puede ser modelado utilizando operaciones de maximización, minimización, adición y multiplicación escalar. Las expresiones compuestas por este tipo de operadores se llaman expresiones min - max - plus - scaling (MMPS).

$$
f := x_i \vee \alpha \vee \max\vee (f_k, f_i) \vee \min\vee (f_k, f_i) \vee (f_k + f_i) \vee \beta \vee f_k \tag{2.10}
$$

donde $i \in \{1,...,n\}, \alpha, \beta \in \mathbb{R}$ y f_k, f_i son nuevamente expresiones MMPS. El símbolo $\vee$ corresponde al "o" lógico y la definición es claramente recursiva, ya que depende de otras expresiones MMPS. Un ejemplo para una expresión MMPS es $5x_1 - 3x_2 + 7 + \max(\min(2x_1, -8x_2), x_2 - 3x_3)$.

Ya definida una expresión MMPS, un sistema MMPS puede ser descrito como:

$$\begin{aligned} x_{k+1} &= M_x(x_k, u_k, d_k) \\ y_k &= M_y(x_k, u_k, d_k) \\ M_c(x_k, u_k, d_k) &\le c \end{aligned} \tag{2.11}$$

donde M_x, M_y y M_c son expresiones MMPS en términos de las componentes x_k, u_k y la variable auxiliar d_k.

2.2 Equivalencia entre modelos híbridos

Se puede demostrar (Heemels, 2001) que los cinco modelos mencionados en 2.2, son equivalentes. Es decir cada sistema puede ser escrito en la forma de otro tipo de los modelos. Para ejemplificar la equivalencia entre estos modelos, considere el siguiente sistema híbrido:

$$x_{k+1} = \begin{cases} x_k + u_k & si \;\; x_k + u_k \le 1 \\ 1 & si \;\; x_k + u_k > 1 \end{cases} \tag{2.12}$$

el cual representa un integrador inferiormente saturado, con las restricciones:

$$-10 \le x_k \le 10, \;\; -1 \le u_k \le 1 \tag{2.13}$$

el modelo presentado está en forma PWA con un espacio de estado/salida de dos dimensiones, particionado por el plano $x_k + u_k = 1$.

Para determinar la representación de la forma MLD del sistema (2.12) se crea la variable binaria $\delta_k \in \{1, 0\}$ y la variable z_k, para obtener:

$$x_{k+1} = z_k \tag{2.14}$$

junto con las desigualdades:

$$
\begin{aligned}
& x_k + u_k + 10\delta_k \le 11 \\
& -x_k - u_k (12 + \varepsilon)\delta_k \le -1 - \varepsilon \\
& -10\delta_k + z_k \le 1 \\
& 12\delta_k - z_k \le -1 \\
& -x_k - u_k + 12\delta_k z_k \le 12 \\
& x_k + u_k + 10\delta_k - z_k \le 10
\end{aligned}
\tag{2.15}
$$

En (2.15) las dos primeras inecuaciones provienen de traducir la expresión $[\delta_k = 1] \Leftrightarrow [x_k + u_k \le 1]$, y las otras de la traducción $z_k = (x_k + u_k)\delta_k + (1 - \delta_k)$ más las restricciones de las variables de estado y de la entrada.

El sistema (2.12) también es representable por un sistema MMPS por:

$$
x_{k+1} = x_k + u_k - \max(0, x_k + u_k - 1) \tag{2.16}
$$

Por último, la formulación LC del sistema (2.18) es:

$$
\begin{aligned}
& x_{k+1} = x_k + u_k - w_k \\
& v_k = -x_k - u_k + w_k + 1 \\
& 0 \le v_k \perp w_k \ge 0
\end{aligned}
\tag{2.17}
$$

donde se han introducido las variables complementarias v_k y w_k.

Es de particular importancia la equivalencia entre sistemas MLD y PWA, que son los utilizados en la formulación de sistemas para control predictivo híbrido. Los modelos MLD son utilizados en las primeras formulaciones de controladores predictivos híbridos basados en programación cuadrática entera mixta (Bemporad y Morari, 1999), y ambos tipos de modelos son utilizados en las formulaciones más recientes basadas en cálculos *offline* con programación multiparamétrica (Bemporad *et al.*, 2002). Esta última formulación basada en los dos tipos de modelos es posible gracias a la equivalencia entre ellos, que permite convertir la expresión del sistema de un modelo al otro, y así utilizar el modelo que más convenga.

3 Control predictivo basado en modelos para sistemas híbridos

3.1 Introducción

El control predictivo basado en modelos (*Model Predictive Control,* MPC) es un conjunto de métodos de control que hacen uso explícito de un modelo del proceso para obtener predicciones de sus salidas futuras. Las acciones de control se calculan utilizando este modelo para minimizar una función objetivo que depende de las salidas futuras predichas del modelo y de las variaciones en la acción de control (Camacho y Bordons, 2004). Estos métodos de control llevan a controladores que tienen básicamente la misma estructura y los siguientes elementos principales:

- Uso explícito de un modelo para predecir la evolución del proceso en los instantes futuros.
- Minimización de una función objetivo.
- Utilización de un horizonte de control finito y deslizante que implica el cálculo de la secuencia de control para todo el horizonte pero con la aplicación de la primera señal de la secuencia y la repetición de todo el proceso en el siguiente instante.

Los diversos algoritmos de MPC difieren unos de los otros en el tipo de modelo para representar el proceso, el tipo de restricciones utilizadas, el tipo de función objetivo a ser minimizada y en el modo de resolver los problemas de optimización (Camacho y Bordons, 2004). Independiente de estas diferencias, MPC presenta una serie de ventajas sobre otros métodos, entre las cuales destacan:

- Es muy atractivo para personal con un conocimiento limitado en la teoría de control pues los conceptos son muy intuitivos y al mismo tiempo la sintonización es relativamente simple.
- Puede ser utilizado para controlar una gran variedad de procesos, incluyendo procesos con dinámicas simples y complejas, con retardos largos, de fase no-mínima e inestables.
- Posee intrínsecamente compensación para tiempos muertos.
- Introduce control pre-alimentado en un modo natural para compensar las perturbaciones medibles.
- La extensión para el tratamiento de restricciones es conceptualmente simple, las cuales ser incluidas sistemáticamente durante el diseño del proceso.
- Es muy útil cuando las referencias futuras son conocidas.
- Es una metodología completamente abierta basada en principios básicos que permiten extensiones futuras.

Sin embargo, estos métodos también posee sus desventajas. Cuando la ley de control se puede encontrar de forma explícita, su derivación requiere un gran esfuerzo de diseño y computacional, mucho mayor que para la sintonización de un PID. En otras ocasiones, la ley no se puede encontrar explícitamente, y es necesario resolver un problema de optimización *online* que puede ser muy demandante computacionalmente, lo que representa un gran problema para procesos con dinámicas rápidas. Además de lo ya mencionado, la mayor de las desventajas es la necesidad de un modelo apropiado del proceso. El algoritmo se basa en la disponibilidad de un modelo del proceso y es independiente de sus parámetros (no así de su estructura), pero obviamente los beneficios obtenidos serán afectados por las discrepancias existentes entre el proceso real y el modelo utilizado.

La formulación de la estrategia básica del MPC se presenta en la Sección 3.2. Luego en la Sección 3.3 se presentan las estrategias particulares para sistemas híbridos. Finalmente, en la Sección 3.4 se concluye acerca de la necesidad de nuevas estrategias para el control predictivo de una gran gama de sistemas.

3.2 Fundamentos de control predictivo basado en modelos (Model based Predictive Control, MPC)

La metodología de todos los controladores pertenecientes a la familia del MPC está caracterizada por la siguiente estrategia, representada en la Figura 1.

1. Las salidas futuras para un horizonte de predicción determinado N_y, llamado el horizonte de predicción, son determinadas en cada instante t utilizando el modelo del proceso. Estas salidas futuras $y(t+k\,|\,t)$ [1], para $k=1,...,N_y$, dependen de los valores conocidos hasta el instante t (entradas y salidas pasadas) y de las acciones de control futuras $u(t+k\,|\,t), k=0,...,N_u-1$, que serán aplicadas al proceso y deben ser calculadas. N_u es el horizonte de control; en el problema de optimización sólo se busca encontrar las N_u acciones de control futuras, y se asume que después de este horizonte las acciones de control se mantienen constantes. Para calcular estas predicciones, se utilizan modelos que en su forma más general son no lineales, y dependen de las salidas y entradas pasadas.

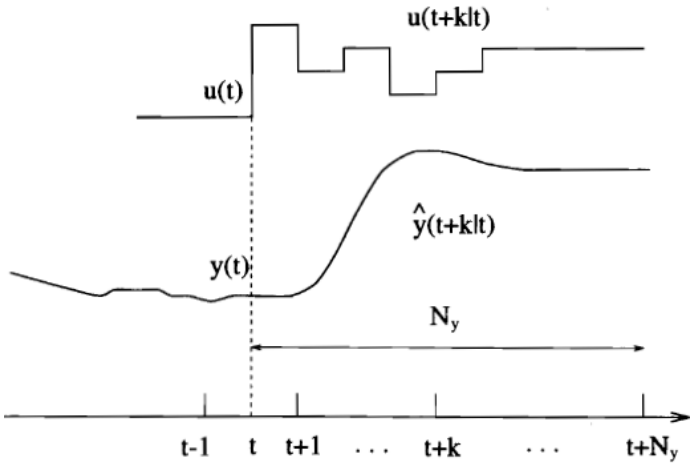


Figura 1: Estrategia de MPC

2. El conjunto de señales de control futuras es calculado al optimizar un criterio (o función objetivo) diseñado para mantener el proceso tan cerca como sea posible a una trayectoria de referencia $r(t+k)$ (que puede ser el mismo *setpoint* o una aproximación cercana de este, por ejemplo después de filtrarlo para evitar los escalones en la referencia). Este criterio por lo general es una forma cuadrática de los errores entre las salidas predichas y las trayectorias de referencia. El esfuerzo de control es incluido en el criterio en la mayoría de los casos. Una solución analítica puede ser encontrada si el criterio es cuadrático o lineal, y el modelo es lineal. De otro modo, se

[1] La notación indica el valor de la variable en el instante $t + k$ calculada en el instante t.

debe utilizar un método iterativo. A continuación se muestra la función objetivo más utilizada, donde el esfuerzo de control se considera como la variación en la acción de control de un instante a otro.

$$J(N_y, N_u) = \sum_{k=1}^{N_y} \delta(k)\left[\hat{y}(t+k\,|\,t) - r(t+k)\right]^2 + \sum_{k=1}^{N_u} \lambda(k)\Delta u(t+k-1)^2 \qquad (3.1)$$

donde $\delta(k), \lambda(k)$ representan pesos relativos de la importancia de cada componente del error en la salida y esfuerzo de control respectivamente, en la función objetivo.

3. La señal de control $u(t\,|\,t)$ es enviada al proceso, mientras que las siguientes señales de control no son utilizadas, pues en el siguiente instante de muestreo $y(t+1)$ ya es conocida y se repite el paso 1 con este nuevo valor y las otras secuencias actualizadas. Entonces se calcula $u(t+1\,|\,t+1)$ (que en principio será diferente al $u(t+1\,|\,t)$ ya calculado, debido a la nueva información disponible) utilizando el concepto de horizonte deslizante.

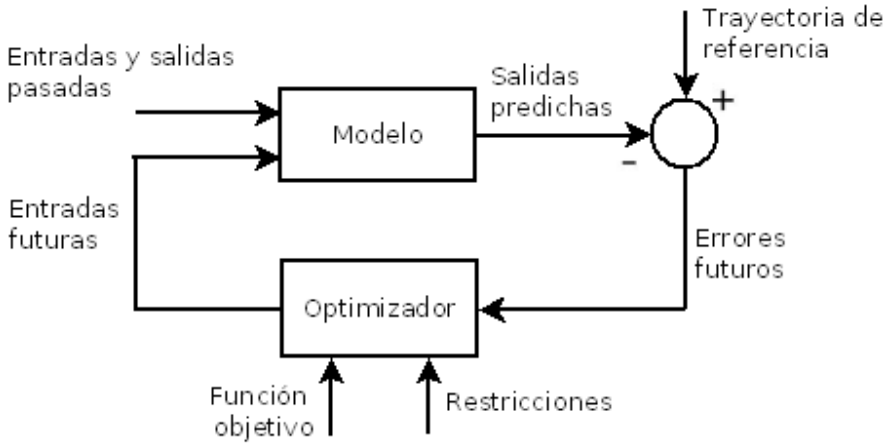


Figura 2: Estructura básica de MPC

Para implementar este tipo de estrategias, se utiliza la estructura básica de la Figura 2. El modelo se utiliza para predecir las salidas futuras del proceso, basado en los valores actuales y pasados y en las acciones de control futuras propuestas. Estas acciones son calculadas por el optimizador tomando en consideración la función objetivo y las restricciones.

En consecuencia, el modelo juega un rol decisivo en el controlador. El modelo escogido debe ser capaz de capturar las dinámicas del proceso para predecir de forma precisa las salidas futuras, siendo tan sencillo de implementar y fácil de comprender como sea posible.

En este trabajo el foco se encuentra en los modelos híbridos. A continuación se presentan las principales técnicas diseñadas y sus aplicaciones para el control predictivo de sistemas híbridos.

3.3 Control predictivo híbrido

Los sistemas híbridos representan una gran clase de sistemas que contienen variables continuas y discretas. En el contexto de MPC, el uso de modelos no lineales con variables continuas y

discretas ha sido considerado para obtener una mejor representación de las no linealidades y de la presencia de ambos tipos de variables en los procesos.

En el ámbito del diseño de sistemas de control predictivo híbrido, Slupphaug *et al.* (1997) y Slupphaug y Foss (1997) describen un control predictivo con entradas continuas y enteras utilizando programación entera mixta. Se demuestra que con este enfoque se logra un mejor desempeño que con una estrategia de control predictivo que separa de las variables continuas y discretas. Estas estrategias son aplicadas para en el control de nivel y temperatura en un sistema de estanques. Bemporad y Morari (1999) presentan un esquema predictivo para sistemas MLD incluyendo restricciones operacionales, que resuelven con programación cuadrática mixta-entera (*mixed integer quadratic programming*, MIQP). La metodología propuesta es aplicada en el control de un sistema de inyección de gasolina, el cual incorpora variables manipuladas enteras.

El problema principal de MIQP es su complejidad computacional para encontrar la solución. Para superar este problema, Thomas *et al*. (1994) proponen particionar el espacio de estados, donde se considera que ciertas variables no cambian, lo que simplifica el problema y disminuye el tiempo computacional. A su vez, Potocnik *et al*. (2005) proponen un algoritmo de control híbrido predictivo con entradas discretas basado en un análisis de alcanzabilidad (capacidad de conjunto de ser alcanzado por el estado de un sistema si es que existe una acción de control admisible), que también está formulado en el espacio de estados. Utilizando este análisis de alcanzabilidad, se poda un árbol de decisión para la solución del controlador predictivo híbrido que logra reducir el costo computacional respecto de la formulación como un MIQP. Los algoritmos son aplicados para el control óptimo de una planta batch de multiproductos (Potocnik *et al*., 2004). Karer et al. (2007) extiende la metodología basada en un análisis de alcanzabilidad para sistemas con entradas discretas utilizando un enfoque de modelación híbrida difusa. Los beneficios del algoritmo de MPC utilizando el modelo híbrido difuso propuesto son verificados en una simulación de un reactor batch y establecen que esta estrategia supera claramente a la que tan sólo utiliza un modelo lineal.

Recientemente se ha demostrado (Bemporad *et al.,* 2002) que un controlador predictivo híbrido se puede expresar como un programa multi-paramétrico, cuya solución resulta en un controlador que es afín a tramos en el espacio de estado. La idea es simple y fue sugerida por primera vez en Zafiriou (1990): el óptimo de un problema de programación cuadrática se alcanza en un conjunto de restricciones activas (el conjunto puede ser vacío), y para todos los puntos del espacio que tengan el mismo conjunto de restricciones activas, la solución es afín.

Borrelli *et al*. (2005) entregan resultados teóricos básicos acerca de la estructura de la solución óptima y el valor óptimo de la función en el problema de control óptimo de sistemas lineales híbridos de tiempo discreto, además de describir como la ley de control óptima puede ser construida combinando programación multi-paramétrica y dinámica, lo que permite mejorar considerablemente la carga computacional respecto de los programas multi-paramétricos simples. Sin embargo, el algoritmo está limitado a modelos PWA (y sus equivalentes), con estados y entradas continuas, y requiere un pesado procesamiento computacional *offline* para sintetizar las leyes de control óptimo para cada elemento de una partición poliédrica del espacio de estados.

Últimamente, muchos trabajos han estado dirigidos a optimizar procedimientos altamente demandantes que ya han sido previamente establecidos. Así, Baric *et al*. (2008) presentan un algoritmo para el cálculo de leyes control óptimas explicitas para sistemas PWA con índices de desempeño poliédricos, que es una extensión del Algoritmo de Borrelli. Este método se encuentra

basado en programación dinámica, y logra mejorar la eficiencia del procedimiento *offline* aprovechando la estructura geométrica del problema de optimización. Sin embargo la metodología también se encuentra limitada para sistemas PWA con entradas y estados continuos. Por su parte, Geyer *et al.* (2008) plantean un método para reducir la complejidad de las particiones poliédricas de los sistemas PWA, basándose en la noción de arreglos de hipar-planos, en el cual se entrega un sistema poliédrico por tramos equivalente que es mínimo en el número de poliedros. Además, la reducción en la complejidad es aumentada considerando un grado de error ajustable. Esta metodología es utilizada para reducir la complejidad de las leyes de control PWA explícitas, manteniendo la optimalidad de las soluciones encontradas, o con un mínimo de reducción en la precisión al utilizar el enfoque basado en el grado de error ajustable. Se logran reducciones de complejidad de hasta un orden de magnitud. La desventaja del método es ser de orden exponencial respecto del número de híper-planos, de modo que los autores plantean el uso de estrategias de dividir-y-conquistar para enfrentar problemas de alta complejidad. Por ejemplo, esta metodología es aplicada para el control de un convertidor AC-DC trifásico de dos niveles, cuya ley PWA de realimentación posee alrededor de 8500 poliédros.

Los enfoques revisados hasta el momento calculan la acción de control de modo *offline* para tener una expresión explícita, o bien la obtienen resolviendo un problema de optimización *online*. La complejidad de los modelos de los procesos puede ocasionar que los cálculos *offline* u *online* sean irrealizables, por asuntos de memoria o las restricciones computacionales de las aplicaciones en tiempo real. Luego, ambas alternativas pueden no ser aplicables en una importante gama de procesos. Por esto, los trabajos más recientes han intentado encontrar una combinación entre un cálculo *offline* y *online* de la acción de control, que permita la aplicación en tiempo real del cálculo *online* y utilizando los resultados de un cálculo previo *offline*. Muchos trabajos han adoptado esta línea pero para el caso MPC continuo. Zeilinger *et al*. (2008) proponen un esquema donde se usa una aproximación de la ley de control óptima PWA que es calculada *offline*, para dar una buena solución inicial (*warm-start*) a la optimización *online*. La optimización ejecuta un número finito de iteraciones de conjuntos activos hasta entregar una acción de control sub-óptima pero factible, que es luego aplicada al sistema. La idea es escoger un buen compromiso entre la complejidad de la aproximación PWA y el número de iteraciones *online* necesarias, para satisfacer restricciones de tiempo de cómputo, almacenamiento y desempeño. Se presentan condiciones que aseguran que la solución sub-óptima es estable, factible, y posee un deterioro acotado. Jones y Morari (2009) toman otro enfoque, que parte de la solución para sistemas PWA propuesta en Borrelli *et al*. (2009), que consiste en encontrar una solución aproximada de la acción de control óptima, y luego interpolarla para encontrar la acción a ejecutar. La idea es encontrar la acción de control óptima en sólo un número limitado de puntos del espacio, y para el resto realizar una simple interpolación en el espacio no convexo. Se presentan además condiciones de estabilidad menos restrictivas que en otro enfoques similares.

Dentro de la variada gama de aplicaciones recientes del control predictivo híbrido, la mayoría se encuentra basada en optimización multiparamétrica. Dentro de estas se puede mencionar a Giorgetti *et al.* (2006), donde se utiliza control predictivo híbrido para controlar la razón aire/combustible y torque en la tecnología de inyección de gasolina, y así lograr que el sistema de inyección pueda operar en dos estados discretos (estratificado y homogéneo). En este trabajo primero se muestra como encontrar la acción de control utilizando programación entera mixta en modo *online*, y luego se convierte ese controlador en una expresión PWA utilizando optimización multiparamétrica, lo que reduce sustancialmente el tiempo requerido para su aplicación práctica. En del Real *et al.* (2007), logran integrar paneles fotovoltaicos con una celda de combustible de modo que se mantenga controlado el voltaje en una micro-red. Bemporad *et al*. (2007)

implementa un sistema de control para una correa transportadora, cuyo estado es medido por sensores infrarrojos. El sistema se comporta como un sistema MLD. Así, se aplica una estrategia de control predictivo híbrido basado en optimización multiparamétrica, cuya salida son las referencias para los controladores locales del sistema. En Beccuti *et al*. (2007) se implementa un control predictivo híbrido basado en optimización multiparamétrica para un convertidor Boost DC-DC, utilizando la configuración híbrida del circuito. Finalmente en Geyer *et al.* (2008) se realiza un modelo un convertidor DC-DC válido para todas las zonas de operación. Bajo este modelo se logra realizar un control óptimo con restricciones

El resto de las aplicaciones, en su mayoría, se basan en la resolución de un problema de optimización *online*. En Negenborn *et al.* (2007) se logra controlar el voltaje en un sistema eléctrico de potencia, donde el modelo se obtiene mediante linealización en el punto de operación, y la acción de control óptima se encuentra resolviendo el problema del control predictivo híbrido mediante la aplicación de programación entera mixta lineal en modo *online*. Julius *et al.* (2007) que logran la regulación del porcentaje de la bacteria Escherichia coli en la lactosa, donde el modelo usado describe la colonia a escala dinámica como un proceso de Markov. Para encontrar la acción de control óptima, también se resuelve un problema de optimización entera mixta lineal en modo *online*. Por su parte, Wei *et al.* (2007) propone una estrategia de control predictivo híbrido para un robot con ruedas considerando deslizamiento de ellas. Las acciones de control óptimas son encontradas adaptando el problema para poder aplicar directamente técnicas clásicas de optimización para problemas no lineales tales como programación secuencial cuadrática (*Sequential quadratic programming*, SQP). La misma estrategia es utilizada más recientemente por Uthaichana *et al.* (2008), basados en un modelo de operación de un vehículo eléctrico, donde se desarrolla un controlador de su potencia mecánica.

A continuación se presentan las principales estrategias para el control predictivo híbrido según la revisión realizada. En 3.3.1 se describe el algoritmo de control predictivo híbrido para sistemas formulados como modelos MLD que está basado en la resolución del problema de optimización *online* con programación cuadrática entera-mixta. En 3.3.2 se presenta la formulación para sistema modelados con PWA y su resolución con optimización multiparamétrica. Luego, en 3.3.3 se presenta la estrategia basada en análisis de alcanzabilidad para sistemas con entradas únicamente discretas. En 3.3.4 se presenta una estrategia de control de sistemas híbridos en la cual puede reducirse el problema MINLP a uno NLP. De las estrategias revisadas, ninguna es para modelos generales, es decir, todas asumen alguna estructura bien definida para los modelos. Por esto en 3.3.5 se presenta una formulación que permita abarcar los modelos más generales posibles de los procesos. Finalmente en la Sección 3.4 se presenta un análisis de las estrategias presentadas, y a partir de este análisis se justifica la necesidad de recurrir a una nueva familia de técnicas, la de algoritmos evolutivos, para resolver el problema de optimización en control predictivo para una clase general de modelos híbridos.

3.3.1 Control predictivo de sistemas MLD basado en programación entera mixta

Para un sistema híbrido MLD descrito por la ecuación (2.1), el problema de control predictivo se puede formular como (Bemporad y Morari, 1999):

$$\min_{\{u_0^{N-1}\}} J(u_0^{N-1}, x(t)) = \sum_{k=1}^{N} \left\| u(t+k-1) - u_r \right\|_{Q_1}^p + \left\| \delta(t+k-1|t) - \delta_r \right\|_{Q_2}^p$$
$$+ \left\| z(t+k|t) - z_r \right\|_{Q_3}^p + \left\| x(t+k|t) - x_r \right\|_{Q_4}^p + \left\| y(t+k|t) - y_r \right\|_{Q5}^p$$

$$s.a. \begin{cases} x(t+k|t) = Ax(t+k-1|t) + B_1 u(t+k-1) + B_2 \delta(t+k-1|t) + B_3 z(t+k|t) \\ y(t+k|t) = Cx(t+k|t) + D_1 u(t+k) + D_2 \delta(t+k|t) + D_3 z(t+k|t) \\ E_1 x(t+k|t) + E_2 u(t+k-1) + E_3 \delta(t+k-1|t) + E_4 z(t+k|t) \le g \\ k = 1, \ldots, N \end{cases} \quad (3.2)$$

sujeto a las ecuaciones (1), donde $\|x\|_Q^P$ denota $x^T Q x$ cuando $p = 2$ (norma-2), $Q\|x\|_p$ para $p = 1$ (norma-1) ó p = ∞ (norma-∞) y $Q_i = Q_i^T > 0, i = 1, \ldots, 5$, son matrices de peso de dimensiones apropiadas y todas las señales se predicen con la información disponible hasta el instante *t* en la forma usual del control predictivo, es decir, $x(k|t) \triangleq x(t+k, x(t), u_0^{T-1})$, y de igual modo para $\delta(k|t),\ z(k|t), y(k|t)$. Los vectores $x, u, \delta, z, x_r, \delta_r, z_r$ son los vectores de estados futuros predichos, movimientos de control, variables lógicas auxiliares, variables reales auxiliares, y sus referencias futuras correspondientes (que por supuesto deben satisfacer las ecuaciones del modelo). Es importante notar que las restricciones operacionales del proceso, que si bien no forman parte del modelo del proceso propiamente tal, pueden ser incluidas en éste para efectos de resolver el problema de optimización. Así, el control predictivo encuentra la acción de control óptima en *N* instantes (*N* es el horizonte de predicción), incluidos el actual y los $N-1$ siguientes, dada una función objetivo que considera costos cuadráticos en la desviación de los valores de las variables respecto de sus referencias para los tiempos presentes y futuros, cumpliendo las restricciones operacionales y de la dinámica del sistema.

Esta formulación del control predictivo híbrido resulta en un problema de optimización de una función cuadrática si $p = 2$, o de una función lineal si $p = 1 \vee p = \infty$, un conjunto de restricciones lineales y con variables de decisión reales y enteras. Si $p = 2$, la formulación es la correspondiente a un MIQP, y si $p = 1 \vee p = \infty$ la formulación corresponde a un problema de programación lineal entera-mixta (*mixed integer linear program*, MILP). Con la excepción de estructuras particulares, los problemas de programación entera con variables binarias se clasifican como NP-completos, así en el peor caso el tiempo de solución crece exponencialmente con el tamaño del problema (Raman y Grossman, 1991). Se debe destacar que en (3.2), en lugar de utilizar norma-2 también se podría utilizar norma-1 o norma-∞, en cuyo caso la formulación correspondería a un problema de programación lineal entera-mixta (*mixed integer linear program*, MILP). Sin embargo, en general es recomendable utilizar la norma-2, pues permite respuestas más suaves de la salida de los procesos.

Fletcher y Leyffer (1995) indican que los métodos de descomposición generalizada de benders (*Generalized Benders' Decomposition*, GBD) (Lazimy, 1985), aproximaciones externas (O*uter aproximations*, OA), ramificar y acotar (*branch and bound*, BB) basado en programación lineal (LP) y programación cuadrática (QP), y ramificar y acotar son los métodos principales para los problemas MIQP. Vale la pena mencionar que estos métodos son generalizables fácilmente para resolver problemas de programación entera mixta no lineal (*Mixed integer non linear programming*, MINLP). Muchos autores están de acuerdo en que los métodos BB son los más

exitosos para problemas MIQP. Fletcher y Leyffer (1995) muestran numéricamente que los métodos BB superan a los demás por un orden de magnitud.

Es importante mencionar que esta estrategia ha perdido popularidad debido a que se ha encontrado que para sistemas PWA (o cualquier sistema MLD dada la equivalencia de éstos con los PWA), en un marco de control predictivo híbrido lineal o cuadrátic,o la ley de realimentación es una expresión PWA dependiente del estado del sistema. De este modo la complejidad disminuye considerablemente desde ser un problema NP a la de simplemente evaluar una expresión PWA. La importancia de presentar esta formulación radica en el hecho de presentar como ha sido resuelto el problema de optimización *online*, y las complejidades que este tiene para sistemas MLD, lo que permite entender las complejidades que aparecen en un caso extendido no lineal.

3.3.2 Control predictivo de sistemas PWA/MLD basado en programación multi-paramétrica

Bemporad *et al*. (2002) presentan la formulación del control predictivo híbrido tomando como base la presentada en el punto anterior; pero con la inclusión de programación multi-paramétrica es posible encontrar una expresión PWA explícita para la ley de control, sin tener la necesidad de resolver un problema MIQP en cada instante de muestreo. Se considera el siguiente sistema PWA restringido (*constrained* PWA, CPWA):

$$\begin{aligned} x(k+1) &= A_i x(k) + B_i u(k) + f_i \\ y(k) &= C_i x(k) + D_i u(k) + g_i \end{aligned} \quad \text{para} \quad \begin{bmatrix} x(k) \\ u(k) \end{bmatrix} \in \tilde{C}_i = \left\{ \tilde{H}_x^i x(k) + \tilde{H}_u^i u \le \tilde{K}^i \right\} \tag{3.3}$$

Y el problema de control predictivo puede ser formulado como:

$$\begin{aligned} \min_{\left\{u_0^{N-1}\right\}} J(u_0^{N-1}, x(t)) = & \sum_{k=1}^{N} \left\| u(t+k-1) - u_r \right\|_{Q_1}^p + \left\| \delta(t+k-1|t) - \delta_r \right\|_{Q_2}^p \\ & + \left\| z(t+k|t) - z_r \right\|_{Q_3}^p + \left\| x(t+k|t) - x_r \right\|_{Q_4}^p + \left\| y(t+k|t) - y_r \right\|_{Q5}^p \\ s.a. & \begin{cases} x(t+k) = A_i x(t+k-1) + B_i u(t+k-1) + f_i \\ y(t+k) = C_i x(t+k) + D_i u(t+k) + g_i \end{cases} \text{si} \begin{bmatrix} x(t+k) \\ u(t+k-1) \end{bmatrix} \in \tilde{C}_i \\ & k = 1, \ldots, N \end{aligned} \tag{3.4}$$

donde $\|x\|_Q^p$ representa la norma-p para $p = 1, 2, \infty$. $Q_i, i = 1, \ldots, 5$ son matrices de peso con las siguientes propiedades: $Q_i = Q_i^T \ge 0, i = 2, \ldots, 5$, $Q_1 \ge 0$, para $p = 2$, y además son de rango completo para $p = 1, \infty$. $k = 1, \ldots, N$ son los pasos de predicción y u_0^{N-1} es la secuencia óptima de acciones de control encontrada por el algoritmo de optimización. De acuerdo a la estrategia de horizonte deslizante, solo el primer elemento de u_0^{N-1} es aplicado al proceso.

El problema de control predictivo presentado en (3.4) también puede ser resuelto aplicando el modelamiento de sistemas MLD. Considerando el modelo MLD equivalente al PWA utilizado, el problema (3.4) se rescribe como:

$$\min_{\{u_0^{N-1}\}} J(u_0^{N-1}, x(t)) = \sum_{k=1}^{N} \left\| u(t+k-1) - u_r \right\|_{Q_1}^{p} + \left\| \delta(t+k-1|t) - \delta_r \right\|_{Q_2}^{p} + \left\| z(t+k|t) - z_r \right\|_{Q_3}^{p} + \left\| x(t+k|t) - x_r \right\|_{Q_4}^{p} + \left\| y(t+k|t) - y_r \right\|_{Q5}^{p}$$

$$s.a. \begin{cases} x(t+k|t) = Ax(t+k-1|t) + B_1 u(t+k-1) + B_2 \delta(t+k-1|t) + B_3 z(t+k-1|t) \\ y(t+k|t) = Cx(t+k|t) + D_1 u(t+k) + D_2 \delta(t+k|t) + D_3 z(t+k|t) \\ E_1 x(t+k|t) + E_2 u(t+k-1) + E_3 \delta(t+k-1|t) + E_4 z(t+k|t) \le g \end{cases} \quad (3.5)$$

El problema (3.5), como ya ha sido mencionado, puede ser formulado como un problema MIQP si se utiliza norma-2 o como un MILP si se utiliza norma-1 ó norma-∞.

La solución óptima del problema de programación multi-paramétrica no es un número como en el caso de la optimización convencional, sino una función óptima $z^*(x)$ de un parámetro x perteneciente a un espacio acotado X. La solución utiliza el concepto de regiones críticas $Cr^i \quad i = 1,...,r$, las cuales generan una partición $\bigcup_{i=1}^{r} Cr^i = X$ del espacio de parámetros, en cada una de las cuales se logra definir una solución óptima afín $z^*(x) = F^i x + g^i$ del problema de optimización.

Dado el valor del estado actual (inicial) $x(t|t)$, el MIQP (o MILP) puede ser resuelto para obtener la secuencia óptima de acciones de control u_0^{N-1}. Si se aplica optimización multi-paramétrica, se puede obtener una expresión explícita para la ley de control óptima $u(x(t|t))$ dado el estado actual. Al utilizar la norma-2 el problema de optimización es tratado como un MIQP multi-paramétrico (mp-MIQP), mientras que para $p = 1, \infty$, el problema de optimización puede ser tratado como un MILP multi-paramétrico (mp-MILP). La solución al mp-MILP (mp-MIQP) es una ley de control definida explícitamente de la forma:

$$u(x(t|t)) = F_i^0 x(t|t) + G_i^0, \; si \; x(t) \in P_i^0 \quad (3.6)$$

donde $P_i^0, i = 1,..., N^0$, es una partición poliédrica del conjunto de estados factibles $x(t|t)$ al tiempo t, y es diferente para normas-1, normas-2 ó normas-∞.

Para este tipo de solución el tiempo necesario para encontrar la acción de control es claramente menor que cuando se requiere resolver un MIQP/MILP en tiempo real, ya que el control consiste sólo en acceder una tabla, en lugar de tener que resolver en cada instante un pesado problema de optimización.

La teoría de estabilidad del control predictivo basado en optimización multi-paramétrica es derivada utilizando la formulación basada en modelos PWA. Sin embargo utilizar esta formulación implica la necesidad de enumerar todas las posibles secuencias de cambios de los modelos afines. Entonces la formulación se cambia por una basada en un modelo MLD, pues de esta forma se puede evitar esta enumeración y utilizar metodologías más eficientes, sin perder las propiedades de estabilidad, gracias a la equivalencia entre modelos PWA y MLD.

Es importante notar que el enfoque mp-MILP (MIQP) resuelve el problema en un espacio extendido (x,u,δ,z) (δ son variables lógicas auxiliares, y z son variables reales auxiliares), que al tener más variables binarias, aumenta la complejidad computacional de modo exponencial. En Baotic *et al*. (2003) se presenta un algoritmo eficiente para calcular la solución para el caso de normas $p=1,\infty$, y en Borrelli *et al*. (2005) se presenta el caso para norma $p=2$. En ambos enfoques el problema es resuelto en un espacio (x,u) y el aumento de complejidad computacional de forma exponencial desaparece. Los algoritmos están basados en programación dinámica y problemas mp-LP/QP. La limitación de estos enfoques es que se encuentran restringidos a sistemas híbridos donde los estados $x(t)$ y entradas $u(t)$ están definidos en $x(t)\in\mathbb{R}^{n_c}$ y $u(t)\in\mathbb{R}^{m_c}$, respectivamente. Cuando este no es el caso, resolver los problemas mp-MILP en el espacio extendido puede no ser factible, y otra metodología es necesaria para enfrentar el problema.

3.3.3 Control predictivo basado en análisis de alcanzabilidad

El enfoque de control predictivo basado en un análisis de alcanzabilidad es válido para procesos híbridos con entradas únicamente discretas, que pueden estar modelados por sistemas PWA, MLD, u otros más generales tales como sistemas difusos o redes neuronales. Esta metodología nace como una eficiente alternativa al control predictivo basado en programación multi-paramétrica que, cuando los sistemas poseen entradas discretas, poseen una altísima complejidad computacional (como ha sido mencionado en el punto anterior). El algoritmo abstrae el comportamiento del sistema híbrido al construir un árbol de evolución. Una función de costos está asociada con cada nodo del árbol, y en base a ésta se explora el árbol. Tan pronto se termina de recorrer el árbol se puede encontrar la acción de control óptima (que minimice los costos en los nodos), ésta se aplica al proceso, y luego el procedimiento se repite para el instante siguiente (Potocnick *et al*., 2005).

La solución a un problema de control en cada tiempo t es la secuencia de control $V_0^{H-1}=\{v(0),...,v(H-1)\}$ donde $u(t)=v(0)$ representa la entrada discreta al sistema en el tiempo t. Si el sistema posee m entradas discretas sin entradas continuas, luego $v(k)\in\{0,1\}^m$ y $V_0^{H-1}\in\{0,1\}^{m\cdot H}$. Como las entradas son binarias, hay $2^{m_b\cdot H}$ combinaciones posibles para V_0^{N-1}. Luego el problema de optimización es NP *hard* y el tiempo computacional requeridos para resolver el problema crece exponencialmente con su tamaño.[2]

En general no todas las combinaciones de entradas son factibles. Un modo de descartar entradas factibles es realizar un análisis de alcanzabilidad. Con este análisis se puede obtener los estados los estados alcanzables del sistema. Si bien enumerarlos todos no sería eficiente, la mayor parte de ellos estarán lejos de la trayectoria óptima. Entonces es razonable combinar el análisis con procedimientos que pueden detectar estados alcanzables que no llevan a una solución óptima y removerlos del procedimiento de exploración. Así, el procedimiento es básicamente una estrategia de BB.

[2] La metodología se ejemplifica para entradas binarias, pero es trivialmente expansible a entradas discretas más generales.

Sea $x(t+k|t)$ el estado al instante $t+k$ $(k=0,...,H-1)$. El análisis de alcanzabilidad calcula todos los estados posibles $x^i(t+k+1|t)$, que son alcanzables en el siguiente instante discreto. Si el sistema posee *m* entradas discretas, entonces pueden existir 2^m posibles siguientes estados. Sin embargo, debido a posibles restricciones en el proceso (o impuestas por la estrategia de control), el número de estados posibles es en realidad menor.

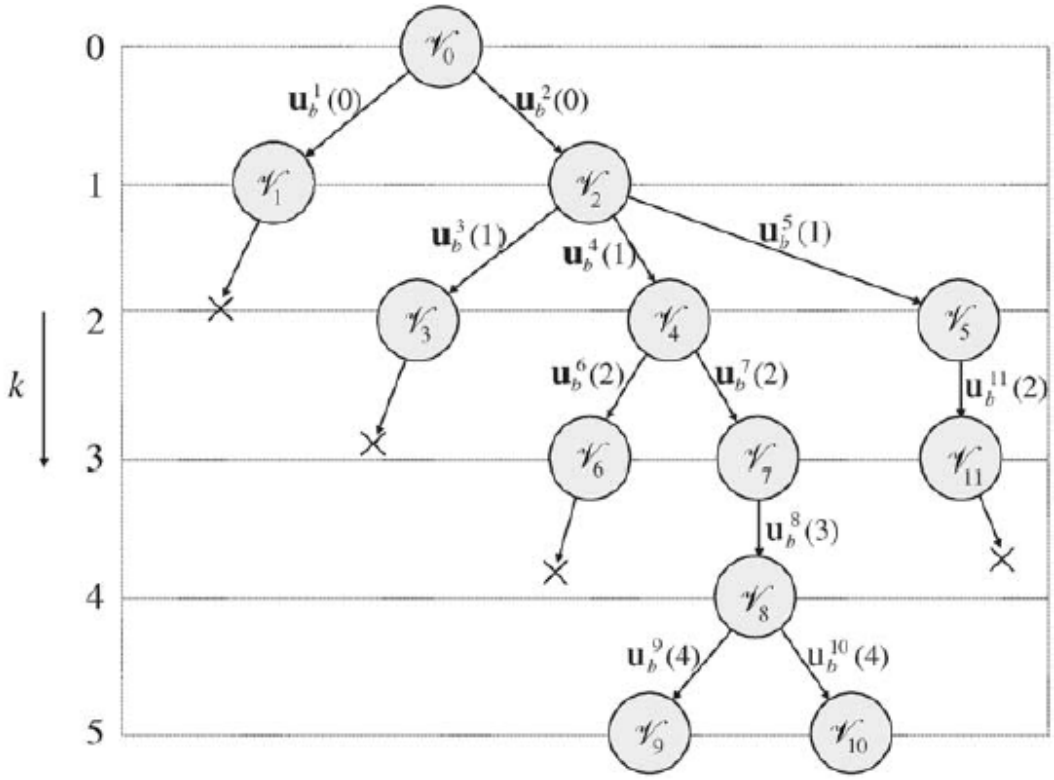


Figura 3: Árbol de evolución.
(Figura tomada de Potocnick *et al.*, 2005)

Entonces, de acuerdo a lo anterior, se puede abstraer la evolución del sistema a lo largo de un horizonte de H pasos (se cambia la notación de horizonte a *H* para no confundir con N_i, que se refiere la nodo *i*) en un árbol de evolución, como se muestra en la Figura 3. Los nodos del árbol representan los estados alcanzables, y las ramas conectan dos nodos si existe una transición posible entre los estados correspondientes. Cada rama tiene una entrada discreta asociada que al ser aplicada al sistema provoca la transición. Para un nodo raíz dado N_0, que representa el estado actual (inicial) $x(t|t)$, los estados alcanzables $x^i(t+k|t)$ son calculados e insertados en el árbol como nodos N_i, mientras que las correspondientes entradas discretas $v^i(k)$ son asociadas con las correspondientes ramas que conectan dos nodos. $i \in \{1,2,...\}$ representa los índices sucesivos de los nodos y las ramas insertadas en el árbol. Para cada nuevo nodo N_i un costo J_i es asociado. La búsqueda de una secuencia de control óptima se pasa a un nuevo nodo inicial, cuya selección está basada en su costo J_i. Tan pronto como se escoja un nuevo nodo inicial se calculan nuevos estados alcanzables. La construcción del árbol de evolución sigue de acuerdo a una estrategia de búsqueda en profundidad (*depth-first*), hasta que se cumpla una de las siguientes condiciones:

- Se llegue al horizonte $(k = H)$.
- El valor de la función de costos en el nodo actual es mayor que el valor óptimo actual ($J_i \geq J_{opt}$, donde inicialmente $J_{opt} = \infty$).

Un nodo que satisfaga una de las condiciones de arriba se marca como *explorado*. Si un nodo satisface la primera condición, el valor asociado de la función de costo se convierte en el valor óptimo actual, y la secuencia de control V_0^{H-1} que lleva del nodo inicial N_0 al nodo actual N_i es la solución actual. Si un nodo satisface la segunda condición, entonces se puede garantizar que esa secuencia de acciones de control no lleva a la solución óptima, ya que al avanzar en el árbol sólo se conseguirá aumenta el valor de la función de costos de la secuencia. Entonces todos los nodos a los que se puede llegar a través de este se marcan como explorados, y así se garantiza que no sigan siendo explorados. . La exploración continua hasta que no haya nodos no explorados en el árbol, y entonces la entrada $u(t) = v(0)$ es aplicada al sistema y luego el proceso se repite en $t+1$.

La función de costos corresponde a la misma utilizada en la formulación del control predictivo.

$$J\left(x(t \mid t), V_0^{H-1}\right) = \left\|x(t+H \mid k) - x_f\right\|_P^p + \sum_{k=1}^{H} \left\|x(t+k \mid t) - x_f\right\|_Q^p + \left\|v(k-1) - v_f\right\|_R^p \qquad (3.7)$$

donde $\|x\|_Q^p$ representa la norma-p para $p = 1, 2, \infty$, y además Q, R y P son matrices de peso que cumplen $Q = Q' \geq 0$, $R = R' \geq 0$ y $P \geq 0$, y además son de rango completo para $p = 1, \infty$.

Así, en cada nodo V_i la función de costos J_i corresponde a esta misma función objetivo, pero truncada hasta el instante de predicción $h < H$ a la que corresponda el nodo.

$$J_i\left(x(t \mid t), V_0^{h-1}, h\right) = \sum_{k=1}^{h} \left\|x(t+k \mid t) - x_f\right\|_Q^p + \left\|v(k-1) - v_f\right\|_R^p \qquad (3.8)$$

Como ya ha sido mencionado, si el valor de la función de costo es más alto que la solución óptima actual ($J_i \geq J_{opt}$) podemos asegurar que al continuar la exploración no se puede encontrar ninguna mejor solución que la solución óptima actual ($J_i \geq J_{opt}$) . Esto es así ya que la función de costos es monótonamente creciente con el paso de predicción. En efecto:

$$\begin{aligned} & J_i\left(x(t \mid t), V_0^h, h+1\right) - J_i\left(x(t \mid t), V_0^{h-1}, h\right) = \\ & = \sum_{k=2}^{h+1} \left\|x(t+k \mid t) - x_f\right\|_Q^p + \sum_{k=2}^{h+1} \left\|v(k-1) - v_f\right\|_R^p - \sum_{k=1}^{h} \left\|x(t+k \mid t) - x_f\right\|_Q^p - \sum_{k=1}^{h} \left\|v(k-1) - v_f\right\|_R^p = \\ & = \left\|x(t+h+1 \mid t) - x_f\right\|_Q^p + \left\|v(h) - v_f\right\|_R^p \geq 0 \quad \text{para } k = 1, \ldots, H-1. \end{aligned} \qquad (3.9)$$

Finalmente, es relevante notar que en esta metodología se puede utilizar cualquier tipo de modelos de sistemas híbridos, ya que no hay ninguna suposición respecto de su estructura, considerando que los procesos sólo aceptan entradas discretas.

3.3.4 Control óptimo de sistemas híbridos no lineales inmerso en optimización continua

Wei *et al*. (2007a) desarrolla una estrategia para control predictivo de sistemas híbridos con *switches* controlados (entradas discretas) y autónomos (estados discretos) basado en la formulación teórica para control óptimo desarrollada por Bengea y de Carlo (2005).

Se consideran sistemas con *switches* controlados y autónomos, cuyo comportamiento se encuentra descrito por cuatro cantidades: (i) el estado discreto $\xi(t) \in D_\xi = \{1, 2, ..., d_\xi\}$, (ii) el estado continuo $x(t) \in \mathbb{R}^n$, (iii) la entrada controlada discreta $\alpha(t) \in D_\alpha = \{1, 2, ..., d_\alpha\}$, y (iv) la entrada controlada continua $u(t) \in \mathbb{R}^m$. El estado discreto del sistema está asociado a los *switches* autónomos. Sin embargo, se considera sólo sistemas *sin memoria*, esto quiere decir que la evolución de los *switches* sólo depende de los estados y entradas continuas presentes, y no de los estados discretos. Así, formalmente la evolución del estado discreto en un sistema sin memoria está definida por una función continua por tramos $\eta : \mathbb{R}^n \times \mathbb{R}^m \to D_\xi$:

$$\xi(t) = \eta(x, u) \tag{3.10}$$

Sea $M_i \subseteq \mathbb{R}^n \times \mathbb{R}^m, i \in D_\xi$ el conjunto de pares (x, u) que corresponden al estado discreto $i \in D_\xi$. Entonces se considera que la ecuación de estado está dada por un conjunto de funciones $f_{(i,j)} : M_i \to \mathbb{R}^n$, $f_{(i,j)} \in C^1$, $i \in D_\xi, j \in D_\alpha$ que dependen del estado discreto i y de la entrada discreta α. Así, la evolución del estado se encuentra dado por:

$$\dot{x}(t) = f_{(\eta(x(t),u(t)),\alpha)}\big(x(t), u(t)\big), \quad x(t_0) = x_0 \tag{3.11}$$

Ahora, como el estado discreto $\xi(t)$ depende completamente del estado continuo $x(t)$ y de la entrada continua $u(t)$, se puede redefinir el conjunto de funciones C^1 por un conjunto de funciones C^1 solamente por tramos, pero que dependen solo de la entrada discreta α (así el efecto del estado discreto se refleja en los distintos tramos C^1 de las funciones de la colección). Así, (3.11) se puede rescribir fácilmente como:

$$\dot{x}(t) = f_{(\alpha)}\big(x(t), u(t)\big), \quad x(t_0) = x_0 \tag{3.12}$$

donde $f_{(\alpha)}$ es C^1 por tramos, como ya ha sido mencionado. Considerando este sistema, se requiere encontrar las acciones de control $\alpha(t)$ y $u(t)$ en los intervalos $\left[t_0, t_f\right]$, tal que se satisfagan los estados iniciales y condiciones terminales: $\big(t_0, x(t_0)\big) \in T_0 \times B_0$ y $\big(t_f, x(t_f)\big) \in T_f \times B_f$. Entonces se define el funcional de optimización:

$$J_C\left(t_0, x_0, u, \alpha\right) = g\left(t_0, x_0, t_f, x_f\right) + \int_{t_0}^{t_f} f^0_{\alpha(t)}\big(x(t), u(t)\big)\, dt \tag{3.13}$$

donde $g\left(t_0, x_0, t_f, x_f\right)$ es una función real C^1 que penaliza la condición final. Mientras que $f_i^0\left(x(t), u(t)\right)$ en una función C^1 por tramos para cada $i \in D_a$, que penaliza la evolución del sistema y las acciones de control aplicadas desde el estado inicial al final. Entonces, se define el problema de control óptimo híbrido (*Hybrid Optimal Control*, HOC):

$$\begin{aligned} &\min_{\alpha,u} J_C\left(t_0, x_0, u, \alpha\right) \\ &s.a. \\ &\dot{x}(t) = f_{(\alpha)}\left(x(t), u(t)\right), \quad x(t_0) = x_0 \\ &\left(t_0, x_0, t_f, x_f\right) \in T_0 \times B_0 \times T_f \times B_f \\ &\alpha(t) \in D_\alpha, u(t) \in \Omega, \forall t \in \left[t_0, t_f\right] \end{aligned} \tag{3.14}$$

Este problema debe ser inmerso en otro más general para ser resuelto. Así se introducen d_α (recordar que $\alpha(t) \in D_\alpha = \{1, 2, ..., d_\alpha\}$) nuevas variables $\alpha_i \in [0,1], i \in D_\alpha$ que satisfacen $\sum_{i=1}^{d_\alpha} \alpha_i(t) = 1$. Se define además un nuevo modelo del sistema (3.15), basado en una colección de campos vectoriales (uno para cada variable α_i), una acción de control continua u_i para cada campo vectorial, y una nueva funcional de optimización (3.16):

$$\dot{x}(t) = \sum_{i=1}^{d_\alpha} \alpha_i f_{(i)}\left(x(t), u_i(t)\right), \quad x(t_0) = x_0 \tag{3.15}$$

$$J_E\left(t_0, x_0, u, \alpha\right) = g\left(t_0, x_0, t_f, x_f\right) + \int_{t_0}^{t_f} \left(\sum_{i=1}^{d_\alpha} \alpha_i f_{\alpha(t)}^0\left(x(t), u_i(t)\right) \right) dt \tag{3.16}$$

Y entonces, el HOC se ha convertido en un problema de control óptimo inmerso (*Embedded Optimal Control*, EOC):

$$\begin{aligned} &\min_{u} J_E\left(t_0, x_0, u, \alpha_i\right) \\ &s.a. \\ &\dot{x}(t) = \sum_{i=1}^{d_\alpha} \alpha_i f_{(i)}\left(x(t), u(t)\right), \quad x(t_0) = x_0 \\ &\left(t_0, x_0, t_f, x_f\right) \in T_0 \times B_0 \times T_f \times B_f \\ &\alpha_i(t) \in [0,1], i \in D_\alpha, \forall t \in \left[t_0, t_f\right] \\ &u(t) \in \Omega, \forall t \in \left[t_0, t_f\right] \\ &\sum_{i=1}^{d_\alpha} \alpha_i(t) = 1 \end{aligned} \tag{3.17}$$

El problema EOC es analizable por la teoría clásica y aplican las condiciones suficientes de la teoría del control óptimo. Además, Bengea y De Carlo (2005) demuestran que si se resuelve el problema EOC y se obtiene una solución existen dos posibilidades: la solución es discreta, es

decir solo un $\alpha_i(t) = 1, \forall t$, y entonces el HOC tiene la misma solución; si la solución no satisface que sólo un $\alpha_i(t) = 1, \forall t$, entonces el HOC no tiene solución, pero se puede encontrar soluciones sub-óptimas que se acerquen arbitrariamente a la solución del EOC.

Para resolver este problema como un control predictivo híbrido, se discretiza el problema (3.17) en el dominio temporal, y se resuelve el problema de optimización con una variación del método de colocación directa (*direct collocation;* von Stryk, 1993). En este método se parametrizan las incógnitas x_i, u_i en bases de dimensión finita. Así las predicciones en el modelo dinámico de x_i son aproximadas por una función afín a tramos:

$$\hat{x}_i(t) = x_i(t_j) + \frac{t - t_j}{t_{j+1} - t_j}\left(x_i\left(t_{j+1}\right) - x_i\left(t_j\right)\right),\ t_j \le t < t_{j+1},\ i = 1,...,n \tag{3.18}$$

Del mismo modo, las acciones de control continuas son:

$$\hat{u}_i(t) = u_i(t_j),\ t_j \le t < t_{j+1},\quad i = 1,...,m \tag{3.19}$$

Con esto, las incógnitas del problema de optimización son $x_i(t_j), u_i(t_j), \alpha(t_j),\ j = 1,...,N$, y siendo N el número de puntos de la partición temporal. El problema de optimización resultante puede formularse como uno de programación no lineal (*Non-Linear Programming*, NLP). Los autores plantean que el problema, a pesar que la ecuación de estado está compuesto por funciones que tienen discontinuidades, puede ser resuelto por métodos convencionales tales como programación secuencial cuadrática (Sequential Quadratic Programming, SQP), ya que estas discontinuidades, que se encuentran sólo en las fronteras de los tramos que sí son continuos, no afectan el desempeño de los métodos.

Finalmente, las soluciones para el problema HOC $u(t), \alpha(t)$ se pueden encontrar a partir de los $\alpha_i(t)$, del siguiente modo:

$$\begin{aligned} u(t_j) = u_i(t_j) &\Leftrightarrow \alpha_i(t_j) = 1 \\ \alpha(t_j) = i \qquad &\Leftrightarrow \alpha_i(t_j) = 1 \end{aligned} \tag{3.20}$$

La metodología es útil en el sentido que tan sólo se requiere resolver un NLP en cada instante para encontrar la acción de control, y que acepta modelos de procesos no lineales, incluso no lineales por tramos (modelos para los que no pueden ser aplicadas otras técnicas tales como las descritas en 3.3.1 ó 3.3.2). Sin embargo, se basa en un supuesto de que el problema (3.17) puede ser resuelto por métodos como SQP, a pesar de estar compuesto por componentes que no son clase C^1 . Además, es bastante restrictivo respecto de los modelos de los procesos que requieren ser sin memoria, y al menos de clase C^1 por tramos.

3.3.5 Control predictivo híbrido no lineal

Las formulaciones presentadas hasta el momento sirven para sistemas híbridos que si bien son no lineales, en cada tramo de operación son lineales. A continuación se propone una formulación más genérica en la cual los distintos regimenes de operación discretos pueden ser no lineales,

donde tanto las variables de entrada, estados y variables de salida pueden ser continuas y discretas. La formulación es útil para procesos que se encuentren modelados por sistemas más generales, por ejemplo sistemas con modelos difusos o neuronales.

El caso más genérico de un sistema híbrido no lineal en tiempo discreto que contenga entradas, estados y salidas con valores continuos y discretos, además de poder exhibir dinámicas discretas se puede expresar del siguiente modo:

$$\begin{aligned} x(t) &= f_i\left(x(t-1), u(t-1)\right) \\ y(t) &= g_i\left(x(t), u(t)\right) \end{aligned} \tag{3.21}$$

donde $x(t) = \left[x(t)^c, x(t)^d \right]$ son los estados, $y(t) = \left[y(t)^c, y(t)^d \right]$ son las salidas, y $u(t) = \left[u(t)^c, u(t)^d \right]$ son las acciones de control, que comprenden sus respectivas componentes continuas y discretas, señaladas por los superíndices *c* y *d* respectivamente. $f_i(\cdot)$, $g(\cdot)$ son funciones que determinan el siguiente estado y la salida respectivamente. Estas funciones además, pueden estar definidas por tramos que dependen tanto del estado como de las entradas. En cada tramo las funciones pueden variar, lo que se indica por el subíndice *i*. Como ya fue mencionado, los tramos dependen tanto del estado como de las entradas, a través de la función $h(\cdot)$:

$$i = h\left(x(t), u(t)\right) \tag{3.22}$$

Así, el sistema híbrido se puede rescribir como:

$$\begin{aligned} x(t) &= f_{h\left(x(t-1),u(t-1)\right)}\left(x(t-1), u(t-1)\right) \\ y(t) &= g_{h\left(x(t-1),u(t-1)\right)}\left(x(t-1), u(t-1)\right) \end{aligned} \tag{3.23}$$

Sin embargo, como se está trabajando con funciones no lineales generales, es posible definir:

$$f\left(x(t), u(t)\right) = f_{h\left(x(t),u(t)\right)}\left(x(t), u(t)\right) \tag{3.24}$$

$$g\left(x(t), u(t)\right) = g_{h\left(x(t),u(t)\right)}\left(x(t), u(t)\right) \tag{3.25}$$

Y por lo tanto (3.23) se rescribe como:

$$\begin{aligned} x(t) &= f\left(x(t-1), u(t-1)\right) \\ y(t) &= g\left(x(t-1), u(t-1)\right) \end{aligned} \tag{3.26}$$

La formulación del control predictivo híbrido se completa definiendo la función objetivo y las restricciones en el problema de optimización. Un caso general consiste en una función objetivo que considera el error de seguimiento y de esfuerzo de control, y restricciones no lineales:

$$
\begin{aligned}
\min_{u=(u(t),\ldots,u(t+N_u-1))} J = & \sum_{k=1}^{N_y} \left\| y(t+k|t) - y_{ref}(t+k) \right\|_{Q_y}^{p} + \sum_{k=1}^{N_u} \left\| \Delta u(t+k-1|t) \right\|_{Q_u}^{p} \\
s.a. \quad & x(t+k|t) = f(x(t+k-1|t), u(t+k-1|t)) \\
& y(k+t\,|\,t) = g(x(k+t-1\,|\,t), u(k+t-1)) \\
& c_1(y(k+t\,|\,t), x(k+t-1\,|\,t), u(k+t-1)) \le 0 \\
& c_2(y(k+t\,|\,t), x(k+t-1\,|\,t), u(k+t-1)) = 0 \\
x(t) \in X \subseteq \mathbb{R}^{n_{xc}} \times \mathbb{Z}^{n_{xd}}, \; & y(t) \in Y \subseteq \mathbb{R}^{n_{yc}} \times \mathbb{Z}^{n_{yd}}, \; u(t) \in U \subseteq \mathbb{R}^{n_{uc}} \times \mathbb{Z}^{n_{ud}}, \\
Q_y \in \mathbb{R}^{n_{yc}} \times \mathbb{R}^{n_{yc}}, \; & Q_u \in \mathbb{R}^{n_{uc}} \times \mathbb{R}^{n_{uc}}
\end{aligned}
\tag{3.27}
$$

donde $\|x\|_Q^p$ representa la norma-*p* para $p = 1, 2, \infty$. N_y es el horizonte de predicción y N_u es el horizonte de predicción. Q_y, Q_u son matrices de peso que satisfacen $Q = Q^T \ge 0$. c_1, c_2 son funciones en general no lineales que definen las restricciones de desigualdad y de igualad respectivamente, que debe satisfacer el proceso durante su operación. n_{xc}, n_{yc}, n_{uc}, n_{xd}, n_{yd}, n_{ud} son las dimensiones de los estados, salidas y acciones de control continuas y discretas.

El problema incluye entonces la función objetivo cuadrática, más un conjunto de restricciones de igualdad que corresponden a la dinámica y salidas del sistema, restricciones de desigualdad que pueden corresponder a requerimientos operativos del proceso, y restricciones de igualdad que pueden obedecer a condiciones terminales u otro tipo de condiciones operativas[3]. En general, todas las funciones presentes aquí son no lineales.

La formulación anterior corresponde a un problema de optimización entera mixta no lineal, ya que posee una función objetivo cuadrática, restricciones no lineales y variables enteras y continuas.

En Grossman (2002) se presenta una revisión de las principales técnicas utilizadas para resolver los problemas MINLP. Estos son ramificar y acotar (*Branch and Bound*, BB), aproximaciones externas (*Outer approximations*, OA), descomposición de benders generalizada (*Generalized benders decomposition*, GBD), planos cortantes extendidos (*Extended cutting planes*, ECP). Estos métodos se basan en la resolución sistemática de versiones relajadas del problema MINLP. En estas se puede relajar la condición de discretas para este tipo de variables, se puede fijar el valor de las variables discretas, o bien simplemente resolver un problema de factibilidad de alguna de estas versiones relajadas. El modo en el que se busque a través de estos problemas relajados depende del método utilizado. Otra familia para resolver problemas MINLP son los métodos basados en lógica (*logic-based methods*) en los cuales se utiliza técnicas de inferencia simbólica o restricciones disjuntivas que se pueden expresar en términos de variables binarias, en una formulación que se conoce como programa disjuntivo generalizado (*Generalized disjunctive program*, GDP).

Estos métodos son los clásicos para resolver problemas MINLP. Sin embargo, estos algoritmos pueden tener problemas de convergencia, estabilidad o de requerir demasiadas iteraciones, dependiendo de la estructura del problema particular. Además, el manejo de múltiples mínimos

[3] En el anexo, sección 12, se presentan las restricciones más comunes utilizadas en control predictivo.

locales y no convexidades en muy complejo. Más aún, para aplicar estas técnicas se requiere que las funciones sean diferenciables, situación que no siempre se da. Se requiere entonces encontrar otra estrategia para poder aplicar esta formulación del problema de control predictivo.

3.4 Conclusiones

De la revisión realizada se pueden distinguir claramente cuatro enfoques para enfrentar los problemas de control predictivo híbrido. El primero consiste en la formulación de los procesos como sistemas MLD o PWA, para los cuales, es posible encontrar la solución mediante la resolución de un problema MIQP/MILP en modo *online*. El segundo se apoya en la misma formulación del primero, pero mediante optimización multi-paramétrica, se puede encontrar una ley de realimentación PWA explícita. El tercer enfoque también se basa en resolver el problema de optimización entera mixta resultante en modo *online*, pero de un modo más eficiente basado en un árbol de evolución del sistema, pero que sólo es válida para sistemas con entradas discretas. En la última estrategia también se resuelve un problema de optimización en modo *online*, pero mediante ciertas adaptaciones sólo se debe resolver un problema de optimización no lineal, en lugar de un problema entero-mixto.

Todas estas estrategias poseen claras ventajas y desventajas. En el caso de las estrategias basadas en optimización multi-paramétrica, tienen la gran ventaja que para encontrar la acción de control durante la ejecución del proceso basta evaluar la expresión PWA, sin tener que resolver ningún problema de optimización. Sin embargo, encontrar esta expresión PWA es altamente costosa, pudiendo no ser posible en problemas con muchas variables discretas. En el caso de las estrategias donde se resuelve un problema de programación entera mixta, se cuenta con la ventaja que se pueden analizar problemas más genéricos que aquellos que se pueden enfrentar con optimización multiparamétrica, pero claramente tienen el serio problema de ser NP *hard*, con una complejidad aumentando exponencialmente con el número de variables discretas. La estrategia basada en un análisis de alcanzabilidad tiene como gran fuerte su eficiencia (respecto de otras técnicas, pero de todos modos es NP *hard*), pero su gran limitante es la reducida variedad de procesos (sólo con entradas discretas) a la que puede ser aplicada. La última estrategia que resuelve el problema de optimización no lineal (sin incluir directamente las variables discretas) no tiene esta característica de ser NP *hard*, pero está limitada a una variedad incluso más reducida de modelos.

La limitación más importante de estas estrategias es la reducida variedad de modelos de procesos a los que pueden ser aplicadas. Por ejemplo, para el enfoque basado en programación multi-paramétrica, quedan fuera aquellos que debido a su complejidad no pueden ser formulados como PWA o MLD. Para el enfoque basado en análisis de alcanzabilidad, si bien se puede aplicar a procesos con una variada gama de modelos (PWA, difusos, neuronales, etc.), se requiere que las entradas del proceso sean únicamente discretas (y también es NP *hard*). La estrategia basada en generar un problema NLP, deja afuera aquellos procesos cuyos modelos no son diferenciables, de modo que no se puede aplicar programación no lineal, o aquellos que si bien pueden ser formulados para que sean diferenciables, son de tal no linealidad, que mediante programación no lineal es altamente probable que se encuentren óptimos locales que no sean lo suficientemente satisfactorios.

Para poder abarcar la mayor cantidad posible de procesos, se construye la formulación presentada en 3.3.5. Si las funciones son diferenciables, es posible aplicar alguna de las estrategias para

problemas MINLP. Sin embargo, estos algoritmos pueden tener problemas de convergencia, estabilidad o bien requerir demasiadas iteraciones. Además, el manejo de múltiples mínimos locales y no convexidades puede llegar a añadir una gran complejidad computacional. Finalmente, claramente estas estrategias no son aplicables si el modelo del proceso por ejemplo no es diferenciable.

Para estos procesos, los algoritmos evolutivos surgen como una opción atractiva para la aplicación de controladores predictivos híbridos. Los algoritmos evolutivos son algoritmos estocásticos de optimización basados en poblaciones, cuya heurística proviene de la teoría evolucionaría de Darwin, utilizando operadores tales como selección, recombinación y mutación. Estos algoritmos poseen buenas cualidades para la optimización de sistemas con muchos óptimos gracias a su enfoque basado en poblaciones, y debido a su heurística de búsqueda, no necesitan operadores de gradiente. Otros algoritmos similares también cumplen estas cualidades, por ejemplo la optimización de enjambre de partículas (*Particle swarm optimization*, PSO). Debido a su naturaleza de búsqueda de soluciones no requieren de características particulares que deban cumplir los modelos de los procesos ni la función objetivo. Si bien éstos no poseen pruebas deterministas de convergencia u optimalidad, por lo general entregan soluciones sub-óptimas de buena calidad, en un tiempo acotado que se puede escoger arbitrariamente según los requerimientos de la aplicación, limitando por ejemplo el número máximo de iteraciones. Esto es altamente útil para sistemas de muchas dimensiones, pues la resolución de los problemas de optimización no es NP *hard*. Así, es posible seleccionar compromisos entre el tiempo computacional requerido para encontrar una solución y la calidad de ésta. Además, los algoritmos evolutivos pueden lidiar eficientemente con las restricciones que aparecen en este tipo de estrategias, u otros aspectos que no pueden manejar bien los otros enfoques, como la presencia de múltiples óptimos locales.

Como se ha recalcado, los algoritmos evolutivos poseen muchas ventajas; sin embargo, también poseen importantes complicaciones. Las principales son la ausencia de pruebas de optimalidad, así como el hecho que se puede llegar a requerir un importante número de evaluaciones de soluciones candidatas para converger a una buena solución. Es importante entonces considerar estas limitaciones para diseñar controladores predictivos híbridos basados en algoritmos evolutivos eficientes. En particular se analizará el efecto que tienen sobre estas características diversas elecciones que se deben realizar, como el tamaño de población, número de iteraciones, representación de soluciones, manejo de restricciones, etc.

En el próximo capítulo (Capítulo 4) se presentan los aspectos más relevantes de los algoritmos evolutivos, que servirán como base para plantear las estrategias de control predictivo en base a algoritmos evolutivos en el capítulo 5.

4 Algoritmos evolutivos

4.1 Introducción

En ciertos problemas de optimización de alta complejidad se requiere algoritmos que los puedan resolver de forma eficiente, aceptando que en ocasiones la solución sólo sea cercana a la óptima, tal como ha sido explicado en la sección anterior. En este contexto, los algoritmos inteligentes de optimización basados en poblaciones resultan ser muy adecuados.

Los algoritmos evolutivos son la familia más grande y reconocida de algoritmos de optimización basados en poblaciones (Back *et al*., 1997) e incluyen: algoritmos genéticos, programación evolutiva, estrategias evolutivas y programación genética. El algoritmo de evolución diferencial (Storn y Price, 1995), que ha sido propuesto recientemente, se ha unido a los algoritmos evolutivos pues es considerada una variante de las estrategias evolutivas (Storn y Price, 1997). Por su parte, la optimización por enjambre de partículas (Eberhart y Kennedy, 1995) es un algoritmo que pertenece a la familia de la inteligencia de enjambres (Eberhart y Kennedy, 2001). Si bien esta rama se desarrolla independientemente y posee los fundamentos propios de la inteligencia de enjambres, que son distintos a los de los algoritmos evolutivos, diversos estudios han encontrado similitudes entre la optimización por enjambre de partículas y otros algoritmos evolutivos, como los algoritmos genéticos y la programación evolutiva (Eberhart y Kennedy, 2001).

Basado en lo anterior, por algoritmos evolutivos clásicos se entenderá que se hace referencia a las cuatro heurísticas inicialmente mencionadas: algoritmos genéticos, programación evolutiva, estrategias evolutivas y programación genética. Mientras que por algoritmos evolutivos se entenderá como la familia que consiste en algoritmos evolutivos clásicos, más evolución diferencial y optimización por enjambre de partículas, siempre y cuando se hable de la aplicación a control predictivo híbrido. Si no es así, se debe entender que por algoritmos evolutivos se hace referencia únicamente a las 4 heurísticas de algoritmos evolutivos clásicos.

Para ser capaces de aplicar los algoritmos evolutivos efectivamente en control predictivo híbrido, a continuación se presentan los aspectos más relevantes de las tres principales familias de algoritmos inteligentes de optimización, que son: algoritmos evolutivos, evolución diferencial y optimización por enjambre de partículas. Además de presentar los fundamentos de las heurísticas, se pone especial énfasis en las características de convergencia, selección de parámetros y aplicación para programación entera mixta de los algoritmos mencionados.

4.2 Algoritmos evolutivos clásicos

4.2.1 Introducción

Los algoritmos evolutivos (*Evolutionary algorithms*, EA) son métodos iterativos de optimización estocástica inspirado en conceptos de la teoría evolutiva de Darwin. Un algoritmo evolutivo simula un proceso evolutivo en una población de individuos con el propósito de evolucionar hacia la mejor solución aproximada del problema de optimización en cuestión. En el ciclo de simulación, tres operaciones se encuentran en juego; selección, recombinación, y mutación. Los

operadores de recombinación y mutación crean nuevas soluciones candidatas, mientras que la selección descarta los individuos de baja calidad, la que es evaluada por un *fitness* o adaptación (valores de *fitness* más altos indican mejor calidad de las soluciones). En la Figura 4 se ilustra la inicialización y el ciclo iterativo en EA.

Históricamente, los EA fueron sugeridos por primera vez en los años 40 (Fogel, 1998). Sin embargo, se considera que los padres de los AE modernos son: Lawrence Fogel, de la programación evolutiva (Fogel *et al*., 1966); Ingo Rechenberg y Hans-Paul Schwefel de las estrategias evolutivas (Rechenberg, 1973); y John Holland, de los algoritmos genéticos (Holland, 1975). Años más tarde, los términos Algoritmos Evolutivos (EA) y Computación Evolutiva (*Evolutionary Computation*, EC) fueron introducidos como términos unificadores para el conjunto de técnicas de optimización inspirada por la teoría de la evolución. Para un una revisión en mayor profundidad se sugiere consultar (Back *et al*., 1997).

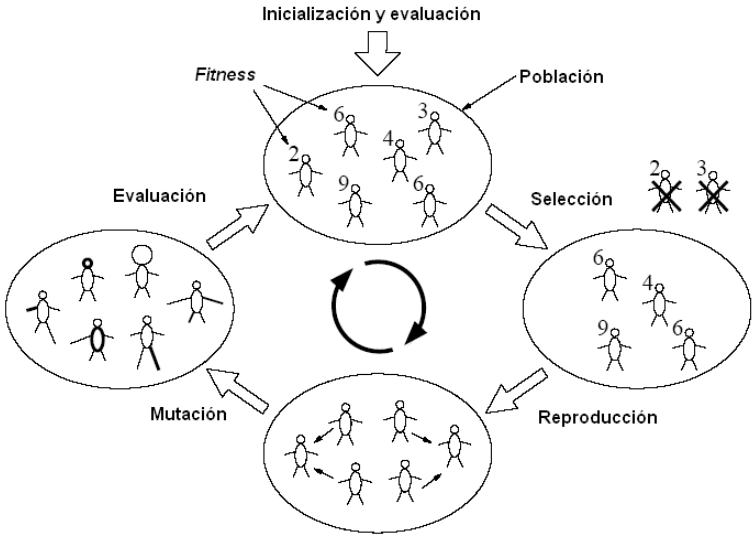


Figura 4: Inicialización y el ciclo iterativo en algoritmos evolutivos.

(Figura tomada de Ursem, 2003).

Los algoritmos genéticos son probablemente los algoritmos evolutivos más utilizados en la actualidad. Sus principales características son: el fuerte énfasis en el operador de recombinación, el uso de un operador de selección probabilística (selección proporcional), y la interpretación de la mutación como un operador de apoyo para la recombinación. Además, la forma original de los algoritmos genéticos se basa en el uso de codificación binaria, en la cual se encuentra basada la teoría de *schematta* y la hipótesis de bloques constituyentes, que justifican el funcionamiento de los algoritmos genéticos. Sin embargo, ésta ha sido adaptada para incluir codificaciones reales para aplicarlo de un modo más directo a espacios de búsqueda que no son binarios.

Las estrategias evolutivas, otra importante familia de los algoritmos evolutivos clásicos, usualmente utilizan codificación real, aunque también han sido utilizadas en problemas discretos. Sus atributos fundamentales son: la diferenciación entre una población de padres y una de descendientes de tamaño superior a la de padres; el énfasis explícito en el operador de mutación con distribución normal; el uso de distintas formas de recombinación; y la incorporación del principio de auto-adaptación para los parámetros que controlan el comportamiento del algoritmo

(tales como las tasas de mutación o recombinación), es decir que estos parámetros evolucionan junto con las soluciones candidatas (o como parte de ellas) en el proceso de optimización.

La programación evolutiva originalmente es desarrollada para evolucionar máquinas de estado finitas para tareas de predicción de series de tiempo, y luego se generalizó para problemas de optimización. Se basa principalmente en: operadores de variación hechos a la medida del problema y generalmente basados en un único padre; selección por torneo que determina un determinado número de sobrevivientes para la siguiente generación a partir de una población del doble de tamaño, formado por los padres y una igual cantidad de descendientes creados por mutación a partir de los padres; y por la auto-adaptación de los parámetros de control al igual que las estrategias evolutivas.

Finalmente, la programación genética (Koza, 1992) el último de los paradigmas de los algoritmos evolutivos clásicos, proviene de los algoritmos genéticos, y se caracteriza por utilizar los principios evolutivos para la generación automática de programas computacionales. Para lograr esto, cada individuo de una población representa un programa computacional completo, que siguen el proceso evolutivo hasta llegar a un programa final que cumple con los requerimientos.

A continuación se presentan los fundamentos de los algoritmos evolutivos, para luego explicar en mayor detalle cada uno de los paradigmas ya presentados.

4.2.2 Fundamentos

Si bien los algoritmos evolutivos son técnicas de optimización iterativa inspiradas en conceptos de la teoría evolutiva de Darwin, el proceso evolutivo en EAs es una representación simplificada del proceso en la naturaleza. Aún cuando muchos términos utilizados en EAs han sido adoptados desde la biología, muy pocos enfoques modernos tienen estos conceptos biológicos implementados en una manera realista. Conceptualmente, un EA mantiene una población de individuos que son seleccionados y creados en un proceso iterativo. Un individuo consiste de un cromosoma, un *fitness* y posiblemente un número de variables auxiliares que permiten controlar el proceso evolutivo, por ejemplo guardando valores como tasas de mutación o *crossover* variables a lo largo de la ejecución del algoritmo. El cromosoma consiste en un número de genes que en su conjunto codifican una solución del problema de optimización. El *fitness* representa la calidad de la solución codificada en el cromosoma del individuo, y usualmente corresponde a la función objetivo en un problema de optimización (o de una función monótonamente creciente respecto de la función objetivo), cuando se trata de problemas de maximización. Sin embargo, en los casos de minimización, a menor función objetivo la solución es de mejor calidad, y en dichos casos el *fitness* usualmente es una función monótonamente decreciente respecto de la función objetivo.

Respecto de la implementación de EAs, existe una gran variedad de estructuras y operadores evolutivos. Sin embargo, todos los EAs tienen una fase de inicialización seguida por una fase iterativa donde la población inicial evoluciona hacia mejores soluciones del problema. En Algoritmo # 1 se muestra un pseudo-código de un algoritmo evolutivo.

En EAs, la población generalmente es inicializada con individuos creados al azar $P(0)$, que son evaluados respecto de la función de *fitness*. Luego de la inicialización, el proceso iterativo es ejecutado hasta que se llega a algún criterio de término. El proceso iterativo consiste de cuatro

pasos. Primero, se incrementa el contador de generaciones (o iteraciones) t. Luego se aplica el operador de *selección* para formar la población $P'(t)$ a partir de la población $P(t-1)$. Esta *selección* es tal que es más probable que sean escogidos los individuos con mejor *fitness* que los individuos con peor *fitness*. Luego, una nueva población $P(t)$ es creada con operadores de recombinación y mutación de las soluciones en la población seleccionada $P'(t)$. El operador de recombinación crea una o dos nuevas soluciones mezclando (*crossover*) los cromosomas de dos o más padres. El operador de mutación altera el cromosoma de un sólo individuo para crear un nuevo individuo. Finalmente se evalúa el *fitness* de la nueva población y se repite el proceso.

Algoritmo # 1: Algoritmo evolutivo estándar
01. $t = 0$. Inicializar Población $P(0)$.
02. Evaluar Población $P(0)$.
03. Mientras no se cumpla condición de término
04. $t = t + 1$
05. Seleccionar población $P'(t)$ a partir de $P(t-1)$.
06. Crear población $P(t)$ a partir de $P'(t)$.
07. Evaluar Población $P(t)$.
08. Finalizar
09. Solución del problema es el mejor elemento de $P(t)$.

Durante el algoritmo, el *fitness* del mejor individuo mejora (idealmente) a medida que avanzan las generaciones y tiende a estancarse cuando se van acabando.

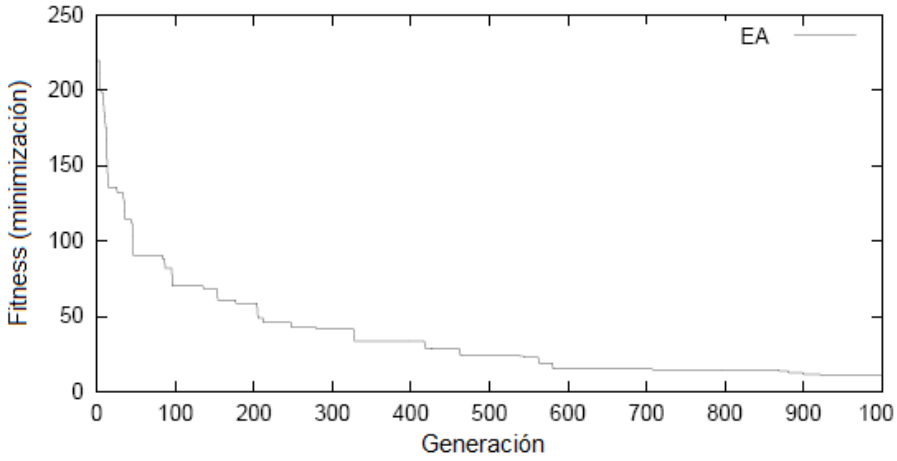


Figura 5: Mejora gradual del *fitness* durante el proceso iterativo de EA.

(Figura tomada de Ursem, 2003).

Idealmente, el estancamiento del proceso coincide con el descubrimiento exitoso del óptimo global. Sin embargo, el estancamiento también puede ocurrir en óptimos locales, que es un resultado indeseado y uno de los principales problemas en EAs y otros algoritmos de búsqueda iterativos. El estancamiento es causado por convergencia genética de los individuos en una parte del espacio de búsqueda, es decir, los genes de todos los individuos son muy similares. En este punto, la mutación es el único modo de explorar otras áreas del espacio de búsqueda, que corresponde a movimientos aleatorias desde el punto de búsqueda actual en el espacio.

4.2.2.1 Terminología básica

La terminología un la computación evolutiva (*Evolutionary computation*, EC) es, en gran medida, tomada desde la biología. Desafortunadamente, no hay acuerdo en gran parte de la terminología básica usada en EC. En general, los investigadores están de acuerdo en el significado de selección, mutación y recombinación. Sin embargo, los términos relacionados con el problema de optimización; el objetivo y la representación están definidos de manera muy vaga y se necesita descripciones más unificadoras y concisas.

A la estructura de los individuos (soluciones candidatas) se les conoce como cromosomas, que son el genotipo manipulado por el GA. La rutina de evaluación decodifica estas estructuras en otra estructura fenotípica (solución candidata) y les asigna un valor de *fitness*, que como ya se ha dicho es una medida de la calidad de la solución. El valor de cada bit en un cromosoma se le conoce como alelo o gen. Sin embargo, cuando un conjunto de bits representa un parámetro en el espacio fenotípico, a este conjunto también se le suele llama gen. Para aclarar términos, en este texto se entenderá por gen al conjunto de bits que representan a un parámetro. Debido a que en la representación binaria varios bits pueden representar un parámetro del espacio fenotípico, la dimensión del problema es en general menor que la dimensión del híper-espacio de búsqueda. En la Figura 6 se presenta una solución candidata para un problema de optimización representada por un cromosoma. El problema de optimización posee dos variables, cada una codificada con una precisión de 8 bits, identificadas como “Gen 1” y “Gen 2”. El espacio de búsqueda en este problema es dos, mientras que la dimensión del híper-espacio de búsqueda es 16.

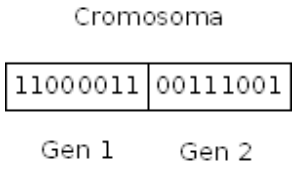


Figura 6: ejemplo de cromosoma.

4.2.2.2 Codificación, mutación y crossover

El modo óptimo de codificación de parámetros del cromosoma de un individuo depende de la definición del problema. En principio, los parámetros del problema pueden ser codificados con representación binaria. Sin embargo, es a menudo conveniente usar codificación de alto nivel e implementar operadores especializados de mutación y recombinación para la codificación en particular. La gran variedad de aplicaciones basadas en EAs han permitido la proliferación de una diversa gama de tipos de codificaciones y operadores. Los utilizados más frecuentemente son codificaciones para dominios numéricos, dominios de permutaciones, dominios matriciales, y dominios funcionales. En esta tesis, sólo se describen los dominios numéricos más relevantes y útiles para el diseño de estrategias de control.

Los dominios numéricos se utilizan en problemas donde el objetivo es encontrar un vector numérico. La mayoría de las aplicaciones de EA se originan en este dominio y por lo tanto una cantidad de investigadores se han dedicado a investigar y desarrollar codificadores y operadores para este dominio. Las dos codificaciones típicas son la codificación con *string* binario, y la codificación con un vector real.

Un problema relevante en la representación numérica de problemas es la precisión de la codificación. La precisión de la representación puede ser mejorada aumentando el número de bits en la representación binaria. Sin embargo, la mejora en precisión también aumenta el espacio de búsqueda, que crece exponencialmente con el número de bits. Por ejemplo, el espacio de búsqueda en un problema con representación de 16 bits es $2^{16} = 65.536$. Para doblar la precisión, los cromosomas deben ser construidos con 17 bits, lo que dobla el tamaño del espacio de búsqueda. La misma consideración se tiene para problemas con codificación real, pues la representación interna de los números de todos modos es binario, y su precisión está dada por el tipo de representación de punto flotante utilizada.

4.2.2.2.1 Strings binarios

La codificación binaria es el modo tradicional de representar parámetros en EAs. La estructura de datos utilizada es un vector de bits con largo fijo L, que corresponde a 2^L soluciones distintas en el espacio de búsqueda.

Para usar representación binaria en dominios numéricos, se debe especificar la función decodificadora que mapea la representación binaria de un gen en un número de punto flotante. La función decodificadora convierte el número binario en un número decimal, que será utilizado para evaluar la función de *fitness*. Supongamos que un gen x está codificado por L bits, luego el valor correspondiente en representación de punto flotante x_{real} está dado por:

$$x_{real} = x_{\min} + \frac{x_{\max} - x_{\min}}{2^L - 1}\left(\sum_{i=0}^{L-1} x[i] \cdot 2^{L-1-i}\right) \tag{4.1}$$

donde $x_{\min}$ y $x_{\max}$ son los valores mínimos y máximos de x, y $x[i]$ es el $i-ésimo$ bit del gen en la codificación binaria. Por ejemplo, si x está codificado por 8 bits, $x_{\min} = -2$ y $x_{\max} = 2$, entonces el número binario $[01100111] = 103$ es traducido como:

$$x_{real} = -2 + \frac{2 - -2}{255} 103 \approx -0.3843 \tag{4.2}$$

Un caso bidimensional puede ser directamente codificado utilizando16 bits, con 8 bits para x_1 y otros 8 para x_2.

$$\left[\underbrace{11010010}_{x_1}\underbrace{00111001}_{x_2}\right]$$

Figura 7: Cromosoma de una función de dos parámetros con precisión de 8 bits.

La otra opción para mapear una codificación binaria a un dominio numérico es la llamada decodificación gris (g*ray decoding*). La ventaja de esta técnica es que parámetros de valor similar en la representación de punto flotante corresponden a números adyacentes en la representación binaria. Por ejemplo, el número binario $[00011111] = 31$ no es adyacente a $[00100000] = 32$ en

la codificación binaria tradicional, aunque 31 y 32 sí son enteros adyacentes. Si 32 es mejor solución que 31 entonces el EA debe cambiar 6 bits en la representación para cambiar el valor de 31 a 32. La función decodificadora Gris resuelve este problema de modo que enteros vecinos son representados por números binarios que sólo difieren en un bit. A continuación se muestran un algoritmo de decodificación gris con complejidad de orden $O(L)$. Sin embargo, en ambas técnicas de decodificación binarias existe el problema que un pequeño cambio en el cromosoma binario puede implicar largos saltos en el espacio de búsqueda de punto flotante, por ejemplo $[00000001]=1$ y $[10000001]=129$.

Algoritmo # 2: Decodificación gris
01. función entero = DecodificaciónGris(string binario x) 02. unos = 0; 03. valor_entero = 0; 04. Para i = 1:abs(x) 05. Si(x[i]==1) 06. unos = unos + 1; 07. Finalizar 08. valor_entero = valor_entero + (unos mod 2)*$2^{abs(x)-i}$ 09. Para 10. Retornar valor_entero;

La codificación binaria es utilizada a menudo, además de problemas numéricos, en problemas combinatoriales. Consideremos por ejemplo el "*problema de la mochila"*. Dada una mochila de determinada capacidad y un conjunto de ítems con un determinado tamaño y valor, se desea encontrar el subconjunto de ítems con mayor valor total que quepan en la mochila. En este problema una codificación natural sería un *string* de unos y ceros de largo igual al número de ítems, donde cada elemento del *string* esta asociados a uno de estos ítems. Así, si el elemento se selecciona para ir dentro de la mochila en una solución candidata, el elemento asociado en el *string* vale 1, y si no se escoge, el elemento del *string* vale 0.

$$\underbrace{1101001000111001}_{L=16}$$

Figura 8: Cromosoma binario para problemas combinatoriales

Por ejemplo en la Figura 8 se muestra una codificación binaria para el problema de la mochila con 16 ítems. Los seleccionados en la solución candidata para entrar en la mochila son los ítems 1, 2, 4, 7, 11, 12, 13 y 16.

4.2.2.2.1.1 Mutación bit-flip

La mutación bit-flip es el operador de mutación más utilizado para problemas con codificación binaria. El procedimiento consiste en una iteración sobre todos los genes, donde el bit *i* en un gen es dado vuelta si un número aleatorio uniforme $r \in U(0,1)$ es más pequeño que un cierto umbral de probabilidad p_m. El principal problema de este operador es su complejidad, que es de orden $O(L)$ para *strings* de bits de largo L, de modo que la implementación directa esta muy ineficiente. Para mejorar esto, se puede notar que la distancia entre 2 bits que han sido cambiados

sigue una distribución geométrica, es decir, si p_m es la probabilidad de cambiar un bit, entonces $T \sim ge(p_m)$ es una variable estocástica que describe la distancia entre bits cambiados. Es posible entonces con esta propiedad generar aleatoriamente la posición de la ubicación del próximo bit que cambiará su valor, en lugar de generar el número aleatorio para cada bit. Entonces, de acuerdo a la distribución geométrica, el número de bits t a saltarse puede ser calculado con la siguiente función:

$$t = 1 + \left\lfloor \frac{\ln(u)}{\ln(1 - p_m)} \right\rfloor \tag{4.3}$$

donde u está uniformemente distribuido de acuerdo a $U(0,1)$. Si la posición t' del siguiente cambio de bit no está en el cromosoma actual, entonces el primer bit cambiado en el siguiente cromosoma mutado debiese ser en el $(t' - L)$-ésimo bit.

En estudios empíricos se proponen valores para $p_m \in [0.001, 0.01]$ (De Jong, 1975; Grefenstette, 1986). Estos valores pequeños eran consecuentes con la creencia de que la mutación sólo servía como un operador de segundo plano que apoya al crossover (Holland, 1975; Goldberg, 1989). Sin embargo en los últimos años se ha demostrado tanto teórica como empíricamente los beneficios de este operador. Algunos de estos descubrimientos son: estudios empíricos muestran que una tasa de mutación grande el inicio con un decrecimiento exponencial con el número de las generaciones es altamente beneficioso (Fogarty, 1989); la confirmación teórica del descubrimiento empírico anterior para funciones de test simples; la prueba de que $p_m = 1/L$ es un valor óptimo para problemas esféricos simples (Bremermann *et al*., 1966; y luego Back, 1993). Por el último argumento, $1/L$ suele utilizarse como una cota inferior para p_m. La importancia de este operador es clara entonces para los algoritmos evolutivos, y no debe ser menospreciada tanto en aplicaciones prácticas como en estudios teóricos, situación que podía darse en el estudio de algoritmos genéticos.

4.2.2.2.1.2 *Crossover* uniforme y de N-puntos.

Un operador de *crossover* ampliamente utilizado para codificación binaria (y también real) es el *crossover* de *n* puntos, que recombina los genes de dos o más padres para crear lo cromosomas de dos descendientes.

En el *crossover* de un punto, los cromosomas de dos padres de tamaño *n* son cortados y re-ensamblados en una posición aleatoria p del cromosoma. El primer descendiente (*offspring*) recibe los genes entre *gen[1]* y *gen[p-1]* del padre 1, y el resto de los genes, es decir del *gen[p]* al *gen[n]* del padre 2. El cromosoma del segundo descendiente es ensamblado con los genes restantes de cada uno de los padres, es decir, del *gen[1]* al *gen[p-1]* son del padre 2 y del *gen[p]* al *gen[n]* son del padre 1. Esta operación es aplicada con una probabilidad p_c para cada par de padres, donde y usualmente se encuentra en el rango $[0.5,1]$. Si no se ejecuta, los descendientes son copias idénticas de los padres.

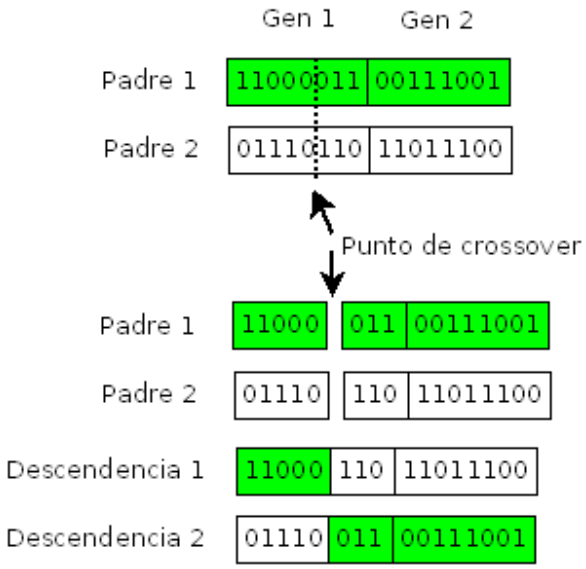


Figura 9: Operador de *crossover* de un punto

La diferencia del *crossover* de *n* puntos y el *crossover* de un punto es el uso de *n* puntos de *crossover* en lugar de uno. En cada punto de *crossover*, el origen del *gen[i]* se alterna entre los dos padres. Se utiliza un valor de *n* entre 1 y 4.

Otro operador de *crossover* es el *crossover* uniforme. En este caso la descendencia es generada seleccionando cada gen aleatoriamente de uno de los padres (con probabilidades iguales para cada padre).

4.2.2.2.2 Vectores con valor real

Otro modo popular de codificar dominios numéricos es representar los genes directamente por número reales (pseudo-reales en estricto rigor, debido a la codificación interna). En este caso, el espacio de búsqueda es un subconjunto del dominio objetivo. De este modo, no se necesita decodificación. La representación directa de los valores reales permite el diseño de operadores de mutación y crossover que están basados en operaciones aritméticas y distribuciones estocásticas. A continuación, se detallan los operadores específicos para esta codificación real.

4.2.2.2.2.1 Mutación Gaussiana y Uniforme

La mayor parte de los operadores de mutación para vectores reales alteran la solución sumando un vector real generado al azar $M = (m_1, m_2, ..., m_n)$ al vector solución x, es decir:

$$x' = x + M \tag{4.4}$$

Es importante destacar que los m_i en M sean generados con una distribución con media cero, de otro modo las soluciones se alejarán debido a la mutación. La elección común para la generación de M es una distribución Gaussiana $N(0, \alpha)$.

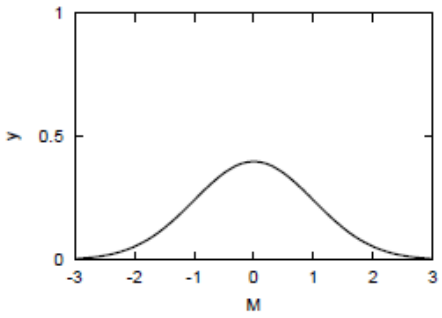
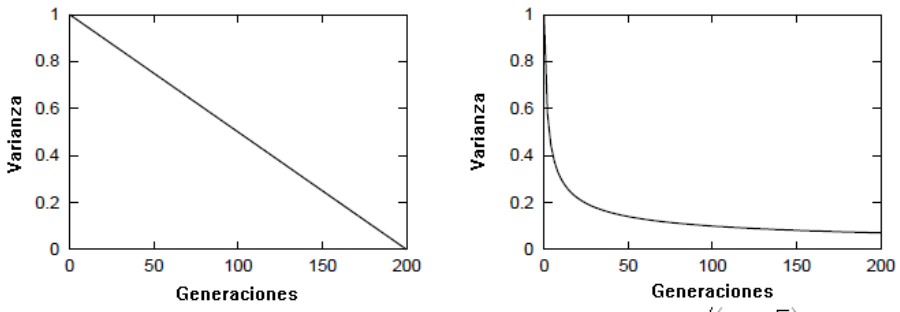


Figura 10: distribución Gaussiana para $N(0,1)$

Una estrategia de mutación un poco menos común está basada en la distribución uniforme $U(-\alpha,\alpha)$, donde M puede tomar cualquier valor entre $-\alpha$ y α con probabilidad uniforme. Un caso especial de mutación uniforme es $x' = M$ con $M \in U\left(x_{\min}, x_{\max}\right)$, que puede ser útil para la codificación de un parámetro enumerable (por ejemplo que pueda tomar distintos valores discretos).

El desempeño del operador de mutación depende fuertemente del parámetro α. Si éste es muy alto el algoritmo tiene dificultades para encontrar la solución fina, mientras que si es muy bajo la población tiene problemas para encontrar la solución global y suele estancarse en un mínimo local. Existen diversas técnicas sugeridas para controlar α, tales como auto-adaptación en estrategias evolutivas (Rechenberg, 1973).

Una solución muy simple pero efectiva es definir α como función de la generación. Una hipótesis bastante aceptada es que en general, la población convergerá a un óptimo local o global. Para mejorar las opciones de encontrar el óptimo global el algoritmo debiese empezar con una estrategia de búsqueda más bien ancha, que gradualmente va adelgazándose a medida que la población converge, es decir, α debe ser calculado usando una función decreciente. Dos funciones que son utilizadas para este fin se presentan en la Figura 11.



(a) $\alpha(t) = 1 - 0.005t$ (b) $\alpha(t) = 1/\left(1+\sqrt{t}\right)$

Figura 11: funciones decrecientes para el cálculo de α **.**

4.2.2.2.2.2 *Crossover* en codificación real

En cromosomas con codificación real también se puede aplicar crossover de *N* puntos o crossover uniforme, del mismo modo que en cromosomas con codificación binaria. La única diferencia es

que el punto de crossover no divide strings de unos y ceros, sino que divide vectores de número reales.

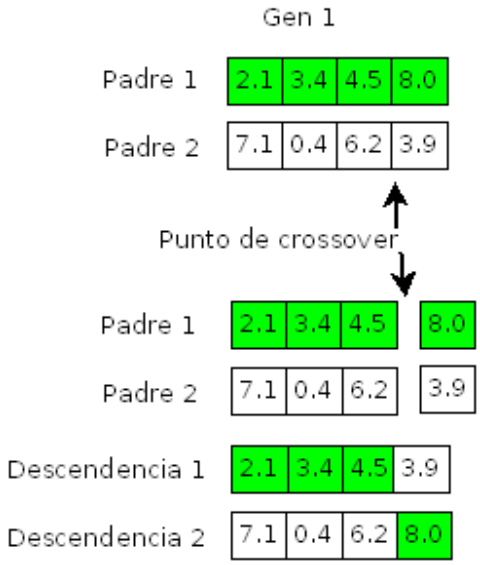


Figura 12: Crossover de 1 punto en cromosomas con codificación real

Crossover aritmético

El *crossover* aritmético es un operador para cromosomas con codificación real donde los cromosomas de los descendientes son generados con la media ponderada de cada gen de los cromosomas de dos padres (Back *et al.*, 2000).

$$x' = \omega x_1 + (1-\omega)x_2 \tag{4.5}$$

donde ω es el peso y x_1 y x_2 son los cromosomas de los padres. Si $\omega = 0.5$ (valor usualmente utilizado) entonces el crossover aritmético calcula el cromosoma del descendiente como la media aritmética de los dos padres. El peso ω es a menudo generado de acuerdo a la distribución uniforme $U(0,1)$, que ubicará el cromosoma de la descendencia numéricamente entre los cromosomas de los padres (figura 2.9b). Una variante del crossover aritmético genera un peso específico ω_i por cada gen x_i en el vector cromosoma $x' = (x'_1, x'_2, ..., x'_n)$.

$$x'_i = \omega_i x_{1i} + (1-\omega_i)x_{2i} \tag{4.6}$$

En esta variante, la descendencia se ubica en una locación aleatoria dentro del hipercubo generado por los dos padres (Figura 13.c). Una tercera variante del crossover aritmético genera la descendencia de $k > 2$ padres. La descendencia se obtiene combinando los padres de acuerdo a un número de pesos, que definen la contribución de cada uno de ellos. La descendencia generada entonces pertenece a la envoltura convexa de los padres, y se genera como sigue:

$$x' = \sum_{j=1}^{k} \omega_j x_j \text{ , donde } \omega_j \in [0,1], \sum_{j=1}^{k} \omega_j = 1. \tag{4.7}$$

En esta configuración, la descendencia es creada en la envoltura convexa definida por los k padres (Figura 13d).

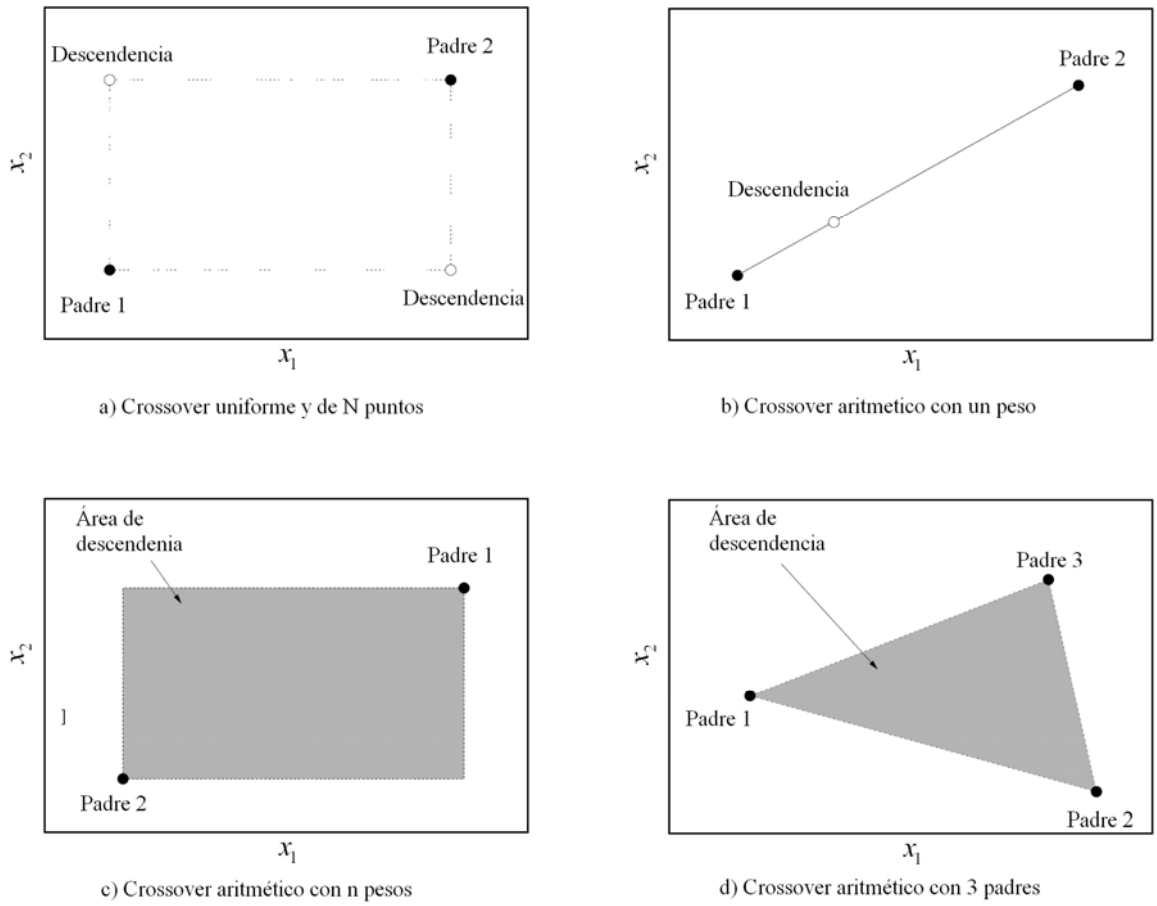


Figura 13: crossover de vectores reales en un problema bidimensional.

(Figura adaptada de Ursem, 2003).

4.2.2.3 Operadores de selección

El operador de selección es un proceso esencial en EAs, que remueve individuos con un *fitness* bajo y lleva a la población hacia soluciones de mejor calidad. En esta sección, se describen los cinco operadores de selección más comunes. Además se describe la selección manual, que es utilizada cuando una descripción formal (matemática) de una función de *fitness* es imposible.

Este operador esencialmente define como el algoritmo actualiza la población de una iteración a la siguiente. En general, el operador de selección puede reemplazar la población entera o sólo una fracción de ésta. La primera opción es utilizada en los llamados algoritmos evolutivos generacionales, mientras que la segunda es utilizada los llamados algoritmos evolutivos de estado permanente (*steady state*). Además de ésta, existen otras diferencias mayores entre los dos enfoques. Primero, el procedimiento de selección en los EAs generacionales es estocástico, mientras que en los EAs de estado permanente es determinístico. Debido a su naturaleza estocástica, en los EAs generacionales el operador de selección podría no escoger los mejores

individuos. Por esto, es en general considerado una buena idea asegurar la supervivencia del mejor individuo. Esta estrategia es considerada como *elitismo* o *k-elitismo* si *k* individuos son escogidos como elite. Segundo, en los EAs generacionales los individuos son clonados al realizar la selección (sin realizar mutación ni crossover) mientras que los EAs de estado permanente seleccionan un subconjunto determinístico de soluciones candidatas luego de haber aplicado operadores de mutación y *crossover*. La selección de estado permanente es usada fundamentalmente por las estrategias evolutivas (Rechenberg, 1973).

Un aspecto importante en los operadores de selección es la presión selectiva. La presión selectiva se puede medir como el número de generaciones que se requieren para que la población se llene de copias del mejor individuo, cuando no interviene ningún otro operador. Si la presión selectiva es alta, la búsqueda se concentrará en los mejores individuos conocidos (en un fenómeno que se conoce como explotación), lo que puede ocasionar una pérdida de diversidad en la población, y se produce una convergencia prematura. En esta situación los genes de unos pocos individuos con alto *fitness* pueden llenar la población, lo que ocasiona que el algoritmo converja a un óptimo local. Como los individuos son muy similares, los operadores de *crossover* no funcionan y el único modo de explorar nueva regiones es mediante mutación, sin embargo la mutación es muy lenta para este propósito, y en general la población se queda estancada en el óptimo local. Si se reduce la presión selectiva se mejoran las capacidades de exploración pues más variedad de material genético es utilizado en el proceso de búsqueda. Sin embargo, una presión selectiva muy baja producirá una convergencia muy lenta.

4.2.2.3.1 Selección por torneo

El operador de selección por torneo crea la siguiente generación realizando un torneo por cada lugar disponible en la población de la siguiente generación. En cada torneo, el proceso escoge k individuos al azar, compara sus *fitness*, y el ganador es el individuo con el mejor *fitness*, quien se gana su lugar en la nueva generación. El tamaño de torneo k es usualmente de dos individuos, y rara vez sobre cinco, dado que esto impondría una presión selectiva muy alta y produciría convergencia prematura. La población original suele encontrarse fija durante la selección de la siguiente generación, lo que permite que los buenos individuos se copien múltiples veces. Sin embargo, esto también puede ocasionar que los mejores individuos nunca sean escogidos para competir en el torneo. Si los individuos seleccionados se van retirando de la población de origen a medida que éstos van entrando al torneo, se puede asegurar que el mejor individuo va a ser seleccionado para la siguiente generación un número de veces igual a T, donde T es el número de pasadas por la población completa (si se acaba la población para hacer torneos y aún faltan individuos para la siguiente generación, se vuelve a rellenar la población con todos los individuos).

Algoritmo # 3: Selección por Torneo
01. función P(t+1) = SelecciónTorneo(P(t)) 02. Para j = 1:largo(P(t)) 03. Escoger dos individuos I_1 e I_2 al azar de $P(t)$ 04. Escoger entre I_1 e I_2 el que tenga mejor fitness e insertarlo en $P(t+1)$ 05. Finalizar 06. Retornar P(t+1)

En el Algoritmo # 3 se presenta un seudo-código para una selección por torneo de tamaño 2. En este caso se requieren 2 pasadas para llenar la población de la siguiente generación. Así, el mejor individuo se llena 2 copias, el peor 0, y el promedio 1.

La selección por torneo fácil de implementar, permite obtener buenos resultados, posee una complejidad computacional bastante baja y requiere definir muy pocos parámetros. Debido a todo esto, es probablemente el operador de selección más usado actualmente.

4.2.2.3.2 Selección proporcional o por Ruleta

En este operador se asigna la probabilidad de supervivencia del individuo de acuerdo a su *fitness*. Esta probabilidad es calculada dividiendo el *fitness* del individuo por la suma total de *fitness* de la población, es decir, la probabilidad de supervivencia del individuo está dada por el *fitness* relativo con respecto al resto de la población.

$$p_{\text{supervivencia}}(I_i) = \frac{fitness(I_i)}{\sum_{j=1}^{N} fitness(I_j)} \tag{4.8}$$

donde se cumple, $\sum_{j=1}^{N} p_{\text{supervivencia}}(I_j) = 1$.

A cada individuo se le asigna un casillero en el intervalo [0,1] de acuerdo a su probabilidad de supervivencia $p_{\text{supervivencia}}$. Un individuo es seleccionada si un número aleatorio cae en el intervalo correspondiente a su casillero. Este método de selección es ilustrado generalmente como una ruleta sesgada, donde los casilleros corresponden a las zonas de la ruleta y los ganadores pasan a la siguiente generación.

Una desventaja de este tipo de selección es que la presión selectiva depende del *fitness* relativo de los individuos en lugar de un parámetro modificable como lo es el tamaño de un torneo. En la selección proporcional unos pocos individuos pueden tomar rápidamente la población entera, pues pueden dominar gran parte de la ruleta y ser copiados frecuentemente en la siguiente generación. O de forma inversa, si la diferencia de *fitness* entre los individuos es baja en términos relativos la presión selectiva es muy baja y se convierte casi en una selección al azar.

4.2.2.3.3 Selección por Ranking

La selección por ranking es una variante de la selección proporcional que es capaz de lidiar los problemas de presión selectiva que posee la segunda estrategia de selección mencionada. En selección por ranking la superioridad del individuo es determinada por una probabilidad fija $p_{\text{supervivencia}}$ de acuerdo a su ranking de *fitness*. El ranking es obtenido luego de ordenar los individuos de acuerdo a su *fitness*. Luego se asigna a cada individuo una probabilidad $p_{\text{supervivencia}}$, que es determinada por el esquema de ranking escogido. Luego la selección se realiza utilizando el enfoque de ruleta o de selección universal estocástica.

La gran dificultad de aplicar este tipo de selección es determinar una buena probabilidad de supervivencia para cada ranking. Un esquema demasiado generoso para soluciones malas puede ocasionar una lenta convergencia, mientras que un esquema que favorezca mucho a las buenas soluciones puede llevar a una pérdida prematura de diversidad genética.

Selección por ranking lineal

Una estrategia para asignar la probabilidad de supervivencia según el ranking de modo lineal se muestra a continuación. Ésta consiste en asignar el número de copias esperadas de un individuo en la siguiente generación, con más copias para los mejores individuos y menos para los peores. El número esperado de copias para un individuo se obtiene entonces como:

$$\alpha(x) = \eta^+ - (\eta^+ - \eta^-)\frac{x-1}{N-1} \tag{4.9}$$

donde $x = 1$ identifica al mejor individuo de la población y $x = N$ identifica al peor individuo. Además $\sum_x \alpha(x) = N, \quad 1 \le \eta^+ \le 2, \quad \eta^- = 2 - \eta^+$, y:

η^+ es el máximo valor esperado de copias para un individuo, que se asigna al mejor de la población (se recomienda usar un valor 1.1 ó 1.2).

η^- es el mínimo valor esperado de copias para un individuo, que se asigna al peor de la población.

Así, la probabilidad de selección del individuo está dada por su número esperado de copias divido por el número total de individuos en la siguiente generación:

$$p_{\text{sup}ervivencia}(x) = \frac{\alpha(x)}{N} \tag{4.10}$$

Selección por ranking no lineal

También existen técnicas para asignar una probabilidad de supervivencia no lineal respecto del ranking.

Un modo posible es asignar probabilidades de supervivencia en base a la distribución geométrica:

$$p_{\text{sup\_geo}} = \alpha(1-\alpha)^{x-1} \tag{4.11}$$

donde $x = 1$ identifica al mejor individuo de la población, mientras que $x = N$ identifica al peor individuo. Esta distribución aparece si la selección ocurre como una serie independiente de pruebas *Bernoulli* sobre los individuos ordenados por ranking, donde la probabilidad de seleccionar el siguiente individuo es α. Como resultado, aparece una distribución geométrica, donde la probabilidad de escoger al peor individuo es $p_{\text{sup\_geo}}(\text{peor\_ind}) = \alpha(1-\alpha)^{N-1}$, mientras que la probabilidad de escoger el mejor individuo es $p_{\text{sup\_geo}}(\text{mejor\_ind}) = \alpha$.

Otra opción, que asigna probabilidades exponenciales basadas en ranking es:

$$p_{\text{sup\_exp}} = \frac{1 - e^{-(N-x)}}{c} \tag{4.12}$$

donde el factor de normalización *c* se escoge para que la suma de probabilidades sea 1, y es por lo tanto una función del tamaño de la población. Este método asigna una probabilidad $p_{\text{sup\_geo}}(\text{peor\_ind}) = 0$ al peor individuo y una probabilidad $p_{\text{sup\_geo}}(\text{mejor\_ind}) = \left(1 - e^{-(N-1)}\right) \Big/ c$.

Estos dos métodos llevan imponen un mayor énfasis en la selección de individuos con *fitness* sobre el promedio, teniendo que asumir probablemente el costo de convergencia prematura.

4.2.2.3.4 Selección Universal Estocástica

Una variación a la selección proporcional es conocida como selección universal estocástica (*stochastic universal selection*, SUS). Esta consiste en que en lugar de lanzar una ruleta *N* veces (donde *N* es el tamaño de la población) se lanza una sola ruleta de *N* punteros equiespaciados entre sí, y así, con una sola tirada se obtienen *N* ganadores. Esta modificación posee la ventaja que es de orden $O(N)$, pues la lista se recorre una vez, mientras que en selección proporcional la lista se recorre *N* veces, y así es de orden $O(N^2)$. Además, es posible asegurar que todos los individuos con una probabilidad de supervivencia mayor a $1/N$ son seleccionados a lo menos una vez, y a lo más el cajón superior de $N \cdot p_{\text{supervivencia}}$.

4.2.2.3.5 Selección de estado permanente

Los algoritmos evolutivos que están basados en selección de estado permanente, conocido como *steady-state* EAs, sólo actualizan una pequeña fracción de la población en cada iteración. Los operadores evolutivos crean λ soluciones potenciales desde la población con tamaño μ. Luego, los $(\mu + \lambda)$ individuos son ordenados y los λ individuos con menor fitness se descartan. Este tipo de selección de estado permanente se conoce como $(\mu + \lambda)$, y es comúnmente utilizada en las estrategias evolutivas.

Estos enfoques son claramente distintos a las estrategias de selección proporcional, por ranking y por torneo. En la selección de estado permanente las poblaciones se traslapan y los individuos que sobreviven son seleccionados de forma determinística, propiedad de la que sólo gozan los individuos de elite en las otras técnicas de selección.

4.2.2.3.6 Selección Manual

En algunas aplicaciones, la calidad de la solución se encuentra basada en una evaluación subjetiva de problemas que son difíciles o imposibles de captar matemáticamente, por ejemplo, la belleza de un diseño. Entonces, el proceso de selección puede ser manejado por un operador humano. El algoritmo muestra las soluciones actuales y le pide al operador que seleccione un subconjunto de la solución presentada. La soluciones escogidas son luego usadas para crear una nueva población y se repite el proceso. Ejemplo de selección manual incluyen la evolución de controladores de robots (Lund *et al*., 1998), procesos de mezcla de colores de comidas (Herdy, 1996), y aplicaciones más experimentales en arte evolutivo (Eiben *et al*., 2001).

4.2.3 Los principales paradigmas en los algoritmos evolutivos

Como ha sido mencionado, los distintos paradigmas de los algoritmos evolutivos se desarrollaron de forma independiente los primeros años, pero recientemente se ha producido una conexión en su desarrollo, siendo a veces difícil distinguir uno de otro paradigma en diversas aplicaciones. En un principio, es posible distinguir tres categorías de EAs: algoritmos genéticos, estrategias evolutivas y programación evolutiva. En los últimos años se ha desarrollado el cuarto paradigma de los algoritmos evolutivos, la programación genética. A continuación se describe las principales características de cada uno de estos paradigmas evolutivos.

4.2.3.1 Algoritmos genéticos

Los algoritmos genéticos (*Genetic algorithms*, GA) son probablemente el principal representante de los algoritmos evolutivos. Fueron descritos originalmente por John Holland (Holland, 1975). En los algoritmos genéticos se enfatiza la recombinación como el operador de búsqueda más importante y se aplica mutación con una probabilidad muy pequeña tan solo como un operador adicional. En general utilizan operadores de selección probabilísticos (como selección proporcional). Si bien en un principio fueron utilizados fundamentalmente con codificaciones binarias, actualmente el trabajo con representaciones reales es bastante común en la medida que es útil para diversas aplicaciones y reduce la carga computacional de los operadores evolutivos.

Tres características distinguen a los GA de otros algoritmos evolutivos, según es descrito por Holland (1975): (i) la representación utilizada (*strings* binarios), (ii) la selección proporcional, (iii) y que el método principal de búsqueda es el *crossover*, siendo esta la característica más distintiva. Muchas implementaciones han utilizado otros métodos de selección, o incluso abandonado la representación binaria en pro de representaciones más amenas. Y aunque muchas variaciones de *crossover* han sido propuestas, casi todas están basadas en el espíritu original del procesamiento de *schematta*. Se debe señalar que las estrategias evolutivas han incorporado *crossover* en su repertorio y así las diferencias entre distintos EAs se han desvanecido (Back *et al*., 1991).

Algoritmo # 4: Algoritmo genético estándar
01. $t = 0$. Inicializar Población $P(0)$.
02. Evaluar Población $P(0)$.
03. Mientras no se cumpla condición de término
04. $t = t + 1$
05. Seleccionar población $C(t)$ a partir de $P(t-1)$.
06. Crear población $C'(t)$ a partir de $C(t)$ utilizando operadores de recombinación y mutación.
07. Construir población $P(t)$ a partir de descendencia $C'(t)$ y completar en caso de ser necesario con población de padres $P(t-1)$.
08. Evaluar Población $P(t)$.
09. Finalizar
10. Solución del problema es el mejor elemento de P(t).

El GA básico comienza con una población inicial $P(0)$ que es generalmente inicializada al azar y se evalúa el *fitness* de cada individuo. Luego alguno de estos individuos son seleccionados (generalmente con selección proporcional) para reproducirse y pasan a un conjunto $C(t)$. A este conjunto se le aplican operadores de recombinación y mutación obteniendo una población $C'(t)$. En general la nueva población $P(t+1)$ es construida tan sólo con la descendencia $C'(t)$ si esta es del mismo tamaño que la población inicial (en GAs se trabaja con poblaciones de tamaño fijo), aunque existen enfoques donde la descendencia es de tamaño menor y el resto de la población se compone escogiendo aleatoriamente miembros de la población de padres, o bien utilizando enfoques elitistas (los mejores padres que falten para completar la población). Así, el pseudo-código del algoritmo genético estándar se presenta en el Algoritmo # 4.

A pesar de que en general los algoritmos evolutivos utilizan selección estocástica, existen enfoques que utilizan selección de estado permanente como los que se utilizan en Estrategias evolutivas tales como $(\mu+\lambda)$ ó (μ,λ) (Whitley 1989; Swiswerda, 1989).

Uno de los puntos relevantes de los algoritmos genéticos es la teoría de *schematta*. El Teorema de *Schematta* entrega una prueba de porqué el paradigma de GA es capaz de resolver un problema de optimización basado en que en promedio, los individuos de las nuevas generaciones tienen un mejor *fitness* que el de las anteriores. La teoría tiene que ver con el hecho que GA trabaja con *strings*, y así hay cualidades relacionadas con el *fitness* inherentes a los *strings* que se utilizan.

4.2.3.1.1 Teoría de schematta

Los cromosomas formados por unos o ceros representan soluciones explícitas (luego de la decodificación) para el problema de optimización involucrado. Sin embargo en determinadas posiciones estos unos o ceros pueden ser reemplazados por "*", que representan un comodín o "*don't care*" (como son llamados en la literatura), que dejan abiertos los valores de esos bits, y así se forman híper-planos que incluyen todos los valores posibles en esos "*".

En forma genérica, los *strings* que contienen "*" se conocen como *schematta*. Un *schema* entonces es un *string* en particular formado por $\{0,1,*\}$, que representa todos los *strings* que son iguales al *schema* en las posiciones que no están definidas por "*". En general cada *schema* representa 2^r *strings*, donde r es el número de bits "*don't care*". Cada cromosoma binario (sin "*") representa un vértice en el híper-cubo del espacio de búsqueda, y es miembro de 2^L-1 híper-planos diferentes. Para *strings* binarios de largo L hay 3^L *hiperplanos* (o *schematta*) posibles.

Una población de individuos entrega información sobre diversos híper-planos. Más aún, muchos individuos entregan información sobre híper-planos de orden bajo[4]. El paralelismo implícito o intrínseco de GA viene del hecho que muchos híper-planos son muestreados cuando se evalúa una población en particular. Y como incluso muchos híper-planos distintos son evaluados al considerar un solo *string*, el efecto acumulativo de muestrear una población entrega información estadística acerca de un conjunto particular de híper-planos. Así, el paralelismo implícito implica que muchas competiciones ente diversos híper-planos son ejecutadas en paralelo. La teoría sugiere que a través del proceso de reproducción y selección, los *schematta* de los híper-planos

[4] El orden de un *schema* está dado por el número de bits fijos en el *string*.

en competencia aumentan o disminuyen su representación en la población de acuerdo al *fitness* relativo de los *strings* que se encuentran en dichos híper-planos. Es además posible predecir si la representación de un *schema* en particular aumentará o disminuirá, si se utiliza selección proporcional. Luego, por el efecto del *crossover* y mutación se puede encontrar una cota inferior en la representación de un híper-plano en la generación $t+1$ a partir de la generación t.

Efecto del operador de selección

Si se utiliza selección proporcional, entonces la probabilidad de selección de un *schema* es $f(S,t)/F(t)$ donde $f(S,t)$ es el *fitness* promedio de los individuos representando al schema S, y $F(t)$ es el *fitness* promedio de la población. Si $\zeta(S,t)$ es el número de representantes del esquema S en el tiempo t, entonces el número de representantes esperados del esquema S en $t+1$ es:

$$\zeta(S,t+1) = \zeta(S,t)\frac{f(S,t)}{F(t)} \tag{4.13}$$

Si además definimos $\varepsilon = \frac{f(S,t)-F(t)}{F(t)}$, donde $\varepsilon > 0$ si el *fitness* promedio del esquema es superior al *fitness* medio de la población, se puede ver que un *schema* con un *fitness* sobre el promedio posee un crecimiento exponencial de representantes en la población a lo largo de la ejecución del algoritmo.

$$\zeta(S,t) = \zeta(S,0)(1+\varepsilon)^t \tag{4.14}$$

Efecto del crossover

El *crossover* en general posee el efecto de destruir *schematta*. Si definimos largo definitorio del esquema S, $\delta(S)$, como la distancia entre las posiciones fijas más lejanas en el *string* , entonces la probabilidad de supervivencia del esquema *S* es:

$$p_s(S) = 1 - \frac{\delta(S)}{L-1} \tag{4.15}$$

Luego, si consideramos además que se ejecuta el *crossover* con una probabilidad p_c, y que un *crossover* bien podría no destruir *schema* en todos los casos, se tiene la cota inferior para la supervivencia del *schema* dada por:

$$p_s(S) \geq 1 - p_c\frac{\delta(S)}{L-1} \tag{4.16}$$

Efecto de la mutación

Si la probabilidad de mutación de un bit es p_m, entonces la probabilidad de supervivencia de un bit en particular es $1-p_m$. Si el orden del schema es $o(S)$, entonces la probabilidad de supervivencia del esquema a la mutación es:

$$p_s(S) = \left(1 - p_m\right)^{o(S)} \tag{4.17}$$

Pero como $p_m << 1$, entonces (4.17) se puede aproximar como $p_s(S) \approx 1 - o(S)\,p_m$.

Teorema de Schematta

A partir de los efectos combinados de selección, *crossover* y mutación, se encuentra la nueva expresión para el cambio en la representación del *schema* S en la generación $t+1$.

$$\zeta(S, t+1) \geq \zeta(S,t)\frac{f(S,t)}{F(t)}\left(1 - p_c\frac{\delta(S)}{L-1} - o(s)\,p_m\right) \tag{4.18}$$

A partir de (4.18) se puede concluir que los *schematta* cortos, de orden bajo y de *fitness* alto tendrán un aumento exponencial en las siguientes generaciones de GA. Lo que sustenta el funcionamiento del algoritmo genético, pues garantiza que en general aumentará el *fitness* de la población a través de los híper-planos definidos por los *schematta*.

Hipótesis de bloques constituyentes

La teoría del funcionamiento de GA se completa con la hipótesis de bloques constituyentes. Esta se basa en los siguiente: "Un algoritmo genético busca un desempeño cercano al óptimo a través de la yuxtaposición de schematta pequeños, de orden bajo y de alto *fitness*, llamados bloques constituyentes" (Michalewicz, 1996).

Los operadores genéticos poseen la habilidad de generar, promover, y yuxtaponer bloques constituyentes para formar *strings* óptimos. El *crossover* tiende a conservar la información genética presente en los *strings*, de modo que cuando los *strings* usados para el *crossover* son muy similares la capacidad de generar nuevos bloques constituyentes disminuye. Por su parte, el operador mutación no es un operador conservador, sino que es capaz de generar nuevos bloques constituyentes.

El operador de selección, por su parte, ayuda a sesgar el proceso hacia los bloques constituyentes que poseen *fitness* más altos, y asegura su representación de generación en generación. Esta hipótesis sugiere que la codificación en GA para su desempeño, y que debe satisfacer la idea de bloques constituyentes pequeños.

Todo esto muestra que GA es en esencia un algoritmo que opera a nivel genotípico, es decir, que opera directamente en la codificación de las soluciones.

4.2.3.2 Estrategias evolutivas

Las Estrategias Evolutivas (ES, *Evolutionary Strategies*) fueron desarrolladas por Rechenberg (1965, 1973) y Schwefel (1965, 1977). En general usan mutaciones con una distribución normal para modificar vectores de valores reales y enfatizan tanto la recombinación como la mutación como operadores de búsqueda esenciales tanto en el espacio de búsqueda como en el espacio de

parámetros del problema. Los operadores de selección son determinísticos, donde el tamaño de la población de padres e hijos generalmente es distinto.

Algoritmo # 5: Estrategia Evolutiva Estándar

01. $t = 0$. Inicializar Población $P(0)$ consistente de μ padres.
02. Evaluar Población $P(0)$.
03. Mientras no se cumpla condición de término
04. $t = t + 1$
05. Para i = 1... λ
06. Escoger $\rho \geq 2$ padres al azar.
07. Recombinar sus pasos de mutación y variables de optimización.
08. Mutar los pasos de mutación y variables de optimización de la descendencia generada en el paso anterior.

$$p_i'(t) = p_i(t) + N(0, \sigma_i(t))$$

$$\sigma_i'(t) = \sigma_i(t) + N(0, \sigma_i(t)/c)$$

donde un valor pequeño de c permite una adaptación grande y viceversa.
09. Evaluar el *fitness* de la descendencia. Añadir descendencia a la población de descendientes $C(t)$.
10. Finalizar
11. Seleccionar los mejores μ individuos para formar la nueva población de padres $P(t)$, a partir de la población de descendientes $C(t)$ o de la unión de descendientes y de padres $P(t-1)$.
12. Evaluar Población $P(t)$.
13. Finalizar
14. Solución del problema es el mejor elemento de P(t).

El marco algorítmico de las estrategias evolutivas modernas es presentado de un modo sencillo con la notación introducida por Schwefel (1977). La abreviación $(\mu + \lambda)$ ES denota una ES que genera λ descendientes a partir de μ padres y selecciona los mejores μ individuos para la siguiente generación a partir de los $\mu + \lambda$ individuos totales. En general, en una ES se utiliza que $1 \leq \mu \leq \lambda < \infty$.

Otro tipo de selección utilizada en ES es la (μ, λ). En ésta $\lambda = k\mu$ para algún $k > 1$. El proceso consiste en que k descendientes son generados de cada padre a través de mutación o posiblemente de recombinación, y los μ mejores descendientes (sin incluir a los padres) son seleccionados para la siguiente generación. Estudios experimentales muestran que $k \approx 7$ es un valor óptimo (Schwefel, 1987). De acuerdo a Back y Schwefel (1993), el método (μ, λ) es preferible al $(\mu + \lambda)$, pues es más robusto frente a medios probabilísticos o cambiantes.

En ES se utiliza mutación Gaussiana. $x' = x + M$, donde M es un número aleatorio perteneciente a una distribución Gaussiana $N(0, \sigma)$. En las primeras implementaciones se observó que el rendimiento del algoritmo mejora si se varía de modo inteligente la desviación estándar o paso de mutación σ de la distribución normal utilizada en la mutación. Así, las ES actuales consisten de

un elemento $x \in \mathbb{R}^n$ que incluye elementos del espacio de búsqueda y otros parámetros individuales que controlan la distribución de la mutación. Entonces, en cada iteración se realiza una mutación para la varianza de cada elemento del vector y de los parámetros propiamente tales del problema de optimización.

Para la generación de la descendencia, aparte de la mutación, se pueden utilizar operadores de recombinación tanto para los parámetros del problema de optimización como para los parámetros del algoritmo que regulan la adaptación del paso de mutación.

De este modo, un pseudo-código para una estrategia evolutiva se presenta en Algoritmo # 5.

Se destaca que las diferencias que en un principio fueron claras respecto de los algoritmos genéticos se han ido desvaneciendo y cada vez más sus implementaciones pueden coincidir en algunos aspectos con la del otro algoritmo, y diferir de la implementación del mismo algoritmo en sus versiones anteriores.

4.2.3.3 Programación evolutiva

La Programación Evolutiva (EP, *Evolutionary Programming*) fue desarrollada originalmente por Fogel (1964). Enfatiza el uso de operadores de mutación y no incorpora la recombinación de individuos. Al igual que en ES en problemas de optimización con variables reales, trabaja con mutaciones de distribución normal y extiende el proceso evolutivo a los parámetros del algoritmo. El operador de selección es probabilístico, y aunque el algoritmo fue inicialmente ideado para evolucionar máquinas de estados finitos, actualmente la mayor parte de las aplicaciones están reportadas en espacios de búsqueda de variables reales.

EP difiere diametralmente en los principios de GA (se puede decir que ES se encuentra entre medio). Mientras GA es un algoritmo principalmente genotípico, EP es fundamentalmente fenotípico. En EP los individuos son juzgados solamente en términos del *fitness*, y no se hace el intento de buscar los bloques constituyentes como en GA. Por lo tanto, no se utiliza el operador de *crossover* (diseñado para recombinar bloques constituyentes) no se utiliza en las formas generales de EP. Además, el operador de mutación (tal como en ES) puede afectar a la solución completa, y no sólo a una variable. Todo esto parte de la suposición que la yuxtaposición de buenos bloques no tiene porqué suponer la formación de buenas soluciones.

El método se desarrolla en un intento de generar inteligencia artificial. El comportamiento inteligente se veía como la habilidad compuesta de: (i) predecir el ambiente, y (ii) traducir las predicciones en respuestas adecuadas de acuerdo a un objetivo determinado. En Para generalizar, el ambiente es descrito como una secuencia de símbolos tomados de un alfabeto finito. El problema es definido como evolucionar algoritmos (en particular máquinas de estados finitos) que operan en la secuencia de símbolos conocidos, de modo de producir un símbolo de salida que maximiza el desempeño del algoritmo, dado que se conoce el próximo símbolo que aparecerá en el ambiente y una función de mérito bien definida (Fogel *et al.*, 1966).

El problema puede ser adaptado de forma sencilla para problemas de optimización continua. La codificación así puede ser tomada de forma natural como un *string* de valores reales. La población inicial es seleccionada al azar respecto a una densidad y se evalúa respecto a un objetivo. La descendencia es creada a partir de estos padres a través de mutación aleatoria. Típicamente, cada componente es perturbada por una variable aleatoria Gaussiana con media

cero y una varianza adaptable. La selección se basa en un método probabilístico, en general un torneo probabilístico. Cada solución se hace competir con oponentes de la población seleccionados al azar. La solución gana si es mejor o igual que sus oponentes en términos de la función objetivo (*fitness*). Típicamente cada solución debe participar en 10 competiciones, pero puede variar de acuerdo al grado deseado presión selectiva (mientras mayor presión, mayor número de competiciones). Las soluciones con más victorias son seleccionadas para convertirse en padres de la siguiente generación. El procedimiento se repite hasta que se obtiene una solución adecuada o el tiempo computacional disponible se agota.

Así, para un problema de optimización continua, EP se implementa como se presenta en Algoritmo # 6.

Algoritmo # 6: Programación evolutiva para optimización continua
01. $t = 0$. Inicializar Población $P(0)$ de tamaño N donde cada solución posee una dimensión igual al número de variables del problema de optimización, y la población $\Sigma(0)$ donde cada elemento contiene la varianza de la mutación de los elementos de $P(t)$.
02. Evaluar Población $P(0)$.
03. Mientras no se cumpla condición de término
04. $t = t + 1$
05. Crear para cada padre una descendencia $P'(t)$ y $\Sigma'(t)$, mediante un operador de mutación. Del siguiente modo: $p_i'(t) = p_i(t) + N(0, \sigma_i(t))$ $\sigma_i'(t) = \sigma_i(t) + N(0, \sigma_i(t)/c)$ donde un valor pequeño de c permite una adaptación grande y viceversa.
06. Evaluar la descendencia $P'(t)$.
07. Realizar torneo. Para cada elemento de $P(t)$ y de $P'(t)$ se realiza un competencia uno contra uno con 10 individuos seleccionados al azar.
08. Seleccionar los N individuos con mayor cantidad de triunfos, y sus correspondientes elementos $\Sigma(t)$ ó de $\Sigma'(t)$ en la nueva población $P(t+1)$ y $\Sigma(t+1)$.
09. Finalizar
10. Solución del problema es el mejor elemento de P(t).

4.2.3.4 Programación genética

La programación genética (*Genetic Programming,* GP) está bastante alejada de los paradigmas típicos de los EA (Koza, 1992). Así, se aplican los principios de búsqueda evolutivos para desarrollar automáticamente códigos de computador, que se encuentran representados jerárquicamente en una estructura de árbol, en lugar de estar en la forma de *strings* binarios o de números reales como en las tres heurísticas anteriores. Otra diferencia es que en GP los individuos (o programas generados) pueden variar en tamaño, forma y complejidad, mientras que las otras tres heurísticas en general utilizan un tamaño fijo para los individuos. A pesar de esto, existe la perspectiva de que GP es un subconjunto de GA que evoluciona programas ejecutables, pues utiliza los mismos operadores de selección, mutación y recombinación.

El objetivo de una implementación de GP es encontrar programas de computador dentro de un espacio de programas potenciales que entregue una salida deseada para un conjunto de entradas dadas. Cada programa es representado como un árbol ordenado, donde las funciones definidas para el problema se encuentran en los nodos del árbol y las constantes o entradas se ubican en las hojas.

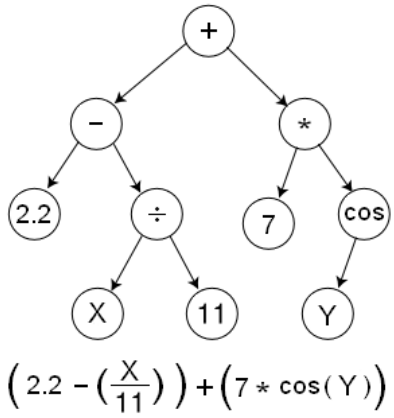


Figura 14: árbol de un programa genético

Los operadores evolutivos utilizados en GP son mutación y *crossover*. El *crossover* se realiza con dos individuos simplemente intercambiando uno de sus nodos (y todas las componentes abajo de este en el árbol) por uno del otro individuo. Por ejemplo en la Figura 15, se intercambian los nodos identificados por la flecha roja, y todas las componentes por debajo de dichos nodos, para formar los nuevos árboles. La mutación en un individuo puede realizar reemplazando un nodo completo (y sus componentes inferiores) por uno generado aleatoriamente o bien reemplazando tan sólo la información del nodo. En la Figura 16, se ejemplifica el segundo caso. En general la selección al igual que en GA se realizada con selección proporcional.

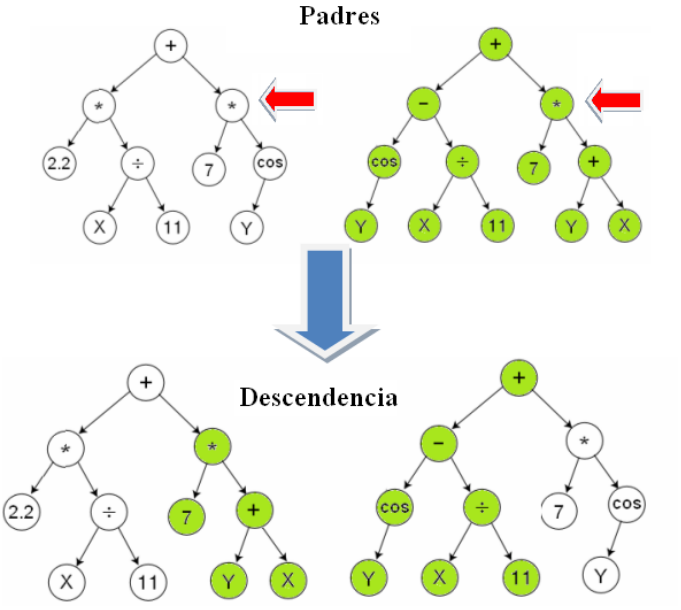


Figura 15: crossover en programación genética.

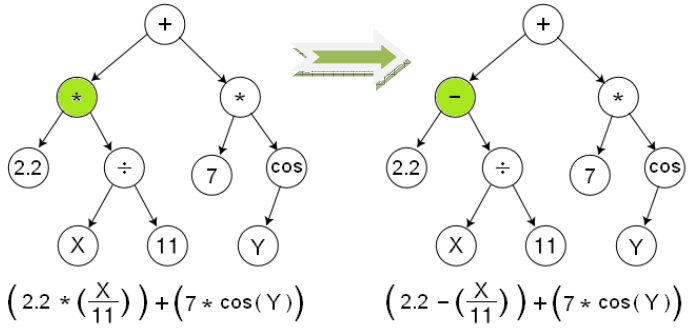


Figura 16: mutación cambiando un operador en programación genética.

Los pasos necesarios para la implementación de un GP son: especificar el conjunto de salidas deseadas; especificar el conjunto de funciones utilizables, que en general pueden ser operaciones binarias (*and*, *or*, *not*) o bien operaciones aritméticas (+, / , etc); especificar la medida de *fitness*, que en general se utiliza siendo inversamente proporcional al error producido por la salida del programa, o en otro tipo de aplicaciones comunes como juegos puede ser el puntaje que el programa obtiene en el juego; seleccionar los parámetros de control del algoritmo, como tamaño de población, número máximo de iteraciones, probabilidad de reproducción, de *crossover*, de mutación y el tamaño máximo permitido en los niveles jerárquicos en las poblaciones de programas; especificar la condición de término, que en general es alcanzar el máximo número de iteraciones.

Luego, un pseudo-código de la implementación de GP se presenta en Algoritmo # 7.

Algoritmo # 7: Programación genética estándar
01. $t = 0$. Inicializar Población de programas $P(0)$.
02. Evaluar Población $P(0)$.
03. Mientras no se cumpla condición de término:
04. $t = t + 1$
05. Seleccionar población $C(t)$ a partir de $P(t-1)$.
06. Crear población $C'(t)$ a partir de $C(t)$ utilizando operadores especializados de recombinación y mutación.
07. Construir población $P(t)$ a partir de descendencia $C'(t)$ y completar en caso de ser necesario con población de padres $P(t-1)$.
08. Evaluar Población $P(t)$.
09. Finalizar
10. Solución del problema es el mejor elemento de P(t).

4.2.4 Dinámica, convergencia y selección de parámetros en algoritmos evolutivos

La sintonía de parámetros para los algoritmos evolutivos es una tarea ardua, puesto que en general los distintos parámetros no son independientes, y probar todas las combinaciones posibles (incluso si se considera un conjunto pequeño de parámetros a estudiar) es un problema

gigantesco. Debido a esto, han existido variados esfuerzos para encontrar ciertos conjuntos de parámetros que funcionen bien en la mayor cantidad posible de algoritmos, y que así sea posible evitar esta sintonía. Algunos esfuerzos en esta línea han sido realizados, y las sugerencias entradas en estos trabajos respecto de los parámetros más importantes se muestran a continuación.

De Jong (1975) encuentra que un valor óptimo para la tasa de mutación (*bit-flip*) es $p_m = 0.001$. Por su parte Back (1993) muestra que utilizar tasa de mutación óptima en un problema unimodal es $p_m = 1/n$, donde n es el número de bits de un cromosoma. Además, Eiben y Smith (2003) dicen que en general la tasa de mutación se escoge para que en promedio, por generación, cambie al menos un bit en toda la población, y a lo más un bit por cromosoma. Back *et al*. (2000) comenta que la sintonía del paso de mutación al usar mutación Gaussiana es bastante compleja y altamente dependiente del problema. Respecto de la probabilidad de crossover (de 1 punto), De Jong (1975) encuentra que el valor óptimo es $p_c = 0.6$, mientras que Grefenstette (1986) muestra que se debe encontrar en el intervalo [0.45,0.95] y Schaffer (1989) concluye que éste debe pertenecer al intervalo [0.75,0.95]. Si se utiliza crossover uniforme, se suele utilizar una tasa $p_c = 0.5$ (Eiben y Smith, 2003) (Back *et al*., 2000). Acerca del tamaño de población en general se utilizan tamaños mayores a 50 para que los algoritmos sean estables con poca variabilidad entre distintas réplicas del algoritmo y posean buenas características de búsqueda global (Eiben y Smith, 2003) (Back *et al*., 2000).

Sin embargo, se debe tener en cuenta que distintas variantes con que se implementen los algoritmos pueden llevar a que el comportamiento con algunos parámetros sea mejor en algunas versiones que en otras, y más aún, las distintas variantes poseen distintos parámetros. Por eso, para tener una mejor idea de qué parámetros usar dependiendo de los operadores a utilizar se sugiere revisar las referencias anteriores y: Eiben y Smith (2003), Lobo *et al*. (2007) y Back *et al*. (2000).

Luego Smit y Eiben (2009) utilizan diversos algoritmos para sintonizar un EA con selección por torneo, elitismo, crossover de un punto y mutación Gaussiana. Se encuentra que conviene utilizar mutaciones con un paso relativamente alto (del orden de 0.6) (espacio de búsqueda normalizado) con una probabilidad pequeña (entre 0.04 y 0.06). Se encuentran además rangos amplios aceptables para el crossover (0.38 a 0.78), entre otros. Mostrando que en general, si se tiene la opción conviene utilizar algún algoritmo de sintonía pues se puede efectivamente mejorar el desempeño de los algoritmos, y consistentemente se encuentran mejores resultados que seleccionándolos a mano.

A pesar de lo que se ha presentado, sido comprobado que para distintos problemas los parámetros óptimos pueden ser muy distintos, y por lo tanto la tarea de sintonización de los parámetros debiese ser realizada cada vez que se quiera resolver un problema de optimización. Esto se recomienda además porque cada variante que se utilice en la implementación cambia la dinámica y finalmente los parámetros óptimos también debieran ser distintos. Por otra parte, ha sido mencionado que los parámetros óptimos para un algoritmo dependen del estado de evolución de la población. Por ejemplo, cuando recién comienza la ejecución del algoritmo, conviene utilizar tasas de mutación altas para favorecer la exploración, mientras cuando se está más avanzado, conviene utilizar tasas de mutación más pequeñas, para favorecer la explotación de las soluciones. El control de parámetros se refiere a las estrategias para utilizar parámetros

cambiantes a lo largo de la ejecución del algoritmo. Algunas alternativas son utilizar parámetros variantes en el tiempo (caso determinístico), parámetros que son función del estado del sistema (parámetros adaptativos), o que los parámetros sean parte de los cromosomas y evolucionen junto con la población (parámetros auto-adaptativos). Para ver una revisión detallada de estas técnicas se sugiere revisar (Eiben y Smith, 2003).

Los temas de la convergencia de la población de los algoritmos evolutivos y la justificación del funcionamiento de este tipo de algoritmos han sido enfocados desde diversas perspectivas. El primer tratamiento de este tema, y uno de los más populares es el basado en la teoría de *schematta*. Este trabajo brinda argumentos de porqué los algoritmos genéticos funcionan, el cual está basado en que los *schematta* con *fitness* sobre el promedio incrementan su representación en la población de generación en generación. Los problemas que tiene esta teoría es que no explica cómo se comporta la población cerca de un óptimo, y además existen críticas respecto de si efectivamente los óptimos son ejemplos de *schematta* de orden menor con un alto *fitness*, como se establece en la hipótesis de bloques constituyentes. Debido a estas limitaciones se ha desarrollado otras formulaciones que intentan explicar el comportamiento, la dinámica y el porqué funcionan de los algoritmos evolutivos.

En general, los trabajos más importantes en esta línea son los que modelan los algoritmos como un sistema dinámico, mediante la mecánica estadística, o como una cadena de Markov.

El tratamiento como un sistema dinámico ha sido fundamentalmente analizado en algoritmos genéticos en espacios de búsqueda discretos. Una de las principales contribuciones a este tema ha sido realizada por Vose (1999). En su análisis se considera un vector de estado de largo *n* (que es tamaño del espacio factible discreto) donde la coordenada *i* representa la proporción de la población que se encuentra en la solución *i* (la formulación considera una población infinita). La actualización del estado se maneja mediante matrices que representan el efecto de los operadores de selección y variación, y la composición de éstas define un campo de fuerza que actúa sobre la población. Este análisis entrega resultados que son consistentes con la teoría de *schematta*, en el sentido que estos proveen un modo natural de agrupar los *strings* en clases de equivalencia al considerar recombinación y mutación. Basado en un análisis similar (Nimwegen *et al*., 1999) se predice precisamente el descubrimiento de nuevos niveles de *fitness* en la búsqueda evolutiva, que dependen exclusivamente de los parámetros del algoritmo genético, al utilizar poblaciones finitas.

Los análisis basados en la mecánica estadística, tal como se realiza en esta rama, se focaliza en describir el comportamiento del sistema por unas pocas variables macroscópicas (Prugel-Bennet, 1994, 2003; Prugel-Bennet y Rogers, 2001). En general, se trabaja con los promedios de los momentos del fitness ($\langle f \rangle, \langle f^2 \rangle, \langle f^3 \rangle, \ldots$ donde $\langle\ \rangle$ es el operador promedio), y se intenta determinar cómo evoluciona el fitness a lo largo de las generaciones, cuya precisión depende del número de momentos utilizados. En general sólo se analiza aspectos como la media y varianza de la población, sin ser capaces de modelar otros comportamientos de interés. A pesar de esto, se puede obtener predicciones con una gran precisión. Por ejemplo, Prugel-Bennet (2003) comparan este enfoque con uno basado en sistemas dinámicos y encuentra mejores predicciones del comportamiento dinámico de la media del *fitness* en la población (pero no en el valor final).

Los algoritmos evolutivos también han sido analizados como si fuesen cadenas de Markov, que son sistemas en los cuales la probabilidad de que el sistema pase a un estado sólo depende del

estado en el instante anterior, independiente de la secuencia de estados anteriores, tal como sucede con los algoritmos evolutivos. En base a este tipo de análisis Eiben *et al*. (1991) muestran que para un algoritmo evolutivo optimizando una función en un espacio discreto finito, la población converge a un óptimo bajo algunas condiciones: cada individuo posee una probabilidad no nula de ser seleccionado; el mecanismo de selección de individuos para la siguiente generación es elitista; y que cada solución puede ser creada con una probabilidad no nula a través de los operadores de variación. En particular se prueba que existe una n-ésima generación en la cual se encontrará el óptimo global. Luego Rudolph (1994) muestra que un algoritmo genético con mutación no nula y elitismo siempre convergerá al óptimo global, y enfatiza que sin utilizar elitismo nunca se convergerá al óptimo global. Posteriormente, Rudolph (1996) extiende los teoremas de convergencia a espacios continuos, donde se muestra que un algoritmo evolutivo con una selección elitista convergerá al óptimo global.

Rudolph (1994) argumenta que hablar de convergencia en espacios discretos no es tan importante pues en un tiempo finito se puede evaluar todas las soluciones posibles, y que en su lugar, es más importante analizar la complejidad computacional requerida para llegar al óptimo. Algunos trabajos en esa línea son realizados por Back (1992) y He y Yao (2002). Los últimos muestran que las poblaciones son altamente importante, pudiendo hacer que en ciertos casos se disminuya la complejidad computacional desde un tiempo exponencial a uno polinomial.

La mayor parte de los trabajos mencionados se han focalizado en los algoritmos evolutivos en espacios discretos. Para algoritmos en espacios continuos, también existen resultados respecto de esta línea. Ya han sido mencionados, por ejemplo, los resultados obtenidos por Rudolph (1996). Además, otros importantes resultados han sido obtenidos al modelar los sistemas mediante dos variables macroscópicas. Una de estas es la tasa de progreso, que mide la distancia del centro de masa de la población al óptimo global. La segunda es la ganancia de calidad, que describe el incremento esperado de fitness entre generaciones. Con el uso de mutaciones Gaussianas y de resultados conocidos de la estadística, se he encontrado con relativa facilidad las ecuaciones que describen la dinámica de las dos variables mencionadas. Basado en estos estudios se ha derivado la regla de éxito de "un quinto"[5] para las estrategias evolutivas $(1+1)\,ES$ (Rechenberg, 1973), y también se ha analizado los principios de auto-adaptación y estrategias multi-miembros (Beyer y Arnold, 2001).

De la revisión realizada se aprecia que existen una serie de recomendaciones estáticas para los parámetros de los algoritmos evolutivos, sin embargo, se advierte también que el buen desempeño de estos depende de la aplicación. Además, para rendimientos óptimos, se debiera utilizar parámetros dinámicos o adaptativos. Respecto de las propiedades dinámicas de los algoritmos se muestra que dadas ciertas condiciones pueden llegar a converger, lo que justifica su aplicación. Sin embargo, las condiciones pueden llegar a ser poco realistas, y por lo tanto se debe tener cuidado en verificar hasta qué grado se pueden extrapolar los resultados y si la utilización de este tipo de métodos efectivamente se justifica por sobre uno convencional.

[5] Esta regla consiste en que si la tasa entre las mutaciones exitosas (aquellas que entregan una solución mejor que la del padre) y todas las mutaciones es mayor que $1/5$, el paso de mutación debe ser aumentado, y en caso contrario debe ser disminuido.

4.2.5 Un nuevo paradigma: evolución diferencial

Evolución diferencial (*Differential Evolution,* DE) es un algoritmo meta-heurístico de optimización en espacios continuos. Fue diseñado gracias a los intentos de Ken Price de resolver el Problema de Ajuste Polinomial de Chebyshev que le fue introducido por Rainer Storn (Storn y Price, 1995). Posee muchas semejanzas con los paradigmas clásicos de EA, y por lo tanto se lo puede considerar como parte de la familia de los algoritmos evolutivos. En efecto, ha sido clasificado como un caso particular de las estrategias evolutivas $(1+1)ES$ (Storn y Price, 1997), pues cada individuo padre genera una descendencia y se realiza una competencia entre ellos para determinar quién es el sobreviviente. Sin embargo, debido a sus diferencias primordiales, como son el distinto modo de realizar la selección y mutación, se lo presenta a continuación por separado de la sección de algoritmos evolutivos, en la Sección 4.3.

4.3 Evolución diferencial

4.3.1 Introducción

El algoritmo de Evolución diferencial (*Differential Evolution,* DE), como casi todos los algoritmos evolutivos, es un optimizador basado en poblaciones que evalúa una función objetivo y, usando esos valores y un procedimiento evolutivo determinado, la población evoluciona en búsqueda del óptimo de dicha función (Price *et al*., 2005). Debido a su simplicidad, efectividad y robustez, DE ha sido aplicado exitosamente en diversas aplicaciones prácticas, por ejemplo; identificación de parámetros (Ursem y Vadstrup, 2003), procesamiento de imágenes (De Falco *et al.*, 2006), clustering (Paterlini y Krink, 2005), ruteo de vehículos (Cao y Lai, 2007), control óptimo, (Raj *et al.*, 2008.).

DE está muy relacionado con los algoritmos evolutivos, pues utiliza operadores de mutación crossover y selección sobre los miembros de su población. En particular, DE ha sido clasificado como un caso particular de las estrategias evolutivas (Storn y Price, 1997a), tanto por su codificación continua, como por la similitud del esquema de selección con $(1+1)ES$. Por otra parte, éste difiere significativamente de los algoritmos evolutivos en el sentido que la información respecto de la distancia y dirección de la población actual es utilizada para guiar el proceso evolutivo. DE utiliza las diferencias entre vectores seleccionados al azar (vector de *diferencias*) como la fuente de variaciones aleatorias para un tercer vector, que es llamado el vector *base*. Se generan soluciones *mutantes* a partir del vector *base* sumándole el vector de *diferencias* multiplicado por un factor de escalamiento. A este proceso se le conoce como la operación de mutación donde el vector objetivo es el que muta. Luego se aplica un paso de recombinación o de *crossover* entre el vector *padre* y el de *prueba*, el cual produce una descendencia (*offspring*) o vector de *prueba* (o *trial*) que es aceptada tan solo si ésta mejora el *fitness* del progenitor.

4.3.2 El algoritmo evolución diferencial

4.3.2.1 El algoritmo básico

Consideremos el problema de optimización continua:

$$\begin{aligned} &\min f(x) \\ &s.a.\ x \in \Omega \subseteq \mathbb{R}^P \end{aligned} \tag{4.19}$$

donde $f(x)$ es la función objetivo que se desea maximizar, sujeto a la restricción de igualdad definida por $g(x) = 0$. La dimensión del problema es p.

DE es un algoritmo de optimización estocástico basado en poblaciones que utiliza *NP* (*number of parameters*) vectores de parámetros (o individuos). *NP* no cambia durante el proceso de optimización. Cada individuo posee una dimensión igual a *P*, que es el número de parámetros a optimizar (dimensión de *x*). De este modo, cada vector corresponde a una solución potencial del problema de optimización. La población se inicializa aleatoriamente en el espacio de búsqueda. En cada iteración la población va evolucionando en camino a encontrar el óptimo, como se explica a continuación.

El algoritmo básico de evolución diferencial es descrito en detalle especificando sus tres operadores evolutivos: mutación, crossover y selección.

Mutación: en la generación t, para cada individuo (o en este contexto *padre*) $x_i(t)$, se crea un vector *mutante*, $v_i(t)$, que se genera creado mutando un vector *base*. El vector *base*, $x_{i_3}(t)$, es seleccionado aleatoriamente, pero asegurando que $i_3 \neq i$. Luego se seleccionan dos individuos al azar, $x_{i_1}(t)$ y $x_{i_2}(t)$ donde $i_1 \neq i_2 \neq i_3 \neq i$, y con ellos se calcula el vector de *diferencias* $x_{i_1} - x_{i_2}$. El vector *mutante* es entonces:

$$v_i(t) = x_{i_3}(t) + F(x_{i_1}(t) - x_{i_2}(t)) \tag{4.20}$$

donde el último término representa el paso de la mutación. $F > 0$ es un factor de escalamiento usado para controlar la amplificación de la variación diferencial (aumenta la diversidad de los vectores mutantes). Es importante mencionar que en este tipo de mutación la generación del vector de *mutante* no depende de ningún modo del *padre, como* se puede apreciar en la ecuación (4.20), pero de todos modos se encuentra asociado exclusivamente a este último.

Crossover: DE utiliza una estrategia de recombinación discreta donde los elementos del vector *padre* $x_i(t)$, se combinan con elementos del vector *mutante* $v_i(t)$, para producir una descendencia o vector de *prueba* $\mu_i(t)$. La estrategia de crossover básica (crossover binomial) es:

$$\mu_{ij}(t) = \begin{cases} v_{ij}(t) & si\ U(0,1) < CR \text{ o si } j = r \\ x_{ij}(t) & \text{si no} \end{cases} \tag{4.21}$$

donde $j = 1,...,P$ se refiere a cada coordenada de cada vector $\mu_i(t)$, P es la dimensión de cada vector, CR es la tasa de *crossover* (*crossover rate*) y $U(0,1)$ corresponde a un número aleatorio con distribución uniforme en el intervalo [0,1]. Así, la coordenada correspondiente de la descendencia es una combinación lineal estocástica de tres individuos cuando $U(0,1) < CR$, si no, toma el mismo valor que en el padre.

Además, $r \in \{1,...,P\}$ se escoge aleatoriamente al principio del *crossover* de cada individuo para asegurar que en al menos una coordenada del padre sea efectivamente reemplazada por la del vector de prueba.

<u>*Selección*</u>: el procedimiento de selección utilizado en DE es muy simple. Si la descendencia $\mu_i(t)$ tiene un mejor *fitness* que el padre, entonces lo reemplaza en la población. Si no es así, entonces el padre permanece en la población. Es decir,

$$x_i(t+1) = \begin{cases} \mu_i(t) & si\ fitness(\mu_i(t)) > fitness(x_i(t)) \\ x_i(t) & \text{si no} \end{cases} \tag{4.22}$$

Se debe notar que para un problema de minimización el *fitness* suele utilizarse como $-f(\cdot)$, así la expresión anterior se rescribe como:

$$x_i(t+1) = \begin{cases} \mu_i(t) & si\ f(\mu_i(t)) < f(x_i(t)) \\ x_i(t) & \text{si no} \end{cases} \tag{4.23}$$

Habiendo definido los operadores básicos, el se procede a describir el algoritmo. Una vez inicializada la población, se entra al ciclo del algoritmo. Se calcula el *fitness* de cada individuo (o padre) x_i y para cada uno de ellos se genera un vector *mutante* v_i mediante mutación. Se realiza un *crossover* entre cada padre x_i y su respectivo vector *mutante* v_i para producir una descendencia μ_i. Finalmente se realiza la selección, donde cada descendencia compite contra su padre y el ganador es seleccionado para aparecer en la siguiente generación. Los ciclos se repiten hasta que se llegue a un criterio de término, y entonces la mejor solución encontrada será la solución del problema de optimización. Los criterios de término pueden ser un número máximo de iteraciones, o un número de iteraciones sin mejora en la solución encontrada. A continuación se describe el algoritmo por medio del siguiente pseudo código:

Algoritmo # 8: Evolución Diferencial
01. Inicialización de la población $x_i(0), i = 1,..., NP$, y parámetros como NP, CR.
02. Mientras no se cumpla el criterio de término:
04. $t = t+1$
05. Calcular el *fitness* de cada individuo.
07. Para $i = 1 : NP$
08. $v_i(t)$ es generado mediante mutación.
09. $\mu_i(t)$ es generado mediante *crossover*
10. $x_i(t+1)$ es generado mediante selección.
12. Siguiente i
13. Finalizar
14. Solución del problema es el mejor $x_i(t)$.

Storn (1996) y Storn y Price (1997b) proponen 10 estrategias distintas para DE basadas en variaciones en el individuo que es perturbado en la mutación (vector *base*), el número de individuos utilizados en los vectores de diferencia en la mutación y el tipo de *crossover* utilizado.

Así, en la literatura de Evolución Diferencial los distintos esquemas son especificados utilizando la notación $DE/a/b/c$, donde a y b especifican el modo de construir el vector *mutante* y c especifica el tipo de *crossover*. La estrategia detallada más arriba es entonces conocida como $DE/rand/1/bin$, donde $rand$ hace referencia a que el vector perturbado por el vector diferencia en la mutación se escoge al azar, el "1" hace referencia a que sólo se utiliza un vector de diferencias, y bin indica que el tipo de *crossover* utilizado es el binomial.

Las distintas opciones para la mutación y el *crossover* utilizados en DE se muestran a continuación.

4.3.2.2 Variantes de Mutación y Crossover

4.3.2.2.1 Alternativas de Mutación

Las distintas variantes de *crossover* utilizadas en DE se diferencian en el modo de escoger el vector *base* y el número de vectores *diferencia* (además de como son escogidos). A continuación se presentan las alternativas más utilizadas.

La primera opción es la que ha sido presentada anteriormente, consiste en que tanto el vector *base* como los utilizados para crear un único vector de diferencias son seleccionados al azar (pero sin coincidir con el padre y siendo todos distintos. El efecto de esto se estudia en (Price *et al.*, 2005), donde se muestra que permite alcanzar buenas cualidades de convergencia). Así esta estrategia se denota $DE/rand/1/*$. Entonces el vector *mutante* se genera como:

$$v_i(t) = x_{i_3}(t) + F(x_{i_1}(t) - x_{i_2}(t)) \tag{4.24}$$

La segunda opción difiere de la primera en que se utilizan dos vectores de diferencia en lugar de uno (entonces se utilizan 4 vectores para generar estas diferencias, pero se mantiene que la selección de todos éstos es aleatoria). Esta estrategia se denota entonces $DE/rand/2/*$, y el vector *mutante* se obtiene como:

$$v_i(t) = x_{i_3}(t) + F(x_{i_1}(t) - x_{i_2}(t)) + F(x_{i_4}(t) - x_{i_5}(t)) \tag{4.25}$$

Utilizar más de una diferencia no ha probado ser un aporte significativo en el desempeño de DE, por lo tanto esta estrategia es mucho menos utilizada que el clásico $DE/rand/1/*$.

En la tercera opción, se propone utilizar el mejor individuo en la población actual como vector *base*, en lugar de escogerlo al azar como en las opciones anteriores. Esta alternativa se denota $DE/best/*/*$ (se deja indefinido el número de diferencias). El vector mutante se obtiene entonces como:

$$v_i(t) = x_{best}(t) + F(x_{i_1}(t) - x_{i_2}(t)) \tag{4.26}$$

La cuarta opción es utilizar como vector *base* una ponderación entre el vector padre y el mejor vector. Se denota $DE/current-to-best/*/*$. El vector mutante se obtiene como:

$$v_i(t) = \lambda x_i(t) + (1-\lambda)x_{best}(t) + F(x_{i_1}(t) - x_{i_2}(t)) \tag{4.27}$$

La última opción presentada es utilizar una ponderación entre el mejor vector y un vector seleccionado al azar como vector *base*. Se denota $DE/rand-to-best/*/*$. El vector mutante se obtiene como:

$$v_i(t) = \lambda x_{i_3}(t) + (1-\lambda)x_{best}(t) + F(x_{i_1}(t) - x_{i_2}(t)) \tag{4.28}$$

En las últimas dos alternativas el parámetro λ es utilizado para controlar la importancia relativa del mejor vector en el vector *base*.

Recientemente se han desarrollado variantes que limitan la elección de los vectores involucrados en la generación del vector mutante a una vecindad, definida por una topología de anillo del elemento actual (Chakraborty *et al*. 2006; Omran *et al.* 2006). Otra variante para escoger el padre se basa en un ranking de los elementos de la población (Sutton *et al.*, 2007). La idea es que al incrementar la presión selectiva de los padres se puede mejorar la habilidad de DE para lidiar con funciones no separables.

4.3.2.2.2 Alternativas de Crossover

En los algoritmos evolutivos el operador de *crossover* usualmente combina atributos de distintos padres. En el caso de DE, dado que el operador de mutación ya se encuentra basado en una recombinación de individuos, el rol del *crossover* es algo distinto. Tan sólo se utiliza para la construcción de una descendencia μ_i, que es generada mezclando las componentes del padre x_i, y el elemento generado por la mutación, el vector *mutante* v_i. Las dos alternativas principales para el *crossover* son el *crossover* binomial, y el *crossover* exponencial.

El *crossover* binomial es el que ya ha sido explicado en la Sección 4.3.2.1, y consiste en que para cada coordenada de la descendencia se realiza una selección aleatoria entre la coordenada correspondiente del padre y la del vector de prueba. Tal como se muestra a continuación:

$$\mu_{ij}(t) = \begin{cases} v_{ij}(t) & si\ U(0,1) < CR \text{ o si } j = r \\ x_{ij}(t) & \text{si no} \end{cases} \tag{4.29}$$

donde $j = 1,...,P$ se refiere a cada coordenada de cada vector $\mu_i(t)$, P es la dimensión de cada vector, p_r es la probabilidad de reproducción, CR es la tasa de *crossover* (*crossover rate*) y $U(0,1)$ corresponde a un número aleatorio con distribución uniforme en el intervalo [0,1].

El nombre de este tipo de *crossover* viene de la propiedad que el número de componentes tomados del vector de prueba tiene una distribución binomial (Price *et al.*, 2005). Este *crossover* binomial es muy similar al *crossover* uniforme utilizado en los algoritmos evolutivos.

La otra opción, el *crossover* exponencial, ha sido diseñada para emular el *crossover* de un punto y el *crossover* de dos puntos utilizados en algoritmos genéticos. Así, la descendencia contiene una secuencia de componentes sucesivos (de una forma circular) tomados desde el vector de pruebas. La estructura del vector de descendencia es descrita en la ecuación (9), donde;

$k \in \{1,2,...,n\}$ es un índice aleatorio, que representa donde parte la secuencia de componentes sucesivos; L es un valor aleatorio en $\{1,2,...,N\}$ que define el largo de esta secuencia; y $\langle j \rangle_N$ es j si $j \leq N$ y $j-N$ si $j > N$.

$$\mu_{ij} = \begin{cases} v_{ij} & \text{si } j \in \{k, \langle k+1 \rangle_N, ..., \langle k+L-1 \rangle_N\} \\ x_{ij} & \text{si no} \end{cases} \tag{4.30}$$

L debe cumplir que $\text{Prob}(L = h) = (1-CR)CR^{h-1}$, que corresponde a la distribución geométrica, la contraparte discreta de la distribución continua exponencial. Esto justifica el nombre del tipo de *crossover* (Price *et al.*, 2005).

El *crossover* exponencial es utilizado en la primera versión de DE (Storn y Price, 1995). Sin embargo, actualmente la mayor parte de las aplicaciones utilizan el *crossover* binomial. Una posible explicación es que en el sitio personal de Rainer Storn (http://www.icsi.berkeley.edu/~storn/code.html: Differential Evolution web page), en la sección dedicada a evolución diferencial, se menciona que el crossover binomial nunca es peor que el exponencial.

La diferencia principal entre ambas estrategias es el hecho que mientras en el caso binomial los componentes heredados del vector de prueba son seleccionados sin un orden establecido, mientras que en el caso exponencial éstos forman una o dos sub-secuencias compactas (ver Figura 17). El impacto de esta diferencia en la efectividad de DE aún no ha sido comprendida a cabalidad. La elección de una variante sobre otra es difícil ya que no hay resultados que establezcan la superioridad de una variante sobre otra en alguna clase de problemas.

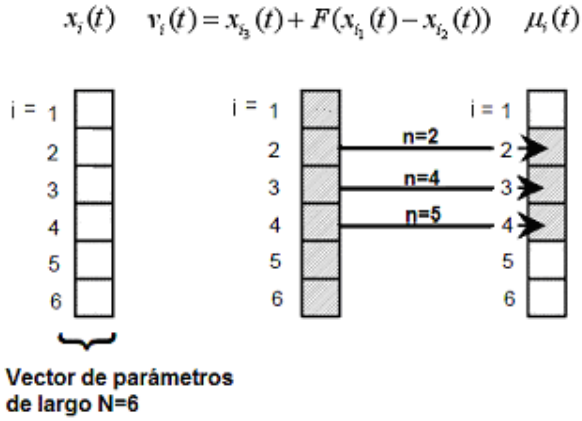


Figura 17: El proceso de crossover exponencial, con k=2, L=3.

4.3.3 Dinámica, convergencia y selección de parámetros en evolución diferencial

Evolución diferencial es un algoritmo evolutivo simple de codificación real, en el cual son dos los procesos principales que manejan la evolución; los procesos de variación, que permiten la exploración en diversas zonas del espacio de búsqueda, y luego el proceso de selección que

asegura la explotación del conocimiento previo del *fitness* de los individuos. Aunque existen muchos trabajos acerca del estudio empírico de la selección de parámetros y el proceso de sintonización en DE en su aplicación a problemas de optimización, además de la influencia de los operadores en las cualidades de exploración y explotación, existe poco acerca del estudio de la dinámica interna de DE, que puede permitir entender cómo y porqué DE logra encontrar el óptimo en diversas funciones numéricas de un modo tan rápido, y menos acerca de la estabilidad del algoritmo.

A continuación se presentan los dos trabajos más importantes que explican de cierto modo la dinámica y las propiedades de convergencia de evolución diferencial. En Dasgupta *et al*. (2008) se presenta un modelo matemático simple del esquema DE/rand/1/bin, que es el enfoque más popular (Price *et al.*, 2005). Se propone un modelo estocástico para la dinámica de los vectores, y se intenta encontrar la velocidad de cada vector a lo largo de las generaciones. Se muestra la similitud del mecanismo de búsqueda de DE con el del método del gradiente, y que por lo tanto puede ser interpretado como un algoritmo de gradiente modificado. De este modo, posee buenas cualidades como una rápida convergencia, sin embargo también posee una cierta tendencia de quedarse estancado en mínimos locales. Dasgupta *et al.* (2009) luego extienden este trabajo desarrollando un análisis de estabilidad basado en Lyapunov para la población en DE. Se muestra que el sistema es estable en el sentido de Lyapunov, donde un valor propio de la función de energía de Lyapunov del sistema es cero y el resto son negativos, y que posee una estimación de la constante de tiempo igual a $1/CR$. Los experimentos muestran que la velocidad de todos los vectores se acerca a cero a medida que se acercan al óptimo, y por lo demás convergen a este, lo que sugiere que la componente asociada al valor propio cero también vale cero y por lo tanto no influye en la dinámica del sistema. Además, se muestra que la estimación de la constante de tiempo es consistente con las mediciones experimentales.

Respecto de la selección de parámetros, en las primeras versiones de DE (Storn, 1996) se plantea que $F \in [0,2]$ y que $CR \in [0,1]$ como cotas generales, pero se recomiendan valores más acotados. Se sugiere que CR sea pequeño (en general $CR = 0.3$), pero si esto ocasiona que el método no converja, se propone utilizar $CR \in [0.8,1]$ en su lugar. Respecto del factor de escalamiento de la mutación se sugiere que $F \in [0.5,1]$. Además, respecto del tamaño de la población se recomienda que se utilice $NP = 10D$.

Después de esto, muchos otros trabajos han sido realizados para encontrar propuestas para los valores de estos parámetros. Ali y Torn (2004) y han encontrado que $CR = 0.5$ y $0.4 \leq F \leq 1$ es una buena elección. Price et al. (2005) argumenta que un límite máximo para F es 1 pues aplicaciones que han utilizado $F > 1$ consumen más tiempo y son menos confiables que cuando $F < 1$. Zaharie (2002) presenta un trabajo significativo, donde se analiza teóricamente la influencia de operadores de variación y sus parámetros en la esperanza de la varianza de la población. La varianza de la población se interpreta como una medida del balance entre exploración y explotación, y esto se utiliza para proponer valores de F. Se encuentra que $F > 0.3$ es un valor crítico para una buena búsqueda. Gamperle *et al*. (2002) por su parte, realizan un estudio con el cual concluyen que $F < 0.4$ no es un valor útil. Chackraborti *et al*. (2001) tiene éxito al resolver problemas utilizando $0.0001 \leq F \leq 0.4$, recomendando $F = 0.2$. Sin embargo, tales valores tan bajos son atípicos, mientras que los límites inferiores sugeridos por Zaharie (2002) y Gamperle et al. (2002) reflejan de mejor forma la norma.

Otros trabajos se han concentrado en la tasa de *crossover*. Si bien CR es una tasa de crossover, también se lo puede considerar como una tasa de mutación, en particular, la probabilidad aproximada que un parámetro puede ser heredado de un mutante. En muchos Gas se recomienda una tasa de mutación $1/D$, indicando que en promedio sólo un parámetro de prueba es mutado por iteración (Price, 2005). Storn y Price (2007) muestran que todas las funciones pueden ser resueltas con $0 \le CR \le 0.2$ o bien $0.9 \le CR \le 1$. Se aprecia que el espacio está bifurcado, tal como se propone inicialmente en Storn (1996). En un principio La razón para la bifurcación del espacio de CR no se entiende, pero en este trabajo se explica que se debe a que las funciones que son resolvibles con CR pequeño son siempre descomponibles, mientras que aquellas que requieren un alto CR no lo son.

Aparte de estos trabajos donde se considera F y CR constantes, existen otros donde se manejan de forma adaptativa (Zaharie, 2003), o como una variables aleatoria (Lampinen y Zelinka, 2000; Price, 2005).

De la revisión realizada se aprecia que existen una serie de recomendaciones para los parámetros F y CR, pero la mayoría se basa en consideraciones empíricas. Además, se muestra que recientemente se ha realizado algunos estudios en base a un modelo estocástico de los vectores que muestra que el algoritmo se comporta cerca del óptimo como un método del gradiente, lo que explica la gran eficacia que ha tenido el algoritmo para resolver diversos problemas. Esta prueba da el sustento teórico, que se suma al sustento empírico, para la aplicación de este algoritmo para problemas complejos altamente no lineales.

4.4 Optimización por Enjambre de Partículas (Particle Swarm Optimization, PSO)

4.4.1 Introducción

La inteligencia de enjambres (*Swarm intelligence*, SI), es un nuevo paradigma de la inteligencia computacional distribuida para resolver problemas de optimización que toma su inspiración original de fenómenos biológicos tales como el comportamiento de enjambres, bandadas y manadas en seres vivos vertebrados (Kennedy and Eberhart, 2001).

La optimización por enjambre de partículas (*Particle Swarm Optimization*, PSO) es un algoritmo de optimización estocástico basado en poblaciones que se encuentra inspirado en el comportamiento de bandadas de aves, cardúmenes, enjambres de abejas, o incluso en el comportamiento social humano (Kennedy y Eberhart, 2001). PSO es una herramienta de optimización basada en poblaciones que puede ser aplicada de un modo sencillo para resolver variados problemas de optimización. Como algoritmo, las principales fortalezas de PSO son su rápida convergencia, que se compara favorablemente con la de otros algoritmos de optimización global tales como algoritmos genéticos (Goldberg, 1989), temple simulado (*Simulated Annealing*, SA, Orosz y Jacobson, 2002). Para aplicar PSO satisfactoriamente, uno de los puntos claves es encontrar como mapear la solución del problema en la partícula de PSO, lo que afecta directamente su factibilidad y desempeño.

De un modo similar a los algoritmos evolutivos, PSO utiliza un mecanismo de búsqueda basado en poblaciones, donde los individuos (o partículas en la literatura de PSO) tienden a seguir al

individuo más fuerte (quién ha encontrado la mejor solución) o a la mejor solución encontrado por ellos mismos. De acuerdo a algunos estudios (Eberhart y Shi, 1998; Angeline, 1998), es posible afirmar que el comportamiento de PSO puede catalogarse como el de un algoritmo evolutivo, en algún lugar entre algoritmos genéticos y programación evolutiva.

4.4.2 Modelo canónico de optimización por enjambre de partículas

El modelo canónico de PSO consiste en un enjambre de partículas, que representa una población de soluciones candidatas al problema de optimización. Las partículas de dimensión *d* se mueven iterativamente por el espacio de solución del problema buscando nuevas soluciones, guiados por el *fitness f* de cada partícula (una medida de calidad de la solución, que para la mayor parte de los problemas de optimización conviene que sea la misma función objetivo por optimizar). Cada partícula *i* tiene una posición representada por un vector x_i, que representa la solución candidata al problema de optimización, y una velocidad representada por el vector v_i. Las coordenadas *j* de los vectores posición y velocidad de la partícula *i* son respectivamente x_{ij} y v_{ij}. Cada partícula recuerda su mejor posición en la historia de la búsqueda en el vector $pbest_i$ (*particle best*), y además, cada partícula conoce la mejor posición de cualquier partícula en la historia de la búsqueda, que se guarda en el vector $gbest$ (g*lobal best*). En la iteración $t+1$, se actualiza la velocidad de cada partícula de acuerdo su velocidad previa, y el seguimiento de $pbest_i$ y de $gbest$. Luego, la nueva posición se calcula como la suma de la posición previa con la nueva velocidad, tal como se muestra en la ecuación (4.31) y en la Figura 18.

$$\begin{aligned} v_{ij}(t+1) &= \omega \cdot v_{ij}(t) + c_1 \cdot \varphi_1 \cdot (pbest_i(t) - x_{ij}(t)) + c_2 \cdot \varphi_2 \cdot (gbest(t) - x_{ij}(t)) \\ x_{ij}(t+1) &= x_{ij}(t) + v_{ij}(t+1) \end{aligned} \tag{4.31}$$

donde *w* es el factor de inercia, c_1 es la constante cognitiva y c_2 es la constante social ($\omega, c_1, c_2 > 0$ son factores de sensibilidad del algoritmo). El comportamiento cognitivo significa que cada partícula del enjambre tiende a irse hacia su mejor posición conocida $pbest_i$, y el comportamiento social significa que cada partícula tiende a irse a la mejor posición conocida por cualquier miembro del enjambre $gbest$. Además, φ_1 y φ_1 son números aleatorios uniformemente distribuidos entre 0 y 1 que ayudan a mantener la diversidad del enjambre. La nueva velocidad de cada partícula es entonces actualizada de acuerdo a su inercia, y a correcciones debidas a su comportamiento social y cognitivo. Finalmente, su nueva posición es la posición actual más la nueva velocidad.

En el modelo PSO las partículas buscan las soluciones en un espacio con un rango $[-s, s]$. Para guiar efectivamente las partículas en el espacio de búsqueda, el movimiento máximo (velocidad máxima) durante una iteración debe encontrarse limitado por un rango $[-v_{\max}, v_{\max}]$. El valor de $v_{\max}$ es $p \times s$, con $0.1 \le p \le 1$, y usualmente se escoge como s, o sea $p = 1$.

El algoritmo consiste en lo siguiente. Luego de inicializar las posiciones y velocidades de las partículas, comienza el ciclo iterativo. Primero se calcula el *fitness* de cada partícula $x(t)$ que está asociado a la solución candidata codificada en su posición. Con esto es posible actualizar la

mejor posición entre todas las partículas, $gbest(t)$, y la mejor posición alcanzada por cada partícula, $pbest_i(t)$. Realizado lo anterior se calcula la nueva velocidad $v_i(t+1)$ y la nueva posición $x_i(t+1)$ de cada partícula, saturando en los límites predefinidos cuando corresponda. Una vez listo esto se pasa a la siguiente iteración. El proceso se repite hasta llegar a uno de los criterios de término, que usualmente son un número fijo máximo de iteraciones, o un número fijo de iteraciones en los que no hay mejora de la solución encontrada. La solución del problema de optimización está dada entonces por $gbest(t)$. El seudo código de PSO que representa el algoritmo recién descrito, para minimizar una función $f(\cdot)$, se muestra en Algoritmo 9 (Nadja y Macedo Mourelle, 2006).

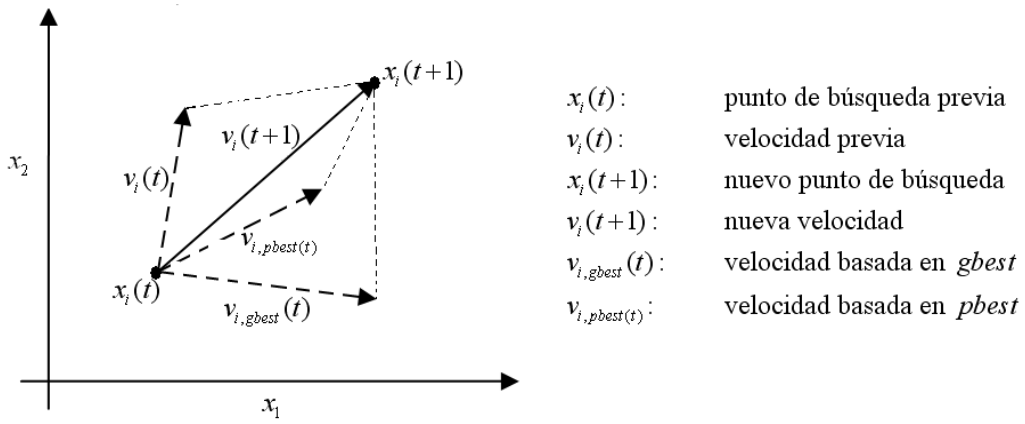


Figura 18: representación del cálculo del nuevo punto de búsqueda de una partícula

Algoritmo # 9: Optimización de Enjambre de Partículas (PSO)
01. Inicializar el tamaño del enjambre de partículas *n*, y otros parámetros como ω, c_1, c_2.
02. Inicializar las posiciones y las velocidades de las partículas aleatoriamente.
03. Mientras no se cumple el criterio de término
04. $t = t+1$
05. Calcular el *fitness* de cada partícula.
06. $gbest(t) = \text{argmin}_{i=1}^{n}\left(f(gbest(t-1), f(x_1(t)),..., f(x_i(t)),..., f(x_n(t))\right)$
07. Para $i = 1:n$
08. $pbest_i(t) = \text{argmin}\left(f(pbest(t-1), f(x_i(t))\right)$
09. Para $j = 1:d$
10. $v_{ij}(t+1) = \omega \cdot v_{ij}(t) + c_1 \cdot \varphi_1 \cdot (pbest(t) - x_{ij}(t)) + c_2 \cdot \varphi_2 \cdot (gbest(t) - x_{ij}(t))$ $x_{ij}(t+1) = x_{ij}(t) + v_{ij}(t+1)$ $x_i \in [-s, s]$ $v_i = [-v_{\max}, v_{\max}]$
11. Siguiente *j*
12. Siguiente i
13. Finalizar
14. La solución al problema de optimización está dada por *gbest(t)*.

El criterio de término usualmente es uno de los siguientes:

- Número máximo de iteraciones: el proceso de optimización termina luego de un número fijo de iteraciones.
- Número máximo de iteraciones sin mejoras: el proceso de optimización termina luego de un número fijo de iteraciones en los que no hay ninguna mejora en la solución encontrada.

El método de PSO descrito es inherentemente para problemas continuos, pero puede ser corregido trivialmente para variables discretas mediante truncación de la solución candidata (tanto al actualizar las posiciones como al evaluar). Existe además una versión del algoritmo para problemas binarios llamado PSO-binario, que es descrito en 4.4.4.2.

4.4.3 Modificaciones de PSO

4.4.3.1 Factor de inercia variable

En el algoritmo canónico de PSO, la actualización de la velocidad de las partículas (ver ecuación (4.31)) depende de tres factores; la inercia de la partícula, el comportamiento social, y el comportamiento cognitivo. La inercia de la partícula se encuentra controlada por ω, llamado "factor de inercia", y se refiere al efecto que posee la velocidad anterior de la partícula en la nueva velocidad. En las primeras versiones del algoritmo este factor de inercia no es considerado, que es equivalente a usarlo como $\omega = 1$ (Eberhart y Kennedy, 1995), y así no hay pérdida de energía en la velocidad, y esta se le suma directamente los efectos sociales y cognitivos para calcular la nueva velocidad. Posteriormente, Shi y Eberhart (1998) proponen incorporar explícitamente este factor de inercia al notar que distintos valores pueden influenciar positivamente el desempeño del algoritmo. Con esto, la ecuación de actualización de la velocidad de las partículas es la presentada en la ecuación (4.31).

En el mismo trabajo, Shi y Eberhart (1998), muestran que el uso de un factor de inercia variable, que decrece linealmente partiendo en 1.4 y terminando en 0.5 hacia el final de las iteraciones provee un mejor desempeño que utilizar un valor fijo de este parámetro. Un valor grande al comienzo de la búsqueda favorece la exploración de nuevas soluciones, mientras que un valor pequeño hacia el final favorece las componentes sociales y cognitivas, que a fin de cuentas permite una mejor explotación de las soluciones ya encontradas. Otros trabajos más recientes proponen que de este modo un valor inicial de 1.2 y su reducción gradual hasta 0 es considerada una buena elección para ω (Eberhart y Shi, 2002). Otros enfoques adaptativos (tales como un controlador difuso) han probado ser efectivos, donde el parámetro puede ser sintonizado de manera adaptable de acuerdo al problema en cuestión (Liu y Abraham, 2005; Shi y Eberhart, 2001).

4.4.3.2 Factor de constricción

Dadas ciertas condiciones, en particular que $\omega = 1$ y que $c_1 + c_2 \geq 4$, el movimiento de las partículas del algoritmo puede explotar, haciendo que todas ellas tiendan a irse hacia el infinito, a no ser que rápidamente cambie *gbest* o *pbest*. Para controlar de un modo efectivo esa situación, se implementa un factor de constricción χ (Clerc y Kennedy, 2002), que multiplica todas las componentes que se utilizan para calcular la velocidad, y así la expresión para la velocidad queda redefinida como:

$$v_{ij}(t+1) = \chi\left(v_{ij}(t) + c_1 \cdot \varphi_1 \cdot (pbest(t) - x_{ij}(t)) + c_2 \cdot \varphi_2 \cdot (gbest(t) - x_{ij}(t))\right) \quad (4.32)$$

Dicho coeficiente es conocido como el coeficiente de constricción tipo 1. Se plantea un modelo dinámico simplificado del enjambre de partículas, y en base a un análisis de valores propios, se tiene que χ está dado por la expresión:

$$\chi = \frac{2\kappa}{\left|2 - \varphi - \sqrt{\varphi^2 - 4\varphi}\right|} \quad (4.33)$$

donde $\varphi = c_1 + c_2$. Si $\kappa = 1$ y $\varphi = 4.1$, entonces el amortiguamiento del sistema es aproximadamente $\chi \approx 0.73$, amortiguando tanto la velocidad anterior como los otros términos (sociales y cognitivos). La condición para utilizar el factor de constricción es que $c_1 + c_2 \geq 4$, pues si no se da esta condición el sistema posee un comportamiento sobre-amortiguado y no hay explosión de los individuos, por lo tanto el factor de amortiguamiento no se encuentra definido. Si $\varphi = c_1 + c_2$ aumenta por sobre 4, la posibilidad de explosión aumenta, por lo tanto es necesario que el amortiguamiento sea mayor. Así, por ejemplo, si $\kappa = 1$ y $\varphi = 5$, entonces $\chi \approx 0.38$. La selección de $\kappa = 1$ permite un amortiguamiento adecuado en la medida que la convergencia es lo suficientemente lenta permitiendo que haya algo de exploración antes que la búsqueda converja. Si este parámetro fuese mayor, la convergencia sería muy rápida, sin permitir exploración.

4.4.3.3 Modelo lbest

El algoritmo PSO puede ser descrito como una población de vectores cuyas trayectorias oscilan en torno a una región que es definida por el mejor resultado de cada partícula y el mejor resultado de alguna "otra partícula". En el modelo canónico esta "otra partícula" es la que ha obtenido el mejor resultado en todo el enjambre, conocida como *gbest*. Otra alternativa es que la otra partícula del que se tome el mejor resultado no sea la mejor de todo el enjambre, sino que la mejor en alguna zona cercana a la partícula en cuestión. Esto constituye el segundo modelo básico, *lbest* (por *local best*), en contraste con el modelo canónico, que también es conocido como el modelo *gbest*. De este modo, la ecuación (1.2) *gbest* es reemplazado por *lbest* en el modelo. Así, la ecuación de actualización de las partículas se re-escribe como:

$$\begin{aligned} v_{ij}(t+1) &= \omega \cdot v_{ij}(t) + c_1 \cdot \varphi_1 \cdot (pbest(t) - x_{ij}(t)) + c_2 \cdot \varphi_2 \cdot (lbest(t) - x_{ij}(t)) \\ x_{ij}(t+1) &= x_{ij}(t) + v_{ij}(t+1) \end{aligned} \quad (4.34)$$

En el modelo *gbest* la trayectoria de búsqueda de cada partícula es influenciada por el mejor punto encontrado por cualquier miembro de la población. Esta mejor partícula actúa como un atractor, arrastrando a todas las partículas hacia él. Eventualmente, todas las partículas convergerán a su posición. El modelo *lbest* permite que cada individuo sea influenciado por un número menor de miembros adyacentes. Las partículas que se encuentren en un subconjunto del enjambre no tienen relación directa con las otras partículas de otros vecindarios. Típicamente las vecindades están compuestas por exactamente dos vecinos, pero pueden ser más. El caso límite cuando el número de vecinos se incrementa a todos menos la misma partícula, se llega al modelo *gbest*. Algunos experimentos muestran que el modelo *gbest* converge más rápido pero tiende a

quedarse estancado en óptimos locales, mientras que el modelo *lbest* converge más lentamente pero puede superar fácilmente el problema de los óptimos locales, pues los individuos exploran distintas regiones (Eberhart y Kennedy, 1995). El modelo *gbest* es recomendado fuertemente para problemas unimodales (que sólo poseen un óptimo), mientras que el modelo *lbest* es recomendado para problemas multimodales (con muchos óptimos locales).

4.4.4 Modelos extendidos para problemas discretos

4.4.4.1 Modelo PSO binario

Como ya ha sido mencionado, la versión canónica de PSO ha sido desarrollada para problemas de optimización con variables continuas. Sin embargo, muchos problemas prácticos de ingeniería son formulados como problemas de optimización combinatorial. El modelo PSO binario es presentado por Kennedy and Eberhart (1997) y está basado en una modificación sencilla del PSO original. En este caso, se mapea el dominio del problema a resolver en un set de string de bits, y una función de fitness. En el PSO binario se define la posición y velocidad de una partícula en términos de cambios de probabilidades de encontrarse en un estado u otro, i.e. sí o no, verdadero o falso, etc. Cuando la partícula se mueve en un espacio de estados restringido a cero y uno en cada dimensión, el cambio de probabilidad en el tiempo se define como:

$$P\left(x_{ij}(t+1)=1\right)=s\left(x_{ij}(t),v_{ij}(t),gbest_j(t),pbest_{ij}(t)\right) \tag{4.35}$$

donde la función de probabilidad usualmente es:

$$s\left(v_{ij}(t+1)\right)=\frac{1}{1+e^{-v_{ij}(t)}} \tag{4.36}$$

Así, en el algoritmo canónico tan solo se reemplaza la ecuación de actualización de la posición en (4.34), manteniendo igual la actualización de la velocidad, como se muestra a continuación:

$$\begin{aligned} v_{ij}(t+1) &= \omega\cdot v_{ij}(t)+c_1\cdot\varphi_1\cdot(pbest(t)-x_{ij}(t))+c_2\cdot\varphi_2\cdot(gbest(t)-x_{ij}(t)) \\ x_{ij}(t+1) &= \begin{cases} 1 \text{ si } \rho\le s\left(v_{ij}(t)\right) \\ 0 \text{ si no} \end{cases} \end{aligned} \tag{4.37}$$

donde $\omega, c_1, c_2, \varphi_1, \varphi_2$ son tal como en el algoritmo canónico, y ρ es una función aleatoria uniforme en el intervalo cerrado [0,1]. V_{max} se utiliza para limitar la exploración de las variables binarias una vez que la población ha convergido. Además, es importante notar que valores altos de V_{max} para variables continuas aumenta el rango de exploración, mientras que ocurre lo contrario para variables binarias.

4.4.4.2 Modelo PSO difuso

En el modelo de PSO difuso, las representaciones de la posición y velocidades de las partículas en PSO se extienden desde vectores reales a matrices difusas (Pang *et al*., 2004). Esto es ilustrado

utilizando problema de asignación de tareas. Los trabajos $J = \{J_1, J_2, ..., J_n\}$ se deben repartir en las máquinas $M = \{M_1, M_2, ..., M_m\}$, y la relación difusa entre M y J se expresa como:

$$X = \begin{bmatrix} x_{11} & x_{12} & \cdots & x_{1n} \\ x_{21} & x_{22} & \cdots & x_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ x_{m1} & x_{m2} & \cdots & x_{mn} \end{bmatrix} \tag{4.38}$$

donde x_{ij} representa el grado de pertenencia del i-ésimo elemento M_i en el dominio M y el j-ésimo elemento J_i en el dominio J. El significado de la relación X es la siguiente: por cada elemento en la matriz X, el elemento

$$x_{ij} = \mu_R(M_i, J_j), i \in \{1, 2, ..., m\}, j \in \{1, 2, ..., n\} \tag{4.39}$$

μ_R es la función de pertenencia, el valor de x_{ij} es el grado de pertenencia que M_j procesará en la solución de asignación factible. Los elementos de la matriz X deben satisfacer las siguientes condiciones:

$$x_{ij} \in [0,1], i \in \{1, 2, ..., m\}, j \in \{1, 2, ..., n\} \tag{4.40}$$

$$\sum_{i=1}^{m} x_{ij} = 1, i \in \{1, 2, ..., m\}, j \in \{1, 2, ..., n\} \tag{4.41}$$

Similarmente la velocidad de la partícula está definida como:

$$V = \begin{bmatrix} v_{11} & v_{12} & \cdots & v_{1n} \\ v_{21} & v_{22} & \cdots & v_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ v_{m1} & v_{m2} & \cdots & v_{mn} \end{bmatrix} \tag{4.42}$$

Entonces se tienen las siguientes expresiones para actualizar las posiciones y las velocidades de las partículas en PSO discreto difuso.

$$\begin{aligned} V(t+1) &= \omega V(t) + (c_1 r_1)\left(X^{pbest}(t) - X(t)\right) + (c_2 r_2)\left(X^{gbest}(t) - X(t)\right) \\ X(t+1) &= X(t) + V(t+1) \end{aligned} \tag{4.43}$$

La matriz de posición puede llegar a violar las restricciones (1.7) y (1.6) luego de algunas iteraciones, de modo que es necesario normalizar la matriz de posiciones. Primero se convierten todos los elementos negativos de la matriz en cero. Si todos los elementos en una columna de la matriz son cero, es necesario re-evaluarlos usando una seria de números aleatorios en el intervalo [0,1]. Luego se realiza la siguiente transformación en la matriz de modo que no se violen las restricciones:

$$X_{normalizado} = \begin{bmatrix} x_{11}\Big/\sum_{i=1}^{m} x_{i1} & x_{12}\Big/\sum_{i=1}^{m} x_{i2} & \cdots & x_{1n}\Big/\sum_{i=1}^{m} x_{in} \\ x_{21}\Big/\sum_{i=1}^{m} x_{i1} & x_{22}\Big/\sum_{i=1}^{m} x_{i2} & \cdots & x_{2n}\Big/\sum_{i=1}^{m} x_{in} \\ \vdots & \vdots & \ddots & \vdots \\ x_{m1}\Big/\sum_{i=1}^{m} x_{i1} & x_{m2}\Big/\sum_{i=1}^{m} x_{i2} & \cdots & x_{mn}\Big/\sum_{i=1}^{m} x_{in} \end{bmatrix} \quad (4.44)$$

Dado que la matriz de posición indica una solución de asignación potencial, debiéramos decodificar la matriz difusa y obtener la solución factible. Un arreglo de *flags* puede ser utilizado para almacenar la solución. En un principio, ninguna de las columnas se encuentra seleccionada, luego para cada columna de la matriz se escoge el elemento que tiene el valor máximo, luego se marca la columna de este elemento como seleccionado, y finalmente el número de la columna se guarda en el arreglo de solución. Una vez que todas las columnas han sudo procesadas, obtenemos la solución factible desde el arreglo de solución y la medida del fitness de las partículas.

4.4.5 Dinámica, convergencia y selección de parámetros en PSO

PSO es un algoritmo de optimización basado en poblaciones que ha demostrado desempeñarse muy bien en una serie de aplicaciones. Muchos trabajos han sido realizados acerca de la selección de sus parámetros óptimos, y de su modificación para beneficiar un tipo de comportamiento u otro (por ejemplo beneficiar exploración o explotación). Sin embargo, el análisis de su dinámica aún se encuentra en una etapa muy incipiente. Esto es debido a que su análisis, a pesar de la simplicidad del algoritmo, presenta diversos desafíos. Primero, su composición por parte de un gran número de elementos (partículas) que interactúan en su camino al óptimo. Mientras la naturaleza de la interacción de sus operadores es bastante simple, comprender su dinámica como un sistema es una tarea no trivial. Segundo, las partículas vienen equipadas con memoria, que produce que una partícula puede ser atraída por un nuevo *gbest* o *pbest*, o incluso ambos, en una iteración determinada, lo que cambia la dirección de movimiento de las partículas. Tercero, la interacción es estocástica, lo que incide que las herramientas convencionales no sean aplicables al problema más allá de comprender de un modo aproximado como pueden llegar a comportarse las partículas.

Se han realizado diversos esfuerzos para entender la dinámica del algoritmo, unos más completos que otros, pero la mayoría considera simplificaciones demasiado fuertes como para poder caracterizar de un modo adecuado el comportamiento del sistema. En general estas simplificaciones incluyen el análisis de un solo individuo, con una sola dimensión, determinismo en la evolución del sistema, y estancamiento en la búsqueda (no se encuentran mejores soluciones que actualicen *gbest* o *pbest*). A pesar de esto, existen algunos esfuerzos que consideran modelos mucho más realistas del algoritmo. A continuación se presenta una revisión de los principales estudios sobre selección de parámetros y dinámica del sistema en PSO.

Shi y Eberhart (1998) propone por primera vez el uso de un coeficiente de inercia en PSO. De acuerdo a los primeros estudios, (Ozcan y Mohan, 1999), este coeficiente no ayudaba mucho al algoritmo, y argumenta que por eso los análisis experimentales muestran que un valor de $\omega = [0.9, 1.2]$ dan los mejores resultados (así se parece a la versión preliminar sin coeficiente de inercia). Sin embargo estudios más recientes muestran que el rol del coeficiente de inercia ω es considerado crítico para el comportamiento de convergencia de PSO (Nedja y Macedo Mourelle,

2006). Este coeficiente es empleado para controlar el impacto de de las previas velocidades en la actual. De este modo, el parámetro ω regula el compromiso entre las habilidades de búsqueda global y de búsqueda local del enjambre. Un factor de inercia grande beneficia una exploración global (buscando en nuevas áreas), mientras que uno pequeño tiende a facilitar la exploración local. Un valor adecuado de ω entrega un balance entre las habilidades de exploración global y local, y consecuentemente se obtiene una reducción del número de iteraciones para encontrar la solución global. Inicialmente se utiliza el factor de inercia constante. Sin embargo, algunos resultados indican que es mejor iniciar el proceso con un valor alto de este parámetro, para promover la exploración global, y gradualmente disminuir su valor para obtener soluciones más refinadas. Así, Shi y Eberhart (1999) proponen el uso de un coeficiente de inercia variable, que parte en 0.9 al comienzo de las iteraciones, y que termine en 0.4. De este modo se favorece una búsqueda global al comienzo de las iteraciones, y al final se privilegia la explotación de las soluciones encontradas hasta el momento. Otros trabajos más recientes proponen que de este modo un valor inicial de 1.2 y su reducción gradual hasta 0 es considerada una buena elección para ω (Eberhart y Shi, 2000). Otros enfoques adaptativos (tales como un controlador difuso) han probado ser efectivos, donde los parámetros pueden ser sintonizados de manera adaptable de acuerdo al problema en cuestión (Liu y Abraham, 2005).

Los coeficientes cognitivos y sociales, c_1 y c_2 respectivamente, no son críticos para la convergencia de PSO (Nadja y Macedo-Mourelle, 2006). Sin embargo una buena sintonización fina puede resultar en una convergencia más rápida y evasión de mínimos locales. Usualmente $c_1 = c_2 = 2$ son usados como parámetros por defecto (Kennedy y Eberhart, 2001). Sin embargo, algunos experimentos muestran que con $c_1 = c_2 = 1.49$ se puede obtener mejores resultados (Clerc, 1999). Otros trabajos muestran que puede ser mejor escoger un coeficiente cognitivo c_1 más grande que el coeficiente social c_2, pero siempre manteniendo $c_1 + c_2 \leq 4$ (Clerc and Kennedy, 2002).

Los conjuntos de parámetros más aceptados en la actualidad son: $\omega = 0.729, c_1 = 2.8\omega, c_2 = 1.3\omega$ (Carlisle y Dozier, 2001); $\omega = 0.729, c_1 = c_2 = 1.494$ (Clerc, 1999); $\omega = 0.6, c_1 = c_2 = 1.7$ (Trelea, 2003); y $\omega = 1, c_1 = c_2 = 2$ (Eberhart y Kennedy, 1995). Dentro de éstos, probablemente el set más utilizados es el propuesto por Clerc.

A continuación se realiza una revisión de los trabajos que han estudiado la dinámica y propiedades de convergencia de PSO. Ozcan y Mohan (1998), realizan los primeros estudios acerca de la dinámica de las posiciones y velocidades de las partículas. Las simplificaciones incluyen el análisis de una sola partícula, unidimensional, con ausencia de estocasticidad, en estancamiento de las soluciones. Como se trabaja con una sola partícula, *pbest* y *gbest* son la misma. En este trabajo se muestra que la partícula se comporta siguiendo una trayectoria sinusoidal, variando aleatoriamente su amplitud y frecuencia. En Ozcan y Mohan (1999), se generaliza el análisis anterior a un sistema con muchas partículas. Se muestra que cuando una partícula se encuentra buscando el óptimo en general cambia su amplitud y frecuencia al encontrarse con otras partículas, y que estos cambios ayudan a la búsqueda. Se muestra también que limitar la velocidad a $v_{\max}$ ayuda a que las partículas atrapen una nueva amplitud y frecuencia. Clerc y Kennedy (2002) realizan un análisis similar, con una partícula, unidimensional, determinístico, y durante estancamiento. Se muestra que bajo estas condiciones este es un sistema dinámico lineal de tiempo discreto. Así, la dinámica del estado de las

partículas (posición y velocidad) puede ser determinada encontrando los valores y vectores propios del sistema lineal. Se muestra que la partícula convergerá al equilibrio si la magnitud de los valores propios son menores o igual a 1. En este trabajo además se compara el desempeño de PSO con el de otros algoritmos evolutivos, y se obtienen resultados comparables en la cantidad de iteraciones requeridas. Sin embargo, el número de partículas utilizadas es ampliamente menor que el número de individuos en algoritmos evolutivos, por lo tanto se muestra que se pueden alcanzar soluciones similares evaluando un número mucho menor de funciones objetivo, resultando en un tiempo computacional mucho menor. Van den Bergh (2001) también realiza un análisis similar, con una partícula, sin estocasticidad y en estancamiento. Se encuentra una formula explícita para la trayectoria de una partícula, y se muestra que esta es atraída hacia un punto fijo. Además, se presenta el conjunto de parámetros que garantiza convergencia. Sin embargo, se dice que este punto fijo podría no ser un óptimo global, ni siquiera local, lo que indicaría que el algoritmo no es un optimizador garantizado. Se dice también que el análisis es válido también en presencia de estocasticidad. Yasuda *et al*. (2003) estudian el mismo sistema simplificado de una partícula unidimensional, en estancamiento y en ausencia de estocasticidad. Se incluye el estudio del factor de inercia. Nuevamente se realiza un análisis basado en valores y vectores propios para determinar la combinación de parámetros para los cuales el sistema es estable, y que tipos de comportamientos puede tener una partícula. Se analiza condiciones para un comportamiento cíclico en detalle. Blackwell (2005), investiga como cambia la distribución espacial en el tiempo. Se adopta un modelo simplificado, pero aún así es una extensión del modelo de Clerc y Kennedy (2002), pues se permiten muchas partículas multidimensionales. Estas pueden interactuar en el sentido que pueden cambiar su *lbest*. Se incluyen la constricción pero no estocasticidad. Se encuentra que la distribución espacial se reduce exponencialmente en el tiempo, lo que indica que el enjambre tiende a converger. Trelea (2003) bajo todas las simplificaciones típicas realiza un análisis de una familia de modelos de partículas de cuatro parámetros donde identifica las regiones en las cuales el algoritmo exhibe diversos comportamientos (estabilidad, oscilaciones armónicas o comportamiento de zigzag). Como resultados de este análisis, se propone el siguiente conjunto de parámetros $\omega = 0.6, c_1 = c_2 = 1.7$.

Los trabajos presentados hasta el momento se basan en modelos estáticos simples, despreciando la clara componente estocástica de este tipo de algoritmos. Sólo unos pocos trabajos han enfrentado la tarea de incluir la presencia de estocasticidad en el estudio de estabilidad de las partículas. Kadirkamanathan *et al.* (2006) incluyen el factor estocástico mediante un análisis de estabilidad basado en la teoría de Lyapunov y de sistema pasivos. Se representa las partículas como un sistema realimentado no lineal, lo que permite aplicar conocimientos de la teoría de control. Se encuentran condiciones suficientes para los parámetros de PSO para garantizar convergencia, y se indica que es conveniente incrementar el valor de los parámetros social y cognitivo cuando se reduce el coeficiente de inercia. Sin embargo, como la teoría de Lyapunov es bastante conservadora, las condiciones encontradas son muy restrictivas, y obligan a que PSO tenga un comportamiento ligeramente oscilatorio.

Jiang *et al.* (2007) también realizan un análisis incluyendo componentes estocásticos, para el sistema de PSO incluyendo muchas partículas multi-dimensionales, pero considerando estancamiento. Se prueba que el sistema es convergente y estable, y se encuentran una zona factible de los parámetros coeficiente de inercia, factor social y cognitivo, que garantizan esta condición. Se comprueba que esta zona coincide con combinaciones recomendadas de parámetros presentadas en otros estudios. Sin embargo, no se puede concluir acerca de la optimalidad del punto de convergencia, ni global ni local. Se analiza que para que una partícula

converja en media cuadrática, su esperanza debe converger, y además, su varianza debe converger a cero. De acuerdo al análisis realizado, se llega a determinar las condiciones sobre los parámetros para que el algoritmo 1 PSO converja en media cuadrática a *pbest*. Si bien el área definida por estas condiciones es pequeña, gran parte de los conjuntos de parámetros utilizados en la literatura caen en esta área, tales como los propuestos por Carlisle y Dozier (2001), Clero (1999), y Trelea (2003). Este trabajo es un gran aporte, pues permite dar una explicación teórica de porqué estos parámetros encontrados empíricamente funcionan. El único que no cae en esta área es el propuesto inicialmente: $\omega = 1, c_1 = c_2 = 2$ (Eberhart y Kennedy, 1995).

Poli y Broomhead (2007) realizan un estudio similar al anterior con las mismas suposiciones, también para un modelo que utiliza coeficiente de inercia. Se hace explícito que para que una partícula converja no basta con que $E(x_t) \to p$, donde $p = \left(c_1 \cdot pbest + c_2 \cdot gbest\right) / \left(c_1 + c_2\right)$, sino que además se requiere que $E((x_t - E(x_t))^2) \to 0$, y esto sucede si y solo sí $pbest = gbest$. Van den Bergh (2002) utiliza el supuesto que $E(x_t) \to p$ implica que $E((x_t - E(x_t))^2) \to 0$ para probar que PSO no tiene garantía de ser un optimizador. Como se prueba que esto no es cierto, la prueba de Van den Bergh no es válida, y así la pregunta de si PSO es un optimizador o no sigue abierta.

Es claro que todos estos trabajos, incluso en los más completos, se asume que el sistema esta en estancamiento. Es claro entonces que hay investigación pendiente en determinar bajo que condiciones en sistema se estanca, bajo cuales puede salir de esta condición, y que pasa con PSO al cambiar a seguir un nuevo óptimo personal y o global. Además, bajo los supuestos más adecuados (una población que se comporta de forma estocástica), no se ha podido probar que el sistema converja a un óptimo, pero sí se ha podido probar que converge. Debido a esto falta evidencia teórica, a pesar que existe la empírica, para justificar plenamente su utilización. No obstante, su facilidad de implementación, lo intuitivo de su heurística y lo exitoso de sus aplicaciones convierten a PSO en una llamativa alternativa para resolver problemas de optimización no lineales. Respecto de los parámetros a utilizar, gran parte de los trabajos que se han realizado para establecer recomendaciones se enfocan en el factor de inercia, social y cognitivo, sin concentrarse de mayor manera en los tamaños de población y número de iteraciones. Por lo tanto, también se puede realizar avances en ese tema.

4.5 Algoritmos evolutivos para programación entera mixta

El control predictivo híbrido no lineal, como ya se ha presentado en la Sección 3.6, se formula como un problema de programación entera mixta no lineal. En la literatura se encuentran reportadas algunas aplicaciones basadas en algoritmos evolutivos para resolver este tipo de problemas. Se realiza entonces una revisión de éstas para conocer como ha sido enfrentado este problema hasta el momento, dividido en las aplicaciones basadas en algoritmos evolutivos clásicos, evolución diferencial y optimización por enjambre de partículas.

4.5.1 Algoritmos evolutivos clásicos para programación entera mixta

En esta sección se presentan los trabajos que han sido realizados para implementar algoritmos evolutivos para resolver problemas de programación entera mixta.

Back y Schutz (1995) diseñan una estrategia evolutiva para la solución de un problema de diseño de sistemas ópticos multicapas, que se formula como un MINLP. Este problema de diseño se plantea sin restricciones; por lo tanto el diseño de la estrategia evolutiva generalizada (como se llamada por los autores) se enfoca en el manejo de las variables continuas y discretas y los operadores de mutación y recombinación, de modo que no se produzcan incompatibilidades y preservar las distintas estructuras de datos. Las soluciones candidatas se codifican explícitamente (con sus valores continuos o discretos según corresponda). Los operadores de recombinación son aplicados para el individuo completo, sin diferenciar las variables continuas de las discretas, sea se utiliza crossover uniforme o bien crossover aritmético. Sólo se toma en cuenta que en el caso del crossover aritmético se trunca inmediatamente al valor correspondiente en las variables discretas. La situación no es la misma respecto de la mutación, pues en este operador si hay diferencias en el tipo de variables. Para las variables continuas se utiliza mutación Gaussiana controlada por la desviación estándar (que también evoluciona como parte del individuo). Sin embargo no es posible utilizar este operador para el caso de variables discretas pues si la desviación estándar cae por debajo de 1, el algoritmo se estancará en los valores presentes en las variables enteras (además que muchas distintas realizaciones pueden arrojar mutaciones iguales). Por lo tanto, la desviación estándar en este caso sirve para determinar si se realiza mutación o no en la variable determinada. Si se realiza mutación, entonces se selecciona alguno de los estados discretos de forma aleatoria con probabilidad uniforme (pero sin considerar el estado actual). Los resultados que se obtienen son satisfactorios frente a diseños anteriores y otras estrategias basadas en GA convencionales.

Cao y Wu (1997) plantean una estrategia para resolver problemas MINLP utilizando programación evolutiva. La estrategia es muy similar al algoritmo canónico, variando en la codificación, las mutaciones y en el manejo de restricciones. Respecto de la codificación, sólo se aceptan soluciones candidatas en el mismo espacio de búsqueda del problema de optimización, es decir las variables continuas se manejan como números reales, las variables enteras se manejan como números enteros, etc. El operador de mutación sigue exactamente esta línea, y debe ser distinto para cada tipo de variables. Para las variables continuas se realiza una mutación Gaussiana con desviación estándar que depende del fitness relativo del individuo. Para las variables binarias se realiza mutación bit flip si un número aleatorio uniforme entre $[0,1]$ es mayor que un cierto umbral. Para las variables enteras se cambia de un entero a otro con un paso aleatorio. Finalmente para las variables discretas se cambia de un valor a otro con una probabilidad uniforme. El manejo de restricciones también debe preservar la categoría de cada tipo de variables. Si bien es cierto que una de las estrategias predilectas para considerar las restricciones es incluirlas como penalizaciones en la función objetivo, ésta tiene los problemas de no comportarse bien cuando las soluciones óptimas se encuentran en la frontera de la región factible y de requerir muchas pruebas para asignar pesos adecuados en las penalizaciones. Por esto, se adopta un esquema conocido como método de restricciones variables. Éste consiste en que si al aplicar el operador de mutación se obtiene una solución infactible, este nuevo individuo se descarta y se aplica el operador nuevamente hasta obtener una solución factible. El método es comparado con las estrategias de Branch and Bound entero mixto con programación secuencial cuadrática (*Mixed integer Branch and Bound using Sequential Quadratic Programming*, MIBBSQP, Sandgren, 1990), programación no lineal entera-discreta-continua (*Integer-discrete-continuous Non-Linear programing*, IDCNLPC, Fu, 1991) y *simulated annealing* (Zhang y Wang, 1993), para un conjunto de problemas de diseño en ingeniería mecánica (un tren de cambios y un estanque de presión) propuestos por Sandgren (1990). Los resultados muestran un mejor desempeño que los otros métodos tanto en calidad de la solución y convergencia.

Una línea con bastantes trabajos para resolver los problemas MINLP se basa en la combinación de *branch and bound* con algoritmos genéticos. En la mayoría de estos trabajos el árbol de búsqueda se arma con las distintas combinaciones de las variables discretas, y las cotas son encontradas resolviendo el problema continuo no lineal con algoritmos genéticos (Cotta *et al*., 1995; Nagar *et al*., 1996), lo que permite ir podando el árbol hasta encontrar el óptimo.

Costa y Oliveira (1999) proponen una estrategia basada en GA para resolver MINLPs, basada en una codificación numérica explícita para las variables continuas y discretas. Este enfoque se diferencia de otros en el modo de manejar las restricciones. Mientras que la mayoría de los métodos estudiados se basan en el uso de funciones de penalización suaves, éstos tienen el problema de que el desempeño depende fuertemente de la definición de pesos para cada una de las restricciones incluidas como penalización en la función objetivo, pues si éstos no están bien definidos puede darse la situación que soluciones infactibles posean mejores fitness que soluciones factibles, lo que claramente guiaría de un modo insatisfactorio la búsqueda de soluciones. Entonces, en esta estrategia se propone un enfoque de penalización sin parámetros apoyada en la selección por torneo, donde en la competición se define que una solución factible siempre derrotará a una infactible, y que la competición entre dos soluciones infactibles vencerá aquella que viole en un menor grado las restricciones (Deb, 1998). Los experimento muestran que esta estrategia posee un mejor desempeño que otras por ejemplo la basada en simulated annealing, propuesta por Cardoso *et al.* (1997).

Dahal *et al.* (2003) estudian cinco métodos híbridos basados en GA para un problema de despacho económico de potencia. Dos de estos métodos son absolutamente genéricos para cualquier tipo de MINLP, mientras que los otros tres utilizan la estructura particular del problema. Estos últimos codifican la parte discreta del problema con GA y luego resuelven la parte continua con heurísticas basadas en el conocimiento del problema en particular. Respecto de los métodos genéricos es, el primer es llamado GA explícito. En este cada individuo codifica una solución que incluye las variables discretas y continuas del problema de optimización. El otro método, llamado GA integrado resuelve el problema en dos etapas. La primera etapa, basado en GA, codifica las soluciones solamente para las variables discretas. Luego, la evaluación de estas funciones depende de resolver el problema resultante con las variables discretas fijas para las variables continuas. Como el sistema estudiado es lineal, este problema es resuelto mediante programación lineal. En ambas estrategias las restricciones son manejadas incorporando penalizaciones en la función objetivo que son crecientes con el grado de no cumplimiento de las restricciones (penalización suave). De estas estrategias, GA integrado es más rápido y siempre encuentra el óptimo. Esto muestra la robustez de GA para trabajar con variables discretas.

Sirikum y Techanitisawad (2006) proponen una estrategia basada en GA muy similar a GA integrado (Dahal *et al.* 2003), para un problema de planificación de expansión de generación de energía. En esta estrategia GA es utilizado para encontrar las combinaciones óptimas de las variables discretas, y para cada una de estas soluciones se debe resolver un problema LP para encontrar los *fitness* de las soluciones candidatas discretas (lo que completa la formulación entera-mixta). La codificación de las soluciones se realiza de modo explícito, donde las variables son incluidas en el cromosoma como los valores binarios o enteros (según corresponda) que se deben encontrar en el problema de optimización. El manejo de restricciones se realiza a través de penalizaciones suaves en la función objetivo. La metodología es comparada con una metodología convencional (LINGO) que debido a la no linealidad del sistema sólo puede garantizar encontrar un óptimo local. Los resultados muestran que para problemas pequeños GA es casi tan bueno

como LINGO en términos de la calidad de la solución, pero computacionalmente es más rápido, y que a medida que aumenta el tamaño del problema la diferencia de la carga computacional se hace exponencialmente favorable para GA, y este pasa a encontrar consistentemente mejores soluciones que LINGO.

4.5.2 Evolución diferencial para programación entera mixta

Evolución diferencial (DE) originalmente es método pensado para problemas de optimización continua. Sin embargo, su excelente desempeño ha llevado a que se diseñen variantes para resolver otro tipo de problemas. Una de las variantes es la resolución de problemas de optimización entera mixta no lineal (MINLP), que corresponde precisamente la formulación genérica para los problemas de control predictivo híbrido no lineal, descrita en 3.6.

Lampinen y Zelinka (1999a) proponen una estrategia para resolver problemas MINLP basada en DE, donde se utiliza la variante DE/rand/1/bin. Se describe el tratamiento necesario para el manejo de las variables discretas y continuas, para lidiar con las restricciones de borde, y con restricciones no lineales.

Para el manejo de restricciones de frontera de cada una de las variables, si luego de los operadores evolutivos una variable cae fuera del rango permitido, se le asigna simplemente el valor de la frontera violada. En el caso de las restricciones más generales, se utiliza un enfoque de penalización suave, que considera un aumento en la función objetivo dado por la distancia a la frontera de la restricción, pero que no se usa para descartarlas como con restricciones duras. Así el espacio de búsqueda permanece continuo, y es posible llegar al óptimo incluso a luego de atravesar de regiones infactibles, pues es capaz de discriminar soluciones más y menos factibles. Además, al aplicar el enfoque basado en penalización suave, no es complicado encontrar soluciones factibles (en lo que se refiere al proceso evolutivo) para que evoluciones hacia el óptimo. Para el manejo de las variables discretas la adaptación es sencilla. Para evaluar la función objetivo simplemente se realiza una truncación de la coordenada correspondiente al individuo cuya función objetivo se está evaluando. Pero esta truncación se realiza sólo a nivel de la evaluación, pues en el proceso evolutivo se utiliza el valor continuo para así mantener la diversidad en la búsqueda. Por lo mismo, para el manejo de las restricciones, se verifica que el valor truncado las cumpla. Este enfoque es luego aplicado a un problema de diseño de un resorte en espiral (Lampinen y Zelinka, 1999b). El rendimiento es comparado con otros métodos ya reportados en la literatura, dentro de los que se cuenta un Branch and Bound con programación secuencial cuadrática (Sandgren, 1990), un algoritmo genético (Chen y Tsao, 1993) y un meta-GA (Wu y Chow, 1995). Se muestra que DE encuentra siempre mejores soluciones que las ya reportadas. Además, se realiza una modificación en los parámetros para privilegiar la velocidad de convergencia por sobre la robustez[6], y se encuentra que mientras la velocidad de convergencia se hace un 325% mayor, sólo un 5% de las soluciones es peor que las ya encontradas, lo que muestra que este enfoque puede ganar mucho en convergencia sacrificando muy poco en robustez.

Lin *et al*. (1999) también proponen un método para resolver problemas MINLP basados en DE. Al igual que Lampinen y Zelinka (1999) incorporan las restricciones como penalizaciones suaves en la función objetivo. La diferencia está en que para el manejo de las variables discretas, en la

[6] En este contexto se entiende por robustez la capacidad que posee el algoritmo para llegar al óptimo global en diversas realizaciones del algoritmo de optimización y no quedarse estancado en óptimos locales.

codificación éstas siempre son manejadas como variables discretas. Así, no realizan truncación para evaluar la función objetivo. Para lograr esto, la truncación se encuentra en los mismos operadores evolutivos, y de este modo la descendencia siempre queda en el dominio que le corresponde. Una característica especial del algoritmo desarrollado por Lin *et al*. (1999), es que se encuentra basado en una variante de evolución diferencial (*hybrid differential evolution,* HDE) desarrollada por Chiou y Wang (1998). Esta es una modificación a DE que incluye operadores de migración y aceleración. El primero sirve para evitar la convergencia prematura (sin tener que aumentar el tamaño de la población y por ende aumentar el tiempo computacional) y consiste en generar una nueva población (salvo el menor individuo) a partir de este sólo si la diversidad de la población ha disminuido drásticamente, y el segundo es utilizado para aumentar la explotación de las soluciones encontradas y así acelerar la convergencia, y consiste en aplicar operadores del método del gradiente si el *fitness* de la mejor solución no disminuye de una iteración a otra[7]. Esta metodología es utilizada exitosamente por Su y Lee (2003) para la re-configuración de redes de distribución.

En los métodos anteriores, la estrategia para la resolución de los MINLP aparece en la codificación/decodificación de las variables, al tratarlas siempre como enteras (codificación) o bien al truncar los valores al momento de evaluar la función objetivo (decodificación). Munawar y Gudi (2005) utilizan una estrategia distinta, al utilizar codificaciones siempre continuas, tanto para las variables continuas como discretas, pero los valores discretos se manejan a través de restricciones. Por ejemplo, una variable binaria $y \in \{0,1\}$ puede modelarse simplemente como una variable continua $x \in [0,1]$, pero añadiendo la restricción de igualdad:

$$x(x-1) = 0, \quad 0 \le x \le 1 \tag{4.45}$$

De este modo, el problema de optimización resultante evade el manejo explícito de variables discretas y su consiguiente complejidad computacional, y resulta simplemente en la formulación de un NLP. Se debe remarcar de todos modos que esta restricción es no convexa, de modo que el problema de optimización es no convexo, aparte de cualquier no convexidad propia. Por esta razón, la aplicabilidad de métodos clásicos como programación secuencial cuadrática o gradiente reducido generalizado es bastante reducida por la alta sensibilidad de la condición inicial y la alta probabilidad de quedar atrapados en óptimos locales, y entonces se utiliza un enfoque evolutivo basado en DE que posee buenas cualidades de búsqueda global. El algoritmo canónico de DE no es capaz de manejar restricciones de igualdad, pero generalmente son manejadas como restricciones suaves en la forma de penalizaciones en la función objetivo. En este trabajo se comparan tres opciones para resolver los MINLP. La primera, llamada *integer-DE* es el enfoque basado en codificación continua y posterior truncación de las variables discretas. La segunda opción, llamada *NLP-DE*, es la basada en la restricción de igualdad de la ecuación (4.45) y su manejo como penalización en la función objetivo. La última opción, *hybrid-DE* ejecuta el algoritmo DE (alguna de las opciones anteriores) hasta que la diversidad de la población o la función objetivo haya alcanzado una disminución considerable de una iteración a otra (punto rodilla), entonces se toman las mejores soluciones y para ellas se toman las variables discretas como fijas, y las variables continuas como punto de partida para un problema NLP, el cual es resuelto por un método convencional como SQP. Se consideran varios puntos iniciales para aumentar la robustez, aprovechando que como se parte en un punto silla ya las soluciones están cerca de los óptimos locales y es altamente probable que SQP demore muy pocas iteraciones (o

[7] Puede encontrar más detalle acerca de estos operadores en el anexo.

tal vez sólo una) para encontrar los óptimos locales. Finalmente, el mejor de los óptimos locales será la solución final.

De las comparaciones se concluye que el método híbrido supera a los otros métodos en tiempo computacional y en la velocidad de convergencia. Sin embargo no es posible concluir que opción entre integer-DE o hybrid-DE es mejor antes de pasar a la etapa NLP, pues se comprueba que *integer-DE* es más rápido en las primeras generaciones pues *NLP-DE* la satisfacción de las restricciones de igualdad de la condición discreta endentece la convergencia, mientras que integer-DE es más lento cerca del óptimo, probablemente debido a la desconexión entre el tratamiento continuo de las variables discreta y la truncación al momento de evaluar la función objetivo.

Siguiendo la misma línea comparativa, Angira y Babu (2006) estudian el rendimiento de DE aplicado a MINLP utilizando el enfoque de truncación para variables discretas, mientras que para variables binarias estudian el método basado en la restricción de igualdad de la ecuación (4.45) y lo comparan con otro basado en codificación continua, muy similar al de truncación. Así, el valor decodificado de la variable binaria y_i utilizando la variable continua x_i es:

$$y_i = \begin{cases} 0 & \text{si } x_i \leq 0.5 \\ 1 & \text{si no} \end{cases} \tag{4.46}$$

Se encuentra que la segunda alternativa es mejor, probablemente a que la inclusión de restricciones y no convexidades en la primera alternativa es dañina para el desempeño del algoritmo.

En este trabajo se utiliza un enfoque basado en penalización de la función objetivo para las restricciones. Se estudia además las estrategias para el manejo de cotas inferiores/superiores durante la evolución de soluciones candidatas. Para esto se prueban dos alternativas; la primera, si alguna variable se sale de su rango, simplemente se satura al rango máximo (o mínimo); y la segunda, se genera nuevamente la evolución de la solución candidata (mediante los operadores de mutación, crossover y selección correspondientes). Se encuentra que el segundo método es más robusto, pero a la vez, más lento. En este sentido, si el óptimo global se encuentra en la cota, la alternativa que asigna el valor de la cota es claramente mejor, pero si un óptimo local es el que se encuentra ahí, el método posee una tendencia a quedarse estancado en ese óptimo local.

4.5.3 PSO para programación entera mixta

En general PSO ha demostrado ser una herramienta altamente exitosa para resolver problemas de optimización en espacios continuos, sin restricciones o con restricciones de desigualdad, pero su modo de operación no es adaptable de un modo directo al manejo de restricciones de igualdad. Además, PSO no ha sido aplicado en gran cantidad para resolver problemas de optimización entera mixta (MINLP). Sin embargo, existen algunos esfuerzos para aplicar PSO en este tipo de problema, cuya presentación es el objetivo de esta sección.

Yiqing *et al*. (2007) desarrollan un algoritmo de PSO mejorado para resolver problemas MINLP con restricciones desigualdad e igualdad. La formulación utilizada en este trabajo es la siguiente:

$$\begin{aligned} &\min f(x,y) \\ &s.a.\ \ g(x,y) \le 0 \\ &\quad\ \ h(x,y) = 0 \\ &x \in X \subset \mathbb{R}^n,\ y \in Y \subset \mathbb{N}^m \end{aligned} \tag{4.47}$$

donde x corresponde a las variables continuas e y corresponde a las variables discretas. $g(x,y)$ y $h(x,y)$ son los vectores que definen las restricciones de igualdad y desigualdad, con dimensiones p y q respectivamente. A partir de las restricciones de igualdad, es posible separar las variables continuas en un conjunto de variables independientes y otras dependientes. Así, $x = [\upsilon, \xi]^T$, $y = [\omega, \psi]$ donde υ, ω son las variables independientes y ω, ψ con las variables independientes. $\upsilon \in \Xi \subset \mathbb{R}^v$, $\omega \in \Omega \subset \mathbb{N}^u$, y entonces $v+u$ es el grado de libertad del sistema, y se debe cumplir que $v+u = n+m-q$. Como ξ y ψ son variables dependientes, están dadas por $\xi = \xi(\upsilon,\omega), \psi = \psi(\upsilon,\omega)$, que se pueden determinar a través de $h(x,y)$. Así, la función objetivo y las restricciones de igualdad se pueden re-escribir como $f(x,y) = F(\upsilon,\omega), g(x,y) = G(\upsilon,\omega)$. Y entonces el problema de optimización se puede re-escribir como:

$$\begin{aligned} &\min F(\upsilon,\omega) \\ &s.a.\ \ G(\upsilon,\omega) \le 0 \\ &\upsilon \in \Xi \cap V, \psi \in \Omega \cap V \end{aligned} \tag{4.48}$$

donde $V = \left\{ (\upsilon,\omega) : h(\upsilon,\xi,\omega,\psi) = 0, \text{ para algún } \xi \in \mathbb{R}^{n-v} \cap X \text{ y } \psi \in \mathbb{R}^{m-u} \cap Y \right\}$.

Este problema en un MINLP con restricciones sólo de desigualdad. Entonces plantean el uso de funciones de penalización para resolver el MINLP, pues este tipo de manejo es útil en restricciones de desigualdad, no así en restricciones de igualdad. Los autores llaman a esto el método de transformación de espacio reducido.

Esta estrategia resulta interesante pues en los problemas de optimización del control predictivo híbrido, las restricciones de igualdad vienen a ser las predicciones del modelo. Así, las salidas en los instantes futuros son las variables dependientes, y es posible resolver el problema tan sólo en términos de las acciones de control, que son las variables independientes.

La estrategia de Yiqing *et al*. (2007), sin embargo, considera el caso más general donde las variables dependientes no pueden ser obtenidas directamente. En este caso, el algoritmo de PSO considera que las partículas son para encontrar las variables independientes. Así, cada vez que se tiene una solución candidata, se ejecuta PSO continuo para encontrar las variables dependientes, donde la función objetivo es la suma de los cuadrados de las componentes de la restricción de igualdad $f^*(x) = \sum_{i=1}^{q} h^2(\upsilon,\xi,\omega,\psi)$. Encontradas las variables dependientes, se puede ejecutar el algoritmo de PSO para el problema verdadero, incluyendo la restricción de desigualdad como una penalización en la función objetivo.

Se plantea que la forma más obvia para enfrentar los MINLP con PSO es actualizando las variables de decisión discretas junto con las continuas. En general, esta estrategia no es muy

efectiva, pues el algoritmo no puede evolucionar adecuadamente en los espacios continuos, ya que la evaluación en las variables discretas modifica el espacio continuo de búsqueda. Entonces, se propone una estrategia basada en una modificación *simulated annealing* realizada por Cardoso *et al*. (1997), que consiste en que las variables discretas se actualizan en la iteración k sólo si un número aleatorio $r_k > \alpha$, $0 < \alpha < 1$ donde α es la probabilidad de que las dimensiones discretas no sean actualizadas. De este modo, las variables continuas sí tienen tiempo de evolucionar en el espacio continuo definido por las variables discretas.

En este trabajo los resultados son comparados con otros enfoques basados en GA y SIMPSA (Costa y Olivera, 2001) para programación entera mixta. Se muestra que en general PSO llega más veces al óptimo global y con un menor número de evaluaciones de funciones objetivo. Entonces se concluye que PSO modificado es una excelente alternativa para resolver problemas de programación entera mixta, aunque se espera continuar con la línea de investigación, pues aún tiene problemas para converger a la solución óptima en problemas de gran escala.

Otros desarrollos de PSO para problemas de optimización entera mixta no lineal se muestran a continuación. Yoshida *et al.* (2000) desarrolla un esquema para un problema de control de potencia reactiva. El problema se resuelve en base a variables de control continuas como los niveles de operación de los reguladores automáticos de voltaje (AVR), y variables discretas como las posiciones de taps de cargadores de transformadores y el número de equipos de compensación de potencia reactiva. El manejo de las variables continuas y enteras se realiza de un modo completamente paralelo, donde cada partícula de PSO contiene en sus distintas dimensiones tanto variables continuas como discretas, todas se actualizan al mismo tiempo. Las variables discretas se manejan directamente por truncación. Además las restricciones son incorporadas directamente en la función objetivo como una penalización cuando no son respetadas. Fukuyama (2005) también utiliza PSO para resolver el problema de planificación en el manejo de potencia reactiva. En este trabajo se comparan PSO con factor de inercia, con factor de constricción, y además estas dos opciones con una modificación que incluye el operador de selección utilizado en GA, llamado HPSO (*Hybrid* PSO). El manejo de las variables discretas se realiza de forma interna, es decir, no se trabaja con variables continuas para luego truncarlas al momento de evaluar la función objetivo, sino que la truncación se realiza incluida en los operadores de PSO. Los resultados muestran que HPSO funciona mejor que las otras estrategias para el problema en particular, encontrando resultados bastante satisfactorios en términos de calidad de solución y robustez. Guo *et al*. (2004) utiliza PSO para problemas MINLP *benchmark* de diseño de sistemas mecánicos: diseño de caja de cambios de un tren, diseño de un estanque de presión y el diseño de un resorte. El manejo de las variables discretas es del mismo modo que en el trabajo anterior, y el manejo de restricciones se realiza en base al método de restricción variable. Este consiste en que si una solución generada es infactible, entonces se descarta y los operadores estocásticos son aplicados nuevamente hasta que resulte una solución factible. El enfoque posee la debilidad de que bien podrían generarse varias veces soluciones infactibles, lo que claramente aumenta la complejidad del algoritmo. Los resultados son comparados con aquellos reportados previamente por otros investigadores, y se muestra que en el primer caso encuentra la mejor solución obtenida previamente, y en los otros supera las soluciones previas. Sin embargo, no se muestran resultados acerca del costo computacional requerido.

4.6 Conclusiones

En esta sección se ha presentado los fundamentos de los algoritmos evolutivos clásicos, evolución diferencial y optimización por enjambre de partículas. Además, se han presentado las propiedades que los hacen recomendables en ciertas circunstancias para su aplicación en control predictivo híbrido, y se ha puesto especial énfasis en sus propiedades de convergencia y justificación de sus características como optimizadores, selección de parámetros y aplicabilidad a problemas de optimización entera mixta.

Respecto de las propiedades de convergencia, se ha discutido efectivamente la capacidad de estos algoritmos para optimizar funciones objetivo. Sin embargo, no hay pruebas que garanticen con probabilidad 1 la convergencia de ellos, pues en general se basan en supuestos poco realistas acerca de los métodos (por ejemplo asumen que son determinísticos cuando en realidad son estocásticos), y cuando son realistas, no hay garantía que la convergencia ocurra en tiempo finito (o al menos aplicable en un sistema de control).

El tema de selección de parámetros es un tema complejo, y si bien existen conjuntos de parámetros aceptados como buenos, se reconoce que en general los parámetros óptimos dependen de los problemas a optimizar. Además, la sintonía de parámetros es costosa computacionalmente, y por lo tanto debe analizarse muy bien si es beneficioso o no incurrir en ese esfuerzo.

Finalmente, diversos estudios han sido analizados respecto de la aplicabilidad de algoritmos evolutivos en problemas enteros-mixtos no lineales. Se ha concluido que los aspectos más importantes a tomar en cuenta son la representación de las soluciones (por los distintos tipos de variables), los operadores de búsqueda (que deben ser acordes con los distintos tipos de variables) y el manejo de restricciones. Las representaciones y el manejo de restricciones, de acuerdo a las estrategias planteadas pueden ser utilizados transversalmente sea en algoritmos genéticos, estrategias evolutivas, optimización por enjambre de partículas, etc., sin menospreciar el hecho que pueden presentar desempeños distintos para las diversas heurísticas.

En base al análisis desarrollado en este capítulo acerca de algoritmos evolutivos, y de los aspectos analizados en capítulos anteriores acerca de sistemas híbridos y control predictivo, se presenta en el siguiente capítulo lo que constituye el aporte central de esta tesis: los aspectos relevantes para el diseño y análisis de algoritmos evolutivos para la resolución de los problemas de optimización que aparecen en control predictivo híbrido de sistemas no lineales. En los capítulos posteriores, se aborda la aplicación de los aspectos previamente presentados en tres casos de estudio (donde se trabaja con simuladores de los procesos reales), que permitirán estudiar la efectividad de las propuestas realizadas y, en base a los resultados, proponer nuevas líneas de trabajo futuro.

5 Control predictivo híbrido basado en algoritmos evolutivos

5.1 Introducción

En la Sección 3.3.5 ya se ha revisado varias formas de resolver distintas formulaciones de control predictivo híbrido a partir de variadas estrategias. Se puede utilizar MILP/MIQP o MINLP cuando se resuelve el problema de optimización de forma *online*, u optimización multiparamétrica cuando el cálculo de la acción de control explícita es realizado *offline*. Todas estas estrategias poseen desventajas. En el caso de las formulaciones como MILP/MIQP o MINLP la resolución del problema de optimización puede ser muy demandante computacionalmente y no ser aplicable en tiempo real. En el caso de la resolución con optimización multiparamétrica, el problema es válido solo para sistemas lineales por tramos, y aún así la complejidad del problema crece rápidamente con el número de variables discretas o con el número de tramos con distintos modelos lineales.

Se plantea entonces la resolución del problema de control predictivo híbrido con algoritmos evolutivos para tener una forma genérica de abordar este problema sobre sistemas no lineales, de un modo que la aplicación en tiempo real sea posible. No se debe perder de vista que los algoritmos evolutivos no pueden garantizar optimalidad, pero de todos modos son una gran herramienta para poder analizar sistemas de alta complejidad los cuales dadas ciertas condiciones como las ya expuestas no pueden ser manejados mediante algoritmos convencionales.

En este capítulo se presentan las alternativas existentes, sus ventajas y desventajas, de aspectos relevantes para el diseño de controladores predictivos híbridos basados en algoritmos evolutivos, como lo son la representación de variables continuas y discretas, manejo de restricciones, sintonía de parámetros y análisis de tiempo computacional. Además, previamente se realiza una revisión bibliográfica de las aplicaciones que han utilizado algoritmos evolutivos en control predictivo, para tomar como punto de partida.

En esta tesis, se ha presentado como alternativas para resolver los problemas de optimización a los algoritmos evolutivos clásicos, evolución diferencial y optimización por enjambre de partículas. Sin embargo, se clarifica que las aplicaciones analizadas en esta tesis son resueltas sólo utilizando algoritmos genéticos (que es parte de los algoritmos evolutivos clásicos) y optimización por enjambre de partículas.

5.2 Revisión bibliográfica

Debido a que la solución del problema de control predictivo no lineal puede ser muy demandante computacionalmente y a que los modelos utilizados pueden ser muy no lineales y con muchos óptimos (entre otros, pero siendo estos los argumentos principales), los métodos de optimización convencionales pueden no entregar desempeños satisfactorios. Fundamentalmente por estos motivos, en la literatura se ha reportado variados esfuerzos por resolver de un modo eficiente los problemas de optimización del control predictivo utilizando métodos de optimización global estocástica, tales como algoritmos evolutivos u optimización de enjambre de partículas.

Uno de los primeros trabajos en los que se estudia la aplicación de algoritmos evolutivos para control predictivo es desarrollador por Onnen *et al*. (1997). En este trabajo se propone un algoritmo genético operadores evolutivos especializados, representación binaria, manejo de restricciones mediante representación factible, e inclusión de la solución en el instante anterior actualizada de acuerdo a la estrategia de horizonte deslizante. Se aplica para el control de un fermentador batch con acciones de control continuas, que son discretizadas para utilizar la representación binaria. Luego, los cromosomas son un string que codifican las acciones de control a lo largo del horizonte de control (enfoque que se utiliza siempre en la aplicación de algoritmos evolutivos para control predictivo). Luego la estrategia basada en GA es comparada con una basada en *branch and bound* y se muestra que encuentran soluciones de calidad similar, pero que GA es mucho más conveniente en tiempo computacional para horizontes de predicción altos. El principal problema de este trabajo es que se proponen criterios de término y un tamaño de población para el algoritmo genético que no están relacionados con la capacidad de cómputos o el tiempo requerido para aplicar las acciones de control. Otro trabajo importante en esta línea es el desarrollado por Fravolini *et al*. (2008) donde se proponen originales ideas para la aplicación de algoritmos evolutivos en control predictivo. Se propone una discretización del espacio factible de las acciones de control, y se muestra que una variación dinámica de esta es beneficiosa frente a una discretización estática. Además se propone una representación entera, con operadores especializacidos, incluyendo elitismo e inclusión de las mejores soluciones anteriores en la nueva población actualizadas de acuerdo a la estrategia de horizonte deslizante. El problema de la implementación en tiempo real sí es considerado mediante la aplicación de una estrategia de realimentación intermitente, que consiste en asignar más de un tiempo de muestreo para resolver el problema de optimización, y por mientras se aplican las acciones de control posteriores al primer instante de la secuencia anterior encontrada. Se concluye que existe un horizonte óptimo en el cual se tiene el compromiso adecuado entre la ganancia de precisión en el algoritmo evolutivo por el tiempo extra, y la pérdida de precisión por no utilizar la información más actualizada para calcular las nuevas acciones de control. Este trabajo sin embargo, no considera el efecto de utilizar distintos tamaños de población y número iteraciones.

En otro tipo de trabajos se estudia la factibilidad de aplicar algoritmos evolutivos, sin proponer estrategias originales ni probar distintas alternativas. Por ejemplo, Na y Hwang (2006) presentan una estrategia de control predictivo basado en modelos para un controlador de potencia PWR con un algoritmo genético donde la predicción se realiza utilizando un modelo CARIMA. Yuzgec *et al*. (2006) proponen un algoritmo genético para resolver el problema de optimización en el control predictivo basado en un modelo no lineal fenomenológico para un proceso de secado. En este trabajo se concluye que la estrategia posee un desempeño satisfactorio, fácilmente adaptable a otros procesos batch, pero que el tiempo de ejecución que se requiere de GA para converger no permite su ejecución en sistemas con un tiempo de muestreo pequeño.

Siguiendo la línea de trabajos del párrafo anterior, existen trabajos que utilizan modelos difusos para la predicción de la salida de los procesos. Sarimveis y Bafas (2003) presentan uno de los primeros trabajos donde resuelven el problema de optimización de MPC basado en modelos de Takagi-Sugeno, utilizando un algoritmo genético especializado. El mismo enfoque es utilizado en Na y Upadhyaya (2006) para resolver el problema de control predictivo de la potencia termoeléctrica de un reactor espacial.

Otros controladores predictivos utilizan un modelo neuronal del proceso para calcular las predicciones del sistema. Shin y Park (1998), con este tipo de modelos utilizan un Algoritmo Genético para resolver el problema de optimización basado en el modelo neuronal, y muestran la

obtención de mejores resultados que al utilizar una técnica de optimización cuasi-Newton. Del mismo modo, Wooley *et al*. (1998) propone un controlador predictivo con GA basado en redes RBF.

PSO también ha mostrado ser eficiente para resolver problemas de control predictivo. Coelho *et al*. (2005) presenta un controlador predictivo basado en modelos lineales recursivos. Se muestra que el enfoque basado en PSO muestra buenos resultados en comparación con GA y métodos cuasi-Newton. Solís *et al*. (2006), desarrolla una estrategia de FMPC basado en PSO con modelos de Takagi-Sugeno (Takagi y Sugeno, 1985) para el proceso, y muestra que se obtienen mejores resultados que utilizando algoritmos genéticos simples o de nichos en un sistema no lineal benchmark. PSO también es usado en el control predictivo basado en modelos neuronales. Wang y Xiao (2005) describen un controlador predictivo basado en PSO utilizando un modelo neuronal RBF y obtienen mejores resultados que GA y métodos cuasi-Newton. Song *et a*l. (2007) desarrollan un controlador predictivo basado en redes neuronales y el problema de optimización es resuelto mediante una modificación de PSO para evitar quedarse estancado en mínimos locales y mejorar las cualidades de búsqueda.

Existen además otras estrategias más producidas para la aplicación de algoritmos evolutivos en control predictivo. Por ejemplo en Fravolini y La Cava (1999) se propone un esquema de control predictivo adaptativo basado en algoritmos evolutivos. La parte adaptativa consiste en la utilización de un algoritmo evolutivo para la actualización de los pesos de una red neuronal que se utiliza para calcular las predicciones. Además, se utiliza otro algoritmo evolutivo para calcular el conjunto de acciones de control óptimas. Se realiza un análisis de la complejidad computacional llegando a la conclusión que ésta aumenta linealmente con el número de variables de decisión del sistema. Además muestran que el sistema es robusto frente a perturbaciones, y el desempeño posee una pequeña desviación estándar mostrando que es confiable y posee un alto grado de repetibilidad. Song y Kusiak (2009), por su parte, proponen un novedoso esquema para el control de un proceso temporal no lineal basado en MPC. Se utilizan técnicas de *data-mining* para aprender las dinámicas del proceso, y luego se utiliza una estrategia evolutiva para resolver el problema de optimización.

Existen otros tipos de usos de algoritmos evolutivos en control predictivo que no consisten en resolver el problema de optimización en tiempo real. Por ejemplo, Goggos y King (1996) proponen una estrategia basada en control por matriz dinámica[8] (*Dynamic Matrix Control*, DMC), donde se utiliza un algoritmo genético para encontrar las matrices de peso y la matriz dinámica del sistema en tiempo real (en cada instante de muestreo), para casos con y sin restricciones.

Respecto de controladores predictivos basados en algoritmos evolutivos para sistemas híbridos, no existen muchos trabajos. Causa *et al*. (2008) presenta un esquema de control predictivo con variables manipuladas discretas con un modelo de Takagi-Sugeno donde resuelve el problema de optimización mediante algoritmos genéticos. Sáez *et al.* (2008) y Cortés *et al.* (2009), presentan una estrategia de control predictivo híbrido basado en algoritmos evolutivos para un problema

[8] DMC es una estrateia de control predictivo basada en un modelo discreto de respuesta al escalón para predecir la salida del modelo. Luego el problema de minimización entrega una solución de mínimos cuadrados, y la ley de control depende de la matriz dinámica cuyos elementos son la respuesta al escalón, dos matrices de pesos y el error de seguimiento (Camacho y Bordons, 1999).

dinámico de vehículos, donde el primero utiliza algoritmos genéticos, y el segundo optimización por enjambre de partículas.

Los algoritmos inteligentes mencionados también han sido utilizados en control óptimo. Michalewicz *et al*. (1990) estudian la aplicación de algoritmos genéticos para problemas de control óptimo en tiempo discreto. Se compara esta estrategia con el modelo de programación matemática GAMS. Mientras GAMS sólo parece funcionar para control óptimo cuadrático lineal o en problemas con un horizonte pequeño, el algoritmo genético en general funciona bastante bien. Luego Dakev *et al*. (1996) proponen convertir el problema de control óptimo continuo en un problema de optimización con restricciones en tiempo discreto, aplicado al control de sistemas multi-cuerpos. El enfoque se encuentra basado en una estrategia multi-objetivo y se utiliza procesamiento paralelo para aliviar la carga computacional de sistemas de alta dimensionalidad. Yucheng *et al*. (2002) desarrollan un algoritmo genético para resolver el problema de control óptimo de la trayectoria de naves espaciales. En este trabajo se muestra que la estrategia de control con algoritmos evolutivos es mucho más sencilla, no necesita análisis algorítmicos complicados, y puede alcanzar resultados muy efectivos. En Arumugam *et al.* (2005) se desarrollan operadores genéticos especializados para el control óptimo de una clase de sistemas híbridos que se encuentra en el contexto de la industria de manufactura. En Arumugam y Rao (2006) se compara el desempeño de GA con codificación real y los operadores especializados y PSO con distintas estrategias para el coeficiente de inercia, para resolver un problema de control óptimo del mismo proceso. Se muestra que el esquema basado en PSO que utiliza un factor de inercia que depende de las posiciones de *Gbest* y *Lbest* para resolver el problema de optimización supera en rendimiento y precisión a GA y PSO con coeficientes de inercia constante y decreciente.

De acuerdo a esta revisión se concluye que existen muchos trabajos donde se justifica la aplicación de algoritmos evolutivos para control predictivo. Sin embargo, no existen muchos trabajos donde se justifique su aplicación para control predictivo híbrido. Además, no se ha realizado un estudio detallado respecto de los aspectos relevantes para su aplicación en tiempo real. Más aún, si bien se presentan algunas estrategias específicas para control predictivo continuo basado en algoritmos evolutivos, no se realiza un estudio comparativo con otras posibles alternativas y los análisis para implementación en tiempo real aún pueden ser mejorados. Estas consideraciones se tienen en cuenta para presentar nuevas alternativas para la aplicación de algoritmos evolutivos para control predictivo, con especial énfasis en sistemas híbridos y su aplicación en tiempo real.

5.3 Formulación

De acuerdo a lo presentado en la Sección 3.3.5, el problema de control predictivo híbrido no lineal se puede formular como sigue.

$$\min_{u=(u(t),\ldots,u(t+N_u-1))} J=\sum_{k=1}^{N_y}\left\|y(t+k|t)-y_{ref}(t+k)\right\|_{Q_y}^2+\sum_{k=1}^{N_u}\left\|\Delta u(t+k-1|t)\right\|_{Q_u}^2$$

$$\begin{aligned} s.a. \quad & x(t+k|t)=f(x(t+k-1|t),u(t+k-1|t)) \\ & y(k+t\,|\,t)=g(x(k+t-1\,|\,t),u(k+t-1)) \\ & c_1(y(k+t\,|\,t),x(k+t-1\,|\,t),u(k+t-1))\le 0 \\ & c_2(y(k+t\,|\,t),x(k+t-1\,|\,t),u(k+t-1))=0 \\ & k=1,\ldots,N_y \end{aligned} \tag{5.1}$$

donde $x_k \in X \subseteq \mathbb{R}^{n_{xc}} \times \mathbb{Z}^{n_{xd}}$, $y_k \in Y \subseteq \mathbb{R}^{n_{yc}} \times \mathbb{Z}^{n_{yd}}$, $u_k \in U \subseteq \mathbb{R}^{n_{uc}} \times \mathbb{Z}^{n_{ud}}$, $Q_y \in \mathbb{R}^{n_{yc}} \times \mathbb{R}^{n_{yc}}$, $Q_u \in \mathbb{R}^{n_{uc}} \times \mathbb{R}^{n_{uc}}$. N_y es el horizonte de predicción y N_u es el horizonte de predicción. Q_y, Q_u son matrices de peso que satisfacen $Q = Q' \ge 0$. c_1, c_2 son funciones en general no lineales que definen las restricciones de desigualdad y de igualad respectivamente, que debe satisfacer el proceso durante su operación. n_{xc}, n_{yc}, n_{uc}, n_{xd}, n_{yd}, n_{ud} son las dimensiones de los estados, salidas y acciones de control continuas y discretas.

El problema incluye entonces la función objetivo cuadrática, más un conjunto de restricciones de igualdad que corresponden a la dinámica y salidas del sistema, restricciones de desigualdad que pueden corresponder a requerimientos operativos del proceso, tanto de las salidas y estados como acciones de control, y restricciones de igualdad que pueden obedecer a condiciones terminales u otro tipo de condiciones operativas. En el caso más general, todas las funciones presentes aquí son no lineales.

La formulación anterior corresponde a un problema de optimización entera mixta no lineal, ya que posee una función objetivo cuadrática, restricciones no lineales y variables enteras y continuas.

Los aspectos más relevantes respecto de los detalles necesarios para la implementación de algoritmos evolutivos para la solución del problema de optimización del control predictivo híbrido se presentan a continuación.

5.4 Representación de las soluciones

En los casos más generales, el espacio factible de las acciones de control a lo largo del horizonte de predicción es $n = N_u \cdot (n_{uc} + n_{ud})$. Es decir, se debe encontrar todas las acciones de control a lo largo del horizonte de predicción, de modo que la función objetivo sea óptima. Así, si $u(t) = [u_{1c}(t),\ldots,u_{nc}(t),u_{1d}(t),\ldots,u_{nd}(t)]$, entonces la solución candidata, en la estructura de partícula por ejemplo si se utilizará PSO sería:

$$x = [u(t), u(t+1),\ldots,u(t+N_u-1)] \tag{5.2}$$

Es importante destacar que no cualquier tipo de representación es compatible con cualquier algoritmo evolutivo, en particular de los operadores utilizados.

5.4.1 Caso continuo

Cuando las acciones de control son únicamente continuas, el problema de optimización es básicamente uno de programación no lineal. Estas variables pueden ser representadas mediante representación real o mediante representación binaria.

Si se utiliza una representación real, los operadores sobre las soluciones candidatas de PSO y DE son los clásicos. En algoritmos evolutivos, la mutación, se suele utilizar mutación *Gaussiana* y uniforme, y el *crossover* aritmético (aunque también se puede utilizar *crossover* de *N* puntos). Luego, la representación de las soluciones candidatas es de dimensión $n = N_u \cdot n_{uc}$ y tienen la forma:

$$x = \left[u(t), u(t+1), ..., u(t+N_u - 1)\right] \tag{5.3}$$

donde $u(t) = \left[u_{1c}(t), ..., u_{nc}(t)\right]$.

Si se utiliza representación binaria (generalmente en algoritmos genéticos), se puede utilizar cualquier operador de mutación y *crossover* para individuos con este tipo de codificación. Sin embargo, ha sido reportado que para problemas continuos con restricciones, sistemáticamente la codificación real es mejor que la binaria (Michalewicz y Schmidt, 2002). Por lo tanto, la representación binaria no es utilizada para la representación de este tipo de variables.

5.4.2 Caso discreto

Las variables discretas se pueden dividir en tres categorías: binarias, enteras o discretas. Las variables binarias son aquellas que sólo pueden tomar los valores 0 ó 1. Las variables enteras son aquellas que sólo pueden tomar valores en el dominio $\mathbb{Z}$, y las discretas son aquellas que pueden tomar valores de un conjunto numerable (cualquiera) de puntos. Así, el caso más general es el de las variables discretas, y las variables enteras y las binarias son casos particulares. A continuación se presentan las distintas opciones que existen para manejar cada tipo de variables.

Representación continua

En este caso se admite que las coordenadas de los individuos tomen valores distintos a los admisibles en el dominio del problema, que son discretos. Por lo general, este método se utiliza en PSO y DE que son inherentemente reales, y por lo tanto se permite que las coordenadas de los individuos tomen valores reales. Luego, para evaluar la función de *fitness*, se fuerza de algún modo la condición de que la solución sea discreta.

Una primera opción para evaluar el *fitness*, en el caso de variables enteras, es directamente aplicar la parte entera de la solución. Sin embargo, esto produce un desequilibrio pues el valor evaluado no siempre es el más cercano al presente en el individuo. Por ejemplo, si las soluciones admisibles son $\{1,2,3\}$, un individuo *i* con coordenada *j* $x_{ij} = 1.98$ se decodificaría como $x_{ij} = 1$, siendo que claramente está más cerca de 2. Por lo tanto, se propone decodificar como el valor

admisible más cercano al valor de la coordenada (lo que es básicamente un redondeo). Luego, el mismo individuo se decodificaría como $x_{ij} = 2$, pues dentro de las soluciones admisibles $\{1, 2, 3\}$, 2 es el más cercano a 1.98. Se plantea utilizar esta idea tanto para variables binarias como para variables discretas (en el caso general). Sin embargo, para el caso general, los valores admisibles podrían no estar distribuidos uniformemente, lo que produciría sesgo en la búsqueda, y por lo tanto no siempre es conveniente aproximar al valor más cercano.

Para hacer más claro esto se presenta el siguiente ejemplo. En el caso del reactor batch, para la válvula de mezcla k_M, los valores admisibles son $\{0, 0.01, 0.02, 0.05, 0.1, 1\}$. Es claro que la repartición de los valores admisibles en el intervalo es no uniforme, y así, casi el 50% del intervalo sería asignado al valor $k_M = 1$.

En este sentido, lo que se propone es distribuir el intervalo $[0,1]$ en un número de intervalos igual al número de valores admisibles, y si una coordenada cae en el casillero correspondiente, entonces la acción de control decodificada será la correspondiente a ese casillero. Así, no hay un mayor sesgo en la búsqueda de las soluciones. En el caso del ejemplo del reactor batch, si se reparte el intervalo de modo uniforme, los valores de k_M al aproximar serían:

$$x_i = \begin{cases} 0 & if\ x < 0.166 \\ 0.01 & if\ 0.166 \le x < 0.333 \\ 0.02 & if\ 0.333 \le x < 0.5 \\ 0.05 & if\ 0.5 \le x < 0.666 \\ 0.1 & if\ 0.666 \le x < 0.833 \\ 1 & if\ 0.833 \le x < 1 \end{cases} \tag{5.4}$$

Otra forma de representar variables discretas, que ha sido presentada para variables binarias por Li (1992), consiste en trabajar libremente con las variables continuas, y añadir la siguiente restricción que fuerce la condición binaria:

$$\begin{aligned} & g(x) = x(x-1) = 0 \\ & x \in \mathbb{R} \end{aligned} \tag{5.5}$$

Es aplicable a todos los algoritmos evolutivos que funcionan con variables continua, tales como PSO o DE. En este caso, la restricción de igualdad se maneja incorporando en la función objetivo una penalización (ver punto 5.6.2.1):

$$\min_{x \in \Omega} J(x) = J_0(x) + \alpha g(x)^2 \tag{5.6}$$

Una desventaja que tiene esta estrategia es que añade no convexidad al problema, y así el riesgo de obtener una solución local aumenta.

Representación exacta

En esta opción las coordenadas de los individuos sólo pueden tomar valores que están en el dominio de la solución (o bien, una codificación de ellos), es decir, los valores binarios, enteros o discretos según corresponda.

Este modo es fácilmente utilizable en algoritmos evolutivos, donde al usar los operadores de mutación y *crossove*r clásicos (aquellos que no son aritméticos, sino coordenada a coordenada) se asegura directamente que las asignaciones de cada coordenada sean las posibles en el conjunto acotado de valores posibles. Es relevante notar que en este caso los distintos tipos de valores binarios son completamente equivalentes, pues no existe ninguna interacción entre los distintos valores, y así tanto las variables binarias como las discretas se pueden manejar como el caso discreto más general.

Este método no es tan claro cuando se utiliza PSO o DE, pues se trata de algoritmos inherentemente continuos. Por lo tanto, para mantener los valores discretos, se debe incluir dentro de los operadores alguna función de aproximación. Por ejemplo, para el caso de variables enteras, en PSO se podría modificar la función de actualización de velocidad aplicando la función cajón inferior:

$$\begin{aligned} v_{ij}(t+1) &= \left\lfloor \omega \cdot v_{ij}(t) + c_1 \cdot \varphi_1 \cdot (pbest(t) - x_{ij}(t)) + c_2 \cdot \varphi_2 \cdot (lbest(t) - x_{ij}(t)) \right\rfloor \\ x_{ij}(t+1) &= x_{ij}(t) + v_{ij}(t+1) \end{aligned} \tag{5.7}$$

donde $\lfloor \cdot \rfloor$ es la función cajón inferior o parte entera. Sin embargo, ya ha sido mencionado que usar el cajón inferior puede no ser la mejor alternativa, y podría ser más deseable redondear al entero más cercano.

Para el caso discreto más general, no se puede usar el cajón inferior, pues los números no son necesariamente enteros. Entonces, se puede hacer dos cosas equivalentes a las presentadas para el caso de representación continua. Una, es aproximar directamente al valor más cercano, pero como en este caso toda la representación en el individuo también debe ser discreta se aproxima al valor discreto correspondiente. Por ejemplo, en el caso de PSO el operador quedaría como:

$$\begin{aligned} v_{ij}(t+1) &= \omega \cdot v_{ij}(t) + c_1 \cdot \varphi_1 \cdot (pbest(t) - x_{ij}(t)) + c_2 \cdot \varphi_2 \cdot (lbest(t) - x_{ij}(t)) \\ x_{ij}(t+1) &= nearest\left((x_{ij}(t) + v_{ij}(t+1), \Omega\right) \end{aligned} \tag{5.8}$$

donde $nearest(x, \Omega)$ es una función que entrega el valor discreto perteneciente al conjunto Ω más cercano al número real x.

La otra alternativa consiste en dividir el intervalo $[0,1]$ en un número de intervalos $I_i = [a_i, b_i]$ igual al número de valores discretos posibles, donde cada intervalo está asociado a un valor discreto. Como la representación debe ser discreta, el valor central del intervalo representará la solución correspondiente. Así, en el caso de PSO el operador sería:

$$
\begin{aligned}
v_{ij}(t+1) &= \omega \cdot v_{ij}(t) + c_1 \cdot \varphi_1 \cdot (pbest(t) - x_{ij}(t)) + c_2 \cdot \varphi_2 \cdot (lbest(t) - x_{ij}(t)) \\
x_{ij}(t+1) &= central\left((x_{ij}(t) + v_{ij}(t+1), I\right)
\end{aligned}
\tag{5.9}
$$

donde $central(x, I)$ entrega la posición central del intervalo $I_i \in I = \{I_1, ..., I_{nd}\}$ al que pertenece el número x. Luego, para evaluar el *fitness*, se utiliza la codificación que relaciona el valor central del intervalo con el valor discreto admisible.

Entonces, en el ejemplo del reactor batch, la representación en las partículas sería:

$$
x_i = \begin{cases}
0.083 & if\ (x_{ij}(t) + v_{ij}(t+1) < 0.166 \\
0.250 & if\ 0.166 \le (x_{ij}(t) + v_{ij}(t+1) < 0.333 \\
0.417 & if\ 0.333 \le (x_{ij}(t) + v_{ij}(t+1) < 0.5 \\
0.583 & if\ 0.5 \le (x_{ij}(t) + v_{ij}(t+1) < 0.666 \\
0.750 & if\ 0.666 \le (x_{ij}(t) + v_{ij}(t+1) < 0.833 \\
0.917 & if\ 0.833 \le (x_{ij}(t) + v_{ij}(t+1) < 1
\end{cases}
\tag{5.10}
$$

Y entonces la codificación que relaciona el valor de las coordenadas con la solución correspondiente en la función objetivo es:

$$
k_M = \begin{cases}
0 & if\ x_{ij}(t) = 0.083 \\
0.01 & if\ x_{ij}(t) = 0.250 \\
0.02 & if\ x_{ij}(t) = 0.417 \\
0.05 & if\ x_{ij}(t) = 0.583 \\
0.1 & if\ x_{ij}(t) = 0.750 \\
1 & if\ x_{ij}(t) = 0.917
\end{cases}
\tag{5.11}
$$

Representación factible

Este esquema de representación nace cuando la ciertas representaciones pueden generar soluciones que no son factibles, pero utilizando otra representación se puede lograr factibilidad para todas las soluciones que se generen.

Para comprender mejor este caso, se analiza nuevamente el caso del reactor batch. En este problema se observan dos acciones de control. Una válvula de mezcla k_M que puede tomar los valores $\{0, 0.01, 0.02, 0.05, 0.1, 1\}$, y otra válvula de entrada q que puede tomar los valores 1 ó 2. De acuerdo con las restricciones del sistema, las únicas entradas admisibles son:

$$
M = \left\{ \begin{bmatrix} 0 \\ 1 \end{bmatrix}, \begin{bmatrix} 0.01 \\ 1 \end{bmatrix}, \begin{bmatrix} 0.02 \\ 1 \end{bmatrix}, \begin{bmatrix} 0.05 \\ 1 \end{bmatrix}, \begin{bmatrix} 0.1 \\ 1 \end{bmatrix}, \begin{bmatrix} 1 \\ 1 \end{bmatrix}, \begin{bmatrix} 1 \\ 0 \end{bmatrix} \right\}
\tag{5.12}
$$

Sin embargo, si se representa como en el caso discreto general, perfectamente podrían aparecer soluciones no factibles, tales como $x = (0.05, 0)$, que no son factibles. Sin embargo, si se realiza la siguiente codificación:

$$1 = \begin{bmatrix} 0 \\ 1 \end{bmatrix}, 2 = \begin{bmatrix} 0.01 \\ 1 \end{bmatrix}, 3 = \begin{bmatrix} 0.02 \\ 1 \end{bmatrix}, 4 = \begin{bmatrix} 0.05 \\ 1 \end{bmatrix}, 5 = \begin{bmatrix} 0.1 \\ 1 \end{bmatrix}, 6 = \begin{bmatrix} 1 \\ 1 \end{bmatrix}, 7 = \begin{bmatrix} 1 \\ 0 \end{bmatrix}, \tag{5.13}$$

se puede generar soluciones siempre factibles si se utiliza la representación con $x \in \{1, 2, ..., 7\}$. Esto corresponde a una representación exacta para esta codificación. Sin embargo, también podría utilizarse una representación con $x \in [0, 7]$, y donde la acción de control corresponde al cajón superior de x, situación que corresponde a la representación continua.

5.4.3 Caso híbrido

Cuando las acciones de control toman valores tanto continuos como discretos, el problema de optimización resultante es entero mixto. Para resolver este problema, existen dos opciones. La primera de ellas consiste en aplicar un algoritmo evolutivo donde las soluciones candidatas contienen tanto las variables continuas como discretas, como se ha presentado al comienzo de 5.4. La segunda opción consiste en resolver un problema de dos etapas, una para las variables discretas, y otro para las variables continuas.

5.4.3.1 Manejo conjunto de variables discretas y continuas

Las soluciones candidatas en los casos más generales poseen una dimensión fenotípica (independiente de la representación interna) $N = N_u \cdot (n_{uc} + n_{ud})$, es decir, se debe encontrar todas las acciones de control a lo largo del horizonte de predicción, de modo que la función objetivo sea óptima. Entonces:

$$x = [u(t), u(t+1), ..., u(t+N_u - 1)] \tag{5.14}$$

donde $u(t) = [u_{1c}(t), ..., u_{nc}(t), u_{1d}(t), ..., u_{nd}(t)]$.

5.4.3.2 Manejo separado de variables discretas y continuas

A diferencia del caso anterior, donde el individuo incluye tanto las variables continuas como las discretas, existe otra alternativa en la cual el problema de optimización se resuelve en dos etapas. En una primera etapa, se prueban las posibles soluciones discretas, que corresponde a un problema de optimización discreto. Luego, para cada solución candidata discreta, las variables continuas están libres, entonces lo que queda por resolver es un problema de optimización continuo, que corresponde a la segunda etapa del algoritmo. El procedimiento descrito se presenta de forma genérica en la Figura 19.

Las técnicas para resolver cada etapa de los problemas de optimización pueden ser escogidas de múltiples formas. Por ejemplo, para la etapa discreta se puede utilizar cualquier algoritmo de optimización discreta, sea óptimo o sub-óptimo como branch and bound, GA, etc. Equivalentemente, para la etapa continua, se puede utilizar cualquier algoritmo de optimización

continua, óptimo o sub-óptimo, como SQP, Newton, GA, PSO, etc. Sin embargo, para que la los análisis presentados en este capítulo tengan sentido, en alguna de las dos etapas se debiera utilizar algún algoritmo evolutivo, como GA o PSO.

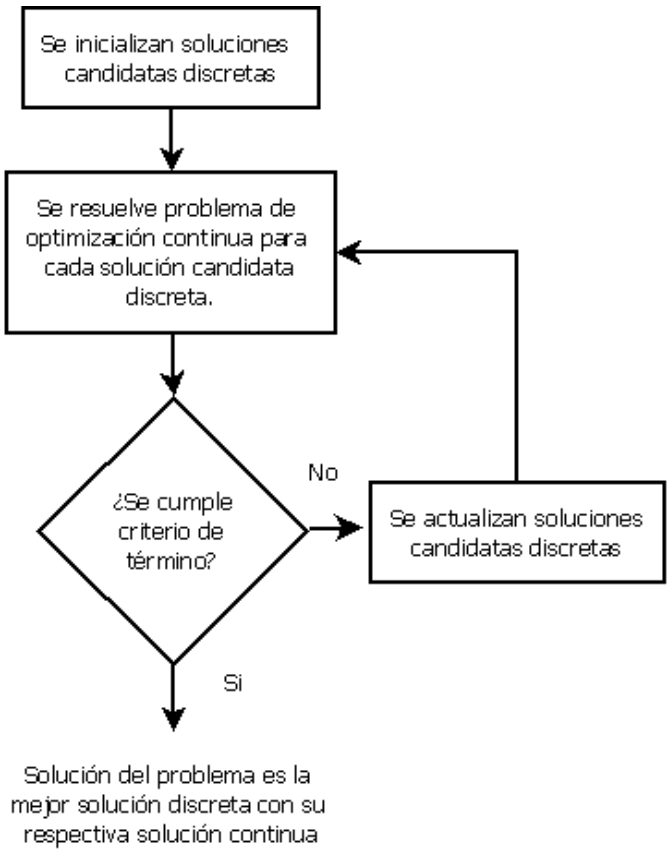


Figura 19. Manejo separado de variables continuas y discretas

5.5 Inicialización de soluciones candidatas

De la literatura se observa, que en general, en los algoritmos evolutivos la inicialización de los individuos o partículas puede tener un importante efecto en la calidad de las soluciones. Esto es bastante evidente en los métodos convencionales de optimización, pues si se trata de un problema con más de un óptimo local, el óptimo que sea encontrado dependerá del punto de partida. Los algoritmos evolutivos, por su naturaleza basada en poblaciones, son bastante menos sensibles al punto de partida, pero de todos modos lo son. En especial, un buen punto de partida puede favorecer a una rápida convergencia del algoritmo, lo cual en el contexto de la aplicación en tiempo real, es altamente favorable.

Por lo general, la usanza más común es inicializar las soluciones candidatas al azar. Sin embargo, en algunos problemas es posible utilizar conocimiento experto en la inicialización de las soluciones.

Si se asume que el estado del sistema no cambia mucho entre instantes de control sucesivos, se puede asumir lo mismo para el espacio de búsqueda del problema de optimización. Luego, la

solución del instante de control anterior debiese ser similar a la del instante de control actual. Por lo tanto, la inclusión de la solución encontrada en el instante anterior en la nueva población puede proveer un buen punto de partida para la búsqueda.

Esta estrategia, sin embargo, en principio no es de gran ayuda durante transientes y cambios de referencia, pues la suposición de que el estado del sistema no cambia entre instantes de control sucesivos no se cumple. A pesar de esto, tampoco deteriora la performance del algoritmo de optimización (gracias al enfoque basado en poblaciones), y en ciertas ocasiones sí brinda ayuda durante transientes (cuando las soluciones en instantes son similares a pesar que el estado del sistema y el espacio de búsqueda cambie).

Ahora se muestra que sí es posible utilizar la solución del instante anterior para que sea beneficiosa en los transientes de un modo consistente. Sea la secuencia de acciones de control encontrada en el instante $t-1$:

$$u^*(t-1|t-1) = \left[u^*(t-1), u^*(t), ..., u^*(t+N_u-2) \right] \tag{5.15}$$

Luego, de acuerdo a la estrategia de horizonte deslizante, la solución en el instante t dada la información en $t-1$ (que serían iguales en ausencia de perturbaciones y con un modelo perfecto) es:

$$u(t|t-1) = \left[u(t|t-1), u(t+1|t-1), ..., u(t+N_u-2|t-1), u(t+N_u-1|t-1) \right] \tag{5.16}$$

Entonces, de acuerdo a la ecuación (5.15), el primer elemento de dicha secuencia se descarta y el resto constituye los primeros N_u elementos de la secuencia $u^*(t|t-1)$, es decir, es la secuencia anterior desplazada. El último elemento es el único que resta, pero al considerar que en la formulación del control predictivo se asume que en los instantes posteriores al horizonte de control las acciones de control permanecen constantes, es trivial notar que la acción de control que falta en la secuencia debe ser igual a la anterior en la nueva secuencia.

$$u(t|t-1) = \left[u^*(t-1), u^*(t), ..., u^*(t+N_u-2), u^*(t+N_u-2) \right] \tag{5.17}$$

Es claro que esta secuencia de acciones de control considera el transiente del sistema, y por lo tanto insertarla en la nueva población claramente otorgaría un buen punto inicial para los algoritmos evolutivos cuando resuelvan los problemas de optimización en los transientes del sistema. Además, cuando el sistema ya ha alcanzado el régimen permanente, en teoría las acciones de control en los distintos instantes de control debieran ser iguales. Debido a esto desplazar las acciones de la secuencia no debiera cambiar mucho las soluciones y por lo tanto esta estrategia también tendría que ser beneficiosa en el régimen permanente.

Finalmente, de acuerdo a lo presentado aquí, se plantean entonces tres alternativas para la inicialización de la población.

1. Inicializar la población completamente al azar.
2. Inicializar la población al azar, salvo un individuo que es igual a la solución del instante de control anterior.

3. Inicializar la población al azar, salvo un individuo que la solución del instante de control anterior desplazada de acuerdo a la estrategia de horizonte deslizante.

La segunda estrategia (incluir solución anterior en población) debiese brindar beneficios en un modo consistente respecto de la primera alternativa (inicializar completamente al azar) mientras el sistema se encuentra en régimen permanente, y la tercera (incluir solución anterior desplazada) brindaría beneficios tanto en régimen permanente como en transientes.

No obstante lo anterior, el efecto en los transientes de incluir la solución anterior desplazada o no desplazada, debiera depender del tiempo de muestreo utilizado y de la dinámica del proceso controlado. En efecto, si el tiempo de muestreo es pequeño comparado con la dinámica del sistema, el cambio en el estado del sistema entre instantes sucesivos debe ser pequeño, y en ese caso no habría mucha variación entre instantes sucesivos en un transiente, y las respuestas utilizando la solución anterior desplazada y no desplazada debiesen entregar resultados similares. Ahora, si el tiempo de muestreo es relativamente grande, entre instantes sucesivos la variación en el estado si es relevante, y entones el efecto de desplazar la solución anterior debiese ser mayor, ya que la solución inserta considera la variación por la dinámica del sistema. En resumen, para tiempos de muestreo grandes, insertar la solución anterior desplazada debiese brindar mayores beneficios que insertar la solución anterior intacta. Mientras que ha medida que disminuye el tiempo de muestreo, este beneficio también disminuye, y ambas estrategias se hacen más semejantes.

5.6 Manejo de restricciones

5.6.1 Conceptos preliminares

El control predictivo en general alcanza su máximo potencial cuando se incorporan las restricciones de los procesos. Esto en general plantea dificultades al resolver los problemas de optimización asociados con algoritmos evolutivos, pues éstos no están creados para resolver problemas con restricciones, y por lo tanto su manejo no es directo. En particular, los operadores de mutación o *crossover*, en el caso de GA y otros similares, ni la actualización de posición en el caso de PSO, entre otros, garantizan que las soluciones resultantes sean factibles, aún cuando los padres si lo sean. Debido a esto se realiza una revisión de las principales técnicas para el manejo de las restricciones, para determinar cuales serían las más adecuadas para el uso en problemas de control predictivo híbrido.

Antes de presentar las distintas estrategias, se formalizarán algunos conceptos necesarios para la compresión de los problemas con restricciones.

Se asumirá que el problema se encuentra dado por las variables $x_1,...,x_n$, cada una con su dominio $D_1,...,D_n$, pudiendo ser continuos o discretos (Eiben y Smith, 2003). Al espacio dado por el producto cartesiano de los dominios de cada variable $S = D_1 \times ... \times D_n$, se le llama espacio de búsqueda libre. La propiedad más importante de este espacio es que verificar la propiedad de pertenencia a este espacio puede ser realizada independientemente para cada variable, y luego se toma la conjunción de los resultados. En general los operadores típicos de mutación y crossover, y de actualización de posición en PSO garantizan que si los padres son factibles la descendencia

también lo será (y en el caso más complicado, basta truncar a la cota no satisfecha). Por lo tanto este tipo de restricción en el espacio de búsqueda no es considerado para el análisis.

Una restricción se entiende como la reducción de las posibles combinaciones de ciertas variables. Una restricción se puede traducir como una expresión lógica ϕ_j, en la cual si una solución x satisface la restricción *j*, entonces $\phi_j(x) = verdadero$. Un problema restringido puede poseer una o más restricciones, y una solución será factible si y sólo si la solución x satisface todas las restricciones, es decir $\phi_j(x) = verdadero, \forall\ j$. En caso contrario, la solución x no es factible.

Los problemas se pueden clasificar en tres tipos distintos. Los problemas de optimización libres (*Free optimisation problems*, FOP) son aquellos en que se optimiza una función objetivo en un espacio de búsqueda libre, sin estar sujeta a ningún tipo de restricciones. Los problemas de satisfacción de restricciones (*Constraint satisfaction prob*lem, CSP) son aquellos donde tan solo se busca una solución en un espacio de búsqueda libre que satisfaga un conjunto de restricciones. Finalmente, un problema de optimización restringido (*constrained optimization problem*, COP) es una combinación de un FOP y de un CSP, es decir, se debe encontrar una solución en el espacio de búsqueda libre donde esta sea factible y tenga un valor óptimo.

Con estos conceptos ya aclarados, se puede presentar las distintas estrategias de manejo de restricciones.

Se comienza entregando una clasificación más bien general de las distintas alternativas.

- Manejo indirecto de restricciones: las restricciones se convierten en objetivos de optimización, o se incorporan dentro de la función objetivo ya existente. Luego de esta transformación, las restricciones dejan de existir, y el problema resultante es un problema de optimización libre.
- Manejo directo de restricciones: las restricciones son forzadas de algún modo en forma explícita durante la ejecución del algoritmo.

Con esta distinción, se tiene que los enfoques indirectos son parte de la preparación del problema antes de ejecutar el algoritmo de optimización. En contraparte, los enfoques directos forman parte del algoritmo para satisfacer las restricciones.

Se debe manifestar que estas opciones no son exclusivas, pudiendo manejarse algunas restricciones de un modo directo y otras de un modo indirecto. En particular, para el modo directo, si bien se presentarán enfoques genéricos, algunas estrategias muy útiles consisten en la creación de operadores especiales que ayuden a satisfacer las restricciones.

Las estrategias más comunes para el manejo de restricciones son:

1. Uso de funciones de penalización para las soluciones infactibles. En el caso que la penalización sea proporcional al número de restricciones violadas o a la distancia de la zona factible, se habla de penalizaciones suaves. En el caso que se trate de penalizaciones de igual magnitud para cada solución infactible, se habla de penalizaciones duras.
2. Uso de mecanismos que "reparan" las soluciones infactibles, es decir, entregan una solución factible, que idealmente es la más cercana posible a la infactible.

3. El uso de una representación adecuada de las variables del problema, sumado a operadores adecuados, que garanticen que las soluciones generadas siempre sean factibles.

De acuerdo a estas estrategias, las alternativas se pueden clasificar del siguiente modo:

1. Manejo directo de restricciones. Coincide con el uso de penalizaciones para las soluciones infactibles.
2. Manejo indirecto de restricciones. Puede ser llevado a cabo de dos modos:
   a. Permitiendo la generación de soluciones infactibles, y luego repararlas.
   b. No permitiendo la generación de soluciones infactibles, preservando la factibilidad a través de representaciones y operadores adecuados.

En la práctica, es común utilizar el conocimiento específico del problema tanto como sea posible, en pos de reducir el tiempo gastado en la generación de soluciones infactibles. Los óptimos globales de un problema de optimización con restricciones con variables continuas en general yace en, o muy cerca, de las fronteras de las regiones factibles e infactibles, y que los algoritmos que buscan específicamente en ellas han reportado resultados prometedores. En esta revisión, por ser general, obviamente no se presenta este conocimiento específico, pero si se mantendrá en mente para las aplicaciones.

A continuación se presentan las principales estrategias utilizadas en la literatura de algoritmos evolutivos para la satisfacción de restricciones. Después, se presenta una revisión de las estrategias utilizadas en control predictivo y un análisis de cuales son o no recomendables para este tipo de aplicaciones.

5.6.2 Estrategias para el manejo de restricciones

5.6.2.1 Funciones de penalización

Al asumir un problema de minimización, como se realiza en el control predictivo, el uso de funciones de penalización consiste en una modificación de la función objetivo tal que (Eiben y Smith, 2003):

$$f'(x) = f(x) + P\big(d(x,F)\big) \tag{5.18}$$

donde F es la región factible, $d(x,F)$ es una métrica de la distancia de un punto infactible a la región factible (puede ser tan simple como un conteo del número de restricciones violadas, o una función lógica que indique si la solución es factible o no) y la función de penalización P es monótonamente no decreciente tal que $P(0) = 0$.

Esta definición asume que es posible evaluar un punto no factible. Sin embargo, en muchos problemas esto no es posible (por ejemplo en el problema de ruteo dinámico de vehículos presentado más adelante, en el capítulo 7). En estos casos se pueden aplicar penalizaciones que aseguren una función objetivo mayor a cualquier solución factible, como se propone en Deb (1998).

Este enfoque es conceptualmente muy simple, lo que ha facilitado su uso extendido, y son ideales para problemas con regiones factibles disjuntas o donde el óptimo global está cerca del límite de

las zonas factibles. Sin embargo, su éxito depende de un balance entre la exploración de zonas infactibles, y no perder tiempo en esta zonas, lo que da una gran importancia en la forma de la función de penalidad y la métrica de distancia. Si la función de penalización es muy severa, entonces la exploración de la región puede ser retrasada o incluso descartada. De igual modo, si la función de penalización es muy pequeña, las soluciones en zonas infactibles pueden dominar a aquellas en regiones factibles, sin permitir una adecuada búsqueda, pudiendo incluso converger a una solución infactible.

En general, para un sistema con m restricciones, la forma de la función de penalización es una suma ponderada:

$$P\left(d(x,F)\right) = \sum_{i=1}^{m} w_i d_i^{\kappa}(x) \tag{5.19}$$

donde κ es una constante definida por el usuario, que en general toma el valor 1 ó 2.

Los enfoques basados en funciones de penalización son clasificadas como constantes, estáticos, dinámicos o adaptativos.

Los enfoques estáticos consisten en el uso de pesos w_j constantes a lo largo de la ejecución del algoritmo. En este enfoque, se han utilizado comúnmente tres métodos: penalizaciones extintivas (donde todos los w_i se definen muy grandes para disminuir considerablemente las soluciones infactibles de la población), penalidades binarias (en las cuales el valor d_i es 1 si la restricción es violada, ó cero en el caso contrario), y las más comunes donde d_i es un operador de distancia (generalmente euclideana). En Goldberg (1989) se ha reportado que la mejor técnica de las tres es la última, pues es la que mejor refleja el grado de infactibilidad de las soluciones. A pesar de todo, el gran problema de estos enfoques es la dificultad de encontrar pesos adecuados, que en general son dependientes del problema deben ser encontrados a través de experimentación, que es altamente costoso en términos de tiempo.

Los otros dos enfoques, dinámicos y adaptativos, nacen para de algún modo evitar el problema de escoger pesos adecuados. Las funciones con penalizaciones dinámicas en general usan pesos dependientes del tiempo $w_i(t) = \left(w_{i0}t\right)^{\alpha}$, donde generalmente $\alpha = 1, 2$ entregan buenos resultados. En este enfoque aún se deben encontrar pesos iniciales adecuados, pero el comportamiento del algoritmo es mucho menos sensible a ellos, y por lo tanto la sintonización es bastante menos difícil que en el caso estático. Los enfoques adaptativos por su parte, utilizan pesos variables que varían a lo largo del tiempo, pero como función del estado de la población en la ejecución del algoritmo y no del tiempo en sí. Por ejemplo, se puede utilizar un incremento fijo Δw que se añade a los pesos correspondientes si en el mejor individuo la restricción correspondiente es violada (Bean y Hadj-Alouane, 1992).

Es importante notar que aparte de estos enfoques clásicos, pueden existir otros que sigan los mismos principios, pero que sean implementados de otro modo. Por ejemplo en Deb (1998) se propone un método de manejo de restricciones muy similar en principios al de penalizaciones. En ambos métodos la idea es que cualquier solución infactible tenga un fitness mayor que cualquier

solución factible. En el trabajo de Deb no se utiliza explícitamente una penalización, sino que, al realizar la selección por torneo, una solución factible siempre le gana a una infactible.

5.6.2.2 Funciones de reparación

Las funciones de reparación son utilizadas para tomar una solución infactible y generar desde ahí una factible, que idealmente, es la más cercana posible a la originalmente infactible. Los esquemas de reparación pueden funcionar en un nivel genotípico o en uno fenotípico. En este contexto, las distintas estrategias pueden clasificarse en los enfoques "Lamarckianos" y los "Baldwinianos". En la evolución Lamarckiana (Ackley y Littman, 1994), cuando se repara una solución infactible, la nueva solución reemplaza a la infactible en la población: en este sentido, esta reparación funciona a un nivel genotípico. En la evolución Baldwiniana (Hilton y Nowlan, 1987), las soluciones infactibles sólo son reparadas para fines de evaluación de la función objetivo, pues el fitness correspondiente es asignado a la solución infactible que se mantiene en la población, y en este sentido, la reparación actúa a nivel fenotípico.

Ha sido reportado que utilizar algoritmos de reparación determinístico entrega los mejores resultados, pues una reparación no determinística añadiría ruido a la evaluación de cada individuo. Sin embargo, algunos autores han encontrado que la adición de ruido puede ayudar a los algoritmos evolutivos a evitar la convergencia prematura. En la práctica, es probable que el mejor método dependa del problema, del tamaño de la población y de la presión selectiva.

Uno de los principales problemas de este enfoque es que, en general, encontrar una solución factible cercana no es sencillo, y es dependiente del problema en particular, pudiendo ser muchas veces tan difícil como el problema original. Un algoritmo genérico que sistematiza y simplifica este problema para optimización en dominios continuos es GENOCOP III (Michalewicz y Nazhiyath, 1995). Este algoritmo funciona manteniendo dos poblaciones, una P_s llamada "puntos de búsqueda", y otra P_r de puntos de referencia, en la cual todos los individuos son factibles. Los puntos en P_r y los puntos factibles en P_s son evaluados directamente. Cuando un punto infactible es generado en P_s este es reparado del siguiente modo: se escoge un punto en P_r y se dibuja una línea desde éste hasta el punto infactible. Luego se muestrea esta línea hasta que se encuentra una solución factible "reparada". Si el nuevo punto tiene un fitness superior al utilizado de P_r, entonces lo reemplaza en esta población. Además, el nuevo punto reemplaza al punto infactible en P_s con una probabilidad pequeña (lo que entrega el balance entre la búsqueda Lamarckiana y Baldwiniana). Vale la pena notar que se utilizan dos métodos distintos para escoger el punto de referencia utilizado en la reparación, siendo ambos estocásticos, de modo que la evaluación es necesariamente ruidosa.

5.6.2.3 Reducción de búsqueda al espacio factible

Un modo de satisfacer las restricciones es asegurando que a través de la representación y de los operadores evolutivos las soluciones generadas sean siempre factibles. Para esto en general se debe tener cuidado de ser capaz de representar todas las zonas factibles. Además, es deseable que cualquier solución sea alcanzable desde cualquier otra utilizando sólo el operador de mutación (una determinada cantidad de veces). En general esta estrategia es especialmente cómoda para trabajar problemas de permutaciones u otros problemas discretos.

Es importante notar que este enfoque, a pesar de ser muy atractivo, no puede ser aplicado sobre todos los tipos de restricciones. Generalmente es muy difícil encontrar o diseñar un operador que garantice que los descendientes sean factibles. Si bien una opción posible es simplemente descartar cualquier punto infactible y aplicar el operador repetidamente hasta que se genere una solución factible, el proceso de verificación de la factibilidad de una solución puede consumir un tiempo computacional importante. Sin embargo, para la larga clase de problemas donde el enfoque es válido, este procedimiento es altamente recomendable.

5.6.3 Consideraciones especiales para control predictivo

5.6.3.1 La ecuación de estado

En la formulación genérica del control predictivo híbrido, la ecuación de estado del sistema es incluida como una restricción de igualdad. De acuerdo a lo estudiado hasta aquí, esta restricción debiera ser tratada con alguna de las técnicas mencionadas. Esto, sin embargo, añadiría una gran complejidad al problema de optimización. Gracias a la estructura causal de los modelos, es posible encontrar todos los valores futuros de los estados y las salidas con la sola información de las acciones de control de la solución candidata. Si se tiene que:

$$\begin{aligned} x_{k+1} &= f(x_k, u_k) \\ y_k &= g(x_k, u_k) \end{aligned} \tag{5.20}$$

donde $x_k = x(t+k|t)$, $y_k = y(t+k|t)$, $u_k = u(t+k|t)$. Luego se puede iterar la expresión de la salida como sigue:

$$\begin{aligned} & y_k = g\left(x_k, u_k\right) \Rightarrow \\ & y_k = g\left(f\left(x_{k-1}, u_{k-1}\right), u_k\right) \Rightarrow \\ & y_k = g\left(f\left(f\left(x_{k-2}, u_{k-2}\right), u_{k-1}\right), u_k\right) \Rightarrow \\ & \vdots \\ & y_k = g\left(f\left(f\left(.....f\left(x_0, u_0\right)...., u_{k-2}\right), u_{k-1}\right), u_k\right) \end{aligned} \tag{5.21}$$

Por lo tanto, la ecuación de estado que se incluye como restricción de igualdad en la formulación del control predictivo híbrido desaparece, al integrarse a la función objetivo. De este modo, el problema de optimización se rescribe como:

$$\begin{aligned} \min_{u=\left(u_0,\ldots,u_{N_u-1}\right)} J = & \sum_{k=1}^{N_y} \left\| g\left(f\left(f\left(.....f\left(x_0, u_0\right)...., u_{k-2}\right), u_{k-1}\right), u_k\right) - y_{ref}(t+k) \right\|_{Q_y}^2 \\ & + \sum_{k=1}^{N_u} \left\| u_k - u_{k-1} \right\|_{Q_u}^2 \\ & s.a.\ \ c_1(y_k, x_k, u_k) \le 0 \\ & \quad\ \ c_2(y_k, x_k, u_k) = 0 \end{aligned} \tag{5.22}$$

Este sistema es claramente más sencillo, pues se tienen que manejar menos restricciones. Este esquema es equivalente al planteado por Yiqing *et al.* (2007). Las restricciones que permanecen son en general restricciones operativas (de desigualdad) y restricciones de región terminal (que pueden ser de igualdad o desigualdad).

5.6.3.2 Recomendaciones

Dependiendo del tipo de restricción se puede establecer ciertas recomendaciones que a priori parecen ser las más adecuadas. De acuerdo a lo analizado en esta sección, el enfoque basado en funciones de penalización es el más complejo de aplicar, debido a la sintonización de las penalidades, y por lo tanto es el que se debe considerar en última instancia. Los esquemas basados en reparación y representación factible, por su parte, son recomendables siempre y cuando sean fácilmente aplicables.

Según lo presentnado en el Anexo (específicamente Sección 12), las restricciones se pueden dividir en: restricciones operativas sobre la salida y/o estados, restricciones operativas sobre las acciones de control, y restricciones finales sobre la salida.

Véase primero las restricciones sobre las salidas y/o estados. Éstas dependen de las acciones de control, que son las soluciones candidatas. Las estrategias de reparación y las de representación factible requieren que sean fácilmente (en términos computacionales) para ser recomendables. Sin embargo, la dependencia de la salida y el estado de las acciones de control en general está dada por una función no lineal, y por lo tanto no es sencillo modificar la acción de control para que esté en el rango admisible, y menos encontrar una representación que asegure que la salida y estado estén en ese rango. En conclusión, para este tipo de restricciones sólo queda como alternativa las funciones de penalización.

Ahora se analizan las restricciones sobre las acciones de control. Como las soluciones candidatas son precisamente las acciones de control, es fácil modificarlas para que cumplan con las restricciones. Así mismo, dependiendo del caso, también pueden ser fácilmente representadas para cumplir las restricciones. Luego, las estrategias de reparación y de representación factible son recomendadas para las restricciones sobre las acciones de control, y por lo tanto las estrategias de penalización no se debieran utilizar. Sin embargo, si se aplican penalizaciones fijas (con valores más grandes que cualquier valor que pueda tomar una solución factible, en lugar de añadir términos de penalización a la función objetivo) cuando una acción de control no es factible, se puede omitir el cálculo de la función objetivo y eso disminuye el esfuerzo computacional del algoritmo. Por lo tanto, en esta situación no es tan claro que no convenga utilizar las estrategias de reparación, y podría ser conveniente probar las tres alternativas.

Finalmente las restricciones finales sobre la salida cumplen las mismas características que el primer tipo de restricciones, y por lo tanto deben ser manejadas mediante las estrategias penalización.

5.7 Aspectos relevantes para la implementación en tiempo real de controladores predictivos en base a algoritmos evolutivos

5.7.1 Reducción de la carga computacional para control predictivo

La aplicación de estrategias de control que conllevan un tiempo computacional relevante para encontrar la acción de control óptima, sobre todo cuando existe un tiempo acotado para aplicar las acciones de control, motiva el deseo de reducir esta carga computacional. En control predictivo existen algunas estrategias para lograr esto, y se presentan a continuación.

Mantener las acciones de control constantes por varios tiempos de muestreo

Veamos que el máximo tiempo de predicción T_{pred} depende de dos factores, del tiempo de muestro T_S, y del horizonte de predicción N_p. Luego, se tiene que $T_{pred} = T_S N_p$. Como en general el tiempo de muestreo no puede ser aumentado, la única forma de aumentar el tiempo de predicción es aumentando el horizonte de predicción. Esto, sin embargo, aumenta la complejidad computacional del problema.

Para superar esta limitación Karen *et al.* (2007) proponen que las acciones de control no cambien en todos los tiempos de muestreo, sino que en su lugar sólo en múltiplos de éste. Existen situaciones donde esto es recomendable, por ejemplo en sistemas donde el tiempo de muestreo es muy pequeño los actuadores pueden sobrecargarse si las acciones de control se actualizan en cada instante.

La propuesta consiste entonces en mantener las acciones de control constantes por Z tiempos de muestro. Con esta estrategia se puede reducir la dimensionalidad del vector de incógnitas en el problema de optimización por un factor Z, cuando el horizonte de predicción se mantiene constante, o bien, aumentar el horizonte de predicción por un factor Z manteniendo la dimensión de las incógnitas del problema.

Es relevante destacar que esta estrategia sirve para cualquier formulación de control predictivo, independiente de si se utilice algoritmos evolutivos o no.

Almacenamiento de soluciones

En los problemas discretos, cuando existe un número limitado de acciones de control, es muy probable que durante la ejecución del algoritmo evolutivo algunas soluciones candidatas aparezcan más de una vez. Basado en esto, si es posible almacenar los *fitness* asociados a estas soluciones, la evaluación de estas mismas soluciones puede ser evitada cuando vuelvan a aparecer. Esta estrategia brinda ahorros de tiempo siempre que el proceso de almacenamiento y búsqueda de las soluciones almacenadas sea computacionalmente eficiente, y considerablemente menor que el costo asociado a calcular su fitness, ya que cada vez que se fuera a evaluar una solución se debe verificar si ya ha sido evaluada o no.

5.7.2 Sintonía basada en análisis multi-objetivo para implementación eficiente de algoritmos evolutivos

El desempeño de los algoritmos evolutivos depende de diversos parámetros que pueden (y en general deben) ser sintonizados, como ya ha sido mencionado. En PSO estos son: el factor de inercia, el factor social, el factor cognitivo, el número de partículas y el número máximo de iteraciones. En el caso de algoritmos genéticos, los parámetros que pueden ser sintonizados son: tasa de crossover, tamaño de la población, número de generaciones y los parámetros de la estrategia de selección, por ejemplo, el número de los padres más adaptados (k -elitismo) que son seleccionados sin modificar para aparecer en la siguiente generación.

Varios estudios han sido realizados para encontrar las configuraciones de parámetros óptimos para las familias de algoritmos de GA y PSO. La mayor parte de estos trabajos tratan acerca de los factores de inercia, cognitivo y social en PSO, y de las tasas de mutación y crossover y de los parámetros relacionados con la presión selectiva en GA. Estos estudios en general concluyen con un conjunto de parámetros recomendados, reconociendo sin embargo que éstos de todos modos dependen del problema específico. A pesar de esto último se evita realizar una sintonía para cada problema particular debido al alto tiempo computacional requerido para esa labor, y en su lugar se utilizan conjuntos de parámetros recomendados o sintonizados manualmente (cuando los parámetros recomendados no parezcan funcionar bien).

Con respecto de los parámetros restantes, el tamaño de la población y el número de generaciones (como son llamadas en la literatura de GA; los nombres equivalentes en la literatura de PSO son el número de partículas y máximo número de iteraciones. Para unificar notación, en el resto del trabajo cuando se hable indistintamente de PSO y GA, se utilizará tamaño de población y número de generaciones), no existen muchos trabajos relevantes para su sintonización, y menos que sean relevantes para los propósitos aquí establecidos. La razón fundamental es que la naturaleza de la mayor parte de los problemas analizados en la literatura de computación evolutiva es muy distinta a la naturaleza de los problemas analizados en esta tesis. En general se analiza problemas estáticos, que se resuelven en modo *offline*, y por lo tanto no existe un requerimiento estricto respecto del máximo tiempo aceptable para poder resolver dichos problemas. Luego, el número de generaciones se escoge de modo que sea lo suficientemente grande para permitir que el algoritmo converja, y el tamaño de la población es escogido de modo que sea lo suficientemente grande para permitir una búsqueda global adecuada, sin hacer al algoritmo excesivamente lento. Por otra parte, los problemas que se analizan en este trabajo, son dinámicos y en tiempo real, por lo que la carga computacional es muy relevante. En general, por la complejidad de los sistemas (y por lo tanto de sus modelos) no hay tiempo suficiente para asegurar que los algoritmos converjan, y entonces el tamaño de la población y el número de generaciones, que están directamente relacionadas con el número de evaluaciones de funciones objetivo (y por lo tanto del tiempo computacional), son críticos para el desempeño del algoritmo.

En esta sección se propone una metodología para sintonizar los tamaños de poblaciones y el número de generaciones, por medio de un enfoque multi-objetivo. El objetivo de este enfoque es encontrar el mejor conjunto de estos parámetros en términos de la calidad de las soluciones y el tiempo computacional requerido para encontrarlas. En problemas de control en tiempo real el tiempo de cómputos es muy importante pues se requiere que las acciones de control sean determinadas tan rápidamente como sea posible (y como cota máxima antes del siguiente instante

de muestreo), y que sean de buena calidad. Como estos dos aspectos son claramente objetivos opuestos, el enfoque multi-objetivo parece ser lo más adecuado para encontrar estos parámetros.

El problema de optimización multi-objetivo que se resuelve es el siguiente:

$$\begin{aligned} &\min_{n,g} \left\{ t_{n,g}, JP_{n,g} \right\} \\ &s.a.\ (n,g) \in P_S \end{aligned} \tag{5.23}$$

donde n es el tamaño de la población, g es el número de generaciones, P_S es el conjunto donde se está buscando los valores óptimos de n y g. $t_{n,g}$ es el promedio del tiempo computacional requerido para resolver el problema de optimización en cada tiempo de muestreo, utilizando una población de tamaño n y un número de generaciones g. $JP_{n,g}$ es el promedio del desempeño del sistema, que se mide como la suma de los errores de seguimiento y cambios en la acción de control sobre toda la simulación, utilizando una población de tamaño n y un número de generaciones g.

Dada la naturaleza estocástica de los algoritmos, y de las perturbaciones de los procesos, se realiza un número lo más alto posible de réplicas con cada combinación de parámetros analizada, para encontrar los promedios del desempeño y del tiempo computacional.

La solución del problema de optimización multi-objetivo es una región llamada conjunto Pareto-óptimo. Para formalizar las nociones previas, es importante definir los siguientes conceptos.

Consideremos $X^i = \left\{ n^i, g^i \right\}$, donde n^i y g^i definen una determinada combinación de los parámetros tamaño de población y número de generaciones. Un punto X^i domina en términos de Pareto a otro punto X^j sí y sólo sí:

$$\left(JP_{n^i,g^i} \le JP_{n^j,g^j} \right) \wedge \left(t_{n^i,g^i} < t_{n^j,g^j} \right) \text{ ó } \left(t_{n^i,g^i} \le t_{n^j,g^j} \right) \wedge \left(JP_{n^i,g^i} < JP_{n^j,g^j} \right). \tag{5.24}$$

Una solución X^i se dice Pareto-óptima si y sólo sí no existe otro punto X^j que domine en términos de Pareto a X^i. El conjunto de todos los valores de las funciones objetivo a minimizar (en este caso $t_{n,g}$ y $JP_{n,g}$) que corresponden a puntos Pareto-óptimos en P_S es $P_F = \left\{ \left(t_{n^i,g^i}, JP_{n^i,g^i} \right) : X^i \in P_S \right\}$. P_F es conocido como la frontera de Pareto. En la Figura 20 se muestra un ejemplo gráfico del problema de optimización multi-objetivo aquí presentado. Cualquier solución $X^i = \left\{ n^i, g^i \right\}$ que pertenece al conjunto Pareto-óptimo no es dominado por ningún otro $X^j = \left\{ n^j, g^j \right\}$. En el ejemplo $X^1 = \left\{ n^1, g^1 \right\}$ es dominado por $X^2 = \left\{ n^2, g^2 \right\}$, y por lo tanto no pertenece al conjunto Pareto-óptimo.

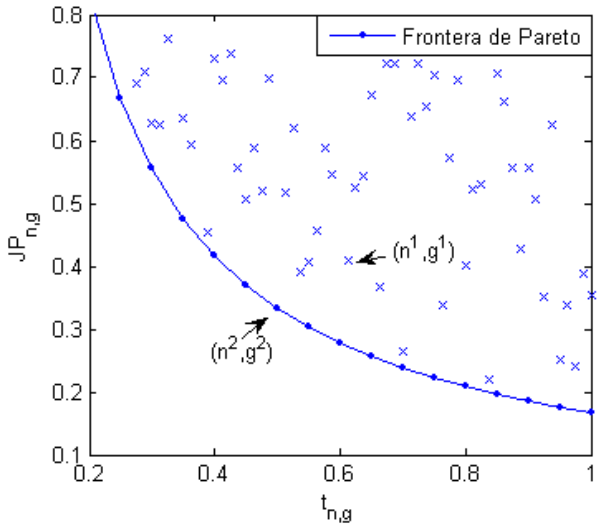


Figura 20. Optimización multi-objetivo para minimización del desempeño del sistema y del tiempo computacional

Es importante notar que el número de evaluación de funciones (que como una primera aproximación, está dado por $(n \cdot g)$) no es el único factor que incide en la calidad de las soluciones. Ellas también dependen de las dinámicas de los algoritmos, por lo tanto el modo en el que se distribuyan estas evaluaciones es muy relevante. De hecho, poblaciones de tamaño grande son utilizadas para favorecer la búsqueda global, mientras que un mayor número de generaciones sirve para encontrar soluciones más refinadas. Luego, pueden existir combinaciones de tamaños de población y número de generaciones que evaluando menos funciones objetivo pueden alcanzar mejor soluciones que otras que evalúan más soluciones candidatas. Las primeras son configuraciones óptimas del conjunto Pareto-óptimo (siempre que a su vez no haya otras combinaciones de parámetros que permitan resolver el mismo problema más rápido y encontrando mejores soluciones), y las segundas son soluciones dominadas. Entonces, con el enfoque multi-objetivo es posible encontrar estas configuraciones óptimas que permiten encontrar mejores soluciones en el mejor tiempo computacional posible.

Una vez que se ha encontrado el conjunto Pareto-óptimo, una sola combinación debe ser escogida de acuerdo a algún criterio para ser la solución del problema. Un criterio natural es escoger la combinación de tamaño de población y número de generaciones que alcance las soluciones de mejor calidad en un tiempo permitido, que puede ser por ejemplo el tiempo de muestreo del sistema o algún otro.

Este análisis puede ser generalizable para cualquier tipo de problemas en tiempo real, donde se deba optimizar una función objetivo en un tiempo limitado, y por lo tanto existe un compromiso entre la cantidad de tiempo disponible para encontrar las soluciones y la calidad de éstas.

5.7.3 Análisis multi-objetivo para comparación de métodos

Aparte de ser útil para comparar sintonizar el número de iteraciones y el tamaño de la población, la metodología puede ser extendida para comparar distintos métodos. Para entender esto se definen los siguientes conceptos:

Se dice que un método M_1 domina a M_2 si para todo punto $\left(t^2_{n^i,g^i}, JP^2_{n^i,g^i}\right)$ que pertenece a la frontera de Pareto P^2_F del método M_2 existe otro punto $\left(t^1_{n^j,g^j}, JP^1_{n^j,g^j}\right)$ que pertenece a la frontera de Pareto P^1_F del método M_1 tal que $\left(t^1_{n^j,g^j}, JP^1_{n^j,g^j}\right)$ domina en términos de Pareto a $\left(t^2_{n^i,g^i}, JP^2_{n^i,g^i}\right)$.

Se dice que un método M_1 domina a M_2 en el intervalo $T=[t_1,t_2]$, si para todo punto $\left(t^2_{n^i,g^i}, JP^2_{n^i,g^i}\right)$ que pertenece a la frontera de Pareto P^2_F del método M_2, donde $t_1 \le t^2_{n^i,g^i} \le t_2$, existe otro punto $\left(t^1_{n^j,g^j}, JP^1_{n^j,g^j}\right)$ que pertenece a la frontera de Pareto P^1_F del método M_1, donde $t_1 \le t^1_{n^i,g^i} \le t_2$, tal que $\left(t^1_{n^j,g^j}, JP^1_{n^j,g^j}\right)$ domina en términos de Pareto a $\left(t^2_{n^i,g^i}, JP^2_{n^i,g^i}\right)$.

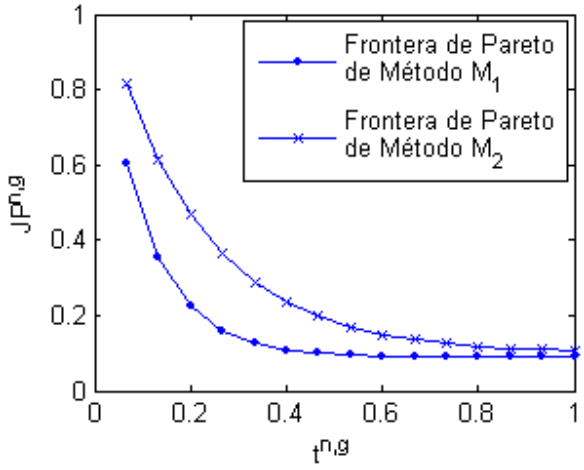


Figura 21. Comparación de métodos en base a análisis multi-objetivo. Método M_1 domina a método M_2.

La situación donde un método domina a otro método se presenta en la Figura 21. De acuerdo a la definición anterior, se tiene que el método M_1 domina al método M_2, pues para cada punto del método M_2 existe al menos un punto del método 1 que encuentra mejores soluciones en un menor o igual tiempo computacional, es decir, que lo domina en términos de Pareto. Si un método domina a otro, es claro que es un mejor método, pues como ya se ha dicho, encuentra mejores soluciones en un tiempo menor.

La situación donde un método domina a otro en un intervalo de tiempo computacional se presenta en la Figura 22. Es posible apreciar que el método M_1 no domina al método M_2, ni vice-versa. Sin embargo, si sólo se consideran las combinaciones de tamaño de población e iteraciones tal que el tiempo computacional sea menor que 0.3 seg., el método M_1 domina al método M_2. Del mismo modo, si sólo se consideran aquellas combinaciones tales que el tiempo computacional es mayor que 0.3 seg., el método M_2 domina al método M_1. Si un método domina a otro en un intervalo no se puede afirmar que sea mejor que el otro, pudiendo sólo afirmar que este método es mejor que el otro cuando la solución debe ser encontrada hasta un tiempo máximo que caiga

dentro del intervalo mencionado. En este ejemplo entonces, el método M_1 es mejor que el M_2 si la solución debe ser encontrada en hasta 0.3 segundos, y el método 2 es mejor si la solución puede ser encontrada en más de 0.3 segundos.

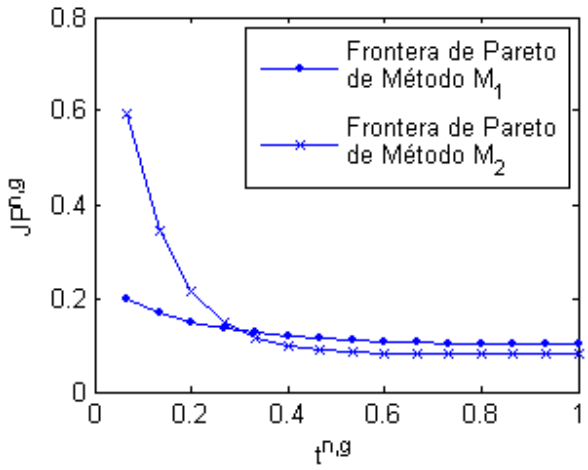


Figura 22. Comparación de métodos en base a análisis multi-objetivo. Método M_1 domina a método M_2 para $t^{n,g} < 0.3$, y método 2 domina a método 1 para $t^{n,g} \geq 0.3$

5.8 Resumen

El capítulo comienza con una revisión bibliográfica de los esfuerzos para aplicar algoritmos evolutivos en control predictivo, y a partir de esta revisión y de las discusiones en capítulos anteriores, se presenta algunos de los aspectos que se ha determinado son más importantes para la aplicación de algoritmos evolutivos en control predictivo híbrido de sistemas no lineales. Simultáneamente, se plantean diversas estrategias para estos aspectos relevantes en base a la aplicación particular trabajada en esa tesis.

Los aspectos que se han considerado en este estudio son: representación de soluciones, inicialización de soluciones candidatas, estrategias para el manejo de restricciones y estrategias para la aplicación en tiempo real. Para cada uno de estos aspectos se entregan diversas alternativas cuya eficacia en general dependen del tipo de problema, y se espera poder determinar tendencias que indiquen que clase de estrategias son recomendables según el tipo de problemas (al menos los similares a los analizados), o si son recomendables para todos los problemas. Luego, en el capítulo siguiente se probará y analizará estas alternativas para poder establecer cuales son las ideales para los procesos estudiados, y así poder establecer recomendaciones para otros problemas similares.

Dentro de las estrategias para la aplicación en tiempo real, se propone una metodología de sintonización basada en optimización multi-objetivo, que permite escoger los parámetros de número de iteraciones y tamaño de población de acuerdo a criterios de calidad de las soluciones y esfuerzo computacional. Este tipo de análisis además, permite comparar distintas estrategias y funciona a modo de un indicador gráfico para la bondad de un método o estrategia. Este indicador es entonces utilizado para evaluar las diversas estrategias propuestas para los aspectos ya mencionados.

Si bien es cierto que en este trabajo se abarca gran parte de los aspectos relevantes para la aplicación de algoritmos evolutivos para control predictivo híbrido, existen otros que no han sido considerados en este estudio. Por ejemplo se ha dejado de lado el tratamiento de temas de estabilidad de los sistemas, y tampoco se ha considerado el análisis de estrategias mixtas entre algoritmos convencionales y algoritmos evolutivos. Reconociendo que son temas importantes, se plantean como trabajo futuro para la construcción de un marco teórico sólido destinado a la aplicación de algoritmos evolutivos en control predictivo híbrido de sistemas no lineales.

6 Caso de estudio: El reactor batch

Para probar las diversas alternativas basadas en algoritmos evolutivos para el control predictivo, se considera como primer caso de estudio el simulador de un reactor batch instalado en una compañía farmacéutica eslovena (cuyo nombre no puede ser revelado por motivos de confidencialidad), y que es utilizado para la fabricación de remedios (Karer *et al*., 2007). En este proceso el objetivo es controlar la temperatura de los ingredientes mezclados en el núcleo del reactor donde se sintetiza el producto final. Para lograr esto, la temperatura debe seguir una trayectoria de referencia predeterminada, de forma tan precisa como sea posible.

Un esquema del reactor batch es presentado en la Figura 23. El núcleo del reactor (a temperatura T), es calentado o enfriado a través de la chaqueta de agua del reactor (a temperatura T_w). A través de la chaqueta circula una mezcla de agua fresca que entra al reactor a través de válvulas *on/off*, y agua de recirculación que ya ha sido utilizada para enfriar o calentar el núcleo del reactor. El agua es bombeada hacia la chaqueta de aguda con un flujo constante Φ. La dinámica del sistema depende de las propiedades físicas del reactor batch, es decir, de la conductividad térmica λ, el área de contacto S; de la masa m y la capacidad específica c de los ingredientes del núcleo y la chaqueta del reactor; y la temperatura ambiente T_0.

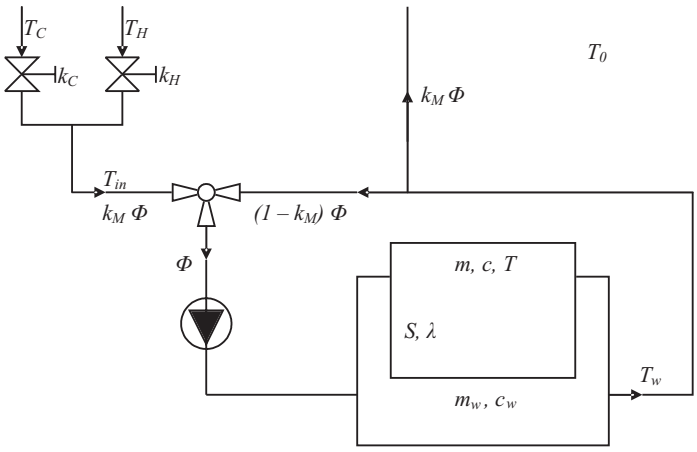


Figura 23. **Diagrama del reactor batch**

Para este estudio se considera un solo reactor batch, pero con dos modelos que consideran distintas entradas al proceso. En un primer modelo el reactor batch sólo posee entradas discretas, caso que permitirá evaluar el uso de algoritmos evolutivos para control predictivo con entradas únicamente discretas. En el segundo modelo el reactor batch posee dos entradas discretas y una continua, caso que permitirá evaluar el uso de algoritmos evolutivos para control predictivo híbrido con entradas continuas y discretas.

6.1 El reactor batch con entradas discretas

La temperatura del agua fresca de entrada T_{in} depende de dos entradas: las posiciones de las válvulas *on*/*off* k_C y k_H, y de la válvula de recirculación discreta k_M. Sin embargo, sólo existen

dos modos de operación. En el primero, definido por $k_C = 1$ y $k_H = 0$, el agua de entrada es helada ($T_{in} = T_C = 12^{\circ}C$), mientras que en el segundo caso, definido por $k_C = 0$ y $k_H = 1$, el agua de entrada es caliente ($T_{in} = T_C = 75^{\circ}C$).

La tasa de la mezcla entre agua fresca con agua de recirculación es controlada por una tercera entrada, la posición de una válvula de mezcla k_M. Existen 6 posibles tasas que pueden ser ajustadas por la válvula de mezcla. La tasa de agua fresca puede ser 0, 0.01, 0.02, 0.05, 0.1 ó 1. Este entonces se trata de un sistema multi-variable con tres entradas discretas (k_M, k_H y k_C), y dos salidas medibles (T y T_w).

Previamente, sobre este sistema se ha aplicado exitosamente control predictivo mediante una estrategia basada en un análisis de alcanzabilidad y *branch and bound* (Karer *et al*., 2007a,b) y mediante el uso de GA (Causa *et al*., 2008). En este trabajo, se propone nuevas estrategias basadas en PSO y además se estudian ciertos aspectos de la implementación basada en GA y las nuevas basadas en PSO, de acuerdo a las propuestas realizadas en este trabajo respecto de la aplicación de algoritmos evolutivos para control predictivo. En todos estos trabajos se utiliza la misma función objetivo y el mismo modelo difuso que son propuestos por Karer *et al*. (2007a).

A continuación se presenta el modelo híbrido difuso para la estrategia de control predictivo. Luego se presenta la estrategia de control predictivo, basada tanto en PSO como en GA, con especial énfasis en el diseño de las representaciones, inicialización de soluciones y estrategias de manejo de restricciones. Posteriormente se presentan los estudios por simulación, donde se comparan los métodos y se sintonizan mediante el análisis multi-objetivo propuesto en 5.7.2. Finalmente se presenta un análisis y conclusiones de los resultados obtenidos.

6.1.1 Modelo híbrido difuso del reactor batch

El procedimiento para el modelamiento del proceso es explicado en detalle en Karer *et al.* (2007a). La temperatura del núcleo del reactor T es influenciada sólo por la conducción calorífica entre el núcleo y la chaqueta de agua. Además, se asume que la conducción de calor es proporcional a la diferencia entras las temperaturas del núcleo y de la chaqueta de agua.

Luego, se considera un subsistema MISO lineal de primer orden, tal como se presenta en (6.1). Los parámetros del sistema se presentan en (6.2).

$$\hat{T}(k+1) = \Theta_C^T \left[T_w(k) \quad T(k) \right]^T \tag{6.1}$$

$$\Theta_C^T = \left[0.0033 \quad 0.9967 \right] \tag{6.2}$$

La temperatura de la chaqueta de agua, por su parte, es influenciada por la temperatura del núcleo T, y las posiciones de las válvulas *on/off* k_C y k_H, y la válvula de mezcla k_M. Se asumen dos modos de operación para el subsistema de la chaqueta de agua, dadas las posiciones de las válvulas *on*/*off*.

- El primer modo de operación ($q = 1$), corresponde al caso cuando el agua de entrada es caliente. Es decir $k_C = 0$ y $k_H = 1$.

- El segundo modo de operación ($q = 2$), corresponde al caso cuando el agua de entrada es fría. Es decir $k_C = 1$ y $k_H = 0$.

Luego, la ecuación de estado discreta es:

$$q(k) = q\left(k_C(k), k_H(k)\right) = \begin{cases} 1 & if \quad k_C(k) = 0 \wedge k_H(k) = 1 \\ 2 & if \quad k_C(k) = 1 \wedge k_H(k) = 0 \end{cases} \tag{6.3}$$

Luego, se definen las funciones de partencia del modelo difuso. El sistema se fuzzifica respecto de la temperatura del agua en la chaqueta del reactor $T_w(k)$. Se utilizan funciones triangulares simples, como se muestran en la Figura 24.

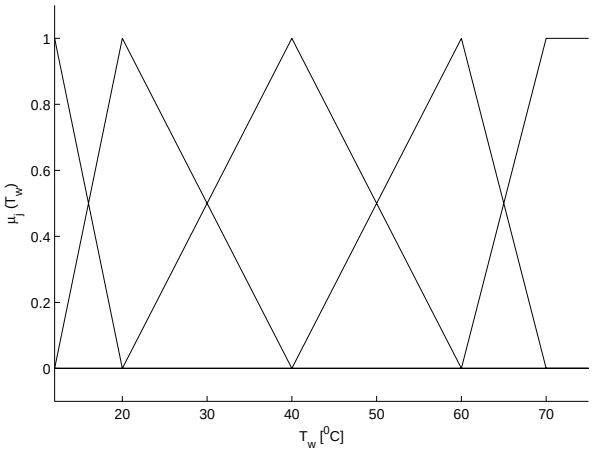


Figura 24. Funciones de pertenencia del modelo difuso del reactor batch.

Tal forma de funciones de pertenencia asegura que los grados de activación normalizados $\beta_j(T_w)$ son iguales los grados de pertenencia $\mu_j(T_w)$ a lo largo de todo el rango de operación de cada regla R^{jd}, respectivamente. Los grados normalizados de activación $\beta_j(T_w)$ entonces generan un vector normalizado de activación $\beta\left(T_w(k)\right) = \beta(k)$. En este caso se consideran 5 funciones de pertenencia ($K = 5$), con máximos a los 12º, 20º, 40º, 60º y 70º, de modo que se cubre todo el rango de operación.

La base de reglas del modelo híbrido difuso se presenta en (6.4). Se asume que cada sistema local correspondiente a cada regla R^{jd} es lineal afín.

$$\begin{aligned} &R^{jd}: \text{if} \quad q(k) \text{ is } Q_d \quad \text{y} \quad T_w(k) \text{ is } A_1^j \quad \text{then} \\ &T_w(k+1) = a_{1\,jd} T_w(k) + a_{2\,jd} T(k) + b_{1\,jd} k_M(k) + r_{jd} \\ &\text{for } j = 1,...5, \text{ and } d = 1,2. \end{aligned} \tag{6.4}$$

La salida del modelo de la temperatura en la chaqueta de agua del reactor se presenta en forma compacta en (6.5).

$$T_w(k+1) = \beta(k)\Theta_w^T(k)\begin{bmatrix} T_w(k) & T(k) & k_M(k) & 1 \end{bmatrix}^T \tag{6.5}$$

$$\Theta_w(k)=\begin{cases}\Theta_{w1} & si \;\; q(k)=1\\ \Theta_{w2} & si \;\; q(k)=2\end{cases}$$

donde Θ_{w1}, Θ_{w2} son matrices que se obtienen al proceso de identificación.

6.1.2 Metodología de solución basada en algoritmos evolutivos

Se propone una estrategia de control predictivo híbrido, basado en la función objetivo propuesta en Karer *et al.* (2007a).

$$J = w_1\sum_{h=1}^{N_y}\left(T(k+h)-T_{ref}(k+h)\right)^2 + w_2\sum_{h=1}^{N_u}\left|\Delta k_M(k+h-1)\right|k_H(k+h-1)+ \\ w_3\sum_{h=1}^{N_u}k_C(k+h)k_H(k+h-1) \tag{6.6}$$

donde $w_1 = 1/15,\ w_2 = 0.03, w_3 = 15$.

En este estudio se considera un horizonte de predicción igual al de control $N_u = N_y = N$. El tiempo de muestreo del modelo de predicción es $T_s = 10\,[s]$. Debido a esto, el problema de optimización debe ser resuelto en 10 [s]. Notar que las entradas sólo pueden cambiar cada 15 tiempos de muestro. Esto significa que la acción de control será mantenida constante durante un tiempo de muestreo de control $T_{cs} = 150\,[s]$.

Como se trata de problemas de minimización, en la aplicación de PSO y GA para resolver este problema, se utiliza como función de *fitness*:

$$F = \frac{1}{J} \tag{6.7}$$

Esta definición asegura que las mejores soluciones, es decir aquellas con una función objetivo menor, posean un mayor *fitness*.

El conjunto de acciones de control posibles $u(k+h-1)$, $h = 1,..., N_u$, se presenta a continuación:

$$M = \left\{\begin{bmatrix}0\\0\\1\end{bmatrix},\begin{bmatrix}0.01\\0\\1\end{bmatrix},\begin{bmatrix}0.02\\0\\1\end{bmatrix},\begin{bmatrix}0.05\\0\\1\end{bmatrix},\begin{bmatrix}0.1\\0\\1\end{bmatrix},\begin{bmatrix}1\\0\\1\end{bmatrix},\begin{bmatrix}1\\1\\0\end{bmatrix}\right\} \tag{6.8}$$

- La primera fila denota la válvula de mezcla $k_M \in \{0, 0.01, 0.02, 0.05, 1\}$.
- La segunda fila es la válvula *on/off* de agua fría $k_C \in \{0, 1\}$.
- La tercera fila denota la válvula *on/off* de agua caliente $k_H \in \{0, 1\}$.

Esto se puede interpretar como un conjunto de restricciones sobre las acciones de control: k_c y k_H deben tomar valores distintos en $\{0,1\}$; y k_M sólo puede tomar los valores $\{0, 0.01, 0.02, 0.05, 0.1\}$ si $k_H = 1$ y $k_C = 0$.

Existen además dos tipos de perturbaciones que afectan la salida de reactores batch en aplicaciones en la vida real. Una perturbación de adición a la temperatura del núcleo del reactor T, y otra de adición a la temperatura en la chaqueta de agua del reactor T_w. Debido a las dinámicas más rápidas de la temperatura de la chaqueta de agua, es mucho más fácil para el algoritmo de control lidiar con estas perturbaciones, por lo tanto en los experimentos se considera sólo las perturbaciones en la temperatura del núcleo.

A continuación se presenta el diseño de controladores basados en algoritmos evolutivos, en particular GA y PSO, en base a los aspectos presentados en el capítulo 5.

6.1.2.1 Diseño del controlador predictivo híbrido basado en GA

El algoritmo genético corresponde al desarrollado en Causa *et al*. (2008). Sin embargo, aquí se presenta con un énfasis distinto, de acuerdo a los aspectos introducidos en el capítulo 5.

Representación de las soluciones y operadores evolutivos

Sólo existen 7 acciones de control posibles entre las combinaciones de las 3 válvulas, como se muestra en la ecuación (6.8). Por lo tanto, se propone una representación que asocia un número a cada una de las 7 acciones de control posibles.

$$0=\begin{bmatrix}0\\0\\1\end{bmatrix}, 1=\begin{bmatrix}0.01\\0\\1\end{bmatrix}, 2=\begin{bmatrix}0.02\\0\\1\end{bmatrix}, 3=\begin{bmatrix}0.05\\0\\1\end{bmatrix}, 4=\begin{bmatrix}0.1\\0\\1\end{bmatrix}, 5=\begin{bmatrix}1\\0\\1\end{bmatrix}, 6=\begin{bmatrix}1\\1\\0\end{bmatrix} \tag{6.9}$$

Luego, para representar una solución candidata completa del problema de optimización, es decir, la secuencia de acciones de control durante el horizonte de predicción N_u, se utiliza un individuo de N_u coordenadas, donde:

$$\begin{aligned} x &= \left(x_1, x_2, \ldots x_{N_u}\right), \quad x_i \in \{0,\ldots,6\} \\ u(k+h-1) &= x_h, \quad h = 1,\ldots,N_u, \end{aligned} \tag{6.10}$$

Para realizar la búsqueda se utilizan operadores de mutación y *crossover*. El operador de mutación utilizado consiste en que con una probabilidad muy pequeña p_m cada coordenada de cada individuo se cambia aleatoriamente a cualquier otra acción de control posible con una probabilidad uniforme para cada alternativa. El operador de recombinación utilizado es el *crossover* de 1 punto, que se aplica para pares de individuos con una probabilidad p_c. Los valores utilizados en esta implementación son $p_m = 0.001$ y $p_c = 0.8$.

Para seleccionar los individuos para cada nueva iteración se utiliza selección universal estocástica basada en *fitness* por ranking con un máximo valor $\eta^+ = 1.2$, de acuerdo a la ecuación (4.9) de la Sección 4.2.2.3.3.

De acuerdo a esta representación y operadores las soluciones generadas son siempre factibles.

Manejo de restricciones

El manejo de restricciones se logra en este algoritmo mediante la estrategia representación factible. En efecto, la representación que enumera cada una de las acciones de control factible, y por lo tanto, las soluciones candidatas generadas siempre serán factibles.

6.1.2.2 Diseño del controlador predictivo híbrido basado en PSO

De acuerdo a lo presentado en el capítulo 5, los aspectos a considerar para la aplicación de PSO al control predictivo híbrido del reactor batch son: la representación de soluciones, manejo de restricciones e inicialización de soluciones candidatas.

Representación de las soluciones

Las decisiones de control en el MPC son las posiciones de dos válvulas a lo largo de todo el horizonte de control N_u. Previamente se ha mencionado que de las tres válvulas que participan en el proceso, dos de ellas siempre toman valores opuestos, y por lo tanto, pueden ser consideradas como una sola válvula o estado discreto $q(k)$ como se presenta en la ecuación (6.3). Si las válvulas *on/off* toman los valores $(k_C, k_H) = (0,1)$, la válvula *on/off* equivalente o estado discreto es $q(k) = 1$, y si toman los valores $(k_C, k_H) = (1,0)$, el válvula equivalente *on/off* es $q(k) = 2$. Con esto en cuenta, se propone la primera alternativa para representar las acciones de control durante el horizonte de predicción.

Opción 1:

La representación propuesta utiliza partículas x_i de dimensión $2N_u$, que codifican las acciones de control durante los siguientes N_u instantes de predicción.

$$x_j = \{u(k), ..., u(k+N_u-1)\}. \qquad (6.11)$$

donde $u(k+h-1) = (k_M(k+h-1), q(k+h-1))$, $h = 1,..., N_u$. La válvula *on/off* puede tomar valores que $\in \{0,1\}$, mientras que los valores de la válvula de mezcla pertenecen al conjunto $\{0, 0.01, 0.02, 0.05, 0.1, 1\}$. Se utiliza partículas continuas en el intervalo [1,2], que se divide de manera uniforme de acuerdo a los valores que puede tomar la válvula. Luego, la codificación para cada válvula es:

Válvula *on*/*off*:

$$q(k) = \begin{cases} 1 & if\ x < 1.5 \\ 2 & if\ x \geq 1.5 \end{cases} \tag{6.12}$$

Válvula de mezcla:

$$k_M = \begin{cases} 0 & if\ x < 0.166 \\ 0.01 & if\ 0.166 \leq x < 0.333 \\ 0.02 & if\ 0.333 \leq x < 0.5 \\ 0.05 & if\ 0.5 \leq x < 0.666 \\ 0.1 & if\ 0.666 \leq x < 0.833 \\ 1 & if\ 0.833 \leq x < 1 \end{cases} \tag{6.13}$$

Para utilizar esta representación se utilizan los mismos operadores clásicos de PSO como son descritos en 4.4.2, más las modificaciones que correspondan para el manejo de restricciones.

Además, para que la heurística funcione de un modo adecuado, los *gbest* y *pbest* se asignan en el centro del intervalo al que corresponde la solución. Por ejemplo, si corresponde actualizar gbest, y la mejor solución corresponde a $q = 1$ y $k_M = 0.1$, $gbest = (1.25, 0.75)$.

Esta representación no garantiza que las soluciones que se generen sean factibles, pues pueden violar las restricciones sobre las acciones de control. Por lo tanto, se requieren estrategias de reparación que se presentan más adelante. Por lo mismo, los métodos que utilicen esta representación se denotan por PSO-R (por requerir una estrategia de reparación).

Opción 2:

La segunda alternativa para representar las soluciones está basada en la misma representación utilizada en GA, pero dado que PSO es un algoritmo inherentemente continuo, se trabaja con partículas continuas y tal como en el caso anterior, se trunca para evaluar la función objetivo.

Se trabaja con partículas x de dimensión N_u, donde cada coordenada x_i puede tomar valores en el intervalo $[-0.5, 6.5]$. Las partículas se mueven libremente en el espacio continuo definido por este intervalo y a las ecuaciones del modelo canónico de PSO (como son definidas en 4.4.2), pero en el momento de evaluar las funciones objetivo se truncan los valores del siguiente modo:

$$u(k+h-1) = \begin{cases} 0 & si\ x_h < 0.5 \\ 1 & si\ 0.5 \leq x_h < 1.5 \\ 2 & if\ 1.5 \leq x_h < 2.5 \\ 3 & if\ 2.5 \leq x_h < 3.5 \\ 4 & if\ 3.5 \leq x_h < 4.5 \\ 5 & if\ 4.5 \leq x_h < 5.5 \\ 6 & if\ 5.5 \leq x_h < 6.5 \end{cases} \quad , \ h = 1, \ldots, N_u \tag{6.14}$$

donde se utiliza la definifición de (6.9) para los valores de $u(k+h-1)$.

Respecto de la asignación de *gbest* y *pbest*, como en este caso los centros de los intervalos corresponden explícitamente al índice que representa a la acción de control en un instante, se utiliza el mismo índice para almacenar en gbest y pbest. Por ejemplo, si corresponde actualizar gbest con la solución $k_H = 1, k_C = 0, k_M = 0.02$, como ésta se encuentra representada por el índice 2, este valor toma la coordenada correspondiente en *gbest*.

Al analizar esta representación, que funciona bastante bien en GA, se desprende que podría funcionar no tan bien en PSO. Para notar esto, en la Figura 25 se presentan los efectos físicos de las acciones de control:

$$\underbrace{0 = \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}}_{\text{Sólo recirculación}}, \underbrace{1 = \begin{bmatrix} 0.01 \\ 0 \\ 1 \end{bmatrix}}_{\text{Agua caliente y mucha recirculación}}, \underbrace{2 = \begin{bmatrix} 0.02 \\ 0 \\ 1 \end{bmatrix}}_{\cdots}, 3 = \begin{bmatrix} 0.05 \\ 0 \\ 1 \end{bmatrix}, \underbrace{4 = \begin{bmatrix} 0.1 \\ 0 \\ 1 \end{bmatrix}}_{\text{Agua caliente y menos recirculación}}, \underbrace{5 = \begin{bmatrix} 1 \\ 0 \\ 1 \end{bmatrix}}_{\text{Agua caliente}}, \underbrace{6 = \begin{bmatrix} 1 \\ 1 \\ 0 \end{bmatrix}}_{\text{Sólo agua fría}}$$

Figura 25: interpretación física de las acciones de control.

De acuerdo a esto podemos ver que la acción "0" corresponde a una entrada tibia, calentándose progresivamente, hasta llegar a la entrada "5" que es la más caliente posible. Finalmente, la entrada "6", que es pura agua fría, es la entrada más helada dentro de las posibles. Se puede notar entonces que el espacio de búsqueda no es el ideal, ya que la acción de control que más enfría al sistema está al lado de la que más lo calienta. Esto afecta a la heurística de PSO, y no a la de GA, pues en la implementación de la primera las partículas viajan en un espacio continuo, y así un movimiento pequeño en el espacio continuo produce un movimiento pequeño en el efecto físico de las acciones de control y vice versa. En la implementación de GA, por su parte, el operador de mutación y crossover utilizados dictan que el espacio adyacente no está dado por el valor de las coordenadas, sino que por el número de coordenadas con valores distintos, y por lo tanto no es relevante como se ordenen las distintas soluciones.

Luego, para evitar esta incongruencia, se plantea una nueva codificación, que se presenta en la Figura 26, junto son sus efectos físicos.

$$\underbrace{0 = \begin{bmatrix} 1 \\ 1 \\ 0 \end{bmatrix}}_{\text{Sólo agua fría}}, \underbrace{1 = \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}}_{\text{Sólo recirculación}}, \underbrace{2 = \begin{bmatrix} 0.01 \\ 0 \\ 1 \end{bmatrix}}_{\text{Agua caliente y mucha recirculación}}, \underbrace{3 = \begin{bmatrix} 0.02 \\ 0 \\ 1 \end{bmatrix}}_{\cdots}, 4 = \begin{bmatrix} 0.05 \\ 0 \\ 1 \end{bmatrix}, \underbrace{5 = \begin{bmatrix} 0.1 \\ 0 \\ 1 \end{bmatrix}}_{\text{Agua caliente y menos recirculación}}, \underbrace{6 = \begin{bmatrix} 1 \\ 0 \\ 1 \end{bmatrix}}_{\text{Agua caliente}}$$

Figura 26: interpretación física de las acciones de control para la nueva representación.

Claramente esta representación no presenta las incongruencias de la alternativa anterior y es posible esperar que funcione mejor. En términos más formales, la segunda representación es define como:

$$u(k+h-1)=\begin{cases}0 & si\ x_h<0.5\\ 1 & si\ 0.5\le x_h<1.5\\ 2 & if\ 1.5\le x_h<2.5\\ 3 & if\ 2.5\le x_h<3.5\\ 4 & if\ 3.5\le x_h<4.5\\ 5 & if\ 4.5\le x_h<5.5\\ 6 & if\ 5.5\le x_h<6.5\end{cases}\quad,\ h=1,...,N_u, \tag{6.15}$$

donde se utiliza la definición presentada en la Figura 26 para los valores de $u(k+h-1)$.:

Las soluciones generadas con estas representaciones son siempre factibles, y por lo tanto no se requieren estrategias extras para el manejo de restricciones, al igual que en el caso del diseño basado en GA. Los métodos que utilicen la primera representación se denotan por PSO-FA (FA por factible, opción A), y los que utilicen la segunda representación se denotan por PSO-FB (FB por factible, opción B).

Manejo de restricciones

La representación dada por la opción 2 consiste en una representación factible, que implícitamente evita que aparezcan las restricciones sobre las acciones de control.

La opción 1, por su parte, no garantiza que se eviten las soluciones candidatas infactibles. Además, como sólo hay restricciones en las acciones de control, es bastante sencillo reparar las soluciones infactibles, y por lo tanto se propone utilizar enfoques de reparación para manejar estas restricciones.

El uso de estas codificaciones y de las funciones típicas de actualización de PSO pueden producir soluciones infactibles. Estas soluciones infactibles son de la forma $(k_M,q)=(x,2),\ x\in\{0.01,0.02,0.05,0.1\}$, como se desprende del conjunto de acciones de control posibles (ecuación (6.8)) y de la definición de q (ecuación (6.12)). Se escoge un enfoque "Baldwiniano" para el manejo de las soluciones infactibles. Para evaluar la función objetivo, se proponen tres procedimientos de reparación para cuando se tiene una solución infactible del tipo $(k_M,q)=(x,2)$. En el primer procedimiento, las posiciones de las válvulas se decodifican como $(k_M,k)=(x,1)$, que significa que la válvula *on/off* es llevada a una posición factible. En el segundo procedimiento, las posiciones de las válvulas se decodifican como $(k_M,q)=(1,2)$, lo que significa que se lleva a la válvula de mezcla a una posición factible. La tercera opción será presentada más adelante.

Es predecible que estos dos procedimientos de reparación sesguen la búsqueda de soluciones a distintas zonas del espacio. Se realizan simulaciones del sistema de control utilizando estos enfoques de reparación, y se encuentra que los resultados concuerdan con esta predicción. Cuando la salida (temperatura del núcleo T) es muy cercana a la referencia, o se encuentra por debajo de ella, las soluciones óptimas en general son de la forma $(k_M,q)=(x,1)$, mientras que cuando la salida se encuentra por encima de la referencia, las soluciones óptimas en general son

de la forma $(k_M, q) = (1, 2)$. Es claro entonces que el primer procedimiento de reparación funciona mejor en la primera situación recién descrita, y que el segundo procedimiento funciona mejor en la segunda situación descrita. Muy relacionado con esto, se tiene el problema que al utilizar el primer procedimiento de reparación el control funciona muy mal cuando la salida esta por sobre la referencia, y utilizando el segundo procedimiento de reparación se de lo propio cuando las salidas están cercanas o por debajo de la referencia. Para superar estas limitaciones, se propone un tercer esquema de reparación: en cada iteración, en el momento de decodificar las partículas para evaluar las soluciones candidatas, se realiza una selección aleatoria con probabilidades uniformes entre los dos procedimientos de reparación. Los resultados muestran la eficacia del nuevo procedimiento (Figura 27).

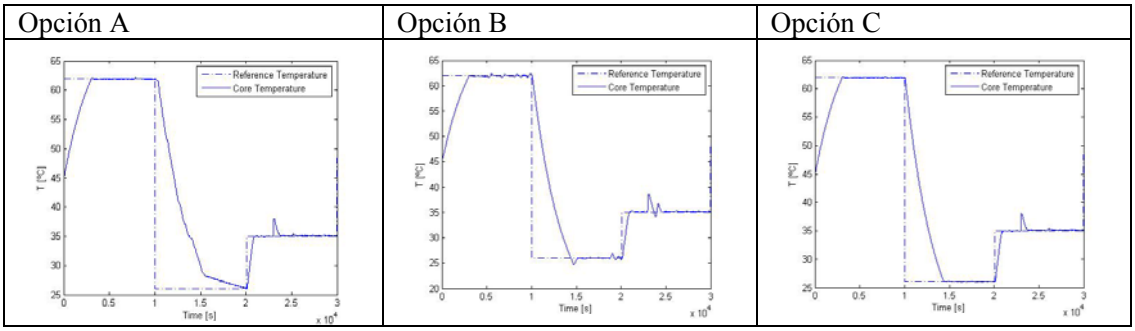


Figura 27: respuesta del proceso utilizando un horizonte de predicción $N = 5$ para diversos esquemas de reparación.

Las estrategias utilizadas se pueden resumir como:

Opción A: $(k_M, q) = (x, 2) \Rightarrow (k_M, q) = (x, 1)$

Opción B: $(k_M, k) = (x, 2) \Rightarrow (k_M, k) = (1, 2)$

Opción C: selección aleatoria con entre las opciones 1 y 2 para cada iteración del algoritmo de optimización.

Finalmente, los métodos que utilizan estas estrategias de reparación se denotan por PSO-RA, PSO-RB y PSO-RC, cuando utilizan las estrategias de reparación A, B y C, respectivamente.

6.1.2.3 Aspectos generales del controlador predictivo basado en algoritmos evolutivos para el reactor batch

En este punto se presentan algunos aspectos respecto de la implementación de controladores predictivos basados en algoritmos evolutivos, completamente independientes se implementa con algoritmos genéticos o con optimización por enjambre de partículas.

Inicialización de las soluciones

El algoritmo evolutivo se ejecuta cada vez que se debe calcular una nueva acción de control, y en ese instante se inicializan los individuos o partículas según corresponda para ejecutar el algoritmo de optimización.

Para acelerar el proceso de convergencia y/o dar un buen punto de partida al proceso de optimización, se propone insertar dentro de la población la solución del problema de

optimización del instante anterior. Dentro de esta estrategia se manejan dos variantes, la primera, insertar esta solución intacta, o bien desplazada de acuerdo a la estrategia de horizonte deslizante, tal como han sido presentadas en 5.5.

Las implementaciones que no utilizan esta estrategia se denotan con el sufijo 1 (por ejemplo: PSO-RB-1), las que utilizan la inserción de la solución anterior intacta se denotan por el sufijo 2 (por ejemplo GA-F-2), y finalmente, las que utilizan la solución desplazada, utilizan el sufijo 3 (por ejemplo PSO-F-3).

Reducción de la carga computacional

Como se trata de un problema discreto, para reducir la carga computacional se almacenan las soluciones ya evaluadas. Así, como es posible que una solución ya visitada vuelva a serlo, se ahorra el tiempo computacional asociado a la evaluación de la función objetivo. Esta estrategia es utilizada en todas las implementaciones probadas.

El almacenamiento se realiza de modo que se guarda el efecto acumulado por cada acción de control. Por ejemplo, si se aplica la acción de control definida como $u=(1,1,2,2)$, según la representación definida en la ecuación (6.14), y sólo se encuentra almacenada la acción de control dada por $u=(1,1,1,1)$, los efectos de las dos primeras componentes ya están almacenados y basta con recalcular los efectos de las dos siguientes componentes. Sin embargo, si se aplica $u=(2,1,1,1)$, hay que recalcular los efectos completos, pues la primera acción de control cambia completamente el efecto del resto.

Considerando esto, los distintos métodos pueden tener distintas tasas de convergencia y por lo tanto pueden evaluar un distinto número de veces las funciones objetivos, a pesar de poseer igual número de iteraciones y tamaño de población. Además, la cantidad de evaluaciones de función objetivo se cuenta por el número de acciones de control dentro del horizonte cuyos efectos deben ser calculados. De este modo, el conteo de evaluaciones para $u=(1,1,2,2)$ es 2, mientras que para $u=(2,1,1,1)$ es 4, si sólo está almacenada la acción de control dada por $u=(1,1,1,1)$.

6.1.3 Estudios por simulación

En esta sección se analiza la aplicación de algoritmos convencionales (óptimos) y algoritmos evolutivos para resolver el problema de optimización discreta del control predictivo híbrido.

Se considera un patrón de operación típico para los procesos de fabricación de remedios. Durante los primeros 8000 segundos la referencia es de 62º C, entonces cambia a 26ºC donde permanece por otros 8000 segundos, y finalmente los últimos 6000 segundos la referencia es de 35ºC. Además, se considera una perturbación persistente en la temperatura del reactor: se aplica un escalón de 3ºC a los 18000 seg. que se añaden a la temperatura del reactor. Los distintos métodos son aplicados para resolver los problemas de optimización instante a instante en la estrategia de control predictivo híbrido, al utilizar un horizonte de predicción $N=5$.

Como medida de la calidad de la respuesta del sistema se utiliza una función de desempeño con las mismas componentes que la utilizada en el controlador predictivo, pero que no considera predicciones, sino que los resultados reales en cada tiempo de muestreo. Es decir:

$$J = w_1 \sum_{h=1}^{K} \left(T\left(k+h\right) - T_{ref}\left(k+h\right) \right)^2 + w_2 \sum_{h=1}^{K} \left| \Delta k_M \left(k+h-1\right) \right| k_H \left(k+h-1\right) + \\ w_3 \sum_{h=1}^{K} k_C \left(k+h\right) k_H \left(k+h-1\right) + \tag{6.16}$$

donde K es el número total de tiempos de muestreo considerados en el estudio. Además, se considera el desempeño en cada uno de los objetivos que componen la función de desempeño, que se desglosan como:

$$J_1 = \sum_{h=1}^{K} \left(T\left(k+h\right) - T_{ref}\left(k+h\right) \right)^2$$
$$J_2 = \sum_{h=1}^{K} \left| \Delta k_M \left(k+h-1\right) \right| k_H \left(k+h-1\right) \tag{6.17}$$
$$J_3 = \sum_{h=1}^{K} k_C \left(k+h\right) k_H \left(k+h-1\right)$$

donde J_1 corresponde al error de seguimiento, J_2 mide el esfuerzo de control de la válvula de mezcla, y J_3 penaliza el cambio de agua caliente a agua fría.

Para obtener las estadísticas de los diversos métodos, se considera 20 réplicas para cada una de las distintas condiciones en que son probados (distintos parámetros, estrategias de inicialización de población, etc.). Además, los estudios han sido realizados con computadores Mac OS X, v10.5.6, con procesadores Core 2 Duo de 2.66 GHz y 4 GB de memoria RAM.

6.1.3.1 Utilizando *branch and bound*

Primero se analiza la respuesta del sistema cuando el problema de optimización es resuelto mediante *branch and bound* (BB) basado en un análisis de alcanzabilidad, tal como es descrito en la Sección 3.3.3.

Es posible apreciar que la respuesta del sistema es bastante buena, con un buen seguimiento y reducido esfuerzo de control (ver Figura 28).

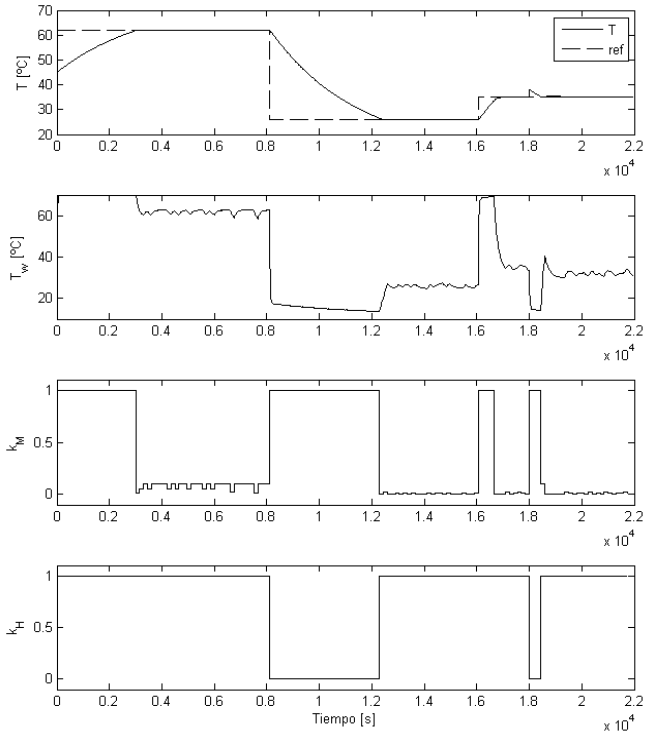


Figura 28: Respuesta del sistema con un controlador predictivo híbrido con branch and bound.

Se puede apreciar también en la Figura 29 que el esfuerzo computacional depende de la zona de operación del proceso, pues ésta incide en la búsqueda que se realiza a través del árbol utilizado en BB. En la Tabla 1 se presenta las estadísticas de función objetivo y sus distintas componentes, y de esfuerzo computacional, que exhibe este método para controlar al reactor. Como el proceso acepta hasta 10 segundos para calcular las acciones de control, es posible utilizar este método en un proceso real. Sin embargo, estas estadísticas se obtienen con un computador de escritorio, cuando en ciertos casos prácticos el control de máquinas y procesos son realizados por procesadores menos poderosos, como DSPs, PICs o PLCs. En estos casos, sobre todo para las zonas de operación que requieren de más evaluaciones, este método no sería el más adecuado, y se requerirían otros métodos, por ejemplo los algoritmos evolutivos.

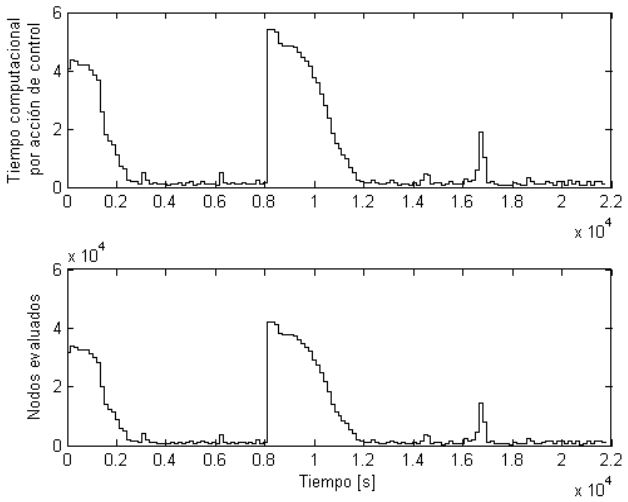


Figura 29: Esfuerzo computacional para BB+NLP a lo largo de la operación del sistema.

Tabla 1: Estadísticas para BB

Estadísticas de desempeño							
J		**J1**		**J2**		**J3**	
Prom	**Desv.**	**Prom.**	**Desv.**	**Prom.**	**Desv.**	**Prom.**	**Desv.**
9848.5	0	9818.3	0	6.48	0	2	0
Estadísticas de esfuerzo computacional por acción de control							
Tiempo computacional [s]				**Nodos evaluados**			
Prom.	**Desv.**	**Min.**	**Max.**	**Prom.**	**Desv.**	**Min.**	**Max.**
1.0044	1.5922	0.047	5.422	7777.9	12345.3	420	42000

6.1.3.2 Usando algoritmos evolutivos

Ya se ha encontrado que BB es aplicable en ciertas condiciones, y en otras no. Por lo tanto, en esta sección se estudia diversas implementaciones basadas en algoritmos evolutivos para estudiar su desempeño, determinar cuales de éstas son más convenientes, y sintonizar dichos métodos.

Se parten los análisis probando sólo dos combinaciones de iteraciones y de tamaño de la población. Posteriormente, como los análisis con sólo dos conjuntos de parámetros no entregan toda la información requerida sobre la calidad de las estrategias, se realiza un análisis multi-objetivo para estudiar los métodos con la mayor información disponible.

6.1.3.2.1 Análisis para tamaños de población y número de iteraciones fijos

En esta sección se comparan los métodos utilizando sólo 2 combinaciones de tamaño de población y número de iteraciones. La estructura de presentación de los resultados es la siguiente: primero, para la representación de PSO que requiere reparación de soluciones infactibles, se estudian las tres estrategias de reparación propuestas, y para la que resulte mejor, se estudian las diversas estrategias de inicialización de población; segundo, se comparan las dos estrategias de representación factible en PSO, y para la que resulte ser mejor, se prueban las distintas estrategias de inicialización de población; tercero, se comparan las distintas estrategias de inicialización de soluciones para la única representación propuesta de GA; posteriormente, se comparan las mejores alternativas de cada una de las representaciones de PSO y GA en un intento en encontrar el mejor método; y finalmente se argumenta la limitación de utilizar tamaños fijos de población y números de iteraciones, y se da paso al análisis multi-objetivo.

1. Comparación de estrategias de reparación para métodos PSO-R

Primero se comparan enfoques de reparación para la primera representación de PSO (Opción 1), considerando que las soluciones se inicializan de un modo completamente aleatorio. Esto corresponde a los métodos PSO-RA-1, PSO-RB-1 y PSO-RC-1, los cuales se diferencian en que las reparaciones son para la válvula *on/off*, para la válvula de mezcla, o aleatoria entre las dos válvulas en cada iteración, respectivamente En la Tabla 2 se presentan las estadísticas de desempeño y esfuerzo computacional para estos tres métodos utilizando 15 partículas y 15 iteraciones, mientras que en la Tabla 3 se presentan las mismas estadísticas para 30 partículas y 30 iteraciones (estos parámetros son los mismos que se han encontrado como óptimos para la implementación basada en GA en Causa *et al*., 2008). Además, de la Figura 30 a la Figura 35 se

presentan las respuestas dinámicas del reactor batch al ser controlado por estos métodos y parámetros.

Tabla 2: Estadísticas para los métodos utilizando 15 partículas y 15 iteraciones

Método	J		J1		J2		J3	
	Prom.	**Std.**	**Prom.**	**Std.**	**Prom.**	**Std.**	**Prom.**	**Std.**
PSO-RA-1	12586.4	1021.03	12444.93	1022.52	14.05	1.770	9.4	1.67
PSO-RB-1	9977.4	96.91	9912.26	84.98	11.51	2.609	4.32	1.38
PSO-RC-1	10498.3	573.98	10081.5	558.98	10.49	1.780	5.12	1.48
Método	**Tiempo computacional**				**Fn. Objetivo evaluadas**			
	Prom.	**Std.**	**Min.**	**Max.**	**Prom.**	**Std.**	**Min.**	**Max.**
PSO-RA-1	0.914	0.308	0.441	1.652	361.3	138.2	153	642
PSO-RB-1	0.898	0.375	0.270	2.140	353.1	168.0	72	670
PSO-RC-1	0.918	0.309	0.435	1.652	363.7	138.8	149	647

Tabla 3: Estadísticas para los métodos utilizando 30 partículas y 30 iteraciones

Método	J		J1		J2		J3	
	Prom.	**Std.**	**Prom.**	**Std.**	**Prom.**	**Std.**	**Prom.**	**Std.**
PSO-RA-1	11071.5	775.08	10697.9	767.54	11.16	1.671	6.88	1.67
PSO-RB-1	9890.6	15.32	9858.6	14.30	6.907	0.786	2.12	0.33
PSO-RC-1	10135.1	286.67	10081.5	277.09	8.59	1.507	3.56	1.08
Método	**Tiempo computacional**				**Fn. Objetivo evaluadas**			
	Prom.	**Std.**	**Min.**	**Max.**	**Prom.**	**Std.**	**Min.**	**Max.**
PSO-RA-1	1.892	0.806	0.952	3.671	755.7	414.18	275	1669
PSO-RB-1	1.908	0.924	0.674	4.773	762.81	472.96	131	1759
PSO-RC-1	1.889	0.812	0.732	3.712	754.41	416.62	158	1697

A partir de estos resultados, en particular de las figuras que muestran la respuesta del sistema con 15 partículas y 15 iteraciones, parece que la mejor estrategia es PSO-RC-1, pues provee un buen equilibrio entre el seguimiento en régimen permanente y en el transiente. Por su parte, PSO-RA-1 sólo se comporta bien en el seguimiento en el régimen permanente, con serios problemas en el transiente, mientras que la característica de PSO-RB-1 es completamente la inversa. Sin embargo, al aumentar a 30 partículas y 30 iteraciones, el método PSO-RB-1 parece mejorar sus limitaciones, mientras que los otros dos siguen exhibiendo sus limitaciones. Por lo tanto, el método PSO-RB-1 parece ser el más robusto. Esto se refleja más claramente en la Tabla 2 y en la Tabla 3, donde se aprecia que el método PSO-RB-1 logra consistentemente mejores desempeños.

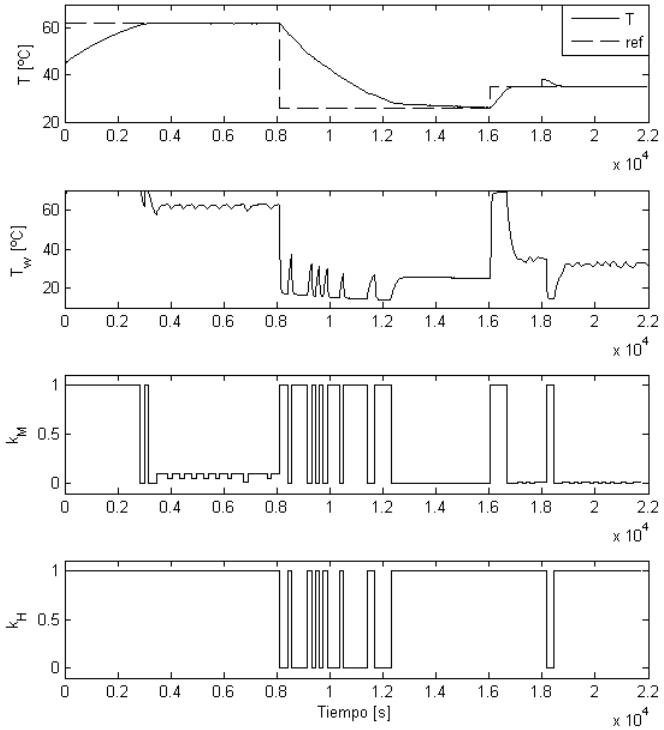


Figura 30: Respuesta del sistema con un controlador predictivo híbrido con PSO-RA-1, 15 partículas y 15 iteraciones.

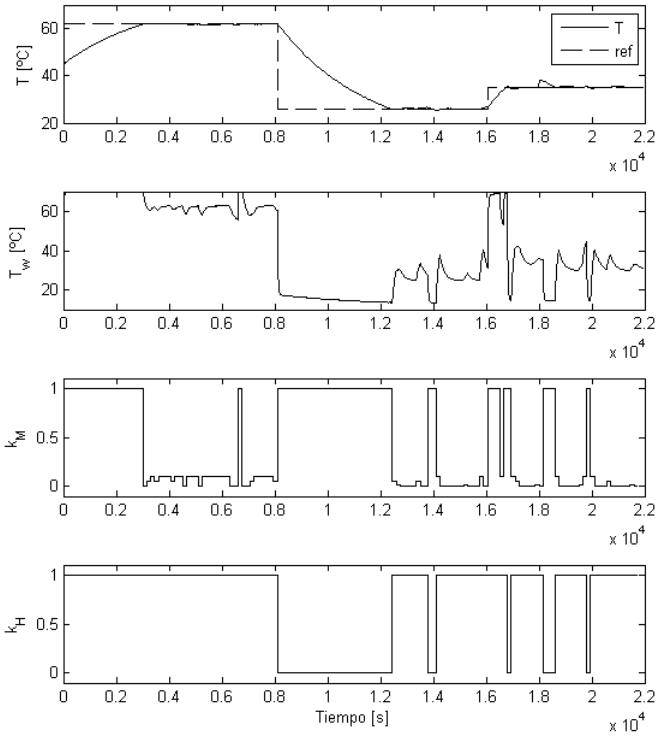


Figura 31: Respuesta del sistema con un controlador predictivo híbrido con PSO-RB-1, 15 partículas y 15 iteraciones.

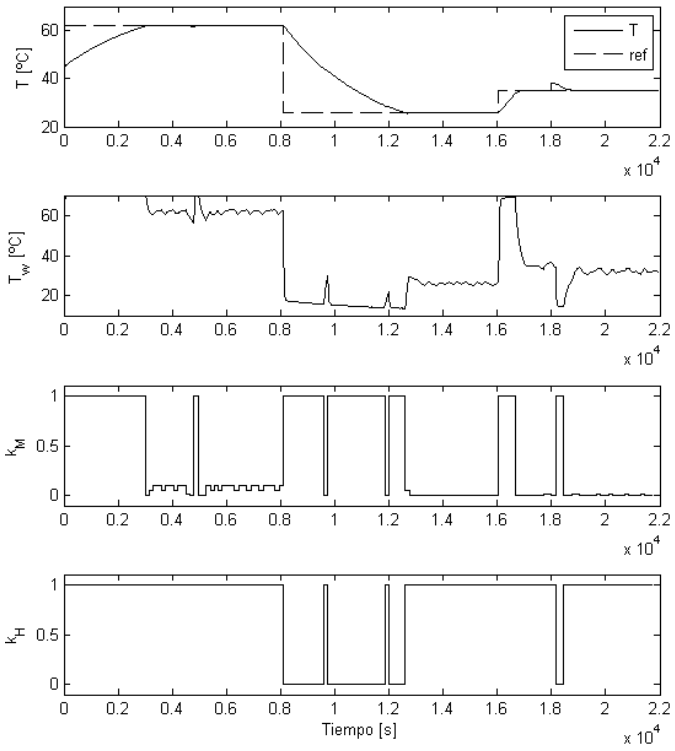


Figura 32: Respuesta del sistema con un controlador predictivo híbrido con PSO-RC-1, 15 partículas y 15 iteraciones.

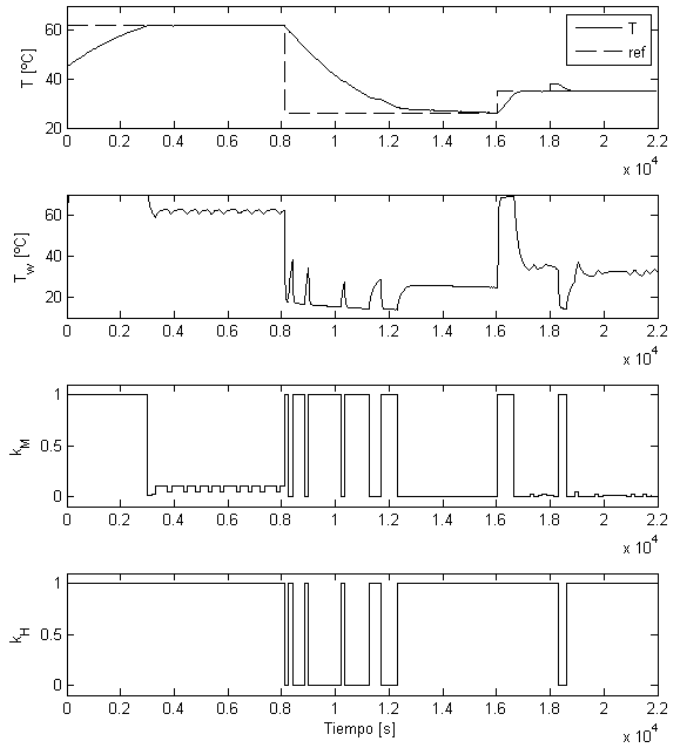


Figura 33: Respuesta del sistema con un controlador predictivo híbrido con PSO-RA-1, 30 partículas y 30 iteraciones.

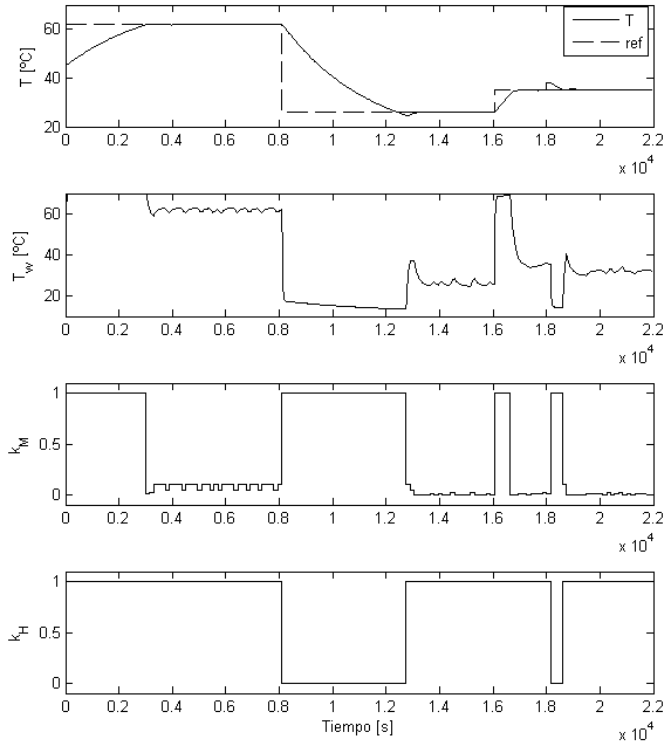


Figura 34: Respuesta del sistema con un controlador predictivo híbrido con PSO-RB-1, 30 partículas y 30 iteraciones.

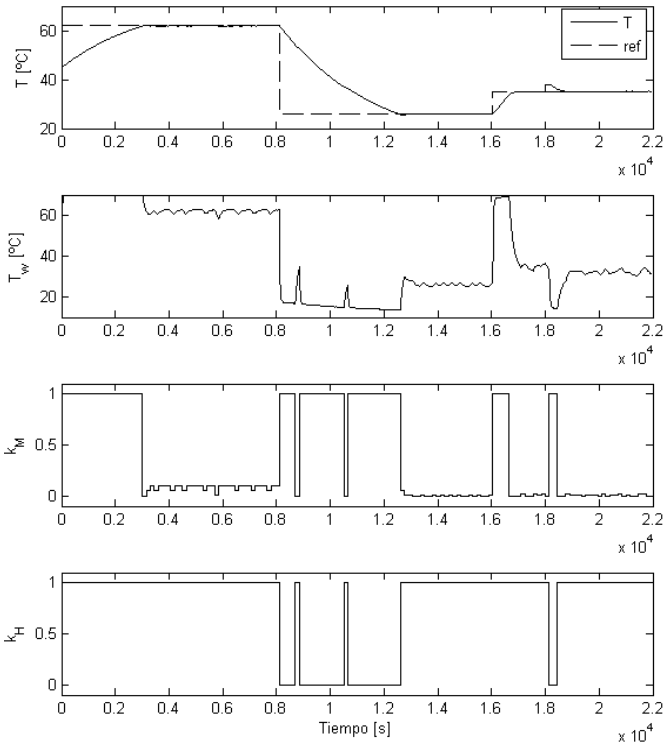


Figura 35: Respuesta del sistema con un controlador predictivo híbrido con PSO-RC-1, 30 partículas y 30 iteraciones.

2. Comparación de estrategias de inicialización de población para método PSO-RB

Ya que se ha determinado que el método PSO-RB-1 es el mejor (en comparación con PSO-RA-1 y PSO-RC-1), se utiliza éste método con 30 iteraciones y 30 partículas para probar la influencia de las distintas estrategias de inicialización de población. Luego, se prueban los métodos PSO-RB-1, PSO-RB-2, PSO-RB-3, que corresponden a los métodos con reparación de la válvula de mezcla, al iniciar la población: aleatoriamente, aleatoriamente pero incluyendo la mejor solución anterior, y aleatoriamente pero incluyendo la mejor solución anterior desplazada, respectivamente. Los resultados se resumen en la Tabla 4.

Tabla 4: Estadísticas para los métodos PSO-RB utilizando 30 partículas y 30 iteraciones

Método	J		J1		J2		J3	
	Prom.	**Std.**	**Prom.**	**Std.**	**Prom.**	**Std.**	**Prom.**	**Std.**
PSO-RB-1	9890.6	15.32	9858.6	14.30	6.907	0.786	2.12	0.33
PSO-RB-2	9879.0	17.13	9848.8	17.13	6.751	0.700	2	0
PSO-RB-3	9882.6	14.48	9852.4	14.48	6.402	0.066	2	0
Método	**Tiempo computacional**				**Fn. Objetivo evaluadas**			
	Prom.	**Std.**	**Min.**	**Max.**	**Prom.**	**Std.**	**Min.**	**Max.**
PSO-RB-1	1.908	0.924	0.674	4.773	762.81	472.96	131	1759
PSO-RB-2	1.775	0.919	0.665	3.753	700.55	473.97	127	1721
PSO-RB-3	1.783	0.94	0.661	4.341	701.03	481.13	125	1730

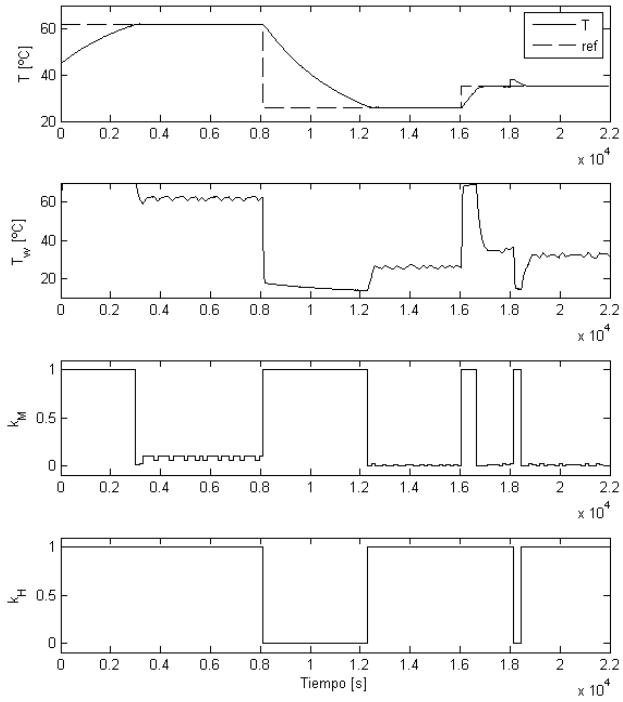


Figura 36: Respuesta del sistema con un controlador predictivo híbrido con PSO-RB-2, 30 partículas y 30 iteraciones.

Resulta claro luego de analizar la Tabla 4 que incluir una solución anterior en la nueva población es un aporte, tanto en términos de calidad de las soluciones como en esfuerzo computacional. De las dos alternativas que incluyen la solución anterior, la mejor opción parece ser PSO-RB-2,

debido a que posee mejores desempeños. Sin embargo, la diferencia es muy pequeña en comparación a la desviación estándar, por lo que establecer conclusiones acerca cual estrategia es la mejor no es posible. De todos modos, lo más probable es que PSO-RB-2 sea mejor. En la Figura 36 entonces se presenta la respuesta dinámica del sistema con este método. Se aprecia que la respuesta mejora en el transiente respecto de utilizar PSO-RB-1 (comparar con Figura 34), pues al llegar a la referencia de 26ºC la llegada es más suave.

3. Comparación de representaciones factibles en PSO

Ahora se analiza la otra representación propuesta en el caso de PSO: la representación factible, en primera instancia inicializando la población al azar. Es decir, se consideran los métodos PSO-FA-1 y PSO-FB-1 (el primero utiliza la representación FA, que no tiene sentido físico, y el segundo la representación FB, que se ha corregido para que sí tenga sentdo físico,tal como es descrito en la Sección 6.2.2.1). Tal como para la otra representación, se estudian los casos con 15 partículas e iteraciones (Tabla 5, Figura 37 y Figura 38), y luego con 30 partículas e iteraciones (Tabla 6, Figura 39 y Figura 40), ambas con soluciones candidatas inicializadas al azar.

Tabla 5: Estadísticas para los métodos PSO-FA-1 y PSO-FB-1 utilizando 15 partículas y 15 iteraciones

Método	**J**		**J1**		**J2**		**J3**	
	Prom.	**Std.**	**Prom.**	**Std.**	**Prom.**	**Std.**	**Prom.**	**Std.**
PSO-FA-1	10508.6	433.9	10424.2	427.4	12.278	2.454	5.6	1.080
PSO-FB-1	9891.60	28.61	9861.4	28.60	6.990	1.13	2	0
Método	**Tiempo computacional**				**Fn. Objetivo evaluadas**			
	Prom.	**Std.**	**Min.**	**Max.**	**Prom.**	**Std.**	**Min.**	**Max.**
PSO-FA-1	1.076	0.382	0.472	1.901	436.77	170.62	170	711
PSO-FB-1	1.215	0.417	0.467	1.816	499.69	186.84	165	711

Tabla 6: Estadísticas para los métodos PSO-FA-1 y PSO-FB-1 utilizando 30 partículas y 30

Método	**J**		**J1**		**J2**		**J3**	
	Prom.	**Std.**	**Prom.**	**Std.**	**Prom.**	**Std.**	**Prom.**	**Std.**
PSO-FA-1	10116.9	336.43	10070.5	332.11	7.402	0.959	3.08	0.954
PSO-FB-1	9887.0	16.87	9856.8	16.87	6.38	0.035	2	0
Método	**Tiempo computacional**				**Fn. Objetivo evaluadas**			
	Prom.	**Std.**	**Min.**	**Max.**	**Prom.**	**Std.**	**Min.**	**Max.**
PSO-FA-1	2.367	1.099	0.983	5.983	1007.7	562.1	297	2020
PSO-FB-1	2.719	1.141	0.937	4.740	1186.9	584.8	272	1998

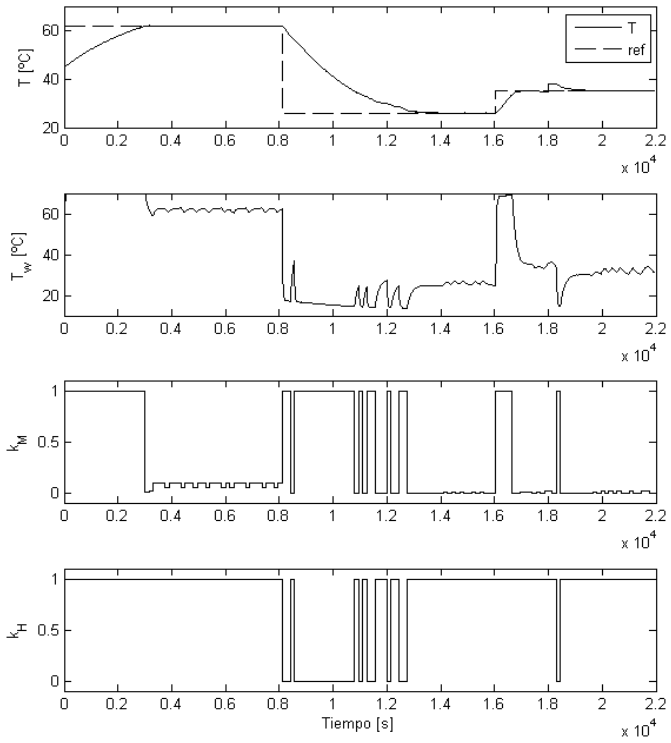


Figura 37: Respuesta del sistema con un controlador predictivo híbrido con PSO-FA-1, 15 partículas y 15 iteraciones.

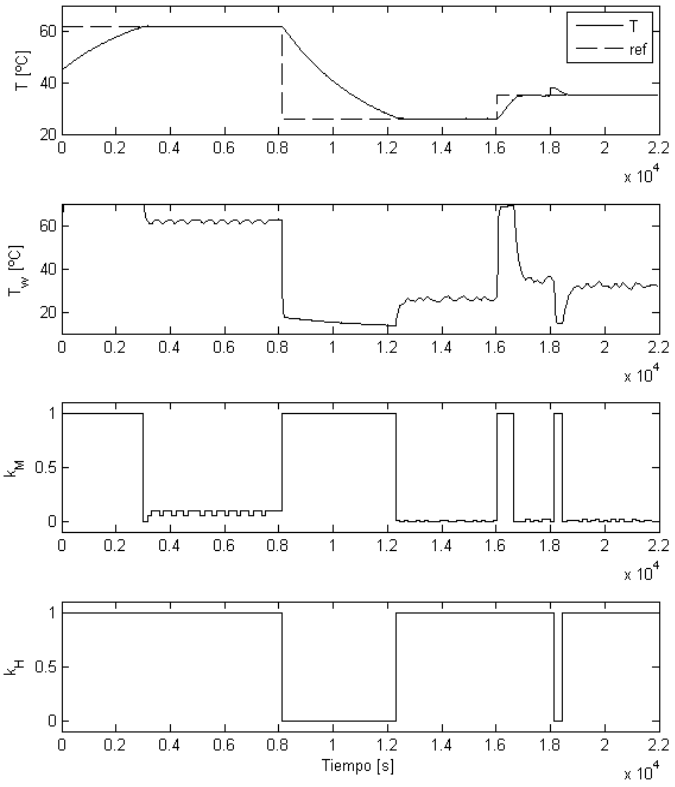


Figura 38: Respuesta del sistema con un controlador predictivo híbrido con PSO-FB-1, 15 partículas y 15 iteraciones.

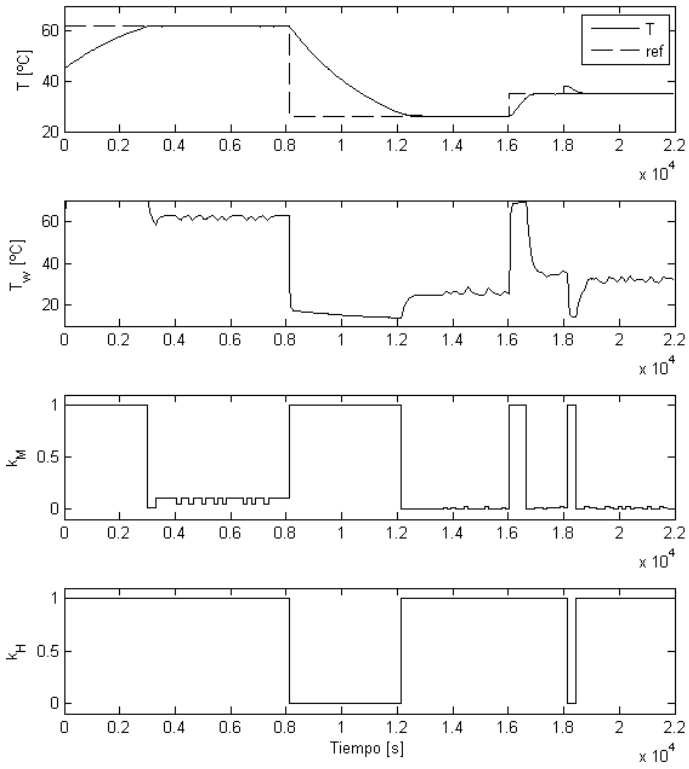


Figura 39: Respuesta del sistema con un controlador predictivo híbrido con PSO-FA-1, 30 partículas y 30 iteraciones.

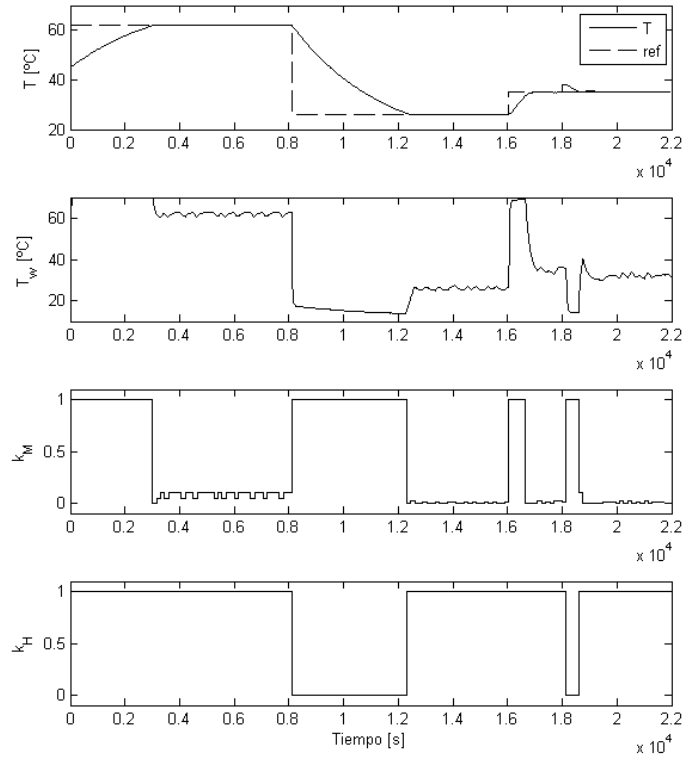


Figura 40: Respuesta del sistema con un controlador predictivo híbrido con PSO-FB-1, 30 partículas y 30 iteraciones.

Se puede apreciar de las figuras anteriores que el método PSO-FB-1 funciona bastante bien para ambas combinaciones de parámetros. De la Tabla 5 y de la Tabla 6 se puede apreciar además que esta representación supera en desempeño a PSO-FA-1 para ambas combinaciones de parámetros.

4. Comparación de estrategias de inicialización para método PSO-FB

Como en el punto se muestra que el método PSO-FB-1 (que usa representación con sentido físico) supera a PSO-FA-1 (cuya representación no posee tal sentido físico), ahora se prueba la representación FB con las 3 alternativas para inicializar la población, y también usando 30 partículas y 30 iteraciones.Entonces, se prueban los métodos PSO-FB-1, PSO-FB-2, PSO-FB-3.

Tabla 7: Estadísticas para los métodos PSO-FB utilizando 30 partículas y 30 iteraciones

Método	**J**		**J1**		**J2**		**J3**	
	Prom.	**Std.**	**Prom.**	**Std.**	**Prom.**	**Std.**	**Prom.**	**Std.**
PSO-FB-1	9887.0	16.87	9856.8	16.87	6.38	0.035	2	0
PSO-FB-2	9890.9	14.84	9860.7	14.85	6.711	0.742	2	0
PSO-FB-3	9887.6	17.25	9854.4	17.25	6.394	0.0485	2	0
Método	**Tiempo computacional**				**Fn. Objetivo evaluadas**			
	Prom.	**Std.**	**Min.**	**Max.**	**Prom.**	**Std.**	**Min.**	**Max.**
PSO-FB-1	2.719	1.141	0.937	4.740	1186.9	584.8	272	1998
PSO-FB-2	2.595	1.144	0.901	4.285	1131.0	590.4	256	1999
PSO-FB-3	2.587	1.139	0.917	4.376	1121.8	585.8	265	1857

Parece no quedar muy claro cual es el mejor de los métodos, ya que la diferencia en el desempeño es muy pequeña. Sin embargo, insertar las soluciones anteriores de todos modos entrega una ayuda, pues si bien no mejora la calidad de las soluciones, sí logra reducir el tiempo computacional. En base a esto parece que insertar la mejor solución desplazada es la mejor opción. Luego, la respuesta dinámica utilizando este método se presenta a continuación.

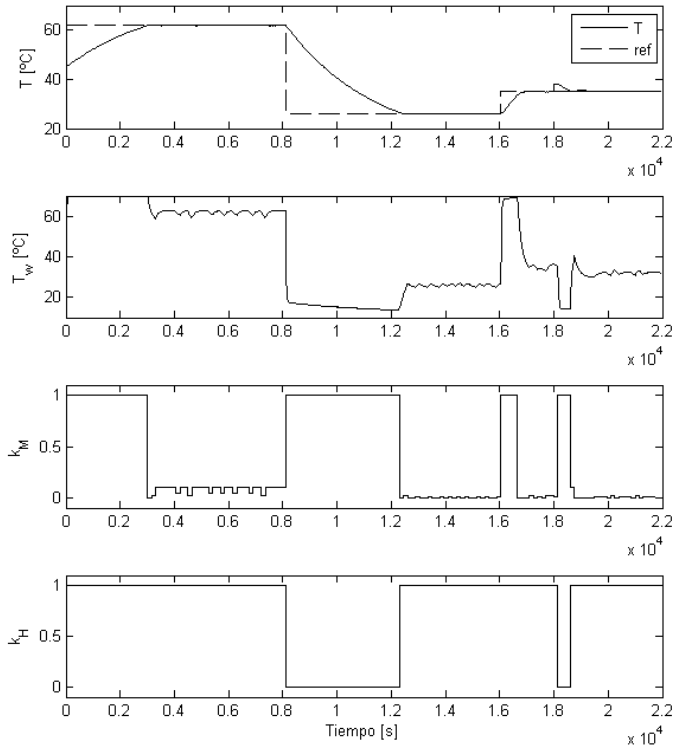


Figura 41: Respuesta del sistema con un controlador predictivo híbrido con PSO-FB-3, 30 partículas y 30 iteraciones.

5. Comparación de estrategias de inicialización para método GA-F

Finalmente se prueba el algoritmo genético, considerando las 3 alternativas de inicialización de población, y las dos combinaciones de parámetros utilizadas. Entonces, se prueban los métodos PSO-FB-1, PSO-FB-2, PSO-FB-3. Los resultados se presentan en la Tabla 8 (15 partículas e iteraciones) y
Tabla 9 (30 partículas e iteraciones).

De estos resultados, se aprecia que con ambas combinaciones de parámetros, GA-F-2 supera el rendimiento de los otros dos casos. Sin embargo la diferencia es muy pequeña respecto de la desviación estándar para poder concluir efectivamente. De todos modos, dado esto, lo más probable es que insertar la solución anterior intacta sea la mejor estrategia de inicialización de población para GA.

Luego se presentan las respuestas dinámicas utilizando GA-F-2 con 15 individuos y generaciones (Figura 42), y con 30 individuos y generaciones (Figura 43). Se puede apreciar que el comportamiento usando sólo 15 individuos y generaciones es claramente insuficiente, proporcionando un muy mal control. Pero cuando se utilizan 30 individuos y 30 iteraciones el comportamiento dinámico ya se aprecia que es adecuado.

Tabla 8: Estadísticas para los métodos GA-F utilizando 15 partículas y 15 iteraciones

Método	J		J1		J2		J3	
	Prom.	Std.	Prom.	Std.	Prom.	Std.	Prom.	Std.
GA-F-1	10478.3	484.68	10412.5	471.2	15.109	471.2	4.36	1.823
GA-F-2	10398.2	349.80	10327.3	321.2	16.479	3.240	4.692	2.15
GA-F-3	10589.3	465.64	10512.8	443.5	16.881	3.095	5.063	2.380
Método	**Tiempo computacional**				**Fn. Objetivo evaluadas**			
	Prom.	**Std.**	**Min.**	**Max.**	**Prom.**	**Std.**	**Min.**	**Max.**
GA-F-1	0.817	0.0922	0.480	1.066	295.54	44.89	129	405
GA-F-2	0.807	0.0918	0.456	1.263	293.10	45.60	111	418
GA-F-3	0.806	0.0904	0.493	1.084	295.18	46.34	132	437

Tabla 9: Estadísticas para los métodos GA-F utilizando 30 partículas y 30 iteraciones

Método	J		J1		J2		J3	
	Prom.	Std.	Prom.	Std.	Prom.	Std.	Prom.	Std.
GA-F-1	9935.9	79.732	9899.7	72.518	7.653	1.345	2.4	0.646
GA-F-2	9905.2	47.624	9871.5	40.145	4.717	1.665	2.231	0.599
GA-F-3	9924.8	77.661	9891.8	72.436	6.970	0.773	2.188	0.403
Método	**Tiempo computacional**				**Fn. Objetivo evaluadas**			
	Prom.	**Std.**	**Min.**	**Max.**	**Prom.**	**Std.**	**Min.**	**Max.**
GA-F-1	2.306	0.275	1.499	3.059	830.92	140.64	420	1218
GA-F-2	2.296	0.264	1.533	2.970	831.71	135.75	437	1186
GA-F-3	2.307	0.277	1.486	3.847	832.76	139.64	415	1195

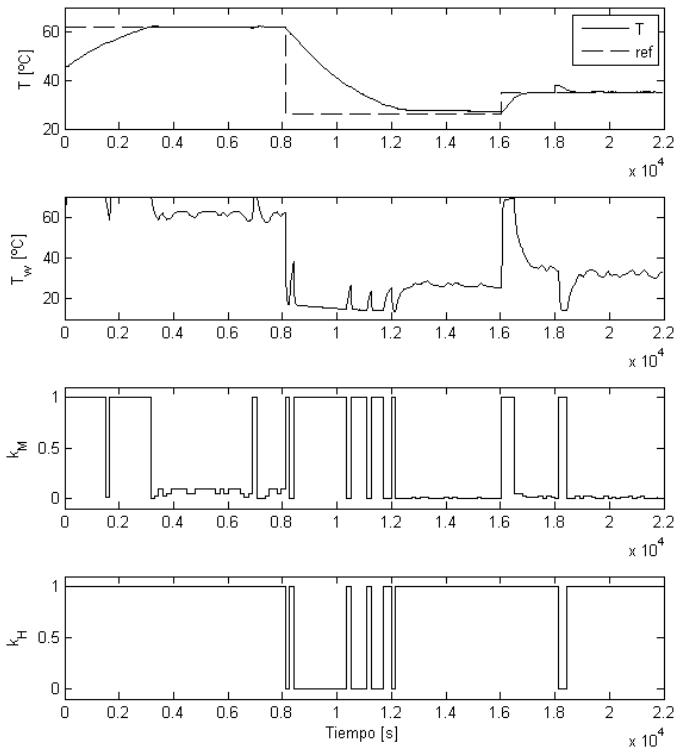


Figura 42: Respuesta del sistema con un controlador predictivo híbrido con GA-F-2, 15 partículas y 15 iteraciones.

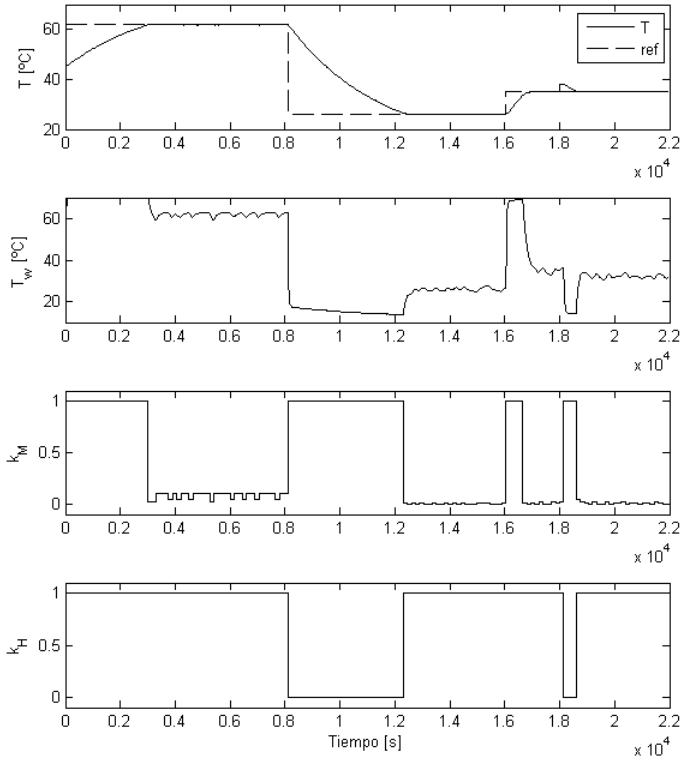


Figura 43: Respuesta del sistema con un controlador predictivo híbrido con GA-F-2, 30 partículas y 30 iteraciones.

6. Comparación de métodos basados en PSO y GA

Finalmente, en la Tabla 10 se presentan los mejores métodos de cada una de las representaciones estudiadas, PSO-RB-2, PSO-FB-3 y GA-F-2, que se han concluido de las pruebas en los puntos 2, 4 y 5. Es claro notar que el rendimiento del algoritmo genético se encuentra muy por debajo de los otros dos métodos. PSO-RB-2 (con reparación de válvula de mezcla e inclusión de solución del instante anterior intacta) por su parte, parece ser el mejor para 30 partículas y 30 iteraciones, mientras que PSO-FB-3 (con representación factible con sentido físico e inclusión de solución del instante anterior desplazada según horizonte deslizante) se muestra como el más sólido para 15 partículas e iteraciones.

Tabla 10: Estadísticas para los mejores métodos utilizando 15 individuos y 15 iteraciones

Método	**J**		**J1**		**J2**		**J3**	
	Prom.	**Std.**	**Prom.**	**Std.**	**Prom.**	**Std.**	**Prom.**	**Std.**
PSO-RB-2	9901.1	34.84	9867.4	31.59	6.895	1.100	2.231	0.439
PSO-FB-3	9878.5	21.23	9848.3	21.22	6.476	0.465	2	0
GA-F-2	10398.2	349.80	10327.3	321.2	16.479	3.240	4.692	2.15
Método	**Tiempo computacional**				**Fn. Objetivo evaluadas**			
	Prom.	**Std.**	**Min.**	**Max.**	**Prom.**	**Std.**	**Min.**	**Max.**
PSO-RB-2	0.855	0.398	0.242	1.596	340.19	181.09	62	680
PSO-FB-3	1.139	0.445	0.418	1.645	471.49	202.07	144	703
GA-F-2	0.807	0.0918	0.456	1.263	293.10	45.60	111	418

Tabla 11: Estadísticas para los mejores métodos utilizando 30 individuos y 30 iteraciones

Método	J		J1		J2		J3	
	Prom.	**Std.**	**Prom.**	**Std.**	**Prom.**	**Std.**	**Prom.**	**Std.**
PSO-RB-2	9879.0	17.13	9848.8	17.13	6.751	0.700	2	0
PSO-FB-3	9887.6	17.25	9854.4	17.25	6.394	0.0485	2	0
GA-F-2	9905.2	47.624	9871.5	40.145	4.717	1.665	2.231	0.599
Método	**Tiempo computacional**				**Fn. Objetivo evaluadas**			
	Prom.	**Std.**	**Min.**	**Max.**	**Prom.**	**Std.**	**Min.**	**Max.**
PSO-RB-2	1.783	0.94	0.661	4.341	701.03	481.13	125	1730
PSO-FB-3	2.587	1.139	0.917	4.376	1121.8	585.8	265	1857
GA-F-2	2.296	0.264	1.533	2.970	831.71	135.75	437	1186

El problema con el análisis realizado hasta este punto es que la información es incompleta. Por ejemplo, al utilizar 15 partículas e iteraciones, si bien el método PSO-FB-3 es el que entrega soluciones de mejor calidad, es el más lento, por lo tanto no queda claro que tan bueno es en realidad. Para tener una completa información para poder comparar las distintas estrategias, a continuación se realiza un estudio multi-objetivo.

6.1.3.2.2 Análisis multi-objetivo para el rango completo de iteraciones y partículas

Hasta ahora sólo se ha comparado los métodos con dos opciones de tamaño de población y número de generaciones. A continuación se presenta un estudio más detallado para distintas combinaciones de tamaños de población y número de generaciones. Este análisis posee dos objetivos. Primero, se establecerá bajo que condiciones un algoritmo es mejor que el otro para este problema de control predictivo. Y segundo, se podrá escoger los tamaños de poblaciones y número de generaciones óptimos de acuerdo a un determinado criterio para cada algoritmo.

Este análisis para la comparación de las distintas estrategias presentadas se realiza en base al enfoque multi-objetivo descrito en la Sección 5.7.2. El análisis, para entender mejor el comportamiento de cada representación, o estrategia de reparación, o etc., se realiza de un modo segmentado similar al realizado en la sección anterior.

1. *Comparación de estrategias de reparación e inicialización de soluciones para métodos PSO-R*

Se parte por los métodos PSO-RA, PSO-RB y PSO-RC (aquellos con una representación que requiere reparación), cuyas fronteras de Pareto se presentan en la Figura 44, Figura 45 y Figura 46, respectivamente, utilizando las tres estrategias de inicialización de población. Al analizar sus fronteras de Pareto, se aprecia que para los métodos PSO-RA, la inclusión de la solución anterior claramente mejora el desempeño del proceso de optimización, ya que estos métodos dominan en términos de Pareto (ver Sección 5.7.3) al que inicializa las soluciones completamente al azar. Sin embargo, la inclusión de la solución intacta beneficia más que la solución desplazada. Para el método PSO-RB la situación es similar, pero es con la inclusión de la solución desplazada la que más beneficia. Finalmente para PSO-RC también se logra mejorar el desempeño del método, pero la diferencia es muy pequeña entre ambas estrategias.

En resumen, las estrategias PSO-RB son las que más se benefician de incluir la mejor solución anterior desplazada, y esto es esperable. Esta estrategia repara la posición de la válvula de mezcla cuando esta es infactible, situación que solo ocurre cuando está dada el agua fría. Examinando el régimen de operación del sistema, esto ocurre en el transiente al ir disminuyendo la temperatura. Por otra parte, también se ha encontrado que la inclusión de la mejor solución desplazada debe funcionar mejor en los transientes, lo que explica el resultado obtenido. Respecto de la estrategia de reparación A, esta repara la posición de la válvula *on*/*off*, de modo que pase a la posición caliente, manteniendo las posiciones de mezcla, que en general se dan en los regímenes permanentes. Por lo tanto, esta reparación no funciona tan bien en los transientes. Finalmente, como la reparación C aplica aleatoriamente las opciones A y B, es esperable que la situación sea la intermedia entre ambas, es decir que se beneficie en el transiente menos que B, pero más que A, situación que efectivamente se da, pues la estrategia de inclusión de la solución desplazada aporta menos que con la reparación B, pero más que con la A.

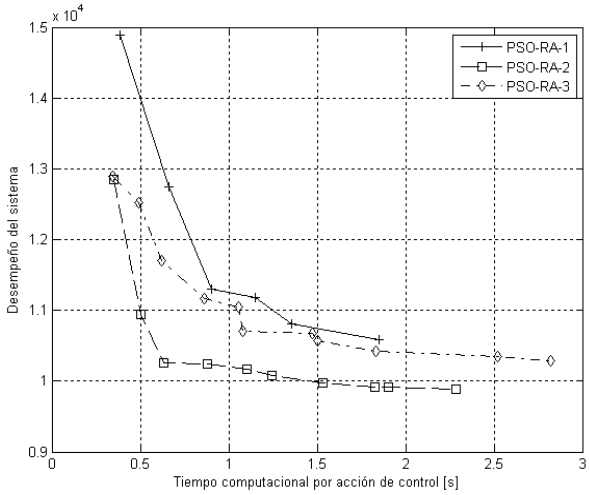


Figura 44: Frontera de Pareto para métodos PSO-RA-1, PSO-RA-2 y PSO-RA-3

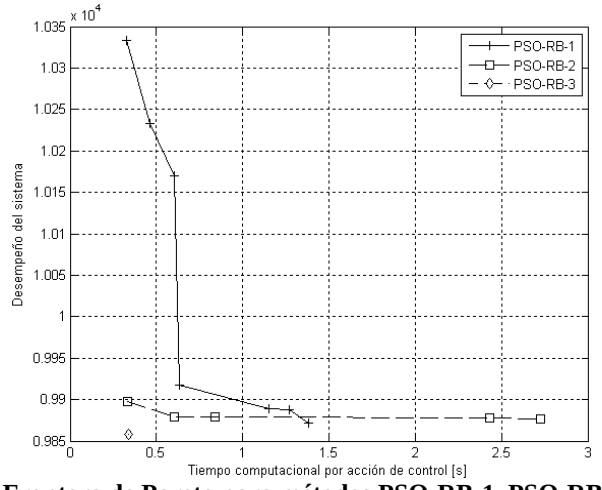


Figura 45: Frontera de Pareto para métodos PSO-RB-1, PSO-RB-2 y PSO-RB-3

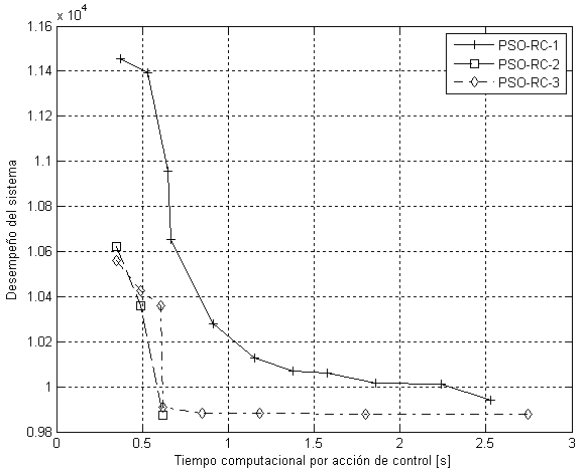


Figura 46: Frontera de Pareto para métodos PSO-RC-1, PSO-RC-2 y PSO-RC-3

Luego, para analizar el mejor método entre PSO-RA, PS-RB y PSO-RC, se grafican simultáneamente todas las fronteras de Pareto de estos métodos con sus mejores opciones de inicialización de población (Figura 47).

De la comparación de la Figura 47 se aprecia que PSO-RB-3 (utiliza reparación en la válvula de mezcla e inserta las soluciones anteriores desplazadas en la nueva población) domina a los demás métodos. Que una de las reparaciones que sesga la búsqueda entregue mejores resultados que la que no sesga, indica que la función objetivo penaliza más un régimen que otro. En este caso, penaliza más la demora en acercarse a la referencia (el transiente) que la capacidad de seguir perfectamente a la referencia. Esto es lógico, debido a la estructura cuadrática de la función objetivo.

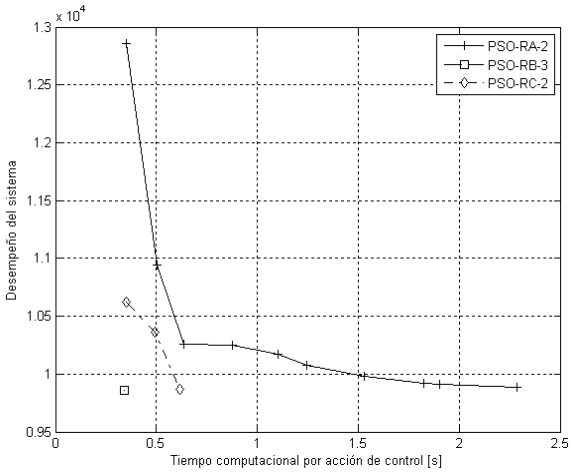


Figura 47: Frontera de Pareto para métodos PSO-RA-2, PSO-RB-3 y PSO-RC-2

2. <u>Comparación de representaciones factibles y estrategias de inicialización de soluciones en PSO</u>

Ahora se analizan las dos estrategias de representación factible de PSO. La frontera de Pareto de los métodos PSO-FA (la primera representación factible) se muestra en la Figura 48. En estos métodos se aprecia claramente que PSO-FA-1 es dominado en términos de Pareto por PSO-FA-2, y que PSO-FA-3 es dominado por PSO-FA-2 (al menos en un rango, mientras que en el resto

parecieran entregar los mismos resultados). Esto significa entonces que insertar las soluciones anteriores efectivamente ayuda al comportamiento de los algoritmos de optimización, y además, la inclusión de la solución anterior intacta ayuda más que la inserción de la solución anterior desplazada.

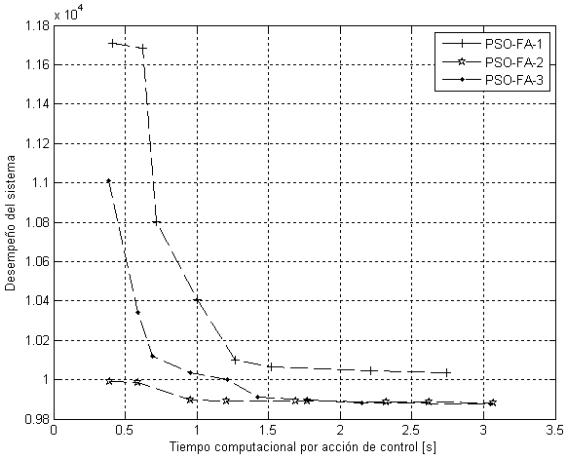


Figura 48: Fronteras de Pareto para los métodos PSO-FA-1, PSO-FA-2 y PSO-FA-3

Las fronteras de Pareto para la segunda representación factible utilizada en PSO (PSO-FB), que posee sentido físico, con las tres estrategias de inicialización de población, se presenta en la Figura 49. Estas fronteras de Pareto se ven anómalas, pues en el método PSO-FB-3 la Frontera solo posee un punto. Además, se produce un efecto distinto al de los demás métodos, y es que la solución que no incluye la mejor solución anterior supera a uno de los que sí la incluye. Sin embargo esto es sólo un efecto producido por la pequeña diferencia entre la exactitud de estos métodos y de sus distintos números de iteraciones y tamaño de población. Como se ve en el Anexo (Figura 104, Sección 13.1) para el método PSO-FB, el desempeño es casi constante a lo largo de todo el rango posible, lo que significa que la representación es bastante buena y no mejora al aumentar la población o las iteraciones. Además, como es consecuente físicamente se comporta mejor al incluir la solución desplazada, que es la que efectivamente debiera ser la solución del nuevo instante (si no hubiese perturbaciones y se utilizara un horizonte de predicción que alcance el régimen permanente) dada la naturaleza de horizonte deslizante del control predictivo.

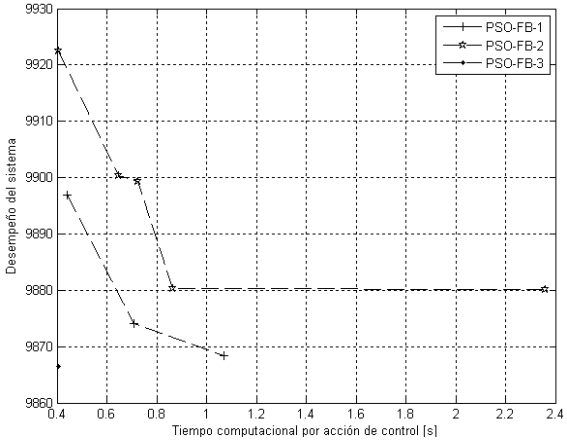


Figura 49: Fronteras de Pareto para los métodos PSO-FB-1, PSO-FB-2 y PSO-FB-3

Luego, para ver cual representación es la mejor se grafican en una misma figura las fronteras con sus respectivas mejores alternativas de inicialización de población (Figura 50). En esta figura se aprecia claramente que PSO-FB-3 es la mejor alternativa. Esto se argumenta por lo que ya se ha dicho, la representación de PSO-FB es consecuente en términos físicos, mientras que la representación de PSO-FA no lo es. Además, incluir la solución anterior desplazada capta de mejor modo las variaciones en los transientes que si se incluye la solución anterior intacta.

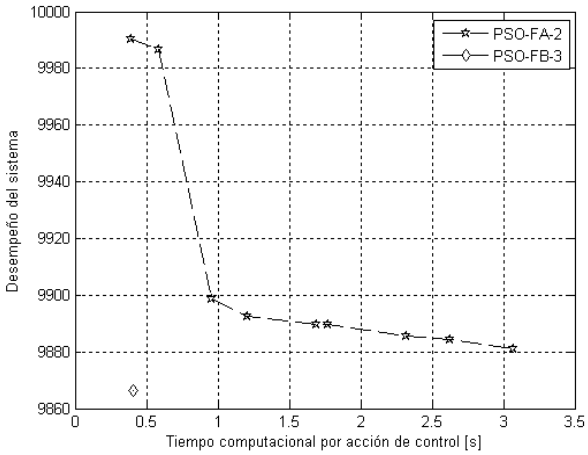


Figura 50: Fronteras de Pareto para los métodos PSO-FA-2 y PSO-FB-3

3. Comparación de estrategias de inicialización de soluciones en GA

A continuación se realiza el mismo análisis de los puntos anteriores, pero para los métodos GA-F. Sus fronteras de Pareto con las tres estrategias de inicialización de soluciones se presentan en la Figura 51. Se puede apreciar claramente que, a diferencia de los métodos PSO-RC, aquí no hay una clara mejora con la inclusión de la mejor solución anterior. Aunque para los métodos que demoran hasta los 0.5 seg. pareciera ser mejor no insertar dicha solución, a contar de ahí levemente parece ser mejor insertar la solución inicial.

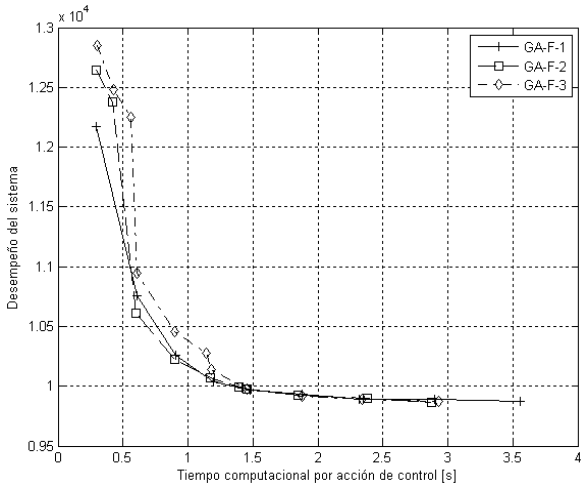


Figura 51: Frontera de Pareto para métodos GA-F-1, GA-F-2 y GA-F-3

El hecho que en PSO la inclusión de una buena solución local mejore los resultados y en GA no lo haga, se puede explicar debido a que la heurística de búsqueda de PSO favorece la búsqueda de soluciones cercanas a las que ya se tienen, mientras que GA no. Esto parece deberse fundamentalmente a que en PSO todos los miembros del enjambre se ven influenciados por *gbest*, que probablemente es la mejor solución del instante anterior, mientras que en GA sólo unos pocos se ven influenciados por esta mejor solución anterior mediante los operadores de crossover.

4. Comparación de estrategias basadas en PSO y GA

Finalmente se comparan todos los métodos. Para esto, se ponen en una sola frontera las mejores alternativas de acuerdo a cada una de las representaciones, según lo desarrollado en los puntos 1,2 y 3. Es decir, se grafican en una sola figura las fronteras de Pareto de los métodos, PSO-RB-3, PSO-FB-3 y GA-F-2. Se puede apreciar que el mejor método es PSO-RB-3 (estrategia con reparación de válvula de mezcla e inclusión de solución anterior desplazada), seguido muy de cerca por PSO-FB-3 (estrategia con representación factible con sentido físico e inclusión de solución anterior desplazada).

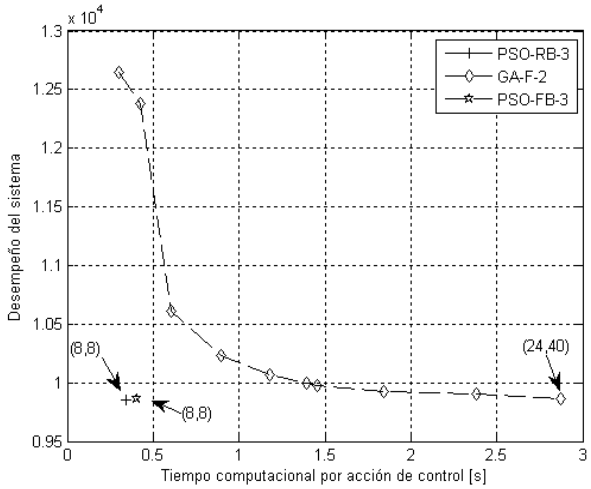


Figura 52: Fronteras de Pareto para los métodos PSO-FA-2 y PSO-FB-3

Una vez comparados todos los métodos, se procede a realizar la selección de parámetros para los mejores, es decir, para PSO-RB-3, GA-F-2 y PSO-FB-3. De acuerdo al análisis propuesto en 5.7.2, como en este proceso se tiene un límite máximo de 10 seg. para aplicar la acción de control, se debe escoger los parámetros que permitan encontrar las mejores soluciones dentro de dicho tiempo. Con ese criterio, se escogerían los siguientes parámetros: para PSO-RB-3, 8 partículas y 8 iteraciones; para PSO-FB-3, 8 partículas y 8 iteraciones; y para GA-F-3, 40 partículas y 24 iteraciones. El rendimiento de estos métodos se presenta en la Tabla 12.

Tabla 12: Estadísticas para los métodos PSO-RB-3, PSO-FB-3 y GA-F-2

Método	J		J1		J2		J3	
	Prom.	Std.	Prom.	Std.	Prom.	Std.	Prom.	Std.
PSO-RB-3 (8 part, 8 ite)	9857.8	37.80	9827.5	37.78	7.71	1.597	2	0
PSO-FB-3 (8 part, 8 ite)	9866.5	55.08	9836.3	55.07	7.949	1.510	2	0
GA-F-2 (24 ind, 40 ite)	9864.8	23.21	9834.7	23.22	6.393	0.083	2	0
Método	**Tiempo computacional**				**Fn. Objetivo evaluadas**			
	Prom.	**Std.**	**Min.**	**Max.**	**Prom.**	**Std.**	**Min.**	**Max.**
PSO-RB-3 (8 part, 8 ite)	0.342	0.120	0.0933	0.6679	140.86	54.03	30	226
PSO-FB-3 (8 part, 8 ite)	0.406	0.110	0.1894	0.5306	172.39	50.16	75	229
GA-F-2 (24 ind, 40 ite)	2.886	0.262	2.037	3.722	935.74	113.9	556	1274

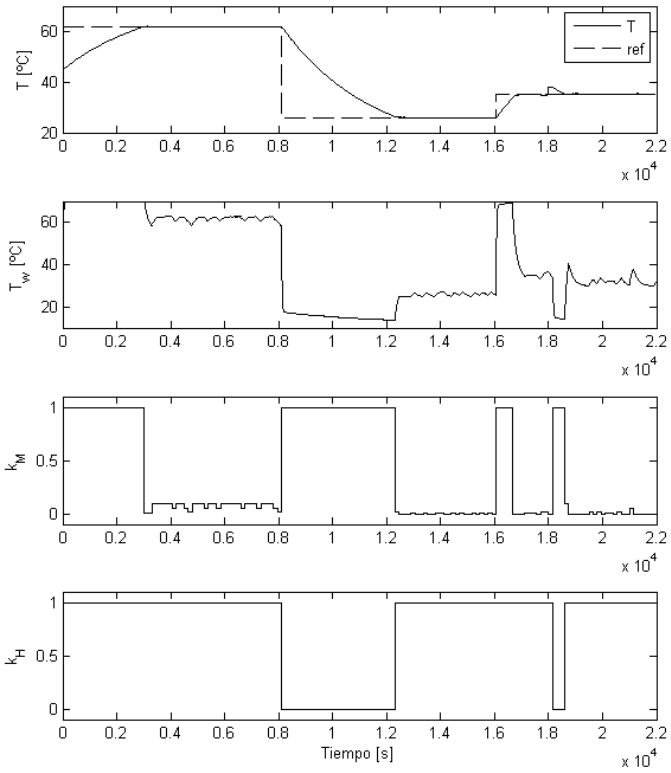


Figura 53: Respuesta del sistema con un controlador predictivo híbrido con PSO-RB-3, 8 partículas y 8 iteraciones.

Se concluye de la Tabla 12 que el método PSO-RB-3 es el mejor, puesto que es el que logra mejores soluciones, y en un menor tiempo computacional, lo que es consistente con lo mostrado en la Figura 52. Finalmente, en la Figura 53 se muestra el comportamiento dinámico de este método, con sus parámetros óptimos, y se aprecia que se obtiene un buen seguimiento con un adecuado esfuerzo de control.

Respecto de los métodos PSO-RB-3 y PSO-FB-3, ya ha sido mencionado que el hecho que las combinaciones de parámetros óptimas sean las que en teoría sean las que debieran entregar una peor calidad (8 iteraciones y 8 partículas), habla claramente de la bondad de la estrategia, ya que incrementar el tamaño de la población o el número de iteraciones no entrega un gran beneficio. En efecto, la desviación estándar de la calidad de las soluciones es grande respecto al beneficio de aumentar las poblaciones o el número de iteraciones (ver Anexo, Figura 101 y Figura 104, Sección 13.1), y en este caso se dio que debido a la variabilidad de la calidad de las soluciones, resultaron ser mejores aquellos casos que debieron ser peores. Luego, como la mejora al aumentar las poblaciones o iteraciones (y por consiguiente el esfuerzo computacional) es tan pequeña, se justifica plenamente el uso de los parámetros que hagan más rápido a los métodos.

6.1.4 Análisis de resultados y conclusiones

En esta sección se ha analizado distintas estrategias para la representación de soluciones para resolver el control predictivo utilizando algoritmos evolutivos, en particular, optimización por enjambre de partículas y algoritmos genéticos.

En PSO, se ha probado tres representaciones. En la primera, se utiliza una coordenada por acción de control real. Al utilizar esta representación pueden aparecer soluciones infactibles, y para solucionar este problema se plantean 3 opciones de reparación: una repara la posición de una válvula en caso de ser necesario, otra que repara la posición de la otra válvula, y una tercera que se ha propuesto para evitar el sesgo de las dos anteriores y consiste en utilizar aleatoriamente la primera o segunda opción. Los resultados muestran que las opciones de reparación que sólo modifican una de las válvulas efectivamente sesgan la búsqueda, y se obtienen resultados no del todo satisfactorios en el régimen que no corresponde a la zona de sesgo de la reparación. En este sentido la opción que utiliza aleatoriamente las reparaciones de una o la otra válvula, mejora el rendimiento en las zonas de mala performance de las otras alternativas, pero deteriora el desempeño en las zonas donde se comportan bien. Por ejemplo, la reparación A tiene un muy buen seguimiento en régimen permanente, pero es muy malo en régimen transiente. Por su parte la reparación B se comporta a la inversa. Al utilizar la opción C, se tiene buen seguimiento tanto en el régimen permanente como en el transiente, pero son de peor calidad que el seguimiento de régimen permanente de la opción A y el seguimiento de régimen transiente de la opción B. Esto se debe a que la búsqueda se ve deteriorada ya que no busca siempre por el mismo espacio. Finalmente, esto se refleja en que los mejores indicadores de desempeño no los obtenga esta opción, sino que alguna de las opciones puras que sesga la búsqueda por donde es más peso se da al indicador de desempeño. En este caso la reparación B sesga la búsqueda por las soluciones que aparecen en el régimen transiente, y estas son las que más importancia tienen en la función de desempeño debido a su naturaleza cuadrática (al ser transiente se encuentran lejos de la referencia y al ser cuadrático cobran mucha más importancia que el régimen permanente donde las salidas están cerca de las referencias), y por lo tanto, es esta reparación la que aparece como de mejor calidad.

Este análisis parece indicar que es más conveniente utilizar funciones de reparación puras (que no cambien aleatoriamente entre una opción y otra) pues permiten una búsqueda de buena calidad, y que en lo posible sesgue la búsqueda hacia el tipo de soluciones que se da en las zonas de operación que más importancia tienen en los indicadores de desempeño. Sin embargo, esto se siente como hacer trampa, porque es como buscar la forma de tener un buen indicador, y no necesariamente el mejor desempeño posible. Inspirado en esto, se plantea como trabajo futuro

investigar acerca de la posibilidad de utilizar distintos enfoques de reparación en distintas zonas de operación, y de ese modo obtener rendimientos óptimos. Por ejemplo, utilizar la reparación A en el régimen permanente y la reparación B en el régimen transiente, y además investigar alguna estrategia óptima de cambio entre estas dos alternativas.

Los esquemas de reparación propuestos se basan en evolución Baldwiniana, es decir que reparan las soluciones sólo para evaluar las funciones objetivo, pero no modifican las partículas o individuos que las codifican. Queda como trabajo futuro evaluar la bondad de las estrategias basadas en evolución Lamarckiana, es decir, aquellas que no solo reparan las soluciones para evaluar las funciones objetivos, sino que además reparan la codificación en los individuos o partículas.

La segunda y tercera representaciones están basadas en la representación factible utilizada en GA que es propuesta en Causa *et al*. (2008). Se aprecia que la representación factible A no es consecuente en términos físicos, en el sentido que no está ordenada de frío a caliente tal como la codificación de las soluciones, y por lo tanto se propone una nueva representación factible B, que sí está ordenada. Los resultados muestran efectivamente que en PSO este orden es relevante, y por lo tanto la representación B es mucho mejor en términos de los resultados obtenidos que la representación A.

En GA se utiliza una única representación que es factible, y es equivalente a la representación factible A utilizada en PSO, pero, a diferencia de esta no se ve afectada por el hecho que las soluciones no estén ordenadas.

Este orden es relevante en PSO puesto que sus funciones de actualización hacen que sea mucho más probable pasar a una zona contigua en el espacio de búsqueda que a una posición lejana, y por lo tanto no tiene mucho sentido que en el espacio de búsqueda la acción de control más caliente sea contigua a la más fría. En GA (al menos la implementación actual) este orden no es relevante pues si una coordenada posee cierto valor, digamos 2, es igualmente probable que al cabo de una generación esta coordenada cambie a 1, a 3 ó a 6. Y en este sentido, el orden de las acciones de control no es relevante para la codificación en GA.

Otro aspecto estudiado es la inicialización de la población, que considera que esta puede ser 100% aleatorio, o además incluyendo una copia de la mejor solución en el instante anterior, pudiendo ser una copia intacta, o desplazada de acuerdo a la estrategia de horizonte deslizante del control predictivo. Se encuentra que en prácticamente todos los métodos la inserción de esta mejor solución anterior ayuda tanto en la calidad de las soluciones como en el tiempo computacional requerido. Se aprecia además que en aquellos métodos que son consecuentes físicamente (por ejemplo la primera representación que codifica cada válvula, o la representación factible B que es reordenada para que siga el orden de más frío a más caliente) se benefician más de insertar la solución anterior desplazada, mientras que aquellos que no tienen esta consecuencia, se benefician más de la solución intacta. Se mencióna además, que este efecto podría amplificarse o reducirse, si el tiempo de muestreo aumentara o disminuyera, de acuerdo a lo analizado en la Sección 5.5. En efecto, en dicha sección se menciona que teóricamente la estrategia que incluye las soluciones desplazadas debiese sacar más provecho de esto, respecto de la estrategia que sólo incluye la solución intacta, al utilizar tiempos de muestreo más bien grandes. En este sentido, sería interesante realizar pruebas utilizando tiempos de muestreo menores y mayores, más para confirmar esta predicción que en beneficio de la aplicación.

Por otra parte, se aprecia que el algoritmo que menos se beneficia de la inserción de soluciones anteriores es GA, pues del mismo modo como es igualmente probable de pasar de una solución a otra que no es contigua, no utiliza información de vecindades, como lo que constituye la información que entrega la inserción de la mejor solución anterior. Pero en este caso, pareciera ser que la causa principal es que en PSO, con una gran probabilidad, todas las partículas utilizarán la información de la solución anterior mediante *gbest*, pues con una alta probabilidad la solución insertada será el *gbest*, mientras que en GA sólo unas pocas utilizarán esta información mediante operadores de crossover. Basado en esto, se plantea como trabajo futuro la posibilidad de explorar la inclusión de más copias de estas soluciones anteriores en GA.

Se ha visto que PSO utiliza la información de las vecindades, mientras que GA no la utiliza mucho. Esto se refleja en el hecho que las implementaciones basadas en PSO se benefician altamente de utilizar las mejores soluciones anteriores mientras que GA prácticamente no saca provecho de esto. Para fortalecer la idea de que se debe a que GA no utiliza la información de las vecindades, y que PSO sí, se verá a continuación que las propiedades dinámicas de GA no cambian a lo largo de los distintos regímenes de operación del sistema, mientras que las de PSO sí cambian. De la Figura 54 a la Figura 59 se presentan gráficos del número de evaluaciones de función objetivo y de la iteración en la que se encuentra la solución que finalmente será la óptima, a lo largo de una ejecución típica del reactor batch donde se han realizado los análisis, para los métodos GA-F-1, GA-F-3, PSO-FB-1 y PSO-FB-3, al utilizar 8 individuos e iteraciones, y 40 individuos e iteraciones. De estas figuras se puede apreciar que estas propiedades prácticamente no dependen del régimen de operación en GA, mientras que en PSO claramente varían con la zona de operación. Esto confirma que PSO hace uso de las vecindades mientras que GA no, y por lo tanto PSO hace un mejor uso de las soluciones anteriores y necesitan representaciones con sentido físico.

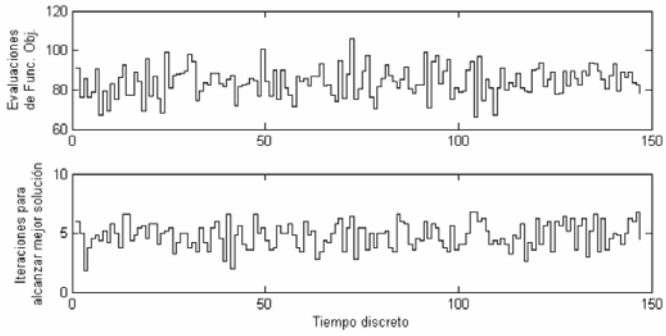


Figura 54: Comportamiento dinámico de GA-F-1 usando 8 individuos y 8 iteraciones

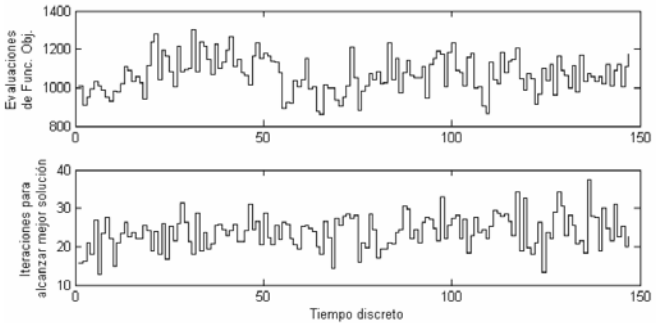


Figura 55: Comportamiento dinámico de GA-F-1 usando 40 individuos y 40 iteraciones

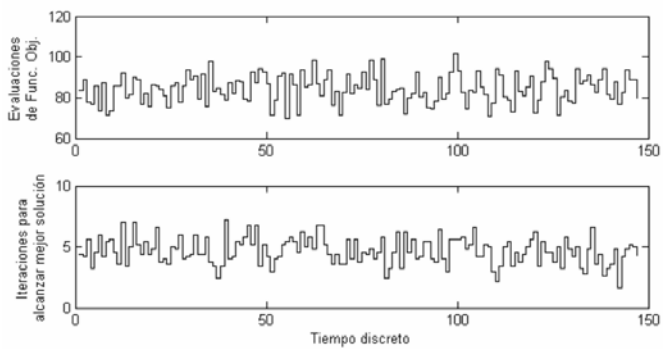


Figura 56: Comportamiento dinámico de GA-F-3 usando 8 individuos y 8 iteraciones

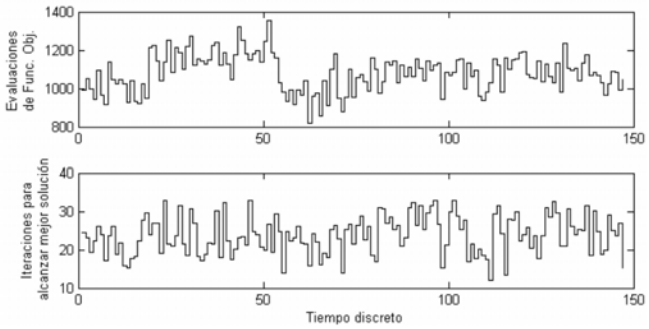


Figura 57: Comportamiento dinámico de GA-F-3 usando 40 individuos y 40 iteraciones

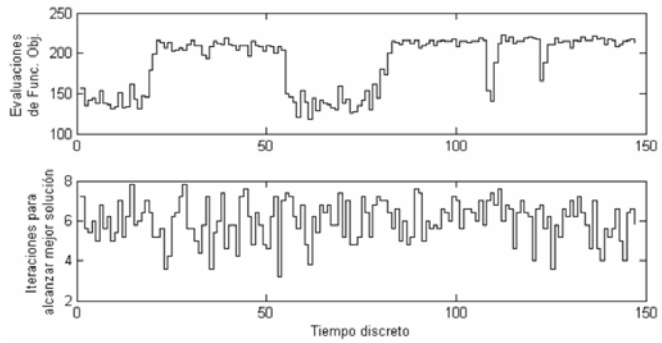


Figura 58: Comportamiento dinámico de PSO-FB-1 usando 8 individuos y 8 iteraciones

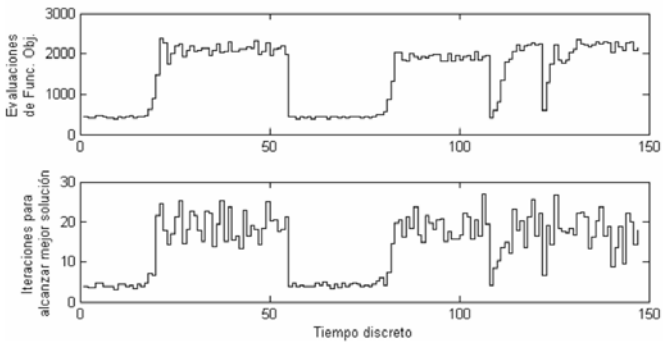


Figura 59: Comportamiento dinámico de PSO-FB-1 usando 40 individuos y 40 iteraciones

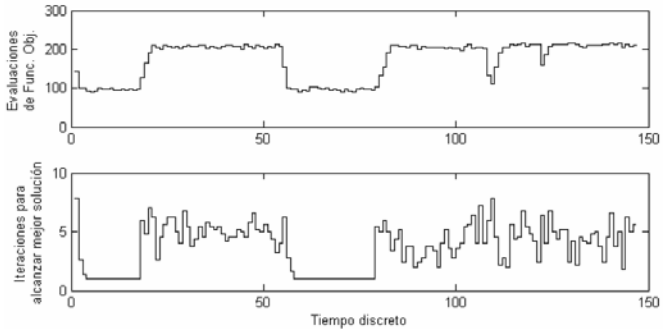


Figura 60: Comportamiento dinámico de PSO-FB-3 usando 8 individuos y 8 iteraciones

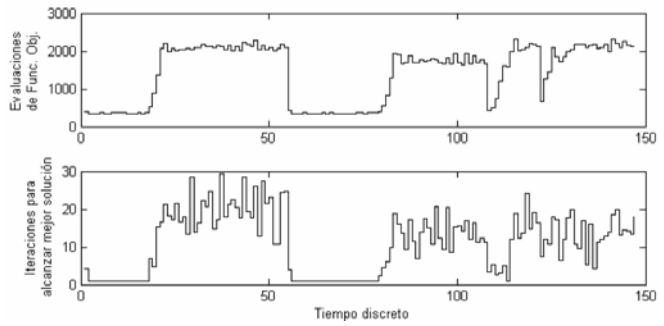


Figura 61: Comportamiento dinámico de PSO-FB-3 usando 40 individuos y 40 iteraciones

Uno de los últimos aspectos analizados para la ejecución del control predictivo híbrido en tiempo real mediante algoritmos evolutivos es el almacenamiento de soluciones ya visitadas. De acuerdo a las figuras anteriores, es claro que en ciertos instantes el algoritmo evalúa menos soluciones que en otros, esto se debe a que se van repitiendo soluciones y por lo tanto no debe evaluarlas de nuevo. Esto se ve reflejado en el tiempo computacional, ya que la mayor parte del esfuerzo está dado por la evaluación de funciones objetivo. Además, en el Anexo (Sección 13.1, Figura 88 a Figura 105) se aprecia que la pendiente del tiempo que demoran los métodos disminuye con el número de iteraciones, y esto se debe a que al utilizar más iteraciones, se almacenan cada vez más soluciones, y en cantidades relativas se evalúan menos soluciones. Finalmente, este aspecto es mucho mejor utilizado en PSO que en GA, pues como utiliza mucho más la información de las zonas cercanas posee una mayor convergencia y debe evaluar muchas menos soluciones que GA.

Respecto de encontrar la mejor alternativa para controlar el reactor, PSO-RB-3 parece ser la mejor alternativa, dentro de las basadas en algoritmos evolutivos, ya que permite encontrar soluciones de muy buena calidad, con un esfuerzo computacional mínimo. Sin embargo, la diferencia con PSO-FB-3 es mínima, y considerando la desviación estándar de ambos métodos, cualquiera podría ser el mejor.

Por otra parte, BB es mejor estrategia que las basadas en algoritmos evolutivos, pues alcanza a resolver los problemas de optimización en el tiempo permitido, y más encima, encontrando la solución óptima de forma garantizada. Sin embargo con mayores restricciones de tiempo (efectivamente un tiempo menor para encontrar las acciones de control, o bien procesador mas limitado), se debiera recomendar otras opciones, tales como PSO-RB-3 o PSO-FB-3, que tienen excelentes rendimientos, incluso tiempos muy reducidos.

Un resultado interesante que se obtiene de los experimentos realizados consiste en que la bondad de las estrategias utilizadas puede depender de la estructura de la función objetivo utilizada. En efecto, en la Sección 6.1.3.2.2 se muestra que una estrategia de reparación (reparación B) supera a las demás porque permite que la búsqueda sea mejor cuando la salida esta lejos de la referencia que cuando está cerca, es decir, justamente la búsqueda es mejor en las zonas más penalizadas en la función objetivo. Siguiendo la misma lógica, si se utiliza una función objetivo con una estructura distinta, otra estrategia de reparación puede resultar la mejor. Del mismo modo, esto puede darse no solo para estrategias de reparación, sino que podría darse para cualquiera de las estrategias a utilizar en un algoritmo evolutivos, y debido a esto resulta interesante en un futuro analizar como influye (y cuándo influye) la estructura de la función objetivo en la calidad de las estrategias para los distintos aspectos relevantes en algoritmos evolutivos para su aplicación en control predictivo híbrido de sistemas no lineales.

Con un énfasis en la relación de calidad con la calidad de los métodos, los análisis realizados mediante los enfoques multi-objetivo son bastante útiles, y sirven efectivamente para entender los efectos de las modificaciones que se puedan hacer a los métodos. Sin embargo, existe el problema que el ruido proveniente de la naturaleza estocástica de los métodos produce en ciertas instancias que sea difícil interpretar los resultados (en particular las fronteras de Pareto que tienen pocos puntos). En ese contexto en particular, utilizar no sólo la frontera, sino que las soluciones sub-óptimas también, entrega más información valiosa que permite interpretar de mejor modo los resultados.

Finalmente, la utilización del enfoque multi-objetivo para utilizarlo como sintonización de los métodos posee ciertas dificultades debidas a la presencia de ruido en los métodos. Por lo tanto, para escoger adecuadamente los parámetros esta selección se debiese realizar considerando también los puntos sub-óptimos, de modo de entender de mejor modo las tendencias de desempeño de los métodos respecto del número de iteraciones y del tamaño de la población. Inspirado en esto, como trabajo futuro se propone investigar la alternativa de proponer una estructura funcional (por ejemplo una exponencial) para la dependencia de la calidad de las soluciones y del tiempo computacional en función del número de iteraciones, y con los datos realizar una interpolación para encontrar estas curvas. Luego con éstas se podría encontrar las fronteras de Pareto, lo que permitiría encontrar los parámetros óptimos con una mayor confianza, y tal vez permita reducir la cantidad de ejecuciones que hay que realizar de los algoritmos, tal como está diseñado el análisis en la actualidad.

6.2 El reactor batch con entradas mixtas

El funcionamiento de este segundo reactor batch es muy similar al descrito anteriormente, siendo diferente tan sólo en las posiciones que pueden tomar las válvulas. En este caso la válvula de mezcla es continua, es decir, puede tomar cualquier valor en el intervalo $[0,1]$.

La temperatura del agua fresca de entrada T_{in} depende de dos entradas: las posiciones de las válvulas *on/off* k_C y k_H. Sin embargo, sólo existen dos modos de operación. En el primero, definido por $k_C = 1$ y $k_H = 0$, el agua de entrada es helada ($T_{in} = T_C = 12°C$), mientras que en el segundo caso, definido por $k_C = 0$ y $k_H = 1$, el agua de entrada es caliente ($T_{in} = T_C = 75°C$).

La tasa de la mezcla entre agua fresca con agua de recirculación es controlada por una tercera entrada, la posición de una válvula de mezcla k_M. Esta válvula puede tomar valores en el intervalo $[0,1]$. Se está trabajando entonces con un sistema multi-variable entero mixto con dos entradas discretas (k_H y k_C), una entrada continua k_M y dos salidas mediales (T y T_w).

De acuerdo a la revisión bibliográfica realizada, se desconoce otras estrategias de control que hayan sido aplicadas a este mismo proceso. Sin embargo, Karer *et al*. (2008), desarrolla una estrategia de control predictivo auto-adaptativo para un reactor batch con las mismas entradas consideradas en este trabajo, pero donde en el fluido cuya temperatura se regula, se produce una reacción exotérmica, situación que no se da en el proceso aquí considerado.

A continuación, se presenta el modelo híbrido difuso para la estrategia de control predictivo. Luego, se presenta la estrategia de control predictivo basada en PSO con especial énfasis en el diseño de las representaciones, inicialización de soluciones y estrategias de manejo de restricciones. Posteriormente, se presentan los estudios por simulación, donde se comparan los métodos y se sintoniza uno de ellos mediante el análisis multi-objetivo propuesto en 5.7.2. Finalmente se presenta un análisis y conclusiones de los resultados obtenidos.

6.2.1 Modelo híbrido difuso del reactor batch con entradas mixtas

Para tener una primera aproximación al control predictivo del reactor batch con variables mixtas, se utiliza el mismo modelo difuso del proceso que considera sólo entradas discretas (ver Sección 6.1.1). Luego de algunas pruebas es posible notar que las zonas de operación del sistema controlado con acciones de control mixtas son muy similares a las del sistema con acciones discretas. Por lo tanto, el modelo utilizado anteriormente es representativo de la operación del nuevo caso, y por lo tanto se utiliza definitivamente como modelo del reactor batch con entradas mixtas.

6.2.2 Diseño de la estrategia de control

Se propone una estrategia de control predictivo híbrido basado en una función objetivo con la misma estructura que la utilizada en el caso con entradas discretas en la Sección 6.1.2, pero utilizando distintos pesos.

$$J = w_1 \sum_{h=1}^{N_y} \left(T\left(k+h\right) - T_{ref}\left(k+h\right) \right)^2 + w_2 \sum_{h=1}^{N_u} \left| \Delta k_M \left(k+h-1\right) \right| k_H \left(k+h-1\right) + w_3 \sum_{h=1}^{N_u} k_C \left(k+h\right) k_H \left(k+h-1\right) \tag{6.18}$$

donde $w_1 = 1/15, w_2 = 0.42, w_3 = 10$. En este estudio se consideran un horizonte de predicción igual al de control $N_u = N_y = N$. Por otra parte, al igual que en el caso con acciones de control discretas, el tiempo de muestreo del modelo de predicción es $T_s = 10\,[s]$. Por lo tanto, el problema de optimización debe ser resuelto en 10 [s]. Además, se considera que las entradas sólo pueden cambiar cada 15 tiempos de muestro, por lo tanto la acción de control será mantenida constante durante un tiempo de muestreo de control $T_{cs} = 150\,[s]$.

Como se trata de problemas de minimización, en la aplicación de PSO para resolver este problema se utiliza como función de *fitness*:

$$F = \frac{1}{J} \tag{6.19}$$

que asegura que las mejores soluciones, es decir con una función objetivo menor, posean un mayor *fitness*.

El conjunto de acciones de control posibles son todas aquellas que satisfacen:

- $k_C = 1$ y $k_H = 0$, ó bien, $k_C = 0$ y $k_H = 1$
- $k_M \in [0,1]$.

Entonces, en cada instante que se debe calcular la acción de control, se ejecuta una realización de un algoritmo de optimización para resolver el problema correspondiente al control predictivo. La solución a dicho problema representa la secuencia de acciones de control óptimas que minimiza la funcional dada por la ecuación (6.18), y se aplica al sistema sólo las acciones de control correspondientes al instante k de acuerdo a la estrategia de horizonte deslizante.

Para resolver el problema de optimización se considera un enfoque tradicional basado en *branch and bound* con programación secuencial cuadrática. Además, se consideran tres enfoques basados en variaciones y adaptaciones de PSO.

A continuación, se presenta el diseño de tres variaciones de PSO para resolver el problema de control predictivo híbrido en base a los aspectos introducidos en el capítulo 5.

6.2.2.1 Diseño del controlador predictivo basado en PSO

Respecto a lo presentado en el capítulo 5, los aspectos a considerar para la aplicación de PSO al control predictivo híbrido del reactor batch son: la representación de soluciones, manejo de restricciones e inicialización de soluciones candidatas.

Representación de las soluciones

Las decisiones de control en el problema de control predictivo son las posiciones de las tres válvulas a lo largo de todo el horizonte de control N_u. Como las dos válvulas discretas toman siempre valores opuestos, estas dos pueden ser representadas por una sola válvula equivalente k_{CH} que puede tomar los siguientes valores.

$$k_{HC} = \begin{cases} 1 & \text{si } k_H = 1 \wedge k_C = 0 \\ 0 & \text{si } k_H = 0 \wedge k_C = 1 \end{cases} \tag{6.20}$$

De este modo, en cada instante las acciones de control quedan completamente determinadas por dos variables, una variable discreta k_{HC} y otra continua k_M. Basado en esto se propone utilizar partículas x_j de dimensión $2N$, que representan las acciones de control en los siguientes N pasos de control.

$$\begin{array}{ccccccc} x = \Big[& x_1 & x_2 & \cdots & x_{2N_u-1} & x_{2N_u} & \Big] \\ & \downarrow & \downarrow & \cdots & \downarrow & \downarrow & \\ & \underbrace{k_M(k) \quad k_{CH}(k)}_{u(k)} & & & \underbrace{k_M(k+N_u-1) \quad k_{CH}(k+N_u-1)}_{u(k+N_u-1)} & & \end{array} \tag{6.21}$$

Como la válvula de mezcla k_M es continua y puede tomar valores en el intervalo $[0,1]$, las coordenadas que representan a estas variables también son continuas y al evaluar la función objetivo se utiliza el mismo valor de la coordenada correspondiente:

$$k_M(k+j) = x_{2j+1} \tag{6.22}$$

La situación es distinta cuando las coordenadas representan a las variables discretas. Para su manejo se consideran tres alternativas: 1) las coordenadas son continuas y se truncan al evaluar la función objetivo, 2) las coordenadas se truncan al momento de actualizar las posiciones, y 3) se utiliza el algoritmo PSO binario.

Opción 1: Coordenadas continuas

Las coordenadas que representan las acciones de control binarias son continuas, y al momento de evaluar la función objetivo se truncan al valor que corresponda.

$$k_{CH}(k+j) = \begin{cases} 0 & if\ x_{2j+2} < 0.5 \\ 1 & if\ x_{2j+2} \geq 0.5 \end{cases} \tag{6.23}$$

Opción 2: Coordenadas discretas

En esta alternativa se utilizan las funciones típicas de actualización de PSO, pero al momento de calcular las nuevas posiciones, éstas se truncan al valor más cercano (0 ó 1). Esto se logra

mediante una modificación a la operación de actualización de la posición de las partículas para las coordenadas que representan a las variables binarias.

$$x_{ij}(g+1) = \begin{cases} x_{ij}(g) + v_{ij}(g+1) & \text{si } j \text{ corresponde a } k_M \\ redondear(x_{ij}(g) + v_{ij}(g+1)) & \text{si } j \text{ corresponde a } k_{HC} \end{cases} \quad (6.24)$$

Luego, al momento de evaluar las funciones objetivo la válvula binaria toma el mismo valor en la coordenada correspondiente de la partícula.

$$k_{CH}(k+j) = x_{2j+2} \quad (6.25)$$

Opción 3: PSO binario

La tercera opción consiste en utilizar para las coordenadas que representan a las variables binarias la función de actualización del algoritmo de PSO binario tal como es descrito en la Sección 4.4.4.1. De este modo, la función de actualización de las posiciones es:

$$x_{ij}(g+1) = \begin{cases} x_{ij}(g) + v_{ij}(g+1) & \text{si j corresponde a } k_M \\ \begin{cases} 1 \text{ si } \rho \le s\left(v_{ij}(t)\right) \\ 0 \text{ si no} \end{cases} & \text{si j corresponde a } k_{HC} \end{cases} \quad (6.26)$$

donde la función de probabilidad en función de la velocidad es:

$$s\left(v_{ij}(t+1)\right) = \frac{1}{1+e^{-v_{ij}(t)}} \quad (6.27)$$

y ρ es un número aleatorio distribuido uniformemente en el intervalo $[0,1]$.

Las variables binarias son las que corresponden a k_{HC} y de este modo las coordenadas correspondientes toman también valores binarios. Por lo tanto, al evaluar la función objetivo, el valor de k_{HC} a lo largo del horizonte de predicción está dado como:

$$k_{CH}(k+j) = x_{2j+2} \quad (6.28)$$

Estas tres variantes se denotan del siguiente modo: por el prefijo R cuando se utilizan partículas continuas para las variables discretas (R-PSO), por el prefijo N cuando las partículas sólo pueden utilizar valores enteros y se utiliza la estructura clásica del algoritmo (N-PSO), y finalmente por el prefijo B cuando se utiliza la actualización del el algoritmo PSO binario (B-PSO).

Manejo de restricciones

De acuerdo a lo presentado en la Sección 6.2.1 la única restricción en este proceso es sobre las válvulas k_C y k_H, y dicta que siempre deben tomar valores opuestos. De acuerdo a la

representación propuesta con la válvula equivalente k_{HC}, las soluciones candidatas son siempre factibles, y en ese contexto el manejo de restricciones se realiza mediante representación factible.

Inicialización de las soluciones

Tal como en el caso con entradas discretas, para acelerar el proceso de convergencia y/o dar un buen punto de partida al proceso de optimización, se propone insertar dentro de la población la solución del problema de optimización del instante anterior. Dentro de esta estrategia se manejan dos variantes, la primera, insertar esta solución intacta, o bien desplazada en un instante de control de acuerdo a la estrategia de horizonte deslizante, tal como han sido presentadas en 5.5.

Las implementaciones que no utilizan esta estrategia se denotan con el sufijo 1 (por ejemplo: R-PSO-1), las que utilizan la inserción de la solución anterior intacta se denotan por el sufijo 2 (por ejemplo N-PSO-2), y finalmente, las que utilizan la solución desplazada, utilizan el sufijo 3 (por ejemplo B-PSO-3).

6.2.3 Resultados por simulación

En esta sección se analiza la aplicación de algoritmos convencionales y algoritmos evolutivos para resolver el problema de programación entera mixta asociado al problema de optimización del control predictivo híbrido.

Se considera el mismo patrón utilizado en la Sección 6.1.3, con un horizonte de predicción $N = 5$. Como medida de la calidad de la respuesta del sistema se utiliza una función de desempeño con las mismas componentes que la utilizada en el controlador predictivo, pero que no considera predicciones, sino que los resultados reales en cada tiempo de muestreo. Es decir:

$$\begin{aligned} J = {} & w_1 \sum_{h=1}^{K} \left(T\left(k+h\right) - T_{ref}\left(k+h\right) \right)^2 + w_2 \sum_{h=1}^{K} \left| \Delta k_M \left(k+h-1\right) \right| k_H \left(k+h-1\right) + \\ & w_3 \sum_{h=1}^{K} k_C \left(k+h-1\right) k_H \left(k+h-1\right) + \end{aligned} \tag{6.29}$$

donde K es el número total de tiempos de muestreo considerados en el estudio. Además, se considera el desempeño en cada uno de los objetivos que componen la función de desempeño, que se desglosan como:

$$\begin{aligned} J_1 &= \sum_{h=1}^{K} \left(T\left(k+h\right) - T_{ref}\left(k+h\right) \right)^2 \\ J_2 &= \sum_{h=1}^{K} \left| \Delta k_M \left(k+h-1\right) \right| k_H \left(k+h-1\right) \\ J_3 &= \sum_{h=1}^{K} k_C \left(k+h\right) k_H \left(k+h-1\right) \end{aligned} \tag{6.30}$$

donde J_1 corresponde al error de seguimiento, J_2 mide el esfuerzo de control de la válvula de mezcla, y J_3 penaliza el de agua caliente a agua fría.

Para obtener las estadísticas de los diversos métodos, se considera 20 réplicas para cada una de las distintas condiciones en que son probados (distintos parámetros, estrategias de inicialización de población, etc.). Además, los estudios han sido realizados con computadores Mac OS X, v10.5.6, con procesadores Core 2 Duo de 2.66 GHz y 4 GB de memoria RAM.

6.2.3.1 Enfoques basados en estrategias convencionales

Primero se analiza la respuesta del sistema cuando el problema de optimización es resuelto mediante *branch and bound* donde los sub-problemas continuos no lineales resultantes son resueltos mediante programación secuencial cuadrática (ver Anexo, Sección 11).

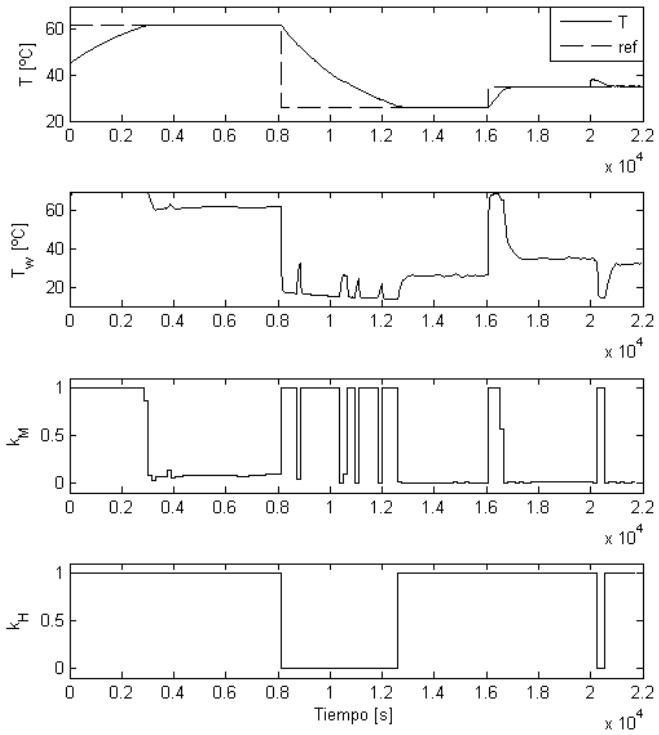


Figura 62: Respuesta del sistema con un controlador predictivo híbrido con branch and bound y programación no lineal con puntos iniciales al azar.

Es posible apreciar (ver Figura 62) que en términos generales el sistema logra un buen seguimiento, y en un tiempo computacional aceptable (ver Figura 63). Sin embargo, en ciertos instantes claramente se queda estancado en soluciones locales (ver entre los 8000 y 12000 segundos en la Figura 62). Para evitar esto, se prueba la alternativa de que el punto inicial de búsqueda sea la mejor solución del instante anterior, tal como se propone para los algoritmos evolutivos. La respuesta dinámica del sistema en este caso se presenta en la Figura 64, donde se observa que en un principio el sistema logra un excelente seguimiento con un pequeño esfuerzo de control, sin embargo, al cambiar la referencia el sistema se queda estancado en la solución local dado por la mejor solución encontrado en el instante anterior.

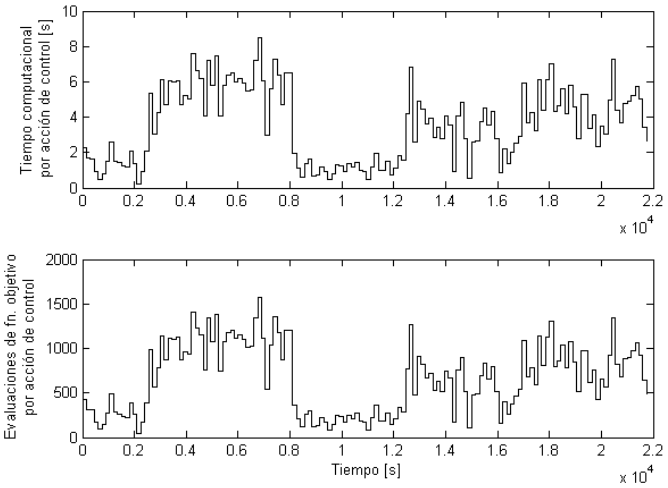


Figura 63: Esfuerzo computacional para BB+NLP a lo largo de la operación del sistema.

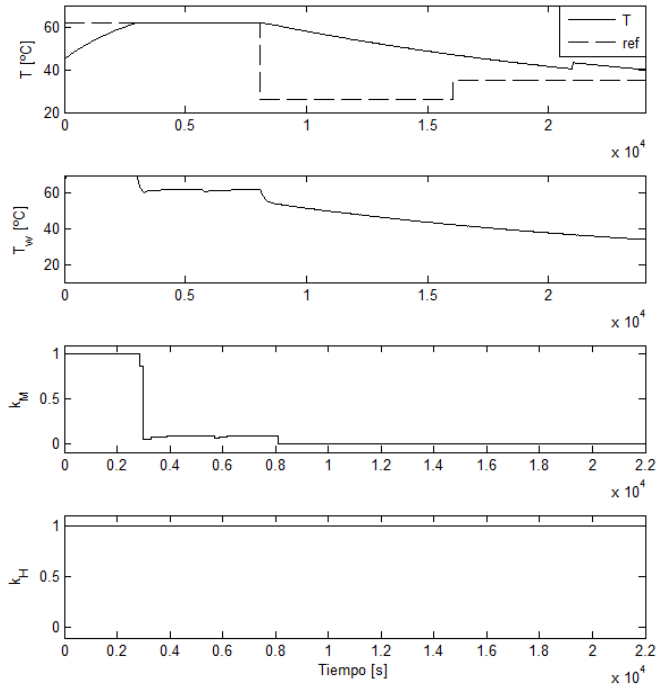


Figura 64: Respuesta del sistema con un controlador predictivo híbrido con branch and bound y programación no lineal con puntos iniciales iguales a solución anterior.

Es entonces claro que esta estrategia de BB+NLP no es completamente adecuada dado que tiene problemas con los óptimos locales.

Se plantea entonces una nueva estrategia de enumeración explícita con programación no lineal (EE+NLP) parar evitar en alguna medida el estancamiento en óptimos locales. El método consiste en probar todas las combinaciones posibles de las variables binarias (válvula k_{HC} a lo largo del horizonte de control), es decir se realiza una enumeración explícita de las opciones definidas por las variables binarias, y manteniendo esos valores fijos se resuelven problemas de programación no lineal para las variables continuas (válvula k_M a lo largo del horizonte de control), encontrando sus valores óptimos para una combinación fija de las variables discretas. Posteriormente se comparan las soluciones para cada combinación de variables binarias con sus

respectivas variables continuas óptimas, entonces se escoge la que obtenga la mejor función objetivo y esa es la que se considera como resultado del problema de optimización. Finalmente, se aplican sólo las soluciones que correspondan al instante k, de acuerdo a la estrategia de horizonte deslizante.

Se prueba dos variantes, al igual que en el caso de BB+NLP. La primera alternativa considera que los puntos iniciales de búsqueda para las distintas realizaciones de NLP son aleatorios. La segunda alternativa considera que estos puntos iniciales son los óptimos que fueron encontrados en el instante anterior. Las respuestas dinámicas del sistema para los dos casos anteriores se presentan en la Figura 65 y en la Figura 67 , mientras que sus esfuerzos computacionales se presentan en la Figura 66 y en la Figura 68. Se observa que el seguimiento utilizando estas dos estrategias es óptimo, con un mínimo esfuerzo de control (al compararlo con BB+NLP), sin apreciarse diferencias considerables entre ambas estrategias. Esto cambia al analizar el esfuerzo computacional, donde se aprecia que la estrategia que da un punto de partida fijo es más lento que la otra estrategia. Esto se puede explicar porque si bien este puede ser un buen punto de partida para las variables binarias que coinciden las óptimas en el instante anterior, al considerar distintas variables binarias, el punto entregado es un mal punto de partida, en promedio peor que uno aleatorio.

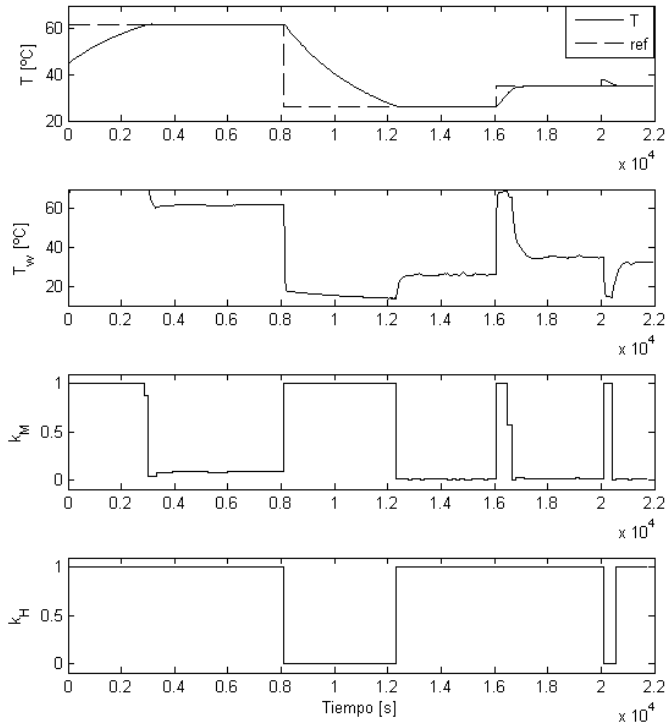


Figura 65: Respuesta del sistema con un controlador predictivo híbrido con EE+NLP con puntos iniciales aleatorios.

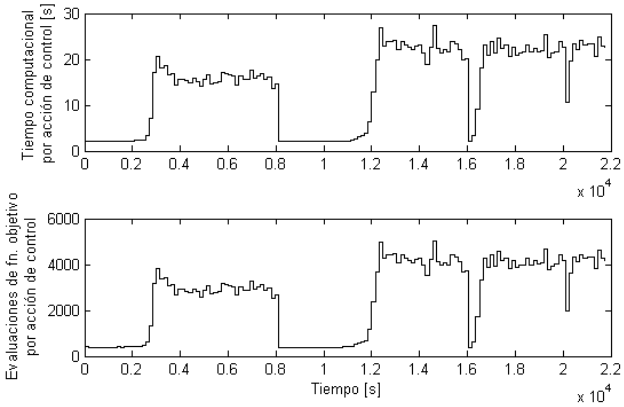


Figura 66: Esfuerzo computacional para con EE+NLP con puntos iniciales aleatorios a lo largo de la operación del sistema.

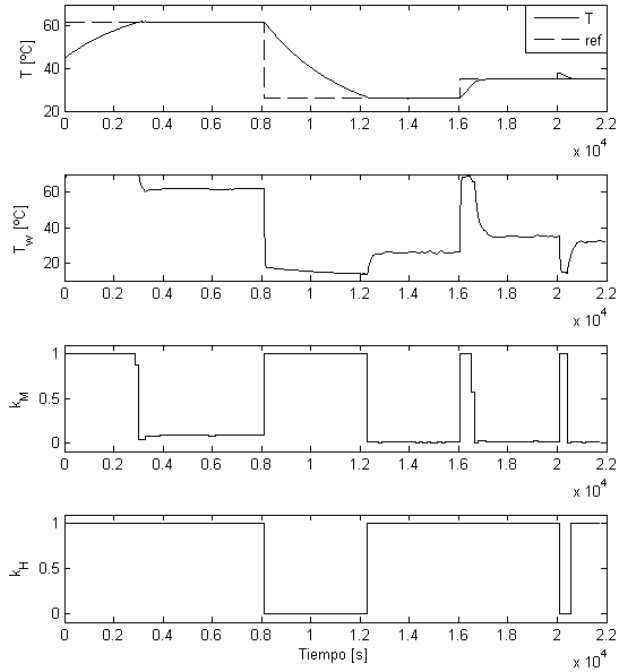


Figura 67: Respuesta del sistema con un controlador predictivo híbrido con EE+NLP con puntos iniciales iguales a solución anterior.

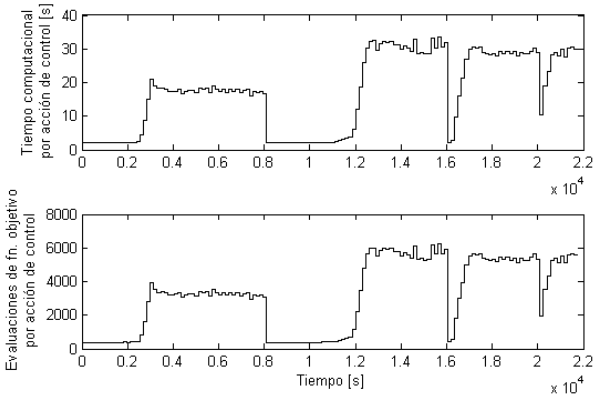


Figura 68: Esfuerzo computacional para con EE+NLP con puntos iniciales iguales a solución anterior a lo largo de la operación del sistema.

A pesar de las excelentes capacidades de seguimiento de las estrategias EE+NLP, éstas no son aplicables al proceso en cuestión, pues existen zonas de operación cuya solución requiere incluso 30 segundos, que es mucho más que los 10 segundos que permite el proceso. Aún si siempre pudiesen ser calculadas en menos de 10 segundos, es conveniente que fuese una fracción pequeña de ese tiempo, pues este análisis es realizado mediante computadores de escritorio poderosos, y lo ideal es implementar este tipo de controladores en otros menos poderosos como DSPs, PICs o PLCs.

Finalmente, para fines de comparación efectivos con las estrategias basadas en algoritmos evolutivos, se presentan las estadísticas completas de estos métodos en lo que respecta a las distintas componentes de la función objetivo, la función objetivo completa y los esfuerzos computacionales.

Tabla 13: Estadísticas de métodos MINLP

Estadísticas de desempeño							
Branch and Bound + NLP							
J		**J1**		**J2**		**J3**	
Prom	**Desv.**	**Prom.**	**Desv.**	**Prom.**	**Desv.**	**Prom.**	**Desv.**
10624.5	969.3	10602.1	969.1	5.56	0.47	2	0
EE + NLP con punto de partida aleatorio							
J		**J1**		**J2**		**J3**	
Prom	**Desv.**	**Prom.**	**Desv.**	**Prom.**	**Desv.**	**Prom.**	**Desv.**
9818.0	1.56	9795.9	1.56	5.11	0.02	2	0
EE + NLP con punto de partida igual a solución previa							
J		**J1**		**J2**		**J3**	
Prom	**Desv.**	**Prom.**	**Desv.**	**Prom.**	**Desv.**	**Prom.**	**Desv.**
9818.4	5.59	9796.2	5.60	5.14	0.02	2	0

Estadísticas de esfuerzo computacional por acción de control de Branch and Bound + NLP							
Branch and Bound + NLP							
Tiempo computacional [s]				**Fn. Objetivo evaluadas**			
Prom.	**Desv.**	**Min.**	**Max.**	**Prom.**	**Desv.**	**Min.**	**Max.**
3.63	2.73	0.1216	14.086	674.4	505.9	23	2608
EE + NLP con punto de partida aleatorio							
Tiempo computacional				**Fn. Objetivo evaluadas**			
Prom.	**Desv.**	**Min.**	**Max.**	**Prom.**	**Desv.**	**Min.**	**Max.**
14.86	9.15	2.09	36.24	2752.4	1694.2	388	6711
EE + NLP con punto de partida igual a solución previa							
Tiempo computacional				**Fn. Objetivo evaluadas**			
Prom.	**Desv.**	**Min.**	**Max.**	**Prom.**	**Desv.**	**Min.**	**Max.**
18.37	11.66	2.12	35.77	3401.7	2158.3	392.62	6624.4

De la Tabla 13 , al compararla con la Tabla 1, se puede apreciar que la justificación de utilizar la válvula de control continua reside en que permite obtener mejores seguimientos (con k_M continuo los J_1 son menores que cuando es discreto) y con menores esfuerzos de control (lo mismo sucede con J_2, mientras que J_3 no cambia).

6.2.3.2 Enfoques basados en PSO

Se ha encontrado que los enfoques convencionales no son aplicables exitosamente al reactor batch con entradas mixtas, sea por el esfuerzo computacional o bien por la calidad de las soluciones. Por lo tanto, se plantean los enfoques basados en PSO para su aplicación, que se espera supere a las ya probadas.

Como en 6.1.3 ya se ha visto que el análisis más completo se obtiene mediante la técnica multi-objetivo, se omite el análisis que considera iteraciones y tamaños de poblaciones fijos y se realiza directamente el análisis ya mencionado.

El análisis se realiza probando cada una de las alternativas y comparándolas utilizando el enfoque multi-objetivo, utilizando como medida de comparación el desempeño real del sistema a lo largo del régimen de operación típico.

El primer estudio consiste en analizar se analiza el efecto de la inclusión de la última solución dentro de la población inicial para cada una de las representaciones consideradas para este problema. Entonces, en la Figura 69 se presentan las fronteras de Pareto para cada uno de los métodos R-PSO (con representación continua para las variables discretas) al utilizar cada una de las alternativas de inicialización de población. En la Figura 70 se presenta lo mismo para los métodos N-PSO (con representación discreta para las variables discretas) y en la Figura 71 para los métodos B-PSO (representación basada en algoritmo PSO binario para las variables discretas).

A partir de estos gráficos, para los métodos R-PSO, N-PSO y B-PSO es claro que la inclusión de la mejor solución anterior en la nueva población entrega beneficios, ya que la frontera queda por debajo de la alternativa que inicializa la población 100% al azar. Sin embargo, si es la solución desplazada la mejor alternativa, o bien la solución intacta, depende de los parámetros de cada método, y del método. En el caso de N-PSO, insertar la solución anterior desplazada es la que entrega los mejores resultados. En R-PSO, para los parámetros que evalúan menos funciones objetivo, es mejor la alternativa con soluciones desplazadas, pero si se trata de parámetros que permiten evaluar mayores funciones objetivos, es mejor insertar la solución anterior intacta. En B-PSO sucede a la inversa, pero en la mayor parte del rango, es mejor la solución anterior desplazada.

Entonces, se concluye que la mejor alternativa es aquella que inicializa la población incluyendo una copia de la mejor solución anterior desplazada.

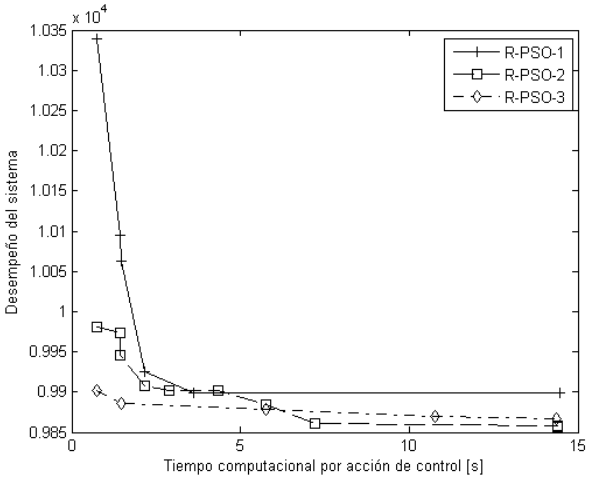


Figura 69: fronteras de Pareto para métodos R-PSO-1, R-PSO-2, R-PSO-3.

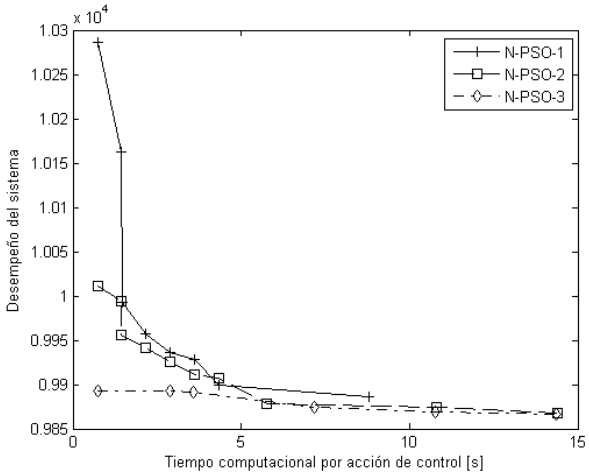


Figura 70: fronteras de Pareto para métodos N-PSO-1, N-PSO-2, N-PSO-3.

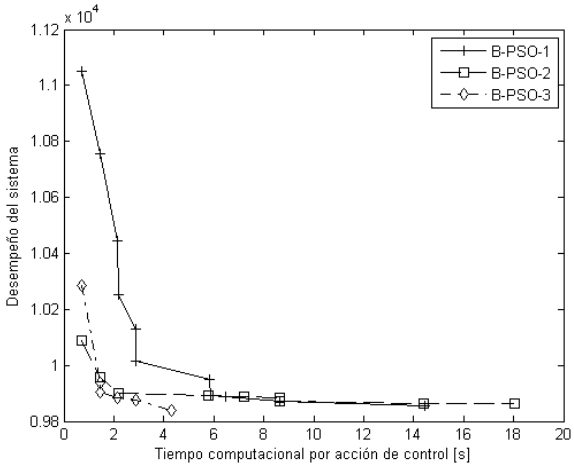


Figura 71: fronteras de Pareto para métodos B-PSO-1, B-PSO-2, B-PSO-3.

Luego, se comparan las distintas representaciones propuestas, utilizando la inclusión de la última solución óptima desplazada, que se ha encontrado es la mejor estrategia para la inicialización de las poblaciones. A continuación se presenta la frontera de Pareto de estos tres métodos en la

Figura 72. A partir de este gráfico es posible notar que el método B-PSO-3 es el que domina en la mayor parte del rango a los otros dos métodos. Por lo tanto, ese es el método escogido.

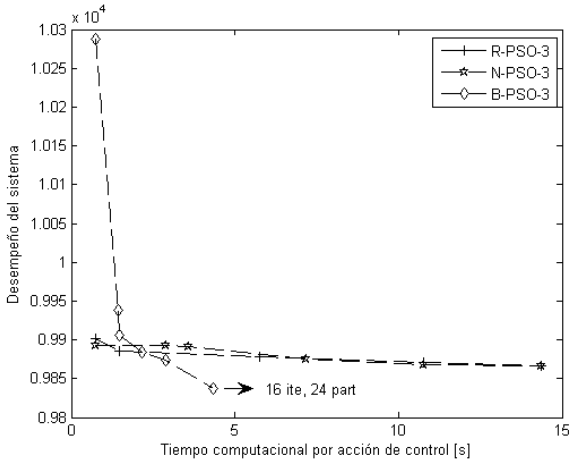


Figura 72: Fronteras de Pareto para los métodos R-PSO-3, N-PSO-3 y B-PSO-3

Ahora, los parámetros a utilizar (el número de iteraciones y tamaño de la población) se deben escoger como una de dos opciones, como el punto silla, o como el punto que entregue mejores soluciones en el tiempo permitido. Como en este proceso existe un límite de 10 segundos para aplicar la acción de control, se utiliza el segundo criterio. De acuerdo a lo presentado en la Figura 72, el punto del método B-PSO-3 que utiliza 24 partículas y 16 iteraciones es el que cumple el criterio. Y no sólo eso, sino que además es el que encuentra la mejor solución, incluso sin la restricción de tiempo. Sin embargo, para el caso de R-PSO-3 y N-PSO-3, los parámetros a utilizar con este criterio serían 16 partículas y 32 iteraciones, y 16 partículas y 40 iteraciones, respectivamente, y en estos casos, no corresponden a la mejor solución del método, como en B-PSO-3.

Corresponde mencionar que en términos rigurosos el punto con 40 partículas y 40 iteraciones (los valores más altos dentro de los considerados) debiesen ser siempre parte de la frontera de Pareto, pues debieran entregar siempre las mejores soluciones. Sin embargo la estocasticidad de los algoritmos hace que no sea siempre así, tal como en este caso. A pesar de esto, es recomendable seguir este criterio, pues ante variaciones poco significativas en la función objetivo, se debe escoger los parámetros que hagan al método más rápido (lo que es consistente con el criterio utilizado).

A continuación se presentan los indicadores de desempeño y esfuerzo computacional del sistema controlado utilizando diversas combinaciones de parámetros para ver como afectan al sistema. Se consideran los métodos R-PSO-3, N-PSO-3 y R-PSO-3, utilizando los parámetros más rápidos (8 partículas y 8 iteraciones), los parámetros escogidos para R-PSO-3 (24 partículas y 16 iteraciones) y los parámetros más lentos (40 partículas y 40 iteraciones). Además, se presentan las respuestas dinámicas para estos mismos métodos, desde la Figura 54 hasta la Figura 59.

Tabla 14: Estadísticas para los métodos utilizando 24 partículas y 16 iteraciones

Método	J		J1		J2		J3	
	Prom.	**Std.**	**Prom.**	**Std.**	**Prom.**	**Std.**	**Prom.**	**Std.**
R-PSO-3	9909.6	28.05	9886.6	28.05	7.177	0.249	2	0
N-PSO-3	9892.7	34.45	9869.7	34.47	6.959	0.346	2	0
B-PSO-3	9837.5	104.29	9813.5	104.10	9.439	0.898	2	0
Método	Tiempo computacional				Fn. Objetivo evaluadas			
	Prom.	Std.	Min.	Max.	Prom.	Std.	Min.	Max.
R-PSO-3	4.341	0.065	4.274	4.660	384	0	384	384
N-PSO-3	4.342	0.055	4.276	4.563	384	0	384	384
B-PSO-3	4.343	0.055	4.278	4.565	384	0	384	384

Tabla 15: Estadísticas para los métodos utilizando 8 partículas y 8 iteraciones

Método	**J**		**J1**		**J2**		**J3**	
	Prom.	**Std.**	**Prom.**	**Std.**	**Prom.**	**Std.**	**Prom.**	**Std.**
R-PSO-3	9901.4	67.493	9874.5	69.60	9.298	1.019	2.3	0.483
N-PSO-3	9892.9	71.04	9868.9	70.94	9.534	0.465	2	0
B-PSO-3	10287.1	334.6	10240.6	336.4	13.092	1.914	4.1	1.370
Método	**Tiempo computacional**				**Fn. Objetivo evaluadas**			
	Prom.	**Std.**	**Min.**	**Max.**	**Prom.**	**Std.**	**Min.**	**Max.**
R-PSO-3	0.729	0.023	0.7	0.874	64	0	64	64
N-PSO-3	0.736	0.033	0.7	1.050	64	0	64	64
B-PSO-3	0.734	0.050	0.7	1.490	64	0	64	64

Tabla 16: Estadísticas para los métodos utilizando 40 partículas y 40 iteraciones

Método	**J**		**J1**		**J2**		**J3**	
	Prom.	**Std.**	**Prom.**	**Std.**	**Prom.**	**Std.**	**Prom.**	**Std.**
R-PSO-3	9868.1	19.34	9845.5	19.36	6.045	0.165	2	0
N-PSO-3	9871.4	21.55	9848.9	21.54	5.986	0.210	2	0
B-PSO-3	9877.0	35.84	9854.0	35.84	7.048	0.330	2	0
Método	**Tiempo computacional**				**Fn. Objetivo evaluadas**			
	Prom.	**Std.**	**Min.**	**Max.**	**Prom.**	**Std.**	**Min.**	**Max.**
R-PSO-3	17.970	0.0738	17.542	18.417	1600	0	1600	1600
N-PSO-3	17.968	0.0737	17.455	18.287	1600	0	1600	1600
B-PSO-3	17.976	0.063	17.542	18.245	1600	0	1600	1600

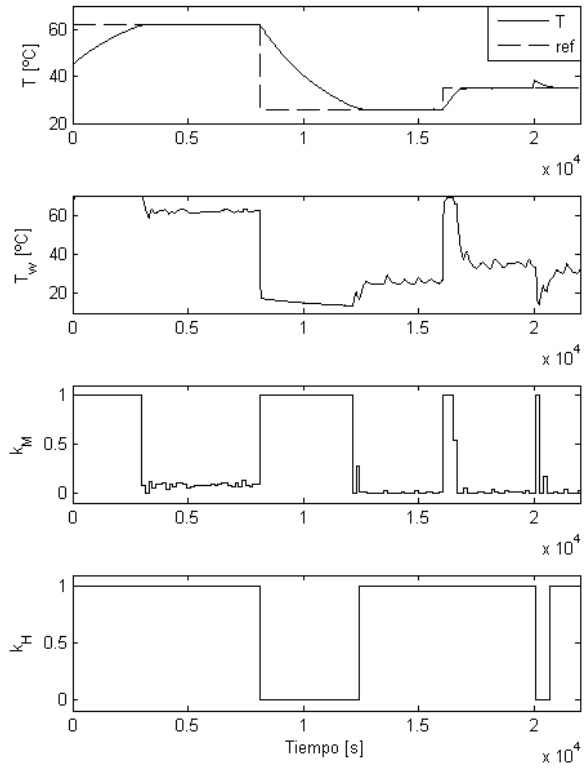


Figura 73: Respuesta del sistema con un controlador predictivo híbrido con R-PSO-3, usando 24 partículas y 16 iteraciones.

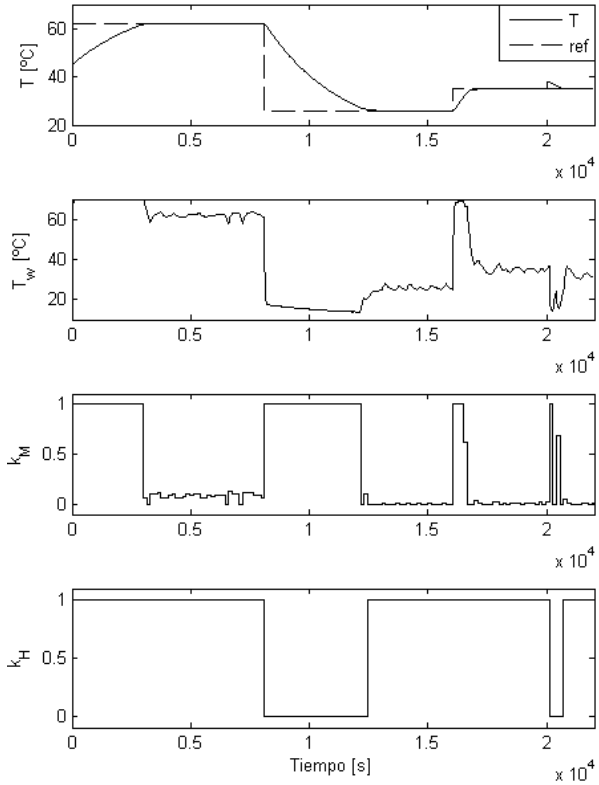


Figura 74: Respuesta del sistema con un controlador predictivo híbrido con N-PSO-3, usando 24 partículas y 16 iteraciones.

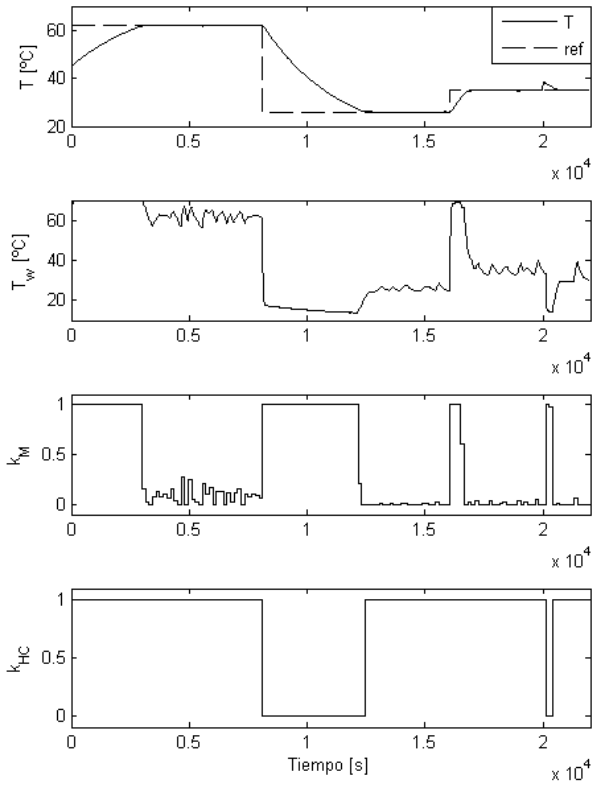


Figura 75: Respuesta del sistema con un controlador predictivo híbrido con B-PSO-3, usando 24 partículas y 16 iteraciones.

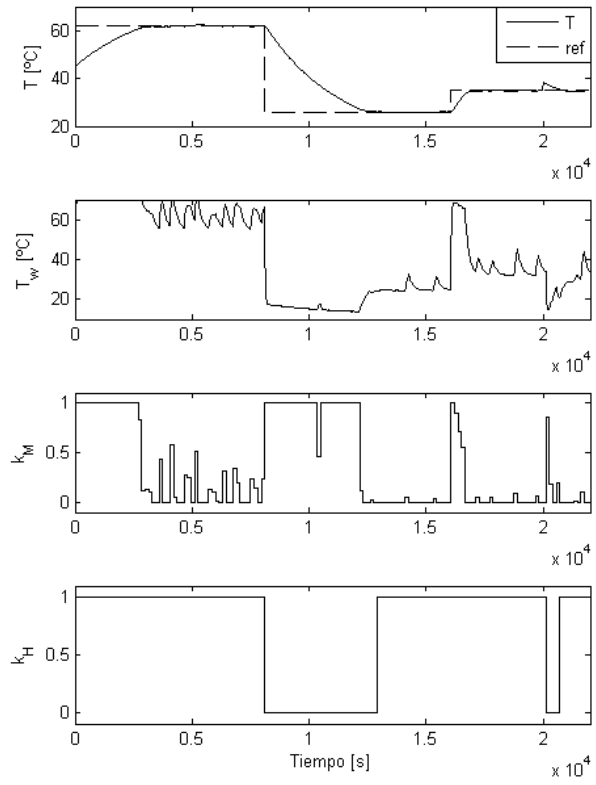


Figura 76: Respuesta del sistema con un controlador predictivo híbrido con R-PSO-3, usando 8 partículas y 8 iteraciones.

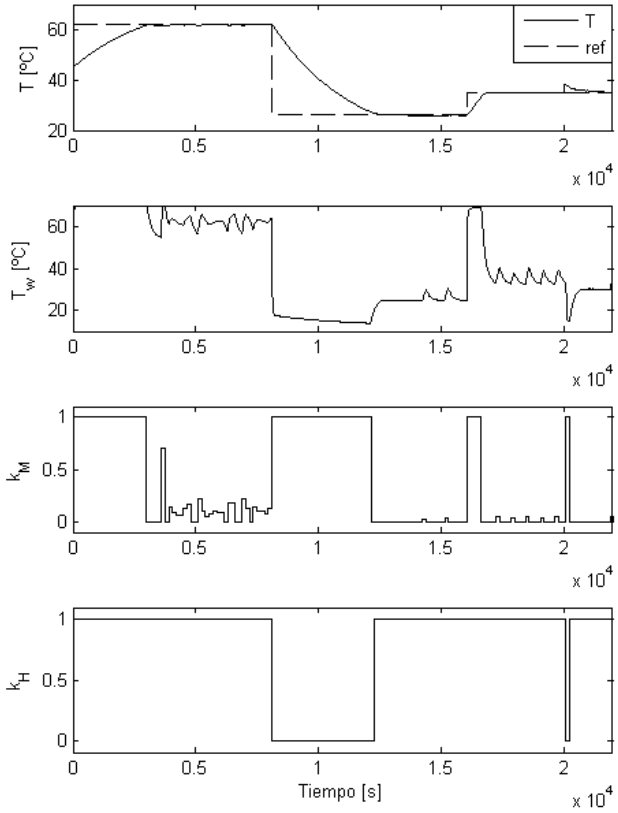


Figura 77: Respuesta del sistema con un controlador predictivo híbrido con N-PSO-3, usando 8 partículas y 8 iteraciones.

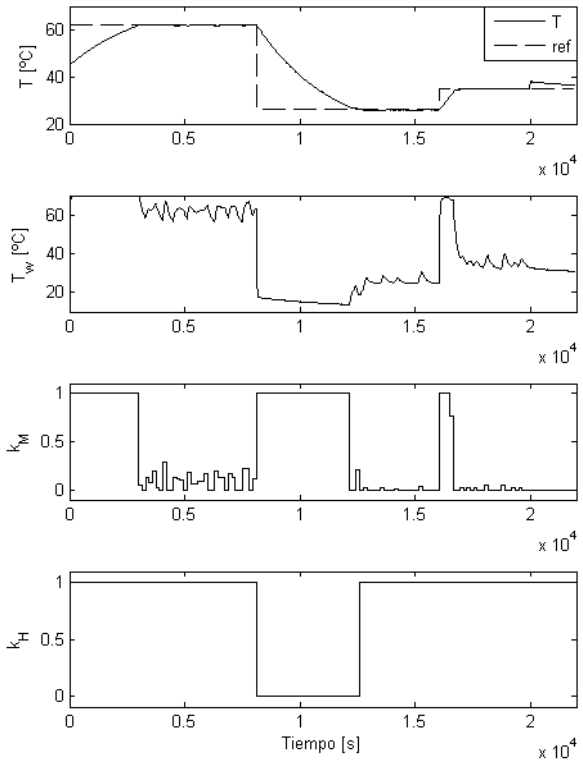


Figura 78: Respuesta del sistema con un controlador predictivo híbrido con B-PSO-3, usando 8 partículas y 8 iteraciones.

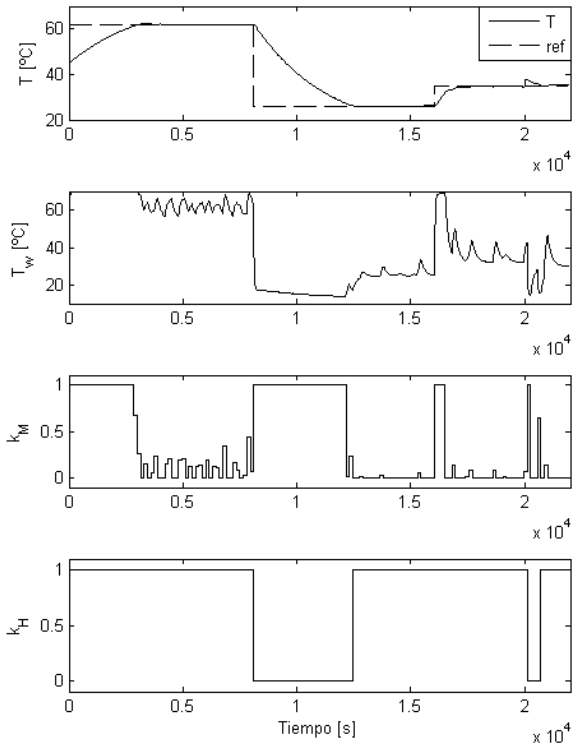


Figura 79: Respuesta del sistema con un controlador predictivo híbrido con N-PSO-3, usando 40 partículas y 40 iteraciones.

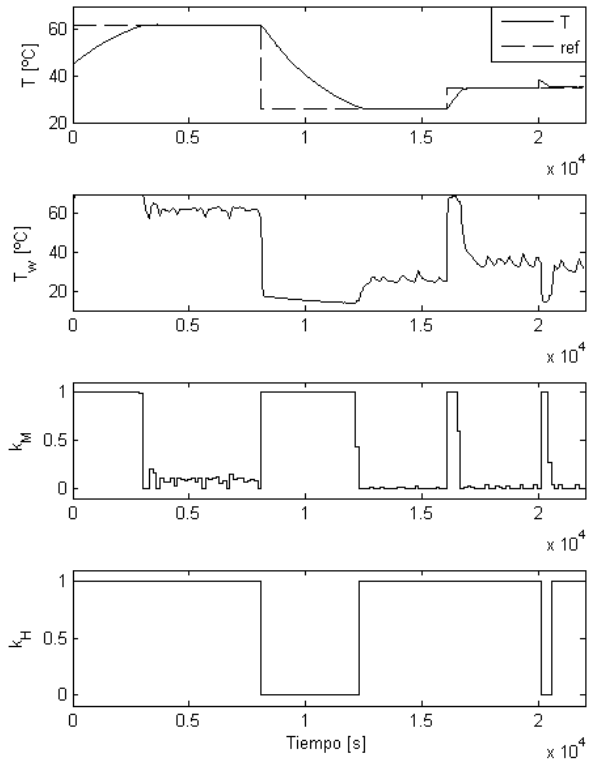


Figura 80: Respuesta del sistema con un controlador predictivo híbrido con R-PSO-3, usando 40 partículas y 40 iteraciones.

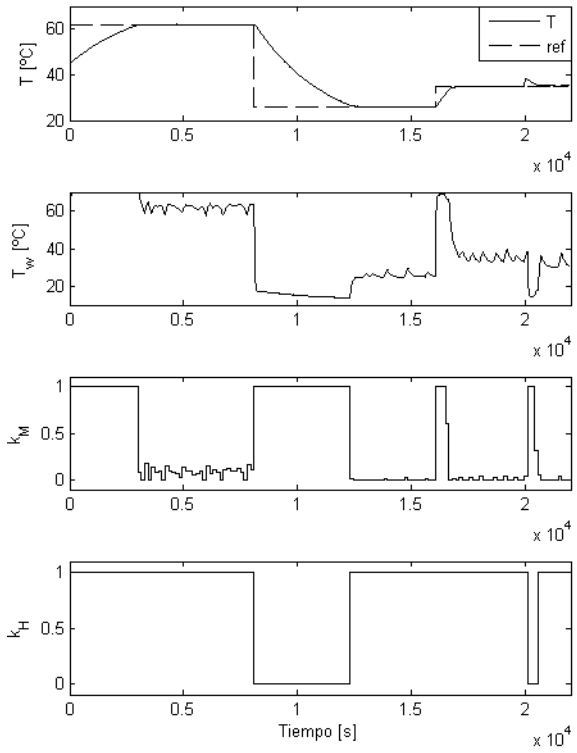


Figura 81: Respuesta del sistema con un controlador predictivo híbrido con B-PSO-3, usando 40 partículas y 40 iteraciones.

Se observa de estas figuras que, al utilizar 24 partículas y 16 iteraciones, todos estos métodos alcanzan mejores seguimientos y desempeño en general en comparación al método BB+NLP, a costa de mayores esfuerzos de control. Se observa también que el método B-PSO-3 con 8 partículas y 8 iteraciones posee un rendimiento bastante limitado, mientras que los otros dos métodos funcionan bastante bien. Al utilizar 40 partículas e iteraciones finalmente, el rendimiento de los tres métodos es bastante parejo, pero es mejor en el caso de R-PSO-3 y N-PSO-3. Se observa además que éstos siguen mejorando sus rendimientos a medida que aumenta el número de partículas e iteraciones (ver Anexo Sección 13.2, Figura 112 y Figura 113), a diferencia de B-PSO-3, que parece en cierto punto la mejora se detiene (ver Anexo Sección 13.2, Figura 114). Esto es congruente con las fronteras de Pareto presentadas en la Figura 72.

Se debe mencionar sin embargo, que ciertas combinaciones de parámetros no deben ser utilizadas pues se pasan de los 10 segundos para calcular las acciones de control, por ejemplo, 40 partículas y 40 iteraciones. Además, como ha sido mencionado, posiblemente una metodología similar pueda ser utilizada en procesos que requieran que las acciones de control estén disponibles en un tiempo menor que 10 segundos, y en dicho caso otros parámetros debiesen ser seleccionados. Por ejemplo si el límite fuese de 3 segundos, se debiera escoger los parámetros que permitan resolver de mejor forma el problema en esos 3 segundos, y para este problema eso implicaría utilizar el método B-PSO-3 con 32 iteraciones y 8 partículas, como se muestra en la Figura 82.

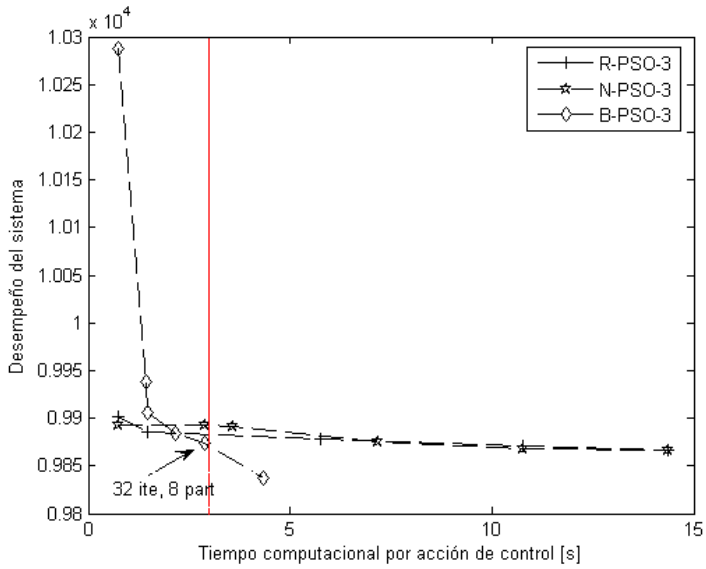


Figura 82: Fronteras de Pareto para los métodos R-PSO-3, N-PSO-3 y B-PSO-3

Es importante señalar del método que se ha escogido como el mejor, es decir, B-PSO-3, que sus resultados (el desempeño del sistema) son los que poseen la mayor desviación estándar, al ser comparados con R-PSO-3 y N-PSO-3, y debido a eso, es posible que no sea el mejor método. Esto se refleja en el hecho que el punto que ha sido escogido es claramente anómalo en comparación a los demás, y no sigue la tendencia de la saturación en el desempeño al aumentar el número de partículas y/o iteraciones (ver Anexo Sección 13.2, Figura 114). A pesar de eso, entrega resultados satisfactorios en un tiempo admisible, y por lo tanto es una elección válida.

Por otra parte, ninguno de los métodos alcanza desempeños similares a los EE+NLP (ver Tabla 13), pero dado que estos no son aplicables debido a su alto esfuerzo computacional, los métodos basados en PSO siguen siendo la alternativa a utilizar.

6.2.4 Análisis de resultados y conclusiones

Para resolver el problema de control predictivo híbrido para el reactor batch con entradas mixtas, se ha considerado diversas alternativas, dentro de las que se cuentan métodos convencionales o exactos, como *branch and bound* con programación secuencial cuadrática para los problemas continuos resultantes y enumeración explícita para las variables binarias con programación no lineal resuelta con programación secuencial cuadrática, y métodos basados en heurísticas, como optimización por enjambre de partículas.

El método de *branch and bound* con SQP en general encuentra buenas soluciones logrando un muy buen seguimiento en un tiempo aceptable para el proceso, pero suele quedarse estancado en soluciones locales que deterioran el comportamiento del proceso. Basado en esto, se prueba entregarle buenos puntos de partida, pero cuando el régimen de operación del proceso cambia, el método nuevamente se queda estancado en soluciones locales, pero de las cuales no logra escapar en el resto de la operación del sistema. Con esto se concluye que el método es fuertemente sensible a las condiciones iniciales, y que la presencia óptimos locales se debe a las variables binarias del proceso que son relajadas en el procedimiento de ramificar y acotar del método BB.

Basado en el último punto del párrafo anterior, se propone resolver el problema de optimización entera mixta mediante un método de enumeración explícita para las variables discretas, donde los problemas de optimización no lineal resultantes son resueltos con SQP. En este caso el sistema efectivamente ya no se queda estancado en soluciones locales y logra rendimientos óptimos. Además, como es esperable, logra un desempeño en términos de calidad de solución, al comparar objetivo por objetivo, superior al obtenido por el método de *branch and bound* utilizado para el reactor con entradas discretas, lo que justifica plenamente el uso de una estructura distinta de acciones de control para el reactor batch. Sin embargo, este método tiene el problema que no logra encontrar las soluciones en el tiempo asignado (10 segundos) y por lo tanto no es aplicable en un proceso real, dadas las capacidades de procesamiento actual.

Otro aspecto que no satisfacen bien estos métodos es la presencia de un sustento teórico (para este problema) respecto de su capacidad de encontrar los óptimos. En general se requieren propiedades sobre las funciones y espacios de búsqueda, tales como convexidad, continuidad y suavidad, sin embargo, en este proceso dichas propiedades no se cumplen, en particular la suavidad, debido a que el modelo del proceso es una estructura difusa de Takagi-Sugeno con distintos modelos para distintas zonas de operación que en general no poseen derivadas continuas. Esta ausencia de derivadas continuas, impone importantes limitaciones sobre la capacidad de los algoritmos, y a fin de cuentas hace que no sean óptimos, lo cual puede ser una causa de la falla en particular del método basado en BB+SQP.

La no aplicabilidad de las estrategias de optimización convencionales para los problemas de optimización de la estrategia de control predictivo híbrido prueba la necesidad de aplicar otro tipo de métodos, y en este caso se propone la aplicación de optimización por enjambre de partículas.

Dentro de esta heurística se proponen tres alternativas. En la primera, R-PSO, tanto las variables continuas como las binarias son representadas por coordenadas que pueden tomar valores continuos, y al decodificar, las variables continuas toman el mismo valor que las coordenadas correspondientes, y en el caso de las binarias, se trunca al valor más cercano. La segunda alternativa, N-PSO, se diferencia de la anterior en que las coordenadas que representan a las variables binarias solo pueden tomar esos valores binarios, y por lo tanto se realiza una truncación al momento de actualizar sus valores. La tercera alternativa, B-PSO, utiliza el algoritmo de PSO convencional (con partículas continuas) para las variables continuas, mientras que utiliza el algoritmo de PSO binario (actualiza de un modo distinto las posiciones) para las variables binarias.

Aparte de las tres alternativas para enfrentar el problema, se proponen las mismas 3 estrategias que en el caso discreto para la inicialización de población: inicializarla de forma 100% aleatoria, incluyendo una copia intacta de la mejor solución anterior, o incluyendo una copia de la mejor solución desplazada de acuerdo al enfoque de horizonte deslizante del control predictivo. Estas estrategias son probadas para cada una de las representaciones propuestas. Para el caso de N-PSO y B-PSO, la inclusión de la mejor solución desplazada es la mejor alternativa, mientras que para el caso de R-PSO al utilizar pocas partículas o iteraciones lo mejor es utilizar esta solución, pero a medida que se utilizan más partículas y/o iteraciones, es mejor insertar la mejor solución anterior intacta. Un tema que no ha sido estudiado mediante simulaciones, pero que sí es analizado en la Sección 5.5, es que la estrategia que incluye las soluciones desplazadas debiese ser cada vez mejor que la estrategia que sólo incluye la solución intacta, a medida que crece el tiempo de muestro. Debido a esto, resultaría interesante estudiar como varía el rendimiento de las

dos estrategias que incluyen la mejor solución anterior en la nueva población (una intacta y otra desplazada), para verificar si efectivamente la predicción teórica se cumple.

Luego, se comparan las distintas representaciones insertando la mejor solución anterior desplazada (ya que es la mejor estrategia en dos de tres representaciones). Se obtiene al realizar el análisis multi-objetivo que al utilizar pocas partículas e iteraciones funciona de mejor modo los métodos R-PSO-3 y N-PSO-3, pero al incrementarlas funciona rápidamente funciona mejor el método B-PSO-3, mientras que los otros dos tienen una mejora en el desempeño más pausado.

Respecto del análisis multi-objetivo, este es altamente útil para comparar métodos y entender los efectos de las variantes realizadas a los métodos, pero posee un elevado ruido debido a la estocasticidad de los algoritmos de optimización. De hecho, al visualizar los espacios de Pareto (ver Anexo Sección 13.2, Figura 114) se aprecia que en particular el punto con 24 individuos y 16 iteraciones de B-PSO-3, que aparece como el mejor según el análisis multi-objetivo es en realidad uno que se obtiene debido a la elevada desviación estándar del desempeño de B-PSO-3, y en teoría no es necesariamente el óptimo. Este ruido es altamente perjudicial sobre todo para los efectos de sintonía de los métodos, y el trabajo futuro descrito en la Sección 6.1.4 acerca de parametrizar las fronteras permitiría reducir este efecto.

Finalmente, respecto del diseño de la aplicación de la válvula de mezcla continua, se justifica su planteamiento ya que permite obtener mejores seguimientos a la referencia a un menor costo de control. Sin embargo, se encuentra que los problemas de optimización mixtos son más complicados de resolver que en el caso únicamente discreto, y estas mejoras son sólo apreciables con EE+SQP, ya que al utilizar PSO la calidad en las soluciones se pierde, y en el mejor caso, la función objetivo de B-PSO-3 está 1 unidad por encima (o sea es peor) de la alcanzada por BB en el caso discreto. Además, la aplicación discreta permite ahorro de tiempo computacional, ya que se pueden almacenar las soluciones ya evaluadas, mientras que no es posible realizar lo mismo en el caso con variables continuas y discretas, ya que no hay cota en la cantidad de soluciones que pueda ser evaluada. Por lo tanto, la aplicación en un proceso real del sistema de control para el reactor batch considerando válvulas de mezcla continuas no se justifica.

No obstante lo anterior, tal vez la comparación sea apresurada, ya que el caso discreto está altamente restringido (con una pequeña gama de acciones de control para elegir), mientras que la aplicación entera mixta no posee ningún tipo de restricciones, y tal vez por esa vía sea posible encontrar mejoras. Por ejemplo, se plantea la opción de trabajar con espacios discretizados con grillas más finas o más gruesas según algún criterio, como ha sido planteado por Fravolini *et al*. (2008) para el caso con variables continuas.

7 Caso de Estudio: Ruteo dinámico de vehículos

El problema de ruteo dinámico de vehículos (*Dynamic Vehicle Routing Problem*, DVRP) es la contraparte del problema de ruteo de vehículos (*Vehicle Routing Problem*, VRP) convencional. El VRP tradicional consiste en la construcción de rutas de costo mínimo para los vehículos de modo que visiten cada destino una sola vez, donde toda la información relevante para la construcción de las rutas no cambia una vez que éstas han sido determinadas, y se asume que está en poder del despachador. El DVRP, por su parte, también consiste en la construcción de rutas de costo mínimo para los vehículos de modo que visiten cada destina una sola vez, pero la información para la construcción de rutas puede cambiar una vez que estas han sido diseñadas, y cuando el sistema se encuentra en operación. Luego, las rutas asignadas deben ser recalculadas tomando la nueva información a medida que va entrando en conocimiento del despachador (Larsen, 2000).

En esta tesis se considera el problema dinámico de recoger y entregar pasajeros[9] (*Dynamic Pickup and Delivery Problem*, DPDP), que es un subconjunto del DVRP. Este problema puede ser formulado con un conjunto de requerimientos de servicio de transporte (identificados por ubicaciones de recogidas y entregas) que son satisfechos por una flota de vehículos ubicados inicialmente en distintos terminales. La dimensión dinámica aparece cuando un sub-conjunto (o el conjunto completo) que los requerimientos no es conocido previamente; luego, estos requerimientos deben ser agendados en tiempo real, en el instante que entran al sistema. El DPDP ha sido de gran interés el último tiempo en el mundo científico e industrial ya que el creciente desarrollo e implementación de eficientes estrategias de optimización *online* permite mejorar significativamente la calidad de las soluciones. El DPDP ha sido intensamente estudiado los últimos 20 años (Psaraftis, 1980; 1988) (Gendrau *et al*., 1999) (Kleywegt y Papastavrou , 1998). La salida final de tal tipo de problemas es un conjunto de rutas de todos los vehículos que varían durante el tiempo. Eksioglu *et al*. (2009) y Berbeglia (2009) presentan una reseña de trabajos recientes acerca de los problemas de recogida y entrega dinámicos, aplicaciones de dial-a-ride y métodos de solución.

Existe una gran cantidad de trabajos para optimizar DPDPs bajo distintas condiciones. Larsen (2000) desarrolla una buena caracterización de diversas variantes del VRP. El problema del vendedor viajero dinámico (*Dynamic Traveller Salesman Problem*, DTSP) introducido por Psaraftis (1988), y el problema del reparador viajero dinámico (*Dynamic Travelling Repariman Problem*, DTRP), que fue introducido por Bertsimas y Van Ryzin (1991) y luego extendido por Berstimas y Van Ryzin (1993a, 1993b), son ejemplos de aplicaciones específicas de este problema (el DPDP). Recientemente, Swihart y Papastavrou (1999) y Thomas y White (2004) formulan y resuelven dos variantes del DTRP. Mitrovic-Minic *et al*. (2004a) introduce el concepto de heurísticas de doble-horizonte para el DPDP con ventanas de tiempo, mostrando que el método puede obtener mejoras en los costos de las rutas asignadas cuando es comparado con métodos clásicos de un horizonte móvil, pero la mejora tiende a disminuir a medida que los problemas crecen en escala. Mitrovic-Minic et al. (2004b) presentan cuatro estrategias de espera para vehículos (manejar-primero, esperar-primero, espera dinámica y espera dinámica avanzada), y concluyen que las estrategias propuestas son mejores en términos de los largos de ruta total que las típicas estrategias de espera basadas en manejar-primero, y que la estrategia de espera

[9] Aunque también podría extenderse a carga.

dinámica avanzada es la más eficiente.

El problema Dial-a-ride (DRP), una versión del DPDP en el cual los requerimientos del sistema son pasajeros que necesitan transporte de un lugar a otro en una ciudad, puede ser modelado utilizando un esquema de control predictivo híbrido (HPC), considerando que los re-ruteos potenciales de vehículos pueden afectar las decisiones actuales, al analizar el costo extra de insertar requerimientos de servicio en tiempo real en las rutas predefinidas de los vehículos mientras éstos se están moviendo. Sáez et al. (2008) y Cortés et al. (2009) plantean una formulación del DRP como un HPC especificando el espacio de variables de estado y modelos de predicción. Se desarrollan dos algoritmos, basados en GA y PSO para resolver el problema en tiempo real.

El problema Dial-a-Ride (DRP) ha sido estudiado intensamente los últimos años. Xiang *et al*. (2008) estudian un DRP considerando un complejo conjunto de restricciones en una red variante con el tiempo, donde utilizan una heurística basada en una segunda función objetivo para escapar de óptimos locales. Respecto a aplicaciones reales, Madsen *et al.* (1995) adaptan las heurísticas de inserción de Jaw *et al*. (1986) para resolver un problema de transporte de ancianos y minusválidos en Copenhague, mientras que Dial (1995) propone un enfoque moderno para el sistema de operación de tránsito dial-a-ride muchos-a-pocos ADART (Autonomous Dial-a-Ride Transit), que actualmente se encuentra implementado en Corpus Christi, EEUU. Gendreau *et al*. (1999) modifican la heurística de búsqueda tabú para resolver el DVRP con ventanas temporales flexibles en un esfuerzo de encontrar un método de solución que maneje distintos DVRPs. Otros métodos más sofisticados basados en búsqueda tabú han sido desarrollados recientemente, tales como búsqueda tabú granular (Toth y Vigo, 2003), y memoria adaptativa basada en búsqueda tabú (Tarantilis, 2005).

Los algoritmos evolutivos también han sido propuestos para resolver variados problemas del tipo DVRP. En particular, GA ha sido aplicado para varias versiones de VRPs, considerando distintas representaciones y operadores genéricos dependiendo del problema específico (para el VRP con un sólo vehículo con límite de capacidad, Sklerc *et al*., 1997; para el DVRP con ventanas temporales flexibles, con múltiples vehículos y con tiempo de viaje dependiente del tiempo, Haghani y Jung, 2005). Zhu *et al*. (2006) proponen una estrategia basada en PSO adaptado para resolver un VRP estático con ventanas temporales. Jih y Yung-Jen (1999) y Osman *et al*. (2005) presentan una comparación exitosa de GA contra programación dinámica en términos de tiempo computacional, donde los primeros resuelven en DVRP con ventanas temporales y restricciones de capacidad, mientras que los segundos resuelven un VRP multi-objetivo. La optimización por colonia de hormigas también ha sido aplicado al DVRP (Montemanni *et al*., 2005; Dréo *et al.*, 2006).

La mayor parte de las metodologías propuestas son de orden exponencial, salvo las basadas en heurísticas del tipo algoritmos evolutivos, PSO, etc., y por lo tanto no son recomendables para instancias muy grandes del problema. En esta tesis se propone una formulación del DPDP basada en control predictivo híbrido (*Hybrid Predictive Control*, HPC), que también resulta ser de orden exponencial. Se propone entonces una metodología de optimización basada en algoritmos evolutivos (Sáez *et al*., 2008; Cortés *et al*., 2009), que permite la aplicación en problemas de mayor tamaño.

Luego, el objetivo principal de esta parte del trabajo de tesis es desarrollar una metodología eficiente y sistemática para resolver la formulación del DVRP como HPC mediante algoritmos

evolutivos ad-hoc. Se aplica además la metodología sistemática para la sintonía de parámetros de los algoritmos evolutivos propuestos basados en un enfoque multi-objetivo, tal como es presentada en la Sección 5.7.2. A continuación se presenta la formulación del problema, donde se incluye el modelo predictivo y la función objetivo utilizada en la formulación de HPC. Luego, se presentan el diseño de la estrategia de optimización basada en algoritmos evolutivos, donde se consideran distintas alternativas para la representación de soluciones y manejo de restricciones. Finalmente, se presentan los estudios por simulación, donde se justifica la aplicación de algoritmos evolutivos frente a otras estrategias convencionales (como enumeración explícita, EE), se comparan las distintas alternativas para el diseño de los algoritmos evolutivos y se aplica el análisis multi-objetivo para la sintonía y comparación de dos de las alternativas de diseño basadas en algoritmos evolutivos.

7.1 Formulación del problema

La modelación del problema DPDP incorpora elementos estocásticos en la llegada de los requerimientos de clientes, que son utilizados para estimar el impacto de reasignaciones futuras en la calidad de servicio de aquellos clientes a quienes ya se ha incorporado en la agenda. La predicción estocástica permite incorporar una medida más realista de los tiempos efectivos de espera y de viaje de los usuarios, en una función objetivo que es coherente con este tipo de predicción (Sáez *et al*., 2008; Cortés *et al*., 2009).

Se asume un área de influencia A, en la que pueden aparecer requerimientos de servicio. Se posee un conjunto de vehículos V compuesta por F vehículos homogéneos con capacidad *Cap*. La flota de vehículos se encuentra en operación moviéndose a través del área de influenza de acuerdo a rutas predefinidas. La demanda de servicio es desconocida, y aparece en tiempo real. Entonces se debe tomar decisiones rápidas de re-ruteo de los vehículos de la flota para satisfacer esta demanda. Se asume que en el tiempo k a cada vehículo $j \in V$ se le asigna una acción de control que corresponde a la secuencia de paradas del vehículo (recogidas y entregas):

$$u_j(k) = S_j(k) = \left[s_j^0(k) \quad s_j^1(k) \quad \cdots \quad s_j^i(k) \quad \cdots \quad s_j^{w_j(k)}(k) \right]^T \tag{7.1}$$

donde el *i*-ésimo elemento de la secuencia representa la parada *i*-ésima del vehículo *j*, y $w_j(k)$ es el número de paradas de la secuencia. La condición inicial, denotada por $s_j^0(k)$, representa la posición del vehículo j en el instante k.

La acción de control completa $u(k) = S(k)$, que corresponde a la decisión del despachador del servicio, se representa por el conjunto de secuencias de paradas asignadas a cada vehículo en el instante k.

$$u(k) = S(k) = \left\{ S_1(k), ..., S_j(k), .., S_F(k) \right\} \tag{7.2}$$

Los vehículos viajan a través de la matriz de secuencias predefinidas $S(k-1)$ mientras no se reciban nuevos requerimientos de servicio. Cuando un nuevo requerimiento de servicio llega al sistema, el controlador o despachador central calcula la secuencia de control en el siguiente paso

$S(k)$ para la flota completa de vehículos, incluyendo las paradas requeridas por el nuevo cliente. Luego, cada secuencia $S_j(k)$ permanece fija durante el intervalo $(k, k+1)$, a no ser que un vehículo llegue a una parada de recogida o entrega predefinida en dicho intervalo, y en ese caso la secuencia se reducirá en tamaño al eliminar de ella la parada que ya ha sido satisfecha.

En este esquema el problema se formula en términos de un tiempo discreto de paso variable (gatillado por eventos), que representan los intervalos de tiempo entre dos requerimientos consecutivo. En este contexto, el controlador predictivo toma decisiones de ruteo cada vez que un nuevo requerimiento entra al sistema, es decir el tiempo discreto pasa de k a $k+1$.

El estado del sistema en el instante k se encuentra asociado con las secuencias previas $S(k-1)$ (la nueva llamada no es considerada). En el problema DPDP las variables del espacio de estados incluyen el tiempo de partida $T_j^i(k)$ y de carga $L_j^i(k)$ estimadas en el tiempo k, luego que el vehículo j abandona la parada i. Se define entonces para cada vehículo $j \in V$ los vectores de tiempo de partida y de carga de los vehículos como se muestra a continuación:

$$L_j(k) = \left[L_j^0(k) \quad L_j^1(k) \quad \cdots \quad L_j^{w_j(k-1)}(k) \right]^T_{(w_j(k-1)+1)\times 1} \tag{7.3}$$

$$T_j(k) = \left[T_j^0(k) \quad T_j^1(k) \quad \cdots \quad T_j^{w_j(k-1)}(k) \right]^T_{(w_j(k-1)+1)\times 1} \tag{7.4}$$

Por lo tanto, el conjunto de variables de estado para el sistema completo en el instante k puede ser escrito como $x(k) = \{L(k), T(k)\}$, donde $L(k)$ y $T(k)$ representan el conjunto de vectores de carga y tiempos de partida respectivamente. Es decir $L(k) = \{L_1(k), ..., L_j(k), ..., L_F(k)\}$ y $T(k) = \{T_1(k), ..., T_j(k), ..., T_F(k)\}$. La salida $y(k)$ del sistema está representada por los tiempos de partida observados (es decir reales y no estimados) de los vehículos en las paradas, $T(k)$.

Las secuencias de los vehículos y las variables de estado deben satisfacer una serie de restricciones que dependen de condiciones reales del DPDP, y se presentan a continuación:

Restricción 1- Precedencia. La entrega de un pasajero no puede sucede antes de su recogida. Por lo tanto, el último nodo de cada secuencia debe ser una entrega.
Restricción 2.- Exclusividad: Un destino $P_j^i(k)$ debe ser visitado solo una vez, y es asignado a sólo un usuario.
Restricción 3.- Consistencia. Si un pasajero se sube en un vehículo específico, debe ser llevado a su destino por el mismo vehículo.
Restricción 4.- Carga máxima. Un vehículo no debe transportar más pasajeros que su capacidad máxima *Cap*.

Bajo esta formulación del modelo se puede plantear un esquema de control en lazo cerrado, como el presentado en el diagrama de flujo de la Figura 83.

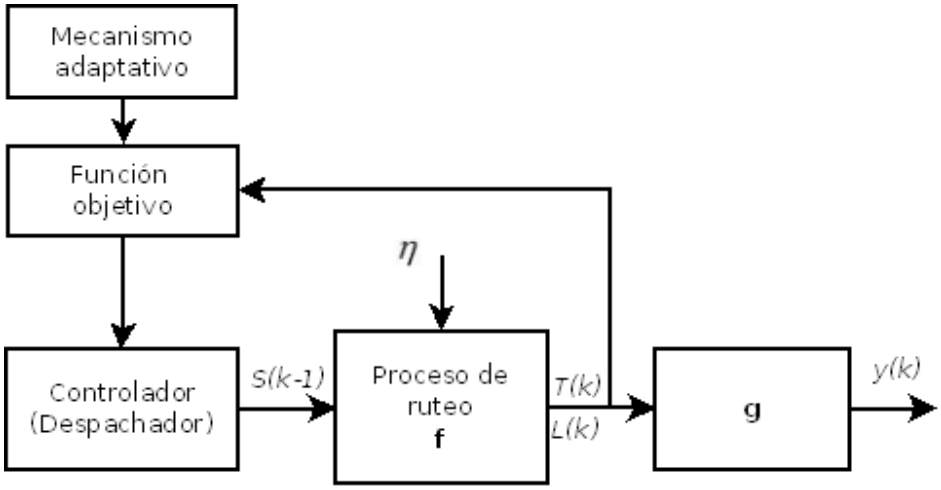


Figura 83: Diagrama de bloques de la estrategia basada en HPC para el DPDP

En esta figura el controlador predictivo se encuentra representado por el despachador y las decisiones de ruteo son resueltas al minimizar una función objetivo que considera el costo de usuario basado en los tiempos totales de viaje y espera de los usuarios, y una componente que representa el costo operacional. El proceso de ruteo se define por la decisión *online* de despacho de la secuencia $S(k-1)$ bajo una cierta demanda μ, que resulta en tiempos de partida observados $y(k)$. El mecanismo adaptativo se incluye debido a los parámetros variantes del sistema y la dimensión variables de las matrices de tiempos de partida y carga de los vehículos.

A continuación se describe el modelo dinámico predictivo, destacando el tratamiento de las componentes de tiempos partida y carga de los vehículos. A su vez, se explicita la función objetivo definida para el diseño de la estrategia de control predictivo híbrido.

7.2 Modelo dinámico predictivo y función objetivo

Como el estado del sistema se encuentra definido por los tiempos de partida $T_j^i(k)$ y cargas estimadas $L_j^i(k)$ en el instante k de cada vehículo k al partir de su parada *i*-ésima. Ambos vectores son estocásticos, pues dependen de la evolución del sistema que es afectado por una demanda incierta. La predicción de estas variables cuando aparece un nuevo requerimiento, está dada por los valores esperados de los vectores de estado para un vehículo j, $\hat{x}_j(k+1)$, como se muestra a continuación:

$$\hat{x}_j(k+1) = \begin{bmatrix} E\{L_j(k+1)/k\} \\ E\{T_j(k+1)/k\} \end{bmatrix} = \begin{bmatrix} \hat{L}_j(k+1) \\ \hat{T}_j(k+1) \end{bmatrix} = \begin{bmatrix} f_L(L_j(k), S_j(k)) \\ f_T(T_j(k), S_j(k)) \end{bmatrix} \quad \forall\, j = 1,...,F \qquad (7.5)$$

donde las funciones f_L y f_T representan el modelo en variables de estado. El sistema dinámico para un vehículo específico j puede ser representado geográficamente por su secuencia $S_j(k)$ calculada en un cierto instante k, y los valores esperados asociados a las variables de estado a

suceder en cada parada en el siguiente instante $k+1$. Esta representación se ilustra en la Figura 84.

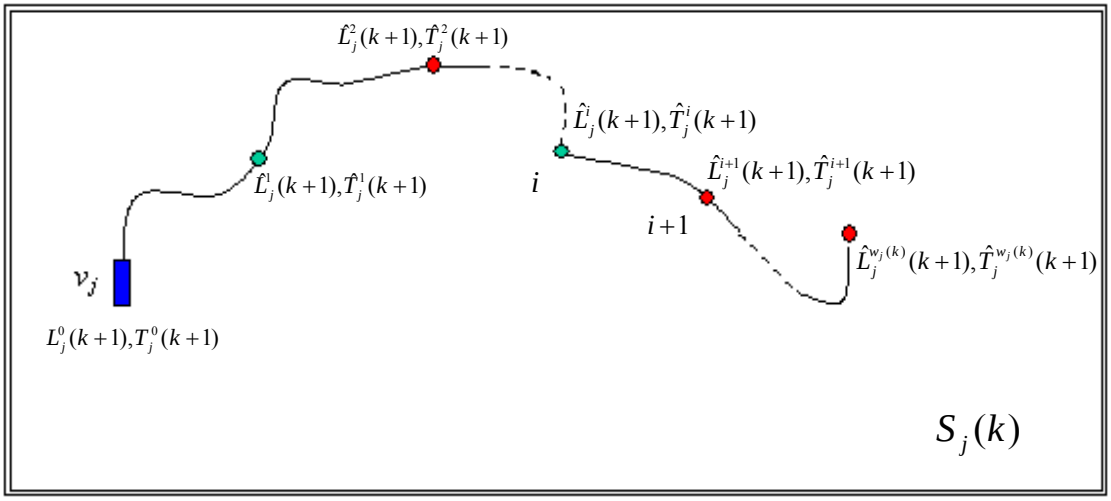


Figura 84: Ruta típica de un vehículo en el instante k y sus variables de estado estimadas en $k+1$.

Los componentes de la acción de control $S_j(k)$ son:

$$S_j(k)=\begin{bmatrix} r_j^0(k) & 1-r_j^0(k) & \Gamma_j^0(k) & label_j^0(k) \\ r_j^1(k) & 1-r_j^1(k) & \Gamma_j^1(k) & label_j^1(k) \\ \vdots & \vdots & \vdots & \vdots \\ r_j^i(k) & 1-r_j^i(k) & \Gamma_j^i(k) & label_j^i(k) \\ \vdots & \vdots & \vdots & \vdots \\ r_j^{w_j(k)}(k) & 1-r_j^{w_j(k)}(k) & \Gamma_j^{w_j(k)}(k) & label_j^{w_j(k)}(k) \end{bmatrix}_{(w_j(k)+1)\times 4} \tag{7.6}$$

donde $r_j^i(k)$ es una variable binaria definida como:

$$r_j^i(k)=\begin{cases} 1 & \text{si la parada } i \text{ de } S_j(k) \text{ es un recogida} \\ 0 & \text{si no} \end{cases} \tag{7.7}$$

La primera y segunda columna de $S_j(k)$ representan un par que identifica si la parada i es una recogida $[1\ 0]$ o bien una entrega $[0\ 1]$. La tercera columna de $S_j(k)$ representa la función de tiempo de viaje, donde $\Gamma_j^i(k)$ es el tiempo total de viaje entre los puntos $i-1$ e i más el retardo por la operación de transferencia en el nodo i. Para estos se asume que los vehículos se mueven a una velocidad constante, y por lo tanto su posición puede ser estimada en cualquier momento. Los valores de la última columna, $label_j^i$, sirven como identificador de pasajeros, que son necesarios para verificar la factibilidad de la secuencia en términos de precedencia (la recogida debe ocurrir antes que la entrega para cada cliente). Finalmente se debe notar que el tamaño de la

matriz de secuencias en (7.6) es $\left(w_j(k)+1\right)\times 4$, donde una fila corresponde a la condición inicial, $w_j(k-1)$ filas para las paradas previamente agendadas, y dos filas con la información (una para la recogida y otra para la entrega) de la última llamada.

Los modelos de predicción de carga y tiempos de partida derivados en Cortés *et al.* (2009) son:

$$\hat{L}_j\left(k+1\right)=A_L\,L_j\left(k\right)+B_L\left(S_j\left(k\right)\right) \tag{7.8}$$

$$\hat{T}_j\left(k+1\right)=A_T\cdot T_j\left(k\right)+B_T\left(S_j\left(k\right)\right) \tag{7.9}$$

donde:

$$A_L=\begin{bmatrix}1 & 0 & \cdots & 0\\ 1 & 0 & \cdots & 0\\ \vdots & \vdots & \ddots & \vdots\\ 1 & 0 & \cdots & 0\end{bmatrix}_{\left(w_j(k)+1\right)\times\left(w_j(k-1)+1\right)},\quad B_L(S_j(k))_{\left(w_j(k)+1\right)\times 1}=B_L^2\cdot\left(S_j(k)\cdot B_L^1\right)$$

$$B_L^1=\begin{bmatrix}1\\ -1\\ 0\\ 0\end{bmatrix}_{1\times 4},\quad B_L^2=\begin{bmatrix}0 & 0 & 0 & \cdots & 0\\ 1 & 0 & 0 & \cdots & 0\\ 1 & 1 & 0 & \cdots & 0\\ \vdots & \vdots & \vdots & \ddots & 0\\ 1 & 1 & \cdots & 1 & 0\end{bmatrix}_{\left(w_j(k)+1\right)\times\left(w_j(k)+1\right)} \tag{7.10}$$

y:

$$A_T=\begin{bmatrix}1 & 0 & \cdots & 0\\ 1 & 0 & \cdots & 0\\ \vdots & \vdots & \ddots & \vdots\\ 1 & 0 & \cdots & 0\end{bmatrix}_{\left(w_j(k)+1\right)\times\left(w_j(k-1)+1\right)},\quad B_T\left(S_j(k)\right)_{\left(w_j(k)+1\right)\times 1}=B_T^2\cdot\left(S_j(k)\cdot B_T^1\right)$$

$$B_T^1\begin{bmatrix}0\\ 0\\ 1\\ 0\end{bmatrix}_{4\times 1},\quad B_T^2=\begin{bmatrix}0 & 0 & 0 & \cdots & 0\\ 0 & 1 & 0 & \cdots & 0\\ 0 & 0 & 1 & \cdots & 0\\ \vdots & \vdots & \vdots & \ddots & 0\\ 0 & 0 & \cdots & 0 & 1\end{bmatrix}_{\left(w_j(k)+1\right)\times\left(w_j(k)+1\right)} \tag{7.11}$$

Tanto la matriz de secuencias $S_j(k)$ y el vector de carga esperada $\hat{L}_j(k+1)$ del vehículo j cambian su dimensión dinámicamente añadiendo dos filas cuando un nuevo requerimiento que

entra al sistema es asignado al vehículo j. Luego, las dimensiones de las matrices A_L, B_L^1, B_L^2 son variables. B_L^1 está diseñado para remover las dos últimas columnas del vector de secuencias, que no son necesarias para representar cambios en las cargas de los vehículos entre los instantes discretos k y $k+1$. Por otra parte, cuando un requerimiento es satisfecho, la primera fila de la secuencia es eliminada. Por lo tanto, el comportamiento adaptativo es modelado con estas técnicas de expansión y reducción del tamaño de las matrices. Similarmente las matrices A_T, B_T^1, B_T^2 asociadas al vector de tiempos de partida cambian sus dimensiones de forma dinámica.

Para el diseño de la estrategia de control predictiva híbrida, se define una función objetivo que representa una medida de los costos de usuario y de los costos operacionales, considerando no solo el requerimiento entrante, sino que además los requerimientos probables futuros. Mediante esta función objetivo el despachador (controlador) calcula las acciones de control óptimas para el horizonte de control N, es decir $u(k)=\{S(k),...,S(k+N-1)\}$, y aplica tan sólo la primera de ellas en base al enfoque de horizonte deslizante.

La función objetivo a minimizar en el DPDP de acuerdo a la estrategia de control predictivo, si se utiliza un horizonte de predicción N, es:

$$\underset{S(k)}{Min}\ J = \sum_{t=1}^{N}\sum_{j=1}^{F}\sum_{h=1}^{H(k+t)} p_h^{\Delta T(k+t)} \cdot \left(\left(C_j(k+t) - C_j(k+t-1) \right) \Big|_{S_j(k+t-2),h} \right) \tag{7.12}$$

y:

$$C_j(k+t)\Big|_{S_j(k+t-2),h} = \sum_{i=1}^{w_j(k+t-1)} \left\{ \underbrace{\left[\hat{L}_j^{i-1}(k+t)+1 \right] \left(\hat{T}_j^i(k+t) - \hat{T}_j^{i-1}(k+t) \right)}_{\text{J travel time}} + \underbrace{r_j^i(k+t-1)\,\alpha \left(\hat{T}_j^i(k+t) - T_j^0(k+t) \right)}_{\text{J waiting time}} \right\} \Bigg|_{S_j(k+t-2),h} \tag{7.13}$$

Donde $k+t$ es el instante en el cual el t-ésimo requerimiento entra al sistema, medido desde el instante k. $H(k+t)$ es el número de requerimientos posibles en el instante $k+t$, y $p_h^{\Delta T(k+t)}$ es la probabilidad de ocurrencia del requerimiento posible h-ésimo, durante el intervalo de tiempo $\Delta T(k+t)$. Es importante notar que los términos asociados a $t=1$ representan los costos de inserción del requerimiento entrante. Luego $H(k+1)=1$ y $p_h^{\Delta T(k+1)}=1$, pues no hay más llamados posibles, sino que el llamado que entra es el único considerado y por lo tanto posee probabilidad 1. Los términos asociados a $t>1$ representan los costos de inserción de los requerimientos potenciales futuros, que en términos concretos son las predicciones del sistema en el enfoque de control predictivo, y diferencian este enfoque de otro que resuelven un problema

estático en cada instante. $C_j(k+t)\big|_{S_j(k+t-2),h}$ es una función asociada el vehículo j en el instante $k+t$, que depende de la secuencia $S_j(k+t-1)$, conocida la secuencia previa $S_j(k+t-2)$, asociada con el requerimiento potencial h con probabilidad $p_h^{\Delta T(k+t)}$. $w_j\left(k+t-1\right)$ es el número de paradas estimadas del vehículos j en la secuencia $S_j(k+t-1)$. $C_j\left(k+t\right)$ puede ser separado en dos componentes: tiempo de viaje ($J_{\text{travel time}}$) y tiempo de espera ($J_{\text{waiting time}}$). La primera componente se calcula como la diferencia entre el tiempo de partida entre paradas consecutivas, multiplicada por la carga del vehículo (número de pasajeros más el conductor), mientras que la segunda considera el tiempo transcurrido entre que se realiza el requerimiento hasta que el vehículo asignado recoge al cliente. Por motivos de flexibilidad y consistencia, la componente de tiempo de espera es ponderada por un coeficiente α. $\hat{L}_j^{i-1}\left(k+t\right)$ denota la carga esperada entre las paradas $i-1$ e i; la diferencia $\hat{T}_j^i\left(k+t\right)-T_j^0\left(k+t\right)$ mide el tiempo de viaje esperado en el segmento $(i-1,i)$, incluyendo el tiempo de transferencia (abordaje o descenso del cliente del vehículo); y la diferencia $\hat{T}_j^i\left(k+t\right)-T_j^0\left(k+t\right)$ mide el tiempo de viaje esperado para llegar a la parada i desde la posición actual más el tiempo de transferencia en dicho punto. Finalmente $r_j^i\left(k+t-1\right)$ corresponde a la misma variable binaria para identificar los puntos de recogida y entrega en la expresión (7.7), pero en este caso está asociado con la secuencia futura $S_j\left(k+t-1\right)$.

Las probabilidades de ocurrencia de cada requerimiento posible $p_h^{\Delta T(k+t)}$ son calculadas en base a datos en tiempo real, históricos, o una combinación de ambos. En esta aplicación particular, las probabilidades se calculan de forma *offline* en base a datos históricos. Para encontrar estas probabilidades, en términos generales, el área de influencia se divide en diversas zonas, y se calcula la tasa de viajes entre cada una de las diversas zonas. Luego se consideran como patrones de requerimientos posibles los viajes entre los centros de masa de las diversas zonas, pero sólo aquellos con una tasa significativa. Luego, la probabilidad de cada requerimiento se calcula como la tasa relativa de dicho patrón respecto de la tasa total de todos los patrones.

El tiempo entre requerimientos utilizado en la función objetivo se escoge no sólo para ser representativo de la tasa real de requerimientos, sino que además para optimizar el desempeño del esquema de control. Entonces, para encontrar el paso óptimo de modo que se cumpla este objetivo se realiza un análisis de sensibilidad a partir de simulaciones para encontrar el paso que minimiza la función objetivo para un horizonte de predicción mayor a uno (Cortés *et al*., 2009). Para la aplicación de una predicción de dos pasos este parámetro se denota por τ y físicamente representa el tiempo esperado para que se registre un llamado esperado. Sin embargo, τ realmente representa el mejor instante para insertar el requerimiento futuro de modo de optimizar la estrategia de ruteo. Los valores óptimos son: $\tau = 4.8$ y $\tau = 2.4$, para particiones del área de influencia utilizando 9 y 4 zonas respectivamente.

Para la estrategia de control que se mostrará más adelante, es relevante ver la estructura particular de la función objetivo cuando se utilizan predicciones a un paso y a dos pasos.

La estrategia a un paso es aquella que utiliza un horizonte de predicción $N=1$. En dicho caso $H(k+1)=1$, pues el primer horizonte de predicción (y la única predicción que se realiza) corresponde al requerimiento que está entrando al sistema. Por lo tanto no se consideran más requerimientos potenciales, y el que ha llegado posee probabilidad 1. Así, la expresión (7.12) para el caso de predicción a un paso se rescribe como:

$$\begin{aligned} J &= \sum_{t=1}^{1}\sum_{j=1}^{F}\sum_{h=1}^{H(k+t)=1} p_h^{\Delta T(k+t)}(k+t)\cdot\left(C_j(k+t)-C_j(k+t-1)\right)\Big|_{S_j(k+t-2),h} \\ &= \sum_{j=1}^{F} \overbrace{p_1^{\Delta T(k+1)}(k+1)}^{=1}\cdot\left(C_j(k+1)-C_j(k)\right)\Big|_{S_j(k-1),1} \\ &= \sum_{j=1}^{F}\left(C_j(k+1)- \overbrace{C_j(k)}^{\text{constante conocida}} \right)\Big|_{S_j(k-1),1} \end{aligned} \tag{7.14}$$

Se debe notar que la diferencia $\left(C_j(k+1)-C_j(k)\right)\Big|_{S_j(k-1),1}$ es evaluada considerando tan solo la acción de control en el instante previo, representada por $S_j(k-1)$. Así, en este caso J representa el costo de inserción cuando el sistema acepta un nuevo requerimiento.

La función objetivo para el caso de predicción a dos pasos es diferente a la previa, pues incluye una predicción de cuales van a ser las asignaciones de las siguientes llamadas probables, y considerando las probabilidades de cada una de ellas. Así, la expresión se (7.12) rescribe como:

$$\begin{aligned} J &= \sum_{t=1}^{2}\sum_{j=1}^{F}\sum_{h=1}^{H(k+t)} p_h^{\Delta T(k+t)}(k+t)\cdot\left(C_j(k+t)-C_j(k+t-1)\right)\Big|_{S_j(k+t-2),h} \\ &= \sum_{j=1}^{F}\Bigg[\cancel{C_j(k+1)\Big|_{S_j(k-1),1}} - C_j(k) + \sum_{h=1}^{H(k+2)} p_h^{\Delta T(k+2)}(k+2)\cdot C_j(k+2)\Big|_{S_j(k),h} - \\ &\qquad \overbrace{\cancel{\sum_{h=1}^{H(k+2)} p_h^{\Delta T(k+2)}(k+2)\cdot C_j(k+1)\Big|_{S_j(k-1),1}}}^{=1} \overbrace{\phantom{\sum_{h=1}^{H(k+2)}}}^{\text{Independent of h}} \Bigg] \\ &= \sum_{j=1}^{F}\left[\sum_{h=1}^{H(k+2)} p_h^{\Delta T(k+2)}(k+2)\cdot C_j(k+2)\Big|_{S_j(k),h} - \overbrace{C_j(k)}^{\text{known constant}} \right] \end{aligned} \tag{7.15}$$

Se puede apreciar entonces que en la expresión final no aparecen los costos de inserción del primer requerimiento, solo el costo inicial, y los costos de inserción del segundo requerimiento. Esto se utilizada para el diseño del algoritmo de optimización basado en algoritmos evolutivos, que se muestra en la siguiente sección.

7.3 Método de solución

El algoritmo de solución para el control predictivo híbrido es originalmente diseñado para la predicción a dos pasos (Cortés *et al*., 2009). Debido a las características del problema se propone un algoritmo de dos capas: una capa para el problema global, y otra capa exclusivamente para el segundo paso de predicción. En cada capa se puede utilizar un algoritmo evolutivo, optimización por enjambre de partículas, o cualquier otro similar, para resolver el problema asociado a ella.

Como se presenta en la sección anterior, la función objetivo a dos pasos (el problema global) para una asignación de paradas de un requerimiento entrante depende también de las asignaciones de los requerimientos futuros posibles (7.15). Luego, para encontrar los *fitness* asociados a cada solución candidata es necesario encontrar las asignaciones óptimas para dichos requerimientos, dado que se conoce la asignación del requerimiento entrante (dada por la solución candidata asociada). La primera capa entonces se encarga de encontrar las asignaciones óptimas para la llamada entrante. El cálculo del *fitness* por su parte, requiere resolver un nuevo problema de optimización, que corresponde a la segunda capa del algoritmo.

La segunda capa corresponde a resolver el problema de encontrar las asignaciones óptimas para las los patrones de requerimientos futuros posibles, dada una asignación (solución candidata) para el requerimiento entrante (es decir de la primera capa). Este problema, a pesar de resolver el segundo paso de predicción, utiliza la función objetivo que corresponde a la predicción a un paso (7.14), pues se asume conocida la asignación del requerimiento entrante.

En la primera capa entonces, se resuelve un problema de predicción a dos pasos, mientras que en la segunda capa se resuelve un problema de predicción a un paso. No obstante, las únicas diferencias entre ambas capas son que el estado del sistema ha cambiado, y que la función objetivo o *fitness* se calcula de distinto modo. Por contraparte, el objetivo de cada capa, que es encontrar asignaciones óptimas para alguna llamada es el mismo, y por lo tanto el diseño del algoritmo de solución es el mismo para ambas capas.

Debido a la naturaleza del sistema, es muy complicado resolver la asignación óptima incluyendo dentro del problema determinar el vehículo óptimo. Por lo tanto, en cada los algoritmos incorporan un paralelismo de modo que resuelven las asignaciones óptimas en cada vehículo, y finalmente se realiza la asignación al vehículo que haya obtenido una menor función objetivo.

A continuación se presenta el diseño de diversas alternativas basadas en algoritmos evolutivos para resolver las asignaciones óptimas para los requerimientos de servicio en cada vehículo, en cada una de las capas. Posteriormente, se presentan en detalle los algoritmos para predicción a dos pasos (capa uno) y para predicción a un paso (capa dos).

7.3.1 Algoritmos evolutivos para asignaciones óptimas

Sea un vehículo j, con una secuencia asociada $S_j(k-1)$. Cuando un nuevo requerimiento entra al sistema en el instante k, el algoritmo genera diversas secuencias posibles $S_j^{\vartheta}(k)$, cada una asociada a un individuo (o solución candidata), que determinan las posiciones de inserción $\vartheta = (pu, de)$ de la llamada entrante, donde pu corresponde a la posición de la recogida y

de corresponde a la posición de inserción de la entrega. Un ejemplo entonces de una secuencia generada por una solución candidata es:

$$S_j^{\vartheta}(k)=\begin{bmatrix} r_j^0(k) & 1-r_j^0(k) & \Gamma_j^0(k) & label_j^0(k) \\ \vdots & \vdots & \vdots & \vdots \\ r_j^{pu}(k) & 1-r_j^{pu}(k) & \Gamma_j^{pu}(k) & label_j^{pu}(k) \\ \vdots & \vdots & \vdots & \vdots \\ r_j^{de}(k) & 1-r_j^{de}(k) & \Gamma_j^{de}(k) & label_j^{de}(k) \\ \vdots & \vdots & \vdots & \vdots \\ r_j^{w_j(k)}(k) & 1-r_j^{w_j(k)}(k) & \Gamma_j^{w_j(k)}(k) & label_j^{w_j(k)}(k) \end{bmatrix}_{(w_j(k)+1)\times 4} \tag{7.16}$$

donde la pu-ésima y de-ésima filas son las posiciones de las nuevas recogidas y entregas, respectivamente.

A continuación de presentan los diversos aspectos relevantes para el diseño de controladores predictivos híbridos en base a algoritmos evolutivos de acuerdo a lo presentado en el capítulo 5. Los diseños para este problema se realizan utilizando GA y PSO.

7.3.1.1 Representación de las soluciones y operadores evolutivos

Si un requerimiento se asigna a un vehículo, las posiciones que pueden tomar *pu* y *de* para generar la secuencia candidata definida por la ecuación (7.16) son:

$$pu \in PU = \{1,..,w_j - 1\} \tag{7.17}$$
$$de \in DE = \{2,...,w_j\} \tag{7.18}$$

Considerando esto, para generar estas secuencias candidatas se presentan las siguientes opciones para la representación de las soluciones en GA y PSO.

GA: como GA es un algoritmo inherentemente discreto se propone una representación discreta. La estructura de cada individuo, y cómo se obtienen las posiciones de inserción se muestra a continuación.

$$x=(x_1,x_2),\ x \in PU \times DE \subset \mathbb{N}^2 \tag{7.19}$$
$$\vartheta = x \tag{7.20}$$

En esta implementación se utiliza el crossover de 1 punto. El operador de mutación, por su parte, implementa una estrategia de desordenamiento local de las secuencias (Sáez *et al.*, 2008; Muñoz-Carpintero *et al.*, 2010) que corresponde a una mutación pequeña. El operador funciona del siguiente modo: Para cada coordenada de cada individuo, si $U(0,1) \le p_m$ se realiza mutación. Para cada x_1 se selecciona un valor al azar en $\{x_1-1, x_1+1\}$, y para cada x_2 se seleccionar un

valor al azar en $\{x_2 - 1, x_2 + 1\}$ y se reemplazan los nuevos valores en las coordenadas correspondientes del individuo.

PSO: como PSO es un algoritmo inherentemente continuo, éste debe ser adaptado para codificar las soluciones discretas. Se presentan dos alternativas para esto.

La primera alternativa es trabajar con partículas continuas, truncando sólo para evaluar las funciones objetivo. Esta representación se define como:

$$x = (x_1, x_2),\ x \in [0, w_j - 1] \times [1, w_j] \subset \mathbb{R}^2 \quad (7.21)$$

$$\vartheta = [\lceil x_1 \rceil, \lceil x_2 \rceil] \quad (7.22)$$

donde $\lceil \bullet \rceil$ es la función cajón superior. Al utilizar esta representación las coordenadas de recogida y entrega se obtienen aproximando hacia arriba las coordenadas de las partículas. Se debe notar lo más natural al utilizar este estilo de representación es almacenar $pbest_i$ y *gbest* como el mismo valor que se decodifica con (7.21), sin embargo esto no es una buena alternativa pues estos atractores no resultan ser simétricos respecto del espacio de búsqueda.

Sea el siguiente ejemplo. Se considera una partícula $x_1 = (1.8, 3.8)$ y otra partícula $x_2 = (2.1, 4.2)$. Sea además el *gbest* como en el caso intuitivo $gbest = (2, 4)$. Es posible notar que x_2 es claramente más cercano a gbest que x_1. Sin embargo, de acuerdo a la decodificación definida en (7.22), x_1 se decodifica igual a *gbest*, mientras que x_2 no.

Debido a lo anterior, se propone que los pbest y gbest sean escogidos como el centro del intervalo que identifica a la mejor solución. Es decir, si la mejor solución corresponde a las posiciones de inserción $\vartheta = (2, 4)$, asociadas a la partícula i, $pbest_i$ y *gbest* debieran almacenarse como $pbest_i = gbest = (1.5, 3.5)$. De este modo, x_1, que genera la misma secuencia que *gbest*, es mucho más cercana a *gbest* que x_2, la cual no genera la misma secuencia.

Finalmente, como esta alternativa utiliza partículas con coordenadas continuas (reales), para identificar los métodos que utilicen esta representación se añade a PSO el sufijo $\mathbb{R}$, resultando la nomenclatura PSO-$\mathbb{R}$.

La segunda opción consiste en forzar que las coordenadas de las partículas tomen los valores discretos definidas en (7.17), a lo largo de todo el proceso evolutivo, de modo que no es necesario truncar al evaluar la función objetivo. Esta representación se define como:

$$x = (x_1, x_2),\ x \in PU \times DE \subset \mathbb{N}^2 \quad (7.23)$$

$$\vartheta = x \quad (7.24)$$

Esta representación es equivalente a la utilizada por GA. Como las coordenadas de las partículas sólo pueden tomar valores enteros se añade el sufijo $\mathbb{N}$ para identificar los métodos que usan esta representación en PSO, quedando como PSO-$\mathbb{N}$.

Para poder garantizar que las partículas efectivamente pertenezcan a $PU \times DE$, los operadores de PSO, en particular el de la actualización de la posición se modifica a:

$$x_{ij}(t+1) = round\left(x_{ij}(t) + v_{ij}(t+1)\right) \tag{7.25}$$

donde $round(\bullet)$ es un operador de redondeo. Este operador permite mantener centradas las soluciones, a diferencia del caso anterior, y por lo tanto los *pbest* y *gbest* pueden ser las mismas posiciones de inserción que se decodifican mediante la ecuación (7.23).

7.3.1.2 Manejo de restricciones

En este sistema se deben satisfacer 4 restricciones, como se presenta en 7.1: precedencia, exclusividad, consistencia y carga máxima. Debido a la estructura de las variables de estado y del algoritmo de solución, es posible asegurar que cada parada se visita una sola vez, se asigna a un solo pasajero, y que un solo vehículo recoge y deja a un pasajero. Gracias a esto se satisfacen automáticamente las restricciones de exclusividad y consistencia, de modo que las restricciones que se deben manejar activamente son la de precedencia y la de carga máxima. Se presenta primero el caso de la restricción de precedencia y luego la de carga máxima.

La restricción de precedencia dice que la recogida debe suceder antes de la demanda. Esto se puede lograr en las soluciones candidatas $\vartheta = (pu, de)$, añadiendo la restricción $pu < de$. Por su parte las restricciones de carga máxima indican que la carga de cada vehículo a lo largo de su trayecto no puede ser mayor que su límite máximo. Esta restricción es más compleja de tratar pues depende además del estado del sistema.

Las alternativas principales para manejar las restricciones son los enfoques de penalización, reparación y representación factible, como se presenta en la Sección 5.6. Como la restricción de carga máxima depende del estado del sistema que es muy no lineal, es muy complicado generar esquemas de reparación de soluciones, o representaciones que garanticen factibilidad de esta restricción. Por lo tanto, la restricción de carga máxima se maneja mediante penalización. Por sencillez se considera un solo peso que garantiza que la función objetivo de una solución infactible sea siempre mayor que el de una solución factible. Por otra parte, todo individuo que genere una solución infactible en carga máxima tendrá el mismo fitness. Así, la función objetivo es:

$$J'(x) = \begin{cases} J(x) & \text{si x es factible} \\ M_1 & \text{si es infactible por carga máxima} \end{cases} \tag{7.26}$$

donde $M_1 >> \max\{J(x) \mid x \text{ es solución factible}\}$ es el peso de la penalización por no satisfacer la restricción de carga máxima.

La restricción de precedencia es de fácil manejo, pues sólo influye en ella la solución candidata y debido a esto es fácilmente reparable. Se propone entonces dos esquemas de reparación, uno basado en evolución Lamarckiana y otro basado en evolución Baldwiniana. Además se propone una estrategia de penalización en la cual cada individuo que genere una solución candidata infactible en términos de precedencia tenga una función objetivo mayor que la de una solución

factible, y mayor además que la de una solución infactible por carga máxima. De este modo, la función objetivo, al utilizar la penalización para la restricción por precedencia se calcula como:

$$J'(x)=\begin{cases} J(x) & \text{si x es factible} \\ M_1 & \text{si es infactible por carga máxima} \\ \mathrm{M}_2 & \text{si es infactible por precedencia} \end{cases} \tag{7.27}$$

Los esquemas de reparación se presentan a continuación.

La reparación Baldwiniana consiste en que si un individuo genera una solución infactible, entonces tanto la solución decodificada como el individuo son reparados para que la solución sea factible. La reparación Baldwiniana se lleva a cabo del siguiente modo:

$$\begin{aligned} &x=(x_1,x_2)\Rightarrow \vartheta=(pu,de) \\ &\text{Reparación Baldwiniana:} \\ &\text{Si } pu<de, \text{ entonces } \vartheta=(pu,de) \\ &\text{Si no (solución infactible en precedencia)} \\ &\quad \text{Si } pu>de, \text{ entonces } y=\frac{pu-de}{2},\ pu=\lfloor pu-y \rfloor,\ de=\lceil de+y \rceil \\ &\quad \text{Si } pu=de, \text{ entonces } de=\min(w_j(k),de+1),\ pu=\max(de-1,1), \\ &\quad \vartheta=(pu,de) \\ &\quad x=(pu,de) \end{aligned} \tag{7.28}$$

En palabras, si la recogida resulta ser mayor que la entrega, éstas son intercambiadas por las posiciones de recogida y entrega factibles más cercanas a las posiciones infactibles, y además, la partícula toma los mismos valores. Si por ejemplo se utiliza PSO-$\mathbb{N}$ o bien GA, el proceso a partir de las partículas hasta la secuencia de paradas se muestra a continuación:

$$\begin{pmatrix} x_1=(1,4) \\ x_2=(1,6) \\ x_3=(5,5) \\ x_4=(6,5) \\ x_5=(6,1) \\ x_6=(3,2) \\ x_7=(1,1) \end{pmatrix} \Rightarrow \begin{pmatrix} \vartheta_1=(1,4) \\ \vartheta_2=(1,6) \\ \vartheta_3=(5,5) \\ \vartheta_4=(6,5) \\ \vartheta_5=(6,1) \\ \vartheta_6=(3,2) \\ \vartheta_7=(1,1) \end{pmatrix} \Rightarrow \text{Reparación} \Rightarrow \begin{pmatrix} \vartheta_1=(1,4) \\ \vartheta_2=(1,6) \\ \vartheta_3=(5,6) \\ \vartheta_4=(5,6) \\ \vartheta_5=(3,4) \\ \vartheta_6=(2,3) \\ \vartheta_7=(1,2) \end{pmatrix} \Rightarrow \begin{pmatrix} \boxed{3^+}\to 1^+\to 2^+\to \boxed{3^-}\to 1^-\to 2^- \\ \boxed{3^+}\to 1^+\to 2^+\to 1^-\to 2^-\to \boxed{3^-} \\ 1^+\to 2^+\to 1^-\to 2^-\to \boxed{3^+}\to \boxed{3^-} \\ 1^+\to 2^+\to 1^-\to 2^-\to \boxed{3^+}\to \boxed{3^-} \\ 1^+\to 2^+\to \boxed{3^+}\to \boxed{3^-}\to 1^-\to 2^- \\ 1^+\to \boxed{3^+}\to \boxed{3^-}\to 2^+\to 1^-\to 2^- \\ \boxed{3^+}\to \boxed{3^-}\to 1^+\to 2^+\to 1^-\to 2^- \end{pmatrix} \tag{7.29}$$

y además:

$$\begin{pmatrix} x_1 \equiv (1,4) \\ x_2 \equiv (1,6) \\ x_3 \equiv (5,6) \\ x_4 \equiv (5,5) \\ x_5 \equiv (3,4) \\ x_6 \equiv (2,3) \\ x_7 \equiv (1,2) \end{pmatrix}$$

La reparación Lamarckiana, por su parte, repara tan sólo la solución candidata decodificada del individuo, sin modificar a este último. La reparación Lamarckiana se lleva a cabo del siguiente modo:

$$\begin{aligned} & x = (x_1, x_2) \Rightarrow \vartheta_{candidato} = (pu, de) \\ & \text{Reparación Lamarckiana:} \\ & \text{Si } pu < de, \text{ entonces } \vartheta = (pu, de) \\ & \text{Si no (solución infactible en precedencia)} \\ & \quad \text{Si } pu > de, \text{ entonces } y = \frac{pu - de}{2},\ pu = \lfloor pu - y \rfloor,\ de = \lceil de + y \rceil \\ & \quad \text{Si } pu = de, \text{ entonces } de = \min(w_j(k), de+1),\ pu = \max(de-1, 1), \\ & \quad \vartheta = (pu, de) \end{aligned} \tag{7.30}$$

En palabras, si la recogida resulta ser mayor que la entrega, éstas son intercambiadas por las posiciones de recogida y entrega factibles más cercanas a las posiciones infactibles para evaluar la función objetivo. Si por ejemplo se utiliza codificación real en PSO, el proceso a partir de las partículas hasta la secuencia de paradas se muestra a continuación:

$$\begin{pmatrix} x_1 = (0.7, 3.9) \\ x_2 = (0.2, 5.2) \\ x_3 = (4.1, 4.9) \\ x_4 = (2.6, 4.9) \\ x_5 = (4.8, 1.4) \\ x_6 = (3.1, 1.1) \\ x_7 = (1.9, 3.2) \end{pmatrix} \Rightarrow \begin{pmatrix} \vartheta_1 = (1,4) \\ \vartheta_2 = (1,6) \\ \vartheta_3 = (5,5) \\ \vartheta_4 = (3,5) \\ \vartheta_5 = (5,2) \\ \vartheta_6 = (4,2) \\ \vartheta_7 = (2,4) \end{pmatrix} \Rightarrow \text{Reparación} \Rightarrow \begin{pmatrix} \vartheta_1 = (1,4) \\ \vartheta_2 = (1,6) \\ \vartheta_3 = (5,6) \\ \vartheta_4 = (3,5) \\ \vartheta_5 = (3,4) \\ \vartheta_6 = (3,4) \\ \vartheta_7 = (2,4) \end{pmatrix} \Rightarrow \begin{pmatrix} \boxed{3^+} \to 1^+ \to 2^+ \to \boxed{3^-} \to 1^- \to 2^- \\ \boxed{3^+} \to 1^+ \to 2^+ \to 1^- \to 2^- \to \boxed{3^-} \\ 1^+ \to 2^+ \to 1^- \to 2^- \to \boxed{3^+} \to \boxed{3^-} \\ 1^+ \to 2^+ \to \boxed{3^+} \to 1^- \to \boxed{3^-} \to 2^- \\ 1^+ \to 2^+ \to \boxed{3^+} \to \boxed{3^-} \to 1^- \to 2^- \\ 1^+ \to 2^+ \to \boxed{3^+} \to \boxed{3^-} \to 1^- \to 2^- \\ 1^+ \to \boxed{3^+} \to 2^+ \to \boxed{3^-} \to 1^- \to 2^- \end{pmatrix} \tag{7.31}$$

7.3.1.3 Inicialización de las soluciones

El procedimiento de inicialización de las soluciones candidatas se realiza aleatoriamente cada vez que entra un nuevo requerimiento al sistema, respetando las ecuaciones (7.19), (7.21) y (7.23) que corresponden a las representaciones de la estrategia de HPC basada en GA, PSO-$\mathbb{R}$ y PSO-$\mathbb{N}$, respectivamente. Estas ecuaciones, sin embargo, no garantizan que se respeten las restricciones de precedencia y carga máxima. Para realizar una búsqueda adecuada, la población

siempre debe incluir al menos una solución candidata factible. Como al final de la secuencia el vehículo siempre queda vacío, ubicar la recogida y entrega al final de la secuencia (en ese orden) garantiza que no se sobrepase la carga máxima de los vehículos y que se respete la restricción se precedencia. En conclusión la población se inicializa con $N_{pob}-1$ individuos al azar y con un individuo $x=\left(w_j-1, w_j\right)$.

Cabe mencionar que no se incluye la mejor solución del instante discreto anterior (ni ninguna similar) dentro de la población inicial puesto que la entrada principal del sistema, la demanda de requerimientos de recogida y entrega, es una perturbación demasiado importante, que produce que el estado del sistema (y por lo tanto el espacio de búsqueda) sea lo suficientemente distinto entre un instante discreto y el siguiente como para impedir que las soluciones óptimas sean similares. Por lo tanto, ésta estrategia propuesta en 5.5 no se utiliza en este sistema.

7.3.1.4 Reducción de la carga computacional

Para reducir la carga computacional del proceso de optimización, se almacenan las soluciones ya evaluadas asociadas a cada decisión de despacho, tal como se presenta en la Sección 5.7.1. Como se trata de un problema discreto, es posible que durante la ejecución del algoritmo evolutivo, una solución ya evaluada tenga que volver a serlo. Luego, como esta solución ya se encuentra almacenada, se ahorra el tiempo computacional asociado a la evaluación de la función objetivo.

7.3.2 Algoritmo para predicción a dos pasos

Este algoritmo es el que resuelve las asignaciones óptimas para la llamada entrante. Debido a la complejidad de ejecutar los algoritmos considerando en una sola solución candidata todos los vehículos y la secuencia que correspondería a cada uno, para cada vehículo se ejecuta una corrida estándar de un algoritmo evolutivo u optimización por enjambre de partículas, y se comparan las soluciones obtenidas para cada vehículo. Luego, la nueva secuencia se aplica al vehículo con que se obtenga una menor función objetivo.

Como la función objetivo en la predicción a dos pasos para una secuencia asignada al requerimiento entrante depende explícitamente de los costos de inserción óptimos de los patrones (ecuación (7.15)), se debe ejecutar el algoritmo de predicción a un paso para encontrar estos últimos (para aquellos patrones se predice a un paso pues ya se asume asignada una solución para la llamada entrante). En base a estas consideraciones, el algoritmo genérico para predicción a dos pasos se presenta en el Algoritmo # 10.

Se debe notar que la representación de soluciones candidatas, manejo de restricciones, etc., son iguales que para el algoritmo a un paso. La única diferencia es que la función objetivo que se evalúa es distinta.

Algoritmo # 10: Algoritmo genérico para predicción a dos pasos

00. Llega un requerimiento en el tiempo k. Se asume conocida la secuencia $S(k-1)$.

01. Para cada vehículo $j \in V$

02. Inicializar población $\left\{ \vartheta_\ell^j \,/\, \ell = 1,...,n \right\}$, donde n es el tamaño de la población.

03. Para $t = 1,..., \text{Max\_iterations}$

04. Decodificar, y reparar si corresponde, cada solución candidata ϑ_ℓ^j, en una secuencia $S_j^{\vartheta_\ell}(k)$.

05. Si $S_j^{\vartheta_\ell}(k)$ es factible.

05. Para $h = 1,..., H(k+2)$ (llamados probables)

06. Asumiendo que $S_j^{\vartheta_\ell}(k)$ se asigna al vehículo j, resolver el problema de predicción a un paso (Algoritmo 11) para el patrón h en el instante $k+1$, un tiempo continuo τ después de haber entrado el requerimiento en el tiempo k.

07. Sea $J_{\vartheta_\ell}^h$ la función objetivo de la solución óptima para la inserción del requerimiento h.

08. Siguiente h

09. Finalizar Si $S_j^{\vartheta_\ell}(k)$ es factible..

10. Evaluar el fitness de cada solución candidata, ϑ_ℓ^j, donde la secuencia generada es $S_j^{\vartheta_\ell}(k)$, del siguiente modo.

$$fitness = \begin{cases} \displaystyle\sum_{j=1}^{F} \left[\sum_{h=1}^{H(k+2)} p_h^{\Delta T(k+2)} (k+2) \cdot J_{\vartheta_\ell}^h \Big|_{S_j(k),h} \right] & \text{si } S_j^{\vartheta_\ell}(k) \text{ es factible} \\ M_1 & \text{no satisface carga máxima} \\ M_2 & \text{no satisface precedencia} \end{cases}$$

donde M_2 se aplica sólo si no corresponde aplicar reparación.

11. Actualizar población de acuerdo a la heurística utilizada.

12. Siguiente t

13. Mejor solución para vehículo j es $S_j^{\vartheta_\ell^*}(k)$

14. Siguiente vehículo j

15. Requerimiento se asigna al vehículo cuya mejor solución $S_j^{\vartheta_\ell^*}(k)$ presente el mejor fitness, en las posiciones definidas por dicha solución.

7.3.3 Algoritmo para predicción a un paso

El algoritmo para predicción a un paso se utiliza para seleccionar un vehículo y su secuencia de paradas para un nuevo requerimiento, sin considerar predicciones adicionales del efecto de otros posibles patrones de llamados. Para los problemas de predicción a dos pasos, se utiliza para calcular los costos de inserción de los patrones asumiendo que ya se ha asignado una solución

para la llamada entrante, y de este modo este algoritmo se utiliza para calcular el calcular el costo de las predicciones del sistema.

Debido a la complejidad de ejecutar los algoritmos considerando en una sola solución candidata todos los vehículos y la secuencia que correspondería a cada uno, para cada vehículo se ejecuta una corrida estándar de un algoritmo evolutivo u optimización por enjambre de partículas, y se comparan las soluciones obtenidas para cada vehículo. Luego, la nueva secuencia se aplica al vehículo con que se obtenga un menor costo de inserción de acuerdo a la ecuación (7.15).

Algoritmo 11: Algoritmo genérico para predicción a un paso
00. Llega un requerimiento en el tiempo k. Se asume conocida la secuencia $S(k-1)$.
01. Para cada vehículo $j \in V$
02. Inicializar población $\left\{ \vartheta_\ell^j / \ell = 1,...,n \right\}$, donde n es el tamaño de la población.
03. Para $t = 1,..., \text{Max\_iterations}$
04. Decodificar, y reparar si corresponde, cada solución candidata ϑ_ℓ^j, en una secuencia $S_j^{\vartheta_\ell}(k)$.
05. Evaluar el fitness de cada solución candidata, ϑ_ℓ^j, donde la secuencia generada es $S_j^{\vartheta_\ell}(k)$, del siguiente modo. $$fitness = \begin{cases} \sum_{j=1}^{F} \left(C_j(k+1) - \overbrace{C_j(k)}^{\text{known constant}} \right)\Big\|_{S_j(k-1),1} & \text{si } S_j^{\vartheta_\ell}(k) \text{ es factible} \\ M & \text{si no} \end{cases}$$ donde M es un valor muy alto que funciona a modo de penalización.
06. Actualizar población de acuerdo a la heurística utilizada.
07. Siguiente t
08. Mejor solución para vehículo j es $S_j^{\vartheta_\ell^*}(k)$
09. Siguiente vehículo j
10. Requerimiento se asigna al vehículo cuya mejor solución $S_j^{\vartheta_\ell^*}(k)$ presente el mejor fitness, en las posiciones definidas por dicha solución.

De acuerdo a lo explicado en 7.3.2 este algoritmo se utiliza siendo llamado desde el Algoritmo 10 para calcular los costos de inserción de los patrones en el segundo instante de predicción respecto de los requerimientos reales que entran al sistema, y por lo tanto corresponde a la 2da capa del algoritmo de solución.

7.3.4 Resumen de métodos de solución

Para aplicar la estrategia de HPC basada en GA o PSO se debe escoger el tipo representación (sólo en el caso de PSO) y de manejo de restricciones (GA y PSO), que serán utilizados tanto en el algoritmo a dos pasos como en el algoritmo a un paso. De acuerdo a las opciones de representación y manejo de restricciones, se puede escoger cualquiera de los siguientes 9 métodos.

- *P-PSO-*$\mathbb{R}$
- *BR-PSO-*$\mathbb{R}$
- *LR-PSO-*$\mathbb{R}$
- *P-PSO-*$\mathbb{N}$
- *BR-PSO-*$\mathbb{N}$
- *LR-PSO-*$\mathbb{N}$
- *P-GA*
- *BR-GA*
- *LR-GA*

Donde los prefijos P, R1, R2 indican que el manejo de restricciones de precedencia está basado en penalización (dura), reparación Baldwiniana, o reparación Lamarckiana, respectivamente. Los sufijos $\mathbb{R}$ y $\mathbb{N}$ indican que las soluciones son codificadas en un dominio continuo o entero, respectivamente. En el caso de GA no se especifica el sufijo, pues la codificación realiza siempre en el espacio discreto.

7.4 Experimentos por simulación

Las simulaciones se estructuran del siguiente modo. Se considera un área urbana de aproximadamente 81 km^2, donde se tienen 4 ó 9 vehículos según se indique, que viajan a una velocidad constante de 20 km/hr. Se realizan simulaciones por periodos de dos o tres horas, donde aparecen 120 requerimientos de servicio en tiempo real.

Se realiza una simulación por un periodo de dos horas. El esquema considera una flota de nueve vehículos, cada uno con capacidad para cuatro pasajeros. Las decisiones de ruteo se toman en tiempo real por el controlador. Se considera un área urbana de aproximadamente 81 km^2, dividida en 4 zonas y que genera 6 patrones posibles de requerimientos. Se asume que los vehículos viajan en línea recta entre paradas y que la velocidad es constante de 20 km/hr.

El experimento base consiste en un simulación de 2 horas (salvo que se indique lo contrario) donde aparecen en tiempo real 120 requerimiento de servicio.

Para evaluar la calidad de las estrategias basadas en algoritmos evolutivos, salvo que se indique lo contrario, se utiliza una función de desempeño que mide la calidad de la respuesta dinámica del sistema, en lugar de la calidad de las funciones objetivos en los problemas de optimización. Esta función de desempeño *JP* consiste en la suma real de los tiempos de espera y viaje de los usuarios, y de los costos operacional de los operadores, con los mismos pesos relativos usados en la ecuación (7.13).

Para obtener las estadísticas de los diversos métodos, se considera 30 réplicas para cada una de las distintas condiciones en que son probados (distintos parámetros, estrategias de inicialización de población, etc.). Además, los estudios han sido realizados con computadores Mac OS X, v10.5.6, con procesadores Core 2 Duo de 2.66 GHz y 4 GB de memoria RAM.

7.4.1.1 Justificación experimental del uso de algoritmos evolutivos

Las primeras pruebas consisten en comparar el desempeño del algoritmo genérico basado en PSO, con un algoritmo de enumeración explícita (EE) para el problema de predicción a dos pasos. Se utiliza en particular la estrategia BR-PSO-$\mathbb{R}$. En esta prueba se evalúa el tiempo computacional que requieren estos métodos para encontrar las soluciones a medida que aumenta el número de requerimientos, manteniendo el tamaño de la flota fijo en 9 vehículos, y usando 9 zonas. Además, la estrategia basada en PSO es probada utilizando: 10 iteraciones y 10 partículas, y 20 iteraciones y 20 partículas.

La Figura 85 muestra, como era esperable, la imposibilidad de utilizar EE en problemas de tamaño real para despacho en tiempo real, pues el tiempo computacional explota rápidamente una vez que el número de requerimientos pasa de los 30.

La única ventaja real de utilizar EE es que permite obtener el óptimo global. Sin embargo, la experiencia en la literatura especializada ha mostrado que en la práctica la diferencia entre el valor óptimo y el encontrado por PSO es bastante pequeño, que es confirmado por pruebas realizadas para este caso, donde la diferencia es del orden de un 2 a 3% cuando se utilizan 10 partículas y 10 iteraciones (Cortés *et al*., 2009). Además, el procedimiento debe ser ejecutado repetidas veces en tiempo real, lo que justifica aún más la fortaleza de utilizar procedimientos como algoritmos evolutivos (en este caso PSO) para resolver los problemas de ruteo de vehículos en tiempo real con el enfoque predictivo.

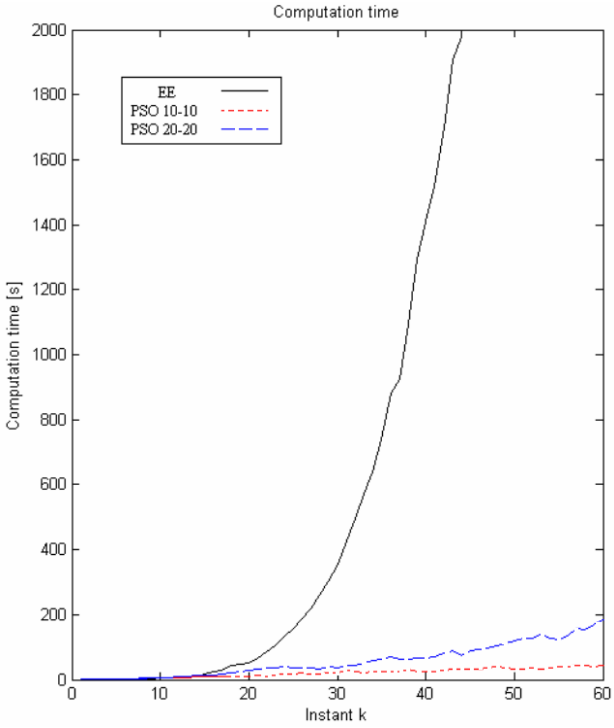


Figura 85: Tiempo computacional versus número de requerimientos

7.4.1.2 Validación de estrategia de control predictivo híbrido

A continuación se realiza un experimento similar al descrito, pero para un periodo de 180 minutos donde se realizan 120 llamados, con el objetivo de comprobar la efectividad del enfoque

predictivo para el problema de ruteo de vehículos, utilizando una estrategia basada en PSO. Pruebas preliminares indican que el uso de 10 partículas y 10 iteraciones entrega un compromiso adecuado entre tiempo computacional y calidad de las soluciones. Se compara entonces los resultados utilizando el enfoque con predicción a un paso (que equivale al caso estático, sin predicciones) con el enfoque de predicción a dos pasos (que considera las predicciones de las llamadas futuras probables), al utilizar 4 zonas y 4 llamadas probables, y 9 zonas y 9 llamadas probables. Se reportan los resultados al utilizar 30 réplicas de cada experimento, desde la Tabla 17 a la Tabla 19.

Tabla 17. Comparación de desempeño de predicción a un paso y a dos pasos, para 9 zonas y 6 llamadas probables

PSO	Tiempo de espera (min)		Tiempo de viaje (min)		Tiempo total (min)	
	Promedio	Desv. Est.	Promedio	Desv. Est.	Promedio	Desv. Est.
Predicción a un paso	21.70	2.22	23.88	1.51	45.58	3.47
Predicción a dos pasos	20.03	3.33	23.17	1.33	43.20	4.23
Ahorro de tiempo	1.67		0.71		2.38	
Mejora %	**7.68**		**2.98**		**5.22**	

Tabla 18. Comparación de desempeño de predicción a un paso y a dos pasos, para 4 zonas y 4 llamadas probables

PSO	Tiempo de espera (min)		Tiempo de viaje (min)		Tiempo total (min)	
	Promedio	Desv. Est.	Promedio	Desv. Est.	Promedio	Desv. Est.
Predicción a un paso	21.92	4.52	23.98	1.53	45.90	5.33
Predicción a dos pasos	19.02	3.36	22.69	1.46	41.71	4.58
Ahorro de tiempo	2.74		1.29		4.19	
Mejora %	**13.23**		**5.38**		**10.05**	

Tabla 19. Comparación del costo operacional por vehículo para predicción a un paso y a dos pasos.

Tiempo de viaje de vehículos (min/veh)	4 zonas, 4 probabilidades		9 zonas, 6 probabilidades	
PSO	Promedio	Desv. Est.	Promedio	Desv. Est.
Predicción a un paso	179.87	11.75	182.39	7.84
Predicción a dos pasos	178.73	8.42	183.41	13.06

La Tabla 17 y la Tabla 18 muestran el desempeño de los dos algoritmos en el periodo de tres horas. Se puede apreciar a partir de los resultados que la estrategia de predicción a dos pasos

sistemáticamente se desempeña mejor que la estrategia a un paso, justificando el uso de la componente predictiva en la metodología de ruteo de vehículos en tiempo real.

Los ahorros más importantes vienen de la componente de tiempo de espera, (13.23% en el mejor caso). De este resultado se puede inferir que los mayores beneficios de las predicciones son debidos a evitar tiempo de espera extra en los puntos de recogida futuras en las secuencias de los vehículos. Los ahorros en tiempo de viaje de los enfoques predictivos son menores que los obtenidos en tiempo de espera (5.38% en el mejor caso), y no se observan mejoras en el costo operacional (Tabla 19). Tal vez esto se deba al peso α utilizado en la función objetivo, lo que sugiere un análisis de este valor en el futuro para reanalizar estos resultados.

Parece además que el caso con menos zonas y menos llamados probables resulta en un esquema más eficiente y con más ahorros al utilizar la estrategia predictiva. Un motivo que explica esta diferencia es que al utilizar menos zonas y menos llamados probables permite una predicción más adecuada de las probabilidades y patrones asociados, permitiendo una mejor predicción de las situaciones futuras.

7.4.1.3 Comparación de PSO, GA y diversos esquemas de representación y manejo de restricciones

En esta sección se realiza un análisis comparativo del desempeño de distintas representaciones y manejos de restricciones para las estrategias de HPC basadas en GA y PSO, como han sido presentadas en 7.3.

Se realiza una simulación por un periodo de dos horas. El esquema considera una flota de nueve vehículos, cada uno con capacidad para cuatro pasajeros. Las decisiones de ruteo se toman en tiempo real por el controlador. Se considera un área urbana de aproximadamente 81 km^2, dividida en 4 zonas y que genera 6 patrones posibles de requerimientos. Se asume que los vehículos viajan en línea recta entre paradas y que la velocidad es constante de 20 km/hr.

El experimento se replica 30 veces para cada variante de los algoritmos. Los indicadores relevantes con la función objetivo para el periodo completo de dos horas, y el tiempo computacional promedio para resolver los problemas de optimización asociados con la decisión de las posiciones de inserción de los requerimientos entrantes, que se muestran en la Tabla 20.

De los resultados resulta claro que las estrategias de HPC basadas en GA encuentran mejores soluciones que las basadas en PSO. Sin embargo, las basadas en GA son bastante más lentas que las basadas en PSO, lo que hace que no sea claro que para la aplicación en tiempo real se justifique su aplicación dado el aumento marginal en la calidad de la solución. A partir de esto se predice, que al menos para esta combinación de parámetros, PSO se desempeña mejor que GA.

Una comparación entra las estrategias basadas en PSO muestra que los métodos PSO-$\mathbb{N}$ alcanzan mejores soluciones en un tiempo más corto que el requerido para las estrategias PSO-$\mathbb{R}$, cuando estas son comparadas utilizando la misma estrategia de manejo de restricciones (ver Tabla 21). Por lo tanto, se concluye que las estrategias PSO-$\mathbb{N}$ son más eficientes entre las estudiadas. Este resultado es coherente con la intuición, al considerar que en un problema de optimización discreto como este, naturalmente sería resuelto de mejor forma con una estrategia discreta. En este sentido, se esperaría que GA funcionara bien para este problema. Esto se da en el sentido que GA encuentra mejores soluciones, pero su elevada demanda computacional es

indeseable en un problema práctico. Este comportamiento parece ser explicado por la naturaleza de la heurística de búsqueda. Es sabido que PSO presenta una convergencia más rápida que GA. Si a esto se le suma que se almacenan las soluciones ya visitadas, se concluye que PSO evalúa menos soluciones, ya que en un momento PSO está evaluando soluciones ya visitadas y así se ahorra el tiempo de la evaluación de la función objetivo. Por su parte GA siempre está buscando nuevas soluciones debido al operador de mutación, y por esto la convergencia no impide buscar nuevas soluciones. Por esto PSO es más rápido que GA. Como se verá más adelante, la heurística de PSO es más recomendable que la de GA para este problema en particular, pues a pesar de la rápida convergencia encuentra mejores soluciones consistentemente.

La Tabla 22 muestra el tiempo computacional extra y la mejora en la calidad de las soluciones para los métodos PSO-$\mathbb{N}$ y PSO-$\mathbb{R}$, cuando una estrategia con penalización es reemplazada por una con reparación. Es claro que, al menos para las estrategias PSO, los enfoques de reparación encuentran mejores soluciones que aquellas con penalización, pero, requieren un mayor tiempo computacional. Esto se debe a que las estrategias de reparación son capaces de evaluar más soluciones candidatas que las otras. Sin embargo, no es posible decidir que estrategia es mejor, dado el compromiso existente entre tiempo computacional y calidad de las soluciones. No obstante, es relevante notar que este comportamiento no se aplica a GA. La estrategia de GA basada en penalización es la mejor en términos de calidad de soluciones entre las variantes analizadas. En este caso el enfoque parece guiar al óptimo de un mejor modo que los otros enfoques para el manejo de restricciones, lo que permite encontrar mejores soluciones, aún evaluando menos soluciones candidatas (al menos que en la segunda estrategia de reparación).

Tabla 20: Estadísticas de desempeño de los algoritmos propuestos para el DPDP

Estadísticas / Métodos	Función de desempeño real (min)				Tiempo computacional (seg)			
	Promedio	Peor caso	Mejor caso	Desv. estándar	Promedio	Peor caso	Mejor caso	Desv. estándar
P-PSO-$\mathbb{N}$	6818.9	7214.6	6378.6	171.39	7.522	8.218	70.319	0.303
BR-PSO-$\mathbb{N}$	6809	7136.1	6487.8	132.79	12.509	14.291	11.686	0.515
LR-PSO-$\mathbb{N}$	6800.1	7041.3	6536.4	119.27	11.530	12.399	10.845	0.335
P-PSO-$\mathbb{R}$	6846.5	7123.5	6544.3	129.52	8.378	9.336	7.700	0.310
BR-PSO-$\mathbb{R}$	6823.6	7168.8	6510.5	156.27	13.468	14.868	12.888	0.455
LR-PSO-$\mathbb{R}$	6810.6	7078.2	6523	107.67	11.763	12.495	10.820	0.335
P-GA	6787.3	7022.4	6582	90.81	27.010	28.551	24.897	0.956
BR-GA	6803	7039.4	6581	107.63	24.764	26.064	23.413	0.643
LR-GA	6796.8	6898.5	6689.8	43.388	31.751	33.353	30.33	0.548

Tabla 21. Comparación entre estrategias PSO-$\mathbb{N}$ y PSO-$\mathbb{R}$.

Cambios en la calidad de los algoritmos al pasar de una estrategia PSO-$\mathbb{N}$ a una PSO-$\mathbb{R}$		
Método	**Tiempo computacional extra [%]**	**Aumento en función de desempeño[%]**
P-PSO	11.38	0.40
BR-PSO	7.67	0.21
LR-PSO	2.02	0.15

Tabla 22. Comparación entre estrategias de penalización y reparación de soluciones infactibles.

Cambios en la calidad de los algoritmos al pasar de una estrategia de penalización a una de reparación		
Método	**Tiempo computacional extra [%]**	**Disminución en función de desempeño [%]**
BR-PSO-$\mathbb{N}$	66.30	0.15
BR-PSO-$\mathbb{R}$	60.75	0.33
LR-PSO-$\mathbb{N}$	53.28	0.28
LR-PSO-$\mathbb{R}$	40.40	0.31
BR-GA	-8.32	-0.23
LR-GA	17.55	-0.14

Los resultados anteriores parecen sugerir que las estrategias basadas en PSO son más adecuadas para la aplicación en tiempo real que aquellas basadas en GA, fundamentalmente por ser considerablemente más rápidas. Además, la representación entera parece ser la mejor opción (por sobre la continua), y la evolución Lamarckiana muestra un mejor desempeño que la evolución Baldwiniana. Sin embargo, sólo es posible asegurar estos resultados para una población de 10 individuos y 10 iteraciones, aunque, es esperable que el comportamiento sea similar utilizando otros tamaños de población y números de iteraciones.

En la próxima sección, se presentan los resultados para la sintonización de parámetros óptimos (tamaño de población y número máximo de iteraciones) basado en un enfoque de optimización multi-objetivo. En principio, se escoge la estrategia LR-PSO-$\mathbb{N}$ por ser la que obtiene las mejores soluciones dentro de las basadas en PSO, pero es importante recalcar que no se está diciendo que por esto sea la mejor estrategia. Como ha sido explicado anteriormente, existe un compromiso entre carga computacional y calidad de las soluciones entre distintas estrategias. Por lo tanto, la selección de uno de los métodos dependería del objetivo del operador. Además, el procedimiento para la sintonía de parámetros será aplicado al método R2-GA. El motivo fundamental es que permitirá ilustrar que la metodología permite escoger el mejor de los métodos. La selección de este en particular es que se trata del análogo a R2-PSO-$\mathbb{R}$ en GA.

7.4.1.4 Sintonía de tamaño de población e iteraciones en base a optimización multi-objetivo

Como ha sido mencionado anteriormente, se realizará el procedimiento de sintonización del tamaño de población y número de iteraciones en base a optimización multi-objetivo para los métodos R2-PSO-$\mathbb{N}$ y R2-GA-$\mathbb{N}$. Cabe mencionar que esta metodología sirve tanto para encontrar los tamaños de población y número de iteraciones óptimos, como para comparar efectivamente las distintas estrategias. Para encontrar los promedios de tiempo computacional y costo real del sistema, se utilizan 50 réplicas para cada combinación considerada de los parámetros a sintonizar. El tamaño de la población y número de iteraciones se prueban en el espacio $\{5,8,11,14,17,20\}^2$. En la Figura 86 se presentan la Frontera de Pareto y otras combinaciones sub-óptimas de tamaño de población y número de iteraciones, donde la calidad de las soluciones es medida de acuerdo al desempeño final real del sistema.

Tabla 23. Tamaños de población y número de iteraciones óptimos de acuerdo a análisis multi-objetivo para LR-PSO-$\mathbb{N}$.

Conjunto Pareto-óptimo para LR-PSO-$\mathbb{N}$			
Tamaño de población	**Número de iteraciones**	**Tiempo computacional por requerimiento (s)**	**Función de desempeño real (min)**
5	5	3,5434	7181,9
5	8	5,0465	6964,8
5	11	6,3738	6905,3
5	14	7,3122	6852,2
5	17	8,1777	6782,7
8	11	10,24	6751,8
11	20	18,201	6738,1
14	20	21,693	6734
17	11	18,141	6746,3
17	17	22,658	6726,4
20	11	20,404	6736

Tabla 24. Tamaños de población y número de iteraciones óptimos de acuerdo a análisis multi-objetivo para LR-GA-$\mathbb{N}$.

Conjunto Pareto-óptimo para LR-GA			
Tamaño de población	**Número de iteraciones**	**Tiempo computacional por requerimiento (s)**	**Función de desempeño real (min)**
5	5	10,846	6958,3
8	5	16,15	6874,9
8	11	32,039	6787,2
8	14	38,128	6785,7
8	17	43,661	6760,1
11	5	20,849	6837,3
11	14	43,776	6759,7
14	5	24,007	6799,2
14	11	40,042	6769,2
17	14	48,91	6758,6
17	17	53,824	6756,5
17	20	58,124	6756,4
20	5	29,215	6790,2
20	14	50,362	6757,9
20	17	55,171	6756,4

De la Figura 86 se observa que para el método LR-PSO-$\mathbb{N}$ el conjunto óptimo de Pareto posee 11 puntos, de los cuales 5 corresponden a una población de tamaño 5, y una a un tamaño 8 (ver Tabla 23). En esta zona la calidad de las soluciones aumenta rápidamente respecto del tiempo computacional, mientras que para combinaciones con poblaciones más grandes, la calidad aumenta muy lentamente. Es posible notar en la Figura 86. que el punto rodilla de la frontera de Pareto del método LR-PSO-$\mathbb{N}$ corresponde al punto que utiliza una población de 8 individuos y 11 iteraciones. Para el método LR-GA, por su parte, no se aprecia un punto rodilla tan claro, y en su lugar la calidad de las soluciones mejora lentamente a medida que el tiempo computacional aumenta. Además, los puntos de la frontera de Pareto están compuestos uniformemente por

puntos con diversos tamaños de poblaciones, que es muy distinto a lo que ocurre en PSO, donde casi la mitad de los puntos corresponde a la población más pequeña dentro de las disponibles. Esto es congruente con el hecho conocido que en general PSO es más rápido que GA porque puede obtener rendimientos similares con poblaciones más pequeñas, por lo tanto evalúa menos soluciones candidatas. En conclusión, PSO puede alcanzar mejores desempeños con poblaciones más pequeñas, mientras que GA necesita poblaciones más grandes, tal como se aprecia en los conjuntos óptimos de Pareto (ver Tabla 24).

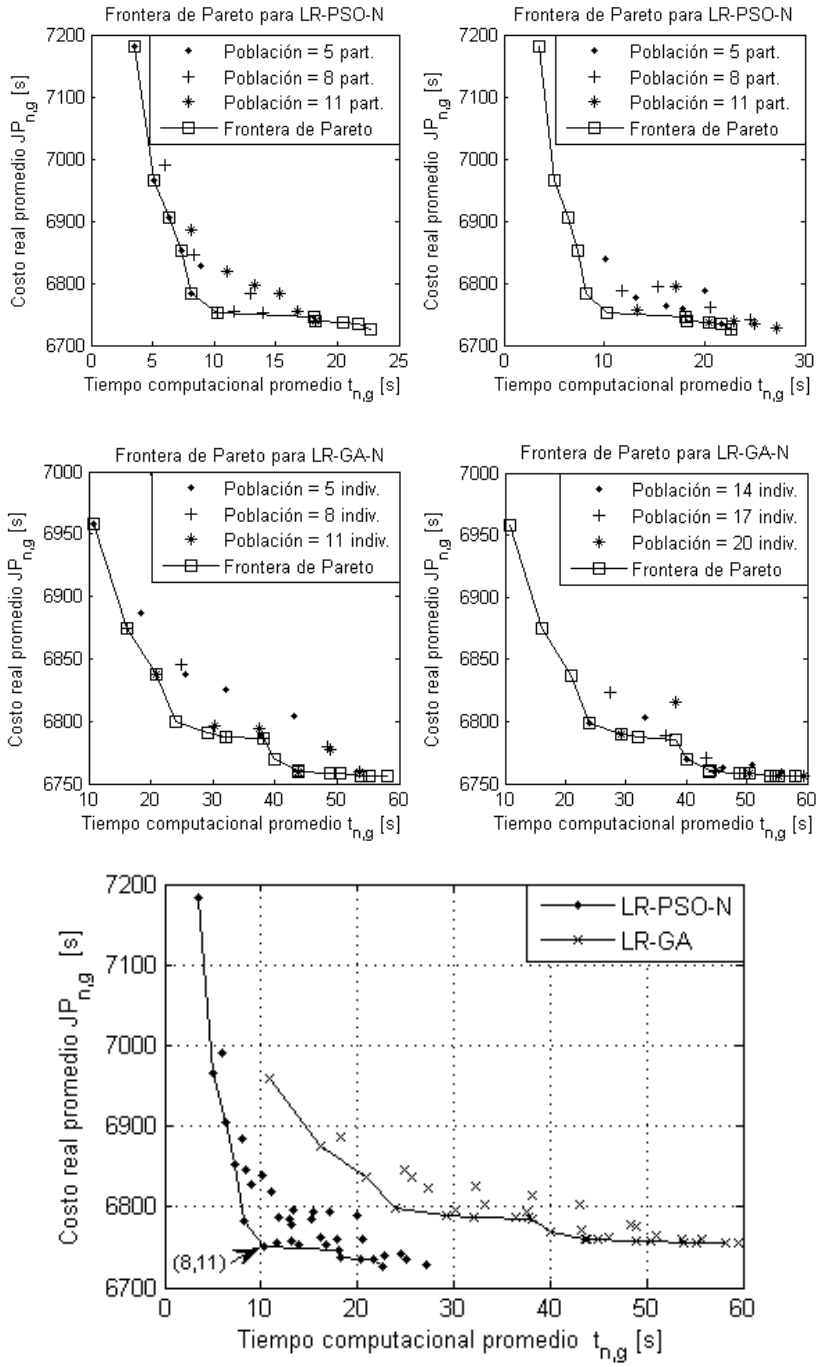


Figura 86. Fronteras de Pareto para las distintas combinaciones de tamaño de población y número de generaciones

Sin embargo, en este sistema el comportamiento descrito anteriormente es ligeramente distinto, pues PSO es más rápido incluso para poblaciones de igual tamaño con un mismo número de iteraciones. Esto ocurre porque a PSO evalúa menos soluciones candidatas que GA. Para entender esto primero hay que notar que en la implementación las soluciones probadas son guardadas, lo que permite no calcular su fitness nuevamente. Además, es sabido que PSO converge más rápido, y por lo tanto es más común que algunas soluciones candidatas se repitan con otras anteriores (dado que el problema es discreto), y entonces para una población y número de iteraciones de tamaños fijos, PSO evalúa menos soluciones candidatas y es más rápido. Por otro lado, si bien PSO no alcanza mejores soluciones que GA para las mismas combinaciones (como se aprecia claramente en el primer análisis para poblaciones de tamaño 10, con 10 iteraciones), sí es mucho más rápido que GA con combinaciones de parámetros tal que la calidad de las soluciones es comparable, o incluso, puede ser más rápido y con mejores soluciones que aquellas encontradas por algún GA. Finalmente, esto se refleja en la Figura 87. Fronteras de Pareto dadas por calidad de soluciones del problema de optimización, para distintas combinaciones de tamaño de población y número de generaciones, donde puede verse claramente que las combinaciones de parámetros de PSO en el conjunto óptimo de Pareto dominan a todas las combinaciones del conjunto óptimo de Pareto del algoritmo basado en GA. En otras palabras, para cada combinación óptima de parámetros utilizada en GA, existe al menos una combinación de parámetros tal que el algoritmo basado en PSO es más rápido, y alcanza mejores soluciones. Se puede concluir entonces que PSO tiene una heurística de búsqueda mucho más adecuada que GA, para el problema en particular.

En el DVRP, la solución debe ser obtenida rápidamente de acuerdo a las necesidades del proveedor del servicio y sus clientes. Entonces, se puede escoger fácilmente los parámetros que alcancen las mejores soluciones en un tiempo razonablemente corto. Por ejemplo, si se define un tiempo máximo de decisión de 10 segundos para cada requerimiento, se puede escoger una población de tamaño 5 con 17 iteraciones (ver Tabla 23), que alcanza la mejor función objetivo (6782 min) dentro de las opciones que se demoran en promedio menos que 10 [s] (8.18 [s] en este caso). También se puede utilizar como criterio el punto rodilla, el cual indicaría que conviene utilizar una población de tamaño 8 y 11 iteraciones.

El análisis anterior ha sido realizado utilizando como medida de calidad de la solución el desempeño final del sistema, luego de que todas las recogidas y entregas han sido satisfechas. Sin embargo, utilizar esa función objetivo para la sintonía de parámetros, de cierto modo evalúa una mezcla entre las estrategias de optimización y la bondad de la estrategia global de ruteo de vehículos. Por ejemplo, una mala calidad de las predicciones (parte de la estrategia global) puede influir en un determinado bajo desempeño de cierta combinación de parámetros. Esto es observable claramente en el hecho que combinaciones con poblaciones grandes y más iteraciones debieran siempre encontrar mejores soluciones, y así aparecer en el conjunto óptimo de Pareto, hecho que no sucede, pues las predicciones no son 100% precisas. Por lo tanto, una solución extremadamente precisa pierde su efectividad si la predicción no lo es tanto. Entonces, para evaluar sólo la estrategia de optimización, se debe utilizar como indicador de la calidad de solución la función objetivo usada en el proceso de optimización. En este caso se debiese eliminar (o atenuar considerablemente) el comportamiento ruidoso que exhiben la fronteras de Pareto. Este comportamiento se presenta en la Tabla 25, la Tabla 26 y la Figura 87. Como se observa en esta figura, en general el comportamiento de PSO es muy similar al caso anterior. La mayor parte de los integrantes del conjunto óptimo de Pareto poseen el menor tamaño de población disponible. Por otra parte, para GA el comportamiento es algo distinto. Antes los

integrantes del conjunto óptimo de Pareto estaban más distribuidos, pero ahora se encuentran más cargados hacia las poblaciones más grandes.

Como era esperado, las fronteras de Pareto parecen ser menos ruidosas. Ahora, al analizar los algoritmos con los valores de la función objetivo encontrados en el proceso de optimización, se puede ver que al utilizar poblaciones más grandes efectivamente se encuentran mejores soluciones (excepto un caso en GA, debido a la estocasticidad sólo del método de optimización) y por lo tanto aparecen en los conjuntos óptimos de Pareto. De hecho, es posible apreciar que hay una variabilidad mucho menor en las tendencias de los valores de la función objetivo utilizada en el proceso de optimización, que en el desempeño final.

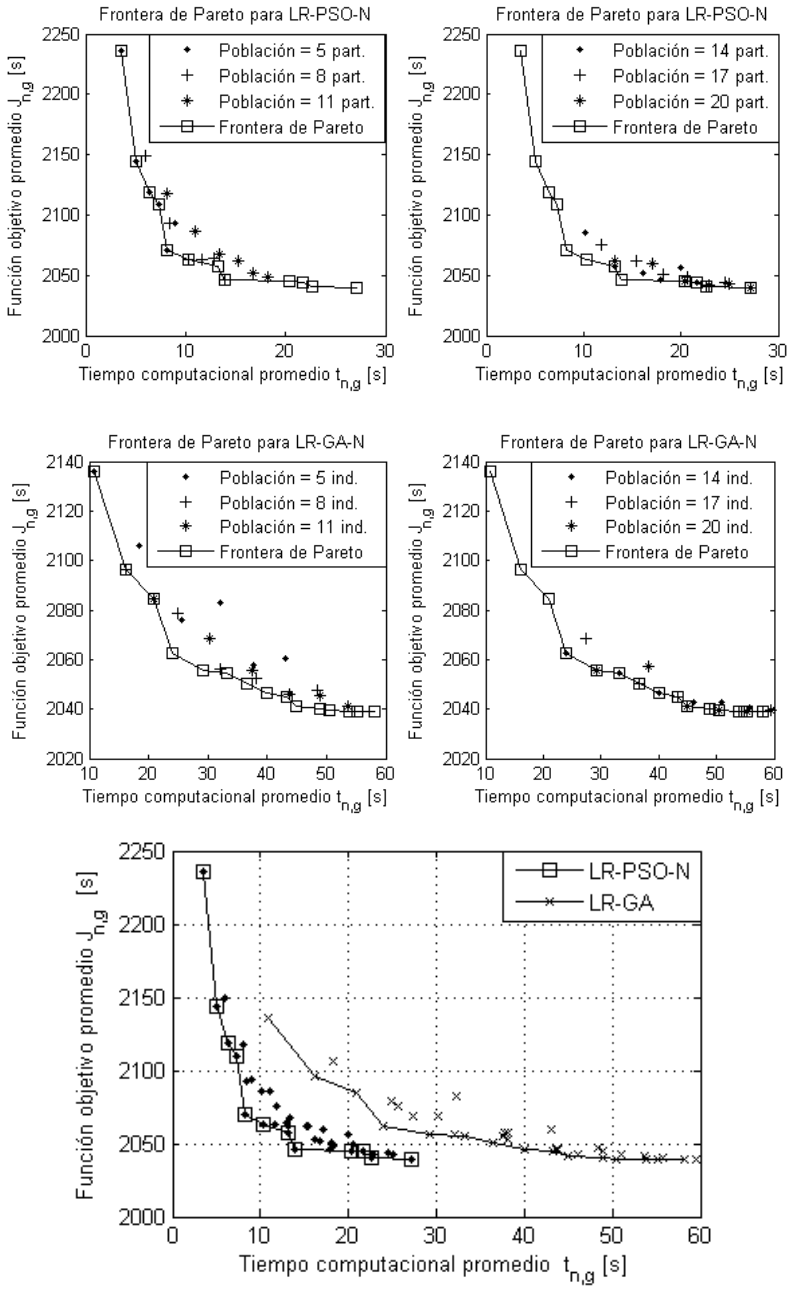


Figura 87. Fronteras de Pareto dadas por calidad de soluciones del problema de optimización, para distintas combinaciones de tamaño de población y número de generaciones

Tabla 25. Tamaños de población y número de iteraciones óptimos según calidad de soluciones del problema de optimización, de acuerdo a análisis multi-objetivo para LR-PSO-$\mathbb{N}$.

Conjunto Pareto óptimo para R2-PSO-$\mathbb{N}$			
Tamaño de población	**Número de iteraciones**	**Tiempo computacional por requerimiento (s)**	**Función objetivo en optimización (min)**
5	5	3,5434	2235,8
5	8	5,0465	2144,3
5	11	6,3738	2118,9
5	14	7,3122	2109,1
5	17	8,1777	2070,4
8	11	10,24	2062,6
8	20	13,908	2045,9
14	8	13,186	2057,6
14	20	21,693	2044,4
17	17	22,658	2040,9
20	11	20,404	2045,4
20	20	27,111	2039,3

Tabla 26. Tamaños de población y número de iteraciones óptimos según calidad de soluciones del problema de optimización, de acuerdo a análisis multi-objetivo para LR-GA-$\mathbb{N}$.

Conjunto Pareto-óptimo para R2-GA			
Tamaño de población	**Número de iteraciones**	**Tiempo computacional por requerimiento (s)**	**Función objetivo en optimización (min)**
5	5	10,846	2136
8	5	16,15	2096,4
11	5	20,849	2084,8
14	5	24,007	2062,3
14	8	33,176	2054,6
14	11	40,042	2046,3
17	8	36,515	2050,4
17	11	43,263	2044,7
17	14	48,91	2040,3
17	17	53,824	2039,1
17	20	58,124	2039,1
20	5	29,215	2055,9
20	11	44,975	2041
20	14	50,362	2039,6
20	17	55,171	2039,1

7.5 Análisis y conclusiones

Se ha presentado diversas estrategias basadas en algoritmos genéticos (GA) y optimización por enjambre de partículas (PSO) para la aplicación de control predictivo híbrido en el problema de ruteo dinámico de vehículos. Se ha justificado la aplicación de este tipo de técnicas debido a la alta demanda computacional de otras técnicas estándar, como la enumeración explícita de las soluciones.

Se ha propuesto una arquitectura a dos niveles para resolver los problemas de optimización a dos pasos; un nivel superior para el primer paso de predicción, y un nivel inferior para el segundo paso de predicción. La separación de las capas se realiza debido a que los espacios factibles para las acciones de control en el segundo paso de predicción son altamente dependientes de las acciones de control en el primer paso de predicción. En efecto, si una llamada se asigna a un vehículo o no, va a cambiar el largo de la secuencia, y debido a esto el espacio factible para las soluciones de los patrones de requerimientos futuros. Por lo tanto, la segunda capa del algoritmo de solución (que encuentra las inserciones para los patrones) se ejecuta dada una solución candidata del primer paso de predicción, y eso permite conocer la calidad de la solución candidata para el requerimiento entrante, cuya solución óptima es la que efectivamente se aplicará como acción de control (u asignación de las nuevas secuencias).

En cada una de estas capas, los algoritmos evolutivos se han utilizado para encontrar las posiciones de inserción óptimas para los requerimientos de recogida y entrega para cada uno de los vehículos por separado. Esta separación de la optimización para los distintos vehículos se debe a que son problemas absolutamente independientes, cuyos espacios de búsqueda no tienen relación alguna. Debido a esta separación, finalmente se asigna el servicio de los requerimientos a aquel vehículo que presente un menor costo predicho para este fin. Como trabajo futuro se propone diseñar alguna arquitectura que utilice algoritmos evolutivos donde las soluciones candidatas, individuos o partículas, representen soluciones para todos los vehículos de la flota y/o pasos de predicción, y sean capaces de lidiar efectivamente con la alta dependencia de los espacios factibles para las soluciones del segundo paso de predicción respecto de las del primer paso, y con la independencia absoluta de los espacios de búsqueda de un vehículo y otro.

Para la implementación basada en PSO se ha propuesto dos alternativas de representación para las variables discretas: manejando las partículas con coordenadas continuas y truncando a los valores discretos para evaluar las funciones objetivos; o bien con partículas directamente discretas. Para GA, debido a su naturaleza discreta, se ha propuesto una representación con coordenadas discretas al igual que en el segundo paso de PSO.

Los algoritmos utilizados para resolver este problema deben manejar 4 tipos de restricciones. Dos de ellas, las de exclusividad y consistencia, son satisfechas implícitamente por la representación propuesta, mientras que las otras dos, las de precedencia y carga máxima, no. La restricción de carga máxima es difícil de manejar mediante reparación y más mediante representación factible, y es por lo tanto abordada mediante un enfoque de penalización. La restricción de precedencia por su parte, es de más fácil manejo ya que debido a la representación sólo afecta a las acciones de control y por lo tanto es más fácil generar esquemas de reparación. Entonces, se proponen tres enfoques, dos de reparación y otro de penalización. Las dos estrategias de reparación se basan en evolución Lamarckiana y en evolución Baldwiniana, mientras que la estrategia de penalización utiliza un único valor muy alto para penalizar las soluciones, y no se debe calcular la función objetivo.

Luego, las distintas alternativas de representación y manejo de restricciones para las implementaciones basadas en GA y PSO son comparadas para un número de iteraciones y tamaño de población fijo. De esta prueba aparece que los métodos con representación entera, tanto de PSO como GA son mejores en términos de calidad de solución que los basados en representación continua en PSO. A su vez, los métodos basados en PSO para este problema parecen ser mejores debido fundamentalmente a que requieren un tiempo considerablemente

mejor que aquellos basados en GA, a pesar de que estos últimos encuentran mejores soluciones. Los métodos basados en PSO son más rápidos que aquellos basados en GA, a pesar de utilizar el mismo tamaño de población y máximo número de iteraciones, ya que PSO converge más rápido y entonces, como se almacenan las soluciones ya evaluadas, no es necesario calcular sus funciones objetivo nuevamente, con lo que se ahorra un tiempo considerable.

Al analizar las diversas estrategias de manejo de restricciones se encuentra lo siguiente. Al comparar las reparaciones Lamarckianas con Baldwinianas se encuentra que consistentemente la primera encuentra mejores soluciones que la segunda. Además, en el caso de PSO, la reparación Lamarckiana, junto con encontrar las mejores soluciones, lo hace en un menor tiempo computacional. Se concluye entonces que la reparación Lamarckiana es más eficiente que la Baldwiniana para PSO. Esto se puede explicar debido a que en el caso de la evolución Lamarckiana las soluciones también se reparan en las partículas, lo que guía de mejor modo la búsqueda, y además, ayuda a una más rápida convergencia. En el caso de GA la evolución Lamarckiana también brinda mejores resultados que la Baldwiniana, pero demora un tiempo mayor que la anterior. Debido a esto en el caso de GA no queda claro que tipo de reparación es la más conveniente. Para poder entender esto se requiere más trabajo, y queda pendiente como trabajo futuro. Respecto de la aplicación de estrategias de penalización versus reparación, en GA se encuentra que las estrategias de penalización permiten encontrar las mejores soluciones de todas las probadas, pero con el alto costo computacional de las estrategias basadas en GA. Respecto de su uso en PSO se encuentra que con el uso de estrategias de penalización encuentran peores soluciones que con estrategias de reparación, pero en un menor tiempo computacional. El comportamiento en el caso de PSO se justifica desde el punto de vista que con un enfoque de penalización se evalúa menos soluciones (pues como las infactibilidades están en las acciones de control, no es necesario predecir el comportamiento del sistema y por lo tanto se evita la evaluación de la función objetivo). Sin embargo en el caso de GA no es tan claro porque se produce ese funcionamiento.

Si bien las comparaciones efectuadas respecto de las distintas representaciones y estrategias de reparación pueden representar tendencias, éstas no son seguras debido a que sólo han sido comparadas con un tamaño de población y número de iteraciones, y como ha sido estudiado en las aplicaciones anteriores, en algunos procesos ayuda más una estrategia con ciertos tamaños de población y números de iteraciones, mientras que es mejor otra estrategia para otros parámetros.

Para superar la limitación recién presentada, se realiza un análisis multi-objetivo que permite comparar distintos métodos con una variedad de tamaños de poblaciones y números de iteraciones. Sin embargo, debido a la alta demanda computacional que se necesita para este análisis, sólo se realiza para dos de los 9 métodos presentados. Con este análisis se comprueba efectivamente que la implementación basada en PSO con representación entera y reparación Lamarckiana es mejor que la implementación basada en GA con la misma representación y reparación. Este análisis también sirve para realizar una sintonía del tamaño de población y número de iteraciones, con el objetivo de tener un algoritmo eficiente en términos de esfuerzo computacional y la calidad de las soluciones. Los parámetros escogibles son aquellos que se encuentran en la frontera de Pareto y se presenta como escogerlos con criterios de punto silla o de un tiempo máximo permitido de cómputo. Si bien el análisis posee problemas con la estocasticidad de los algoritmos evolutivos, permite seleccionar parámetros que cumplen cercanamente con los requerimientos.

8 Conclusiones

En este trabajo de tesis se ha analizado la aplicación de algoritmos evolutivos para el control predictivo híbrido de sistemas no lineales. Este análisis se basa en la aplicación por simulación de diversas estrategias que utilizan algoritmos evolutivos para el control predictivo híbrido de tres procesos: un reactor batch con entradas discretas, un reactor batch con entradas mixtas y sistema de ruteo dinámico de vehículos.

Dadas las condiciones en las que ha sido realizada la tesis, tanto para el reactor batch con entradas mixtas como para el sistema de ruteo dinámico de vehículos, se justifica la aplicación basada en algoritmos evolutivos pues las estrategias óptimas son prohibitivas en términos computacionales, o bien no son capaces de resolver los problemas de optimización satisfactoriamente. Distinto es el caso del reactor batch con entradas discretas, pues el método de *branch and bound* resuelve satisfactoriamente en el tiempo permitido. Sin embargo, si se contara con procesadores menos potentes o con menos tiempo para aplicar las acciones de control de modo que no fuese aplicable este método, se justificaría la aplicación de algoritmos evolutivos.

No obstante lo anterior, independiente de si se justifica o no la aplicación de algoritmos evolutivos versus un método convencional, para cada uno de los procesos ha sido posible desarrollar estrategias basadas en algoritmos evolutivos que controlan satisfactoriamente los procesos (seguimientos y esfuerzos de control adecuados encontrados en un tiempo de cómputos aceptable).

Se ha propuesto un análisis multi-objetivo para manejar de un modo eficiente la relación entre calidad de las soluciones y esfuerzo computacional. Este análisis consiste en determinar los conjuntos Pareto-óptimos en términos de desempeño del sistema controlado y el tiempo computacional requerido para encontrar las soluciones, que se encuentran en función del tamaño de población y máximo número de iteraciones del algoritmo evolutivo. De este modo se puede encontrar el conjunto de combinaciones óptimas de tamaño de población y número de iteraciones. Finalmente, el tamaño de la población y el máximo número de iteraciones a utilizar se puede escoger de acuerdo a distintos criterios como un punto rodilla o bien un máximo tiempo aceptable para encontrar la acción de control. El método conceptualmente está bien desarrollado, pero tiene la complicación que la estocasticidad de los algoritmos evolutivos influye bastante en la medición de la media de la calidad de las soluciones obtenidas con cada combinación de parámetros. Por lo tanto, el conjunto de parámetros que se encuentra en la frontera de Pareto para un método puede no ser la verdadera, ya que las soluciones encontradas se encuentran influenciadas por el ruido. Además, el método demanda un alto esfuerzo computacional, ya que se debe realizar varias réplicas, para muchas combinaciones de parámetros, para cada método. A pesar de esto, el análisis sirve efectivamente para encontrar parámetros que entreguen un buen comportamiento (ya que no se puede asegurar que sea el óptimo debido a la estocasticidad) en términos de eficiencia y calidad de las soluciones. Además, el alto esfuerzo computacional para aplicar esta sintonía se justifica en sistemas complejos donde los problemas de optimización deben ser resueltos repetidamente y la limitante para el tiempo de cómputos es relevante.

El análisis multi-objetivo resulta útil también para comparar distintas estrategias (heurísticas, representación, manejo de representaciones, etc.) basadas en algoritmos evolutivos. Este análisis se puede interpretar como un indicador gráfico. Al asumir que la calidad del algoritmo evolutivo

se encuentra determinada por su frontera de Pareto, una estrategia será mejor que otra si los puntos de su frontera dominan a los puntos de la frontera de Pareto de la otra estrategia. De este modo, una estrategia puede ser mejor que otra en un determinado rango del conjunto de parámetros óptimos, pero puede ser superada por la otra estrategia en otro rango. Dado lo anterior, el análisis multi-objetivo se convierte en una excelente herramienta para analizar estrategias basadas en algoritmos evolutivos, especialmente si se utilizan para resolver problemas de optimización en un tiempo pequeño que no permite asegurar la convergencia.

Mediante este análisis, se ha logrado comprobar efectivamente que al incluir la solución del instante discreto anterior en la nueva población para encontrar la nueva acción de control se mejoran los métodos, al ser posible encontrar mejores acciones de control en un menor tiempo. Se ha encontrado también que los métodos basados en PSO se benefician más que GA de esta estrategia. Esto se debe a que la solución del instante anterior inserta en la nueva población generalmente comienza como el *gbest* en PSO, y por lo tanto todas las partículas hacen uso de la información, mientras que en GA sólo unas pocas la utilizan. Además, en ningún caso estas estrategias tienen efectos adversos, por ejemplo de guiar la población hacia un óptimo local, pues justamente la naturaleza basada en población hace a estos métodos robustos frente a estas complicaciones. Además, en la mayoría de los métodos la utilización de la solución anterior desplazada de acuerdo a la estrategia de horizonte deslizante del control predictivo brinda mayores beneficios que la utilización de dicha solución intacta. Esto es esperable, pues dicha alternativa es capaz de responder mejor a los transientes (y del mismo modo en régimen permanente) que al usar la solución intacta, y está inspirada en la estrategia de horizonte deslizante propia del control predictivo.

En esta tesis se ha estudiado además diversas estrategias para el manejo de restricciones y representación de soluciones. Se ha visto que para las variables discretas es mejor alternativa representarlas de modo discreto, tal como es su naturaleza, pues pareciera que así se guía de mejor manera la búsqueda de nuevas soluciones. Para el caso de las estrategias de manejo de restricciones, de acuerdo a la teoría, se aconseja se eviten las estrategias basadas en penalización, siempre que se encuentren representaciones factibles y/o funciones de reparación que no sesguen la búsqueda, y en el caso de las funciones de reparación, que no sean complejas de un punto de vista computacional. Sin embargo, no ha sido posible ratificar estas recomendaciones, debido a que no se ha probado todas las alternativas en todos los procesos estudiados (por restricciones de tiempo para probar todas las alternativas), y además a que la teoría esta expuesta desde un punto de vista que siempre se calcula la función objetivo, lo que no se cumple en los procesos analizados. Esto último sucede por dos razones: primero, cuando las soluciones son discretas, en los sistemas estudiados se almacenan las soluciones ya probadas, por lo tanto si se las visita de nuevo es posible evitar su cómputo; y segundo, cuando las infactibilidades se producen a nivel de la acción de control y se utiliza estrategias de penalización dura (como son las implementaciones realizadas en estos procesos), esta penalización se aplica con un valor muy alto y no es necesario calcular la función objetivo. A pesar de esto, se puede proponer ciertos consejos para las representaciones y funciones de reparación:

- Si se utiliza estrategias de reparación, se recomienda que sea basada en evolución Lamarckiana, pues guía de mejor modo la búsqueda de nuevas soluciones. Además, acelera la convergencia, lo que permite reducir el tiempo computacional si se está almacenando soluciones ya visitadas.

- En PSO, se recomienda que las representaciones utilizadas sean consistentes con la física del problema. Así, si una partícula se mueve poco, el efecto físico de la acción de control también se mueve poco.

También, se ha mostrado que es posible mejorar el rendimiento de los controladores predictivos basados en algoritmos evolutivos mediante el almacenamiento de soluciones ya visitadas. Se confirma esto al notar que las pendientes del tiempo computacional requerido para resolver los problemas de optimización son decrecientes, cuando si no se almacenan las soluciones estas son constantes (o crecientes si existen problemas computacionales respecto del manejo de memoria).

Finalmente, corresponde hacer una mención al trabajo futuro que se ha derivado de esta tesis, y del trabajo no realizado que es relevante para la aplicación de algoritmos evolutivos en control predictivo híbrido de sistemas no lineales.

De los experimentos realizados se ha observado que la función objetivo parece influir en la calidad de una alternativa u otra de diseño del algoritmo evolutivo. Esto se basa específicamente en el hecho que para el reactor batch discreto, una estrategia de reparación es mejor que las otras, y justamente se da que la búsqueda con esa estrategia es mejor en la misma zona que la función objetivo aplica las penalizaciones más duras. Basado en esto, resulta interesante estudiar en un trabajo futuro como la estructura de la función objetivo afecta el desempeño de las diversas estrategias basadas en algoritmos evolutivos.

En base a las limitantes observadas de las estrategias aplicadas para los procesos, esta tesis ha dejado abiertas las siguientes ramas para trabajo futuro:

- Mejorar el análisis multi-objetivo para que sea más robusto respecto a la estocasticidad de los algoritmos evolutivos.
- Mejorar el análisis multi-objetivo de modo que no haya que utilizar tan sólo un subconjunto de los números de iteraciones o tamaños de población posibles, sino que se pueda utilizar todo el espectro posible (definido por cotas mínimas o máximas). Como el esfuerzo computacional aumentaría considerablemente en dicho caso, se podría utilizar un enfoque basado en optimización multi-objetivo evolutiva.
- El diseño, análisis e implementación de estrategias mixtas entre algoritmos evolutivos y técnicas de programación no lineal y/o entera mixta para el control predictivo híbrido no lineal.
- El diseño, análisis e implementación de estrategias de discretización del espacio factible en las variables continuas, de acuerdo a la evolución temporal del proceso.

Los primeros dos puntos se podrían lograr mediante un precidimiento que consista en una parametrización de la calidad de las soluciones (función objetivo o de desempeño) respecto del número de iteraciones y del tamaño de población, utilizando alguna estructura de función (exponencial negativa, por ejemplo), y posterior regresión de los parámetros de dicha función. De este modo, para calcular la calidad media de un algoritmo al utilizar una combinación de parámetros se estaría utilizando todas las réplicas simuladas, en lugar de sólo las réplicas simuladas con los parámetros correspondientes, tal como se ha hecho en este trabajo. Luego, dado que el efecto es tal como si se utilizaran más réplicas, el efecto del ruido debiera ser reducido. Además, como se logra una parametrización respecto del número de iteraciones y del tamaño de la población, se puede trabajar en todo el rango, realizando pocas iteraciones por

combinación de parámetros, en lugar de trabajar en un subconjunto del rango, y realizando muchas iteraciones por combinación de parámetros, tal como se ha hecho en esta tesis. La calidad de este procedimiento, sin embargo, depende en principio de que se encuentre una estructura funcional acorde con las características del algoritmo.

Los siguientes aspectos son relevantes para la aplicación de algoritmos evolutivos en controladores predictivos no lineales, y como no han sido tratados en esta investigación, quedan propuestos como trabajo futuro:

- Estudiar experimentalmente la predicción teórica que dice que la inclusión en la nueva población de la mejor solución en el instante anterior, desplazada según la estrategia de horizonte deslizante, es cada vez más beneficiosa que el caso que no desplaza dicha solución, a medida que aumenta el tiempo de muestreo.
- Encontrar condiciones de estabilidad para sistemas con controladores predictivos híbridos basados en algoritmos evolutivos, y estudiar la influencia de cada opción de diseño del algoritmo sobre la estabilidad del sistema.
- Analizar la influencia de distintos horizontes de predicción en el desempeño de sistemas con controladores predictivos híbridos basados en algoritmos evolutivos.

9 Referencias

[1] Ackley, D., Littman, M. (1994). A case for Lamarckian evolution. En: Artificial Life III, Langton CG (Ed). Reading, MA: Addison-Wesley, 3-10.

[2] Ali, M., Torn, A. (2004). Population set based global optimization algorithms: Some modifications and numerical studies. Computers and Operations Research 31, 1703 – 1725.

[3] Alur, R., Coucoubetis, C., Henzinger, T., Ho, P., Nicollin, X., Olivero, A., Sifakis, J., Yovine, S. (1995). The algorithmic analysis of hybrid systems. Theoretical Computer Science 138, 3-34.

[4] Angeline, P. (1998). Evolutionary optimization versus particle swarm optimization: Philosophy and performance differences. Evolutionary Programming VII, 601-610.

[5] Angira, R., Babu, B. (2006). Optimization of process synthesis and design problems: A modified differential evolution approach. Chemical Engineering Science 61, 4707-4721.

[6] Arumugam, M., Rao, M., Ramaswamy, P. (2005). New hybrid genetic operators for real coded genetic algorithm to compute optimal control of a class of hybrid systems. Applied Soft Computing 6, 38-52.

[7] Arumugam, M., Rao, M., (2006). On the performance of the particle swarm optimization algorithm with various inertia weight variants for computing optimal control of a class of hybrid systems. Discrete Dynamics in Nature and Society Volume, 1–17.

[8] Back, T. (1993). Optimal Mutation Rates in Genetic Search. En: Forrest (Editor), Proceedings of the Fifth International Conference on Genetic Algorithms, 2–8.

[9] Back, T., Hoffmeister, F., Schwefel. H. (1991). A survey of evolution strategies. En: Proc. 4th Int. Conf. on Genetic Algorithms (San Diego, CA). Eds: Belew, R., Booker, L. Morgan Kaufmann, 2–9.

[10] Back, T.,Schwefel, H. (1993). An overview of evolutionary algorithms for parameter optimization Evolut. Comput. 1, 1-23.

[11] Back, T. (1992). The interaction of mutation rate, selection, and self adaptation within a genetic algorithm. En: Manner, R., and Manderic (Eds). Parallel problem solving from nature, 2, Amsterdam: North Holland.

[12] Back, T., Schutz, M. (1995). Evolution strategies for mixed-integer optimization of optical multilayer systems. Proceedings of the fourth annual conference on evolutionary programming, McDonnel, J., Reynolds, R., Fogel, D., Eds. MIT Press, Cambridge, MA, 33-51.

[13] Back.T., Fogel, D., Michalewicz, Z. (1997). Handbook on Evolutionary Computation. IOP Publishing Ltd and Oxford University Press.

[14] Back, T., Fogel, D., Michalewicz, T. (2000). Evolutionary Computation 1. Bristol, UK: Institute of Physics Publishing.

[15] Baric, M., Grieder, P., Baotic, M. and Morari, M., (2008). An efficient algorithm for optimal control of PWA systems with polyhedral performance indices. Automatica 44, 296 – 301.

[16] Bean, J., Hadj-Alouane, A. (1992). A dual genetic algorithm for bounded integer problems. University of Michigan ,Tech. Rep., 92 (53).

[17] Beccuti, A., Papafotiou, G., Frasca, R., Morari, M. (2007). Explicit Hybrid Model Predictive Control of the dc-dc Boost Converter**.** Power Electronics Specialists Conference, PESC, IEEE, 17-21 June, 2503 - 2509.

[18] Bemporad, A., Morari, M. (1999). Control of systems integrating logic, dynamics, and constraints. Automatica, 35(3), 407–427.

[19] Bemporad, A., Borrelli, F., Morari, M., (2002). On the optimal control law for linear discrete time hybrid systems. En: Proc. 5th International Workshop, Hybrid Systems: Computation and Control, 25 - 27 March, Stanford, California, USA, Tomlin C. J. and Greenstreet M. R. (eds.), Lecture Notes in Computer Science 2289, 105 - 119, Springer Verlag.

[20] Bemporad, A., Di Cairano, S., Henriksson, E., Johansson, K. (2007). Hybrid model predictive control based on wireless sensor feedback: An experimental study**.** Conference on Decision and Control, 46th IEEE ,12-14 Dec, 5062 – 5067.

[21] Bengea, S., DeCarlo, R. (2005). Optimal control of switching systems. Automatica (41), 11-27.

[22] Berbeglia, G., Cordeau, J., Laporte, G. (2009). Dynamic pickup and delivery problems. In Press European Journal of Operational Research. doi:10.1016/j.ejor.2009.04.024.

[23] Bertsekas, D. (1999). Nonlinear Programming: 2nd Edition. Athena Scientific, Belmont, Massachusetts.

[24] Bertsimas, D., Van Ryzin, G. (1991). A Stochastic and Dynamic Vehicle Routing Problem in the Euclidean Plane. Operations Research 39, 601-615.

[25] Bertsimas, D., Van Ryzin. G. (1993). Stochastic and Dynamic Vehicle Routing Problem in the Euclidean Plane with Multiple Capacitated Vehicles. Operations Research 41, 60-76.

[26] Bertsimas, D., Van Ryzin, G. (1993). Stochastic and Dynamic Vehicle Routing with General Demand and Interarrival Time Distributions. Applied Probability 25, 947-978.
[27] Beyer, H., Arnold, D. (2001). Theory of evolution strategies: A tutorial. En: L. Kallel, B. Naudts, A. Rogers. Eds. Theoretical Aspects of Evolutionary Computing, 109-134. Springer, Berlin, Heidelberg. New York.
[28] Blackwell, T. (2005). Particle swarms and population diversity. Soft Computing 9, 793–802.
[29] Borrelli, F., Baotic, M., Bemporad, A., Morari, M., (2005). Dynamic programming for constrained optimal control of discrete-time linear hybrid systems. Automatica 41, 1709-1721.
[30] Bremermann, H., Rogson, M., and Salaff, S. (1966). Global Properties of Evolution Processes. En: Pattee *et al*. (eds), Natural Automata and Useful Simulations, 3–41. Spartan Books.
[31] Camacho, E., Bordons, C. (2004). Model Predictive Control. Second Edition, Springer-Verlag.
[32] Cao, E., Lai, M. (2007). An Improved Differential Evolution Algorithm for the Vehicle Routing Problem With Simultaneous Delivery and Pick-up Service. Third International Conference on Natural Computation, ICNC 3, 436-440.
[33] Cao, Y., Wu, Q. (1997). Mechanical design optimization by mixed-variable evolutionary programming. Proceedings of the 1997 IEEE Conference on Evolutionary Computation, pp. 443–446.
[34] Cardoso, M., Salcedo, R., Feyo de Azevedo, S., Barbosa, D. (1997). A simulated annealing approach to the solution of MINLP problems. Computers & Chemical Engineering 21, 1349–1364
[35] Causa, J., Karer, G., Nuñez, A., Saez, D., Skrjanc, I., Zupancic, B. (2008). Hybrid fuzzy predictive control based on genetic algorithms for the temperature control of a batch reactor. Computers & chemical engineering 32, 3254-3263.
[36] Chakraborty, U., Das, S., Konar, A. (2006). Differential evolution with local Neighborhood. In: Proc. of IEEE Congress on Evolutionary Computation (CEC-2006), IEEE Press, 73957402.
[37] Chen, J., Tsao, Y. (1993). Optimal design of machine elements using genetic algorithms. Journal of the Chinese Society of Mechanical Engineers, 14(2), 193–199.
[38] Chiou, J., Wang, F. (1998). A hybrid method of differential evolution with application to optimal control problems of a bioprocess system. En: Proc. IEEE Evol. Comput. Conf., 627–632.
[39] Clarke, D., Scattolini, R. (1991). Constrained receding-horizon predictive control. IEE Proceedings 138(4), 347-354.
[40] Clerc, M. (1999). The swarm and the queen: towards a deterministic and adaptive particle swarm optimization. Proc. I999 ICEC, Washington, DC, 1951 - 1957.
[41] Clerc, M., Kennedy, J. (2002). The particle swarm-explosion, stability and convergence in a multidimensional complex space. IEEE Transactions on Evolutionary Computation 6(1), 58-73.
[42] Clerc, M. (2006). Stagnation analysis in particle swarm optimisation or what happens when nothing happens. Technical Report CSM-460, Department of Computer Science, University of Essex. Edited by Riccardo Poli. Also available from http://clerc.maurice.free.fr/pso/.
[43] Coelho, J., de Moura Oliveira, P., Boaventura Cunha, J. (2005). Greenhouse air temperature predictive control using the particle swarm optimisation algorithm. In Computers and Electronics in Agriculture 49, 330-344.
[44] Cortés CE, Sáez D, Núñez A, Muñoz-Carpintero D. Hybrid Predictive Control for a Real-time Routed Transit System. Transportation Science 43, 27-42.
[45] Costa, L.,Olivera, P. (2001). Evolutionary algorithms approach to the solution of mixed integer non-linear programming problems. Computers & Chemical Engineering 25, 257–266.
[46] Cotta, C., Aldana, J., Nebro, A., Troya, J. (1995). Hybridizing Genetic Algorithms with Branch and Bound Techniques for the Resolution of the TSP. En: Pearson *et al*., (eds.), Artificial Neural Nets and Genetic Algorithms. Proceedings of the International Conference on Artificial Neural Nets and Genetic Algorithms. Ales, France, 1995, 277–280.
[47] Dahal, K., Galloway, S., Aldridge, C. (2003). Developing GA-based Hybrid approaches for a Real-world Mixed-integer Scheduling Problem. The 2003 Congress on Evolutionary Computation 3, 1887-1894.
[48] Dakev, N., Chipperfiield, A., Fleming, P. (1996). An evolutionary approach for path following optimal control of multibody systems. En: International conference on evolutionary computation, 512–516.
[49] Dasgupta, S., Das, S., Biswas, A., Abraham, A. (2009). On Stability and Convergence of the Population-Dynamics in Differential Evolution. AI Communications 22, 1–20.
[50] David, R., Alla, H. (1992). Petri nets and Grafcet: Tools for modeling discrete event systems, Prentice Hall.
[51] De Falco, I., Della Cioppa, A., Tarantino, A. (2006). Automatic Classification of Handsegmented Image Parts with Differential Evolution. In Rothlauf *et al.* (eds), EvoWorkshops 2006, LNCS 3907, 403-414.
[52] De Jong, K. (1975). An Analysis of the Behavior of a Class of Genetic Adaptive Systems. PhD thesis, University of Michigan.

[53] De Schutter, B., De Moor, B. (1999). The extended linear complementarity problem and the modeling and analysis of hybrid systems. En: Antsaklis et al. (eds), Hybrid Systems V, Lecture Notes in Computer Science 1567, 70–85. Springer.
[54] De Schutter, B., van den Boom, T. (2000). On model predictive control for max-min-plus-scaling discrete event systems. Technical Report bds:00-04, Control Lab, Fac. ITS, Delft Univ. Techn., Delft, The Netherlands.
[55] del Real, A., Arce, A., Bordons, C. (2007). Hybrid model predictive control of a two-generator power plant integrating photovoltaic panels and a fuel cell. Decision and Conference on Control, 2007 46th IEEE 12-14 Dec, 5447 – 5452.
[56] Dial, R. (1995). Autonomous Dial a Ride Transit – Introductory Overview. Transportation Research Part C 1995, 3, 261-275.
[57] Doma, M., Taylor, P., and Vermeer, P. (1996), Closed loop identification of MPC models for MIMO processes using genetic algorithms and dithering one variable at a time: Application to an industrial distillation tower. Comput. Chem. Eng. 20(2), 1035–1040.
[58] Dréo J, Pétrowski A, Siarry P, Taillard E. Metaheuristics for Hard Optimization Methods and Case Studies. Springer-Verlag, 2006.
[59] Eberhart, R., Kennedy, J. (1995). A new optimizer using particle swarm theory. En: Proc. 6th Int. Symp. Micromachine Human Sci 1, 39–43.
[60] Eberhart, R.,Shi, Y. (1998). Comparison between genetic algorithms and particle swarm optimization. En: Proc. 7th Conf. Evol. Program, 1447, 611–616.
[61] Eberhart, R., Shi, Y. (2000). Comparing inertia weights and constriction factors in particle swarm optimization. In Proc. Congr. Evolutionary Computation, San Diego, CA, July 2000, 84–88.
[62] Eiben, A., Nabuurs, R., Booij, I. (2001). The Escher evolver: Evolution to the people. Chapter 17, 425–439. Morgan Kaufmann publishers.
[63] Eiben, A., Smith, J. (2003). Introduction to Evolutionary Computing. Springer.
[64] Eiben, A., Aarts, E., Van Hee, K. (1991). Global convergence of genetic algorithms: a Markov chain analysis. En: Schwefel y Manner (eds). Proceedings of the 1st Conference on Parallel Problem Solving from Nature. Lecture Notes in Computer Science 496, 4-12. Springer, Berlin, Heidelberg, New York.
[65] Eksioglu, B., Volkan, A., Reisman, A. (2009). The vehicle routing problem: A taxonomic review. In Press Computers & Industrial Engineering. doi:10.1016/j.cie.2009.05.009.
[66] Ferrari-Trecate, G., Muselli, M., Liberati, D., Morari, M. (2000). Identification of piece-wise affine and hybrid systems. Technical report, ETH Zuerich. Tech. Rep. AUT00-21, http://www.control.ethz.ch.
[67] Fletcher, R., Leyffer, S. (1995). Numerical experience with lower bounds for MIQP branch-and-bound. Technical Report. Dept. of Mathematics, University of Dundee, Scotland, U.K. SIAM J. Optim., submitted. http://www.mcs.dundee.ac.uk:8080/sley¤er/miqp_art.ps.Z.
[68] Floudas, C. (1995). Nonlinear and Mixed Integer Optimization. Oxford Academic Press.
[69] Fogarty, T. (1989). Varying the probability of mutation in the genetic algorithm. In Proc. 3rd Int. Conf. on Generic Algorithms (Fairfax, VA. I9H9) Sehaffer (ed). 104-102.
[70] Fogel, L. (1964). On The Organization of Intellect. PhD Dissertation, University of California.
[71] Fogel, D.,Editor (1998). Evolutionary Computation: The Fossil Record. IEEE Press.
[72] Fogel, L., Owens, A., Walsh, M. (1966). Artificial Intelligence through Simulated Evolution. John Wiley & Sons.
[73] Fravolini, M., La Cava, M. (1999). Evolutionary Algorithms for adaptive predictive control. En: Proc. IEEE International Conference on Emerging Technologies and Factory Automation, Barcelona Spain, 55-61.
[74] Fu, J., Fenton, R., Cleghorn, W. (1991). A mixed integer-discrete-continuous programming method and its application to engineering design optimization. Engineering Optimization 17(3), 263-280.
[75] Fukuyama, Y. (2005). Comparative studies of particle swarm optimization techniques for reactie power allocation planning in power systems. Electrical Engineering in Japan 153(1), 34-41.
[76] Gendreau, M., Guertin, F., Potvin, J., Taillard, E. (1999).Parallel Tabu Search for Real-Time Vehicle Routing and Dispatching. Transportation Science 33, 381-390.
[77] Geyer, T., Torrisi, F., Morari, M. (2008). Optimal complexity reduction of polyhedral piecewise affine systems. Automatica 44(7), 1728-1740
[78] Geyer, T., Papafotiou, G., Morari, M. (2008). Hybrid Model Predictive Control of the Step-Down DC–DC Converter**.** Control Systems Technology, IEEE Transactions.
[79] Giorgetti, N., Ripaccioli, G., Bemporad, A., Kolmanovsky; I., Hrovat, D. (2006). Hybrid Model Predictive Control of Direct Injection Stratified Charge Engines**.** Transactions on Mechatronics, IEEE/ASME, 11(5), 499 - 506
[80] Glover, F., Laguna, M. (1997). Tabu Search. Kluwer, Norwell, MA.

[81] Goggos, V. King, R. (1997). Stochastic predictive control of mechatronic systems. Mechatronics 7, 129–140.
[82] Goldberg, D. (1989). Genetic Algorithms in search, optimization and machine learning. Addison-Wesley Publishing Corporation, Inc.
[83] Grefenstette, J. (1986). Optimization of control parameters for genetic algorithms. IEEE Transactions on Systems, Man, and Cybernetics, 16(1), 122–128.
[84] *Grossmann*, I. (2002). Review of nonlinear mixed-integer and disjunctive programming techniques. Opt. Eng. 3, 227-252
[85] Guo, C., Hu, J., Ye, B., Cao, Y. (2004). Swarm intelligence for mixed variable design optimization. Journal of Zhejiang University Science, 5(7), 851-860.
[86] Haghani, A., Jung, S. (2005). A dynamic vehicle routing problem with time-dependent travel times. Computers & Operations Research 32(11), 2959-2986.
[87] He, J., Yao, X. (2002). From an Individual to a Population: An Analysis of the First Hitting Time of Population-based Evolutionary Algorithms, IEEE Transactions on Evolutionary Computation, 6(5), 495-511
[88] Heemels, W., De Schutter, B., Bemporad, A. (2001). Equivalence of hybrid dynamical models. Automatica 37(7), 1085–1091.
[89] Herdy, M. (1996). Evolution Strategies with Subjective Selection. En Voigt *et al*. (eds), Parallel Problem Solving from Nature IV (PPSN-1996), Lecture notes in computer science 1141, 22–31.
[90] Hinton, G., Nowlan, S. (1987). How learning can guide evolution. Complex Systems 1, 495-502.
[91] Holland, J. (1975). Adpatation in Natural and Artificial Systems. University of Michigan Press.
[92] Jaw J, Odoni A, Psaraftis H, Wilson N. (1986). A heuristic algorithm for the multivehicle many-to-many advance-request dial-a-ride problem. Transportation Res. B: Methodological, 20, 243-257.
[93] Jiang, M., Luo, Y., Yang, S. (2007). Stochastic convergence analysis and parameter selection of the standard particle swarm optimization algorithm. Information Processing Letters 102, 8–16.
[94] Jih, W., Yun-Jen, J. (1999). Dynamic Vehicle routing using hybrid genetic algorithms. Proceeding of the IEEE International Conference on Robotics & Automation, Detroit, Michigan, 453-458.
[95] Johansson, M. (2002). Piecewise Linear Control Systems: A Computational Approach. Series: Lecture Notes in Control and Information Sciences 284. Springer, Berlin, Heidelberg..
[96] Jones, C. N., Morari, M. (2009). Approximate Explicit MPC using Bilevel Optimization. Proceedings of the European Control Conference, Budapest, Hungary.
[97] Julius, A., Sakar, M, Bemporad, A., Pappas, G. (2007). Hybrid model predictive control of induction of Escherichia coli. Conference on Decision and Control, 46th IEEE 12-14 Dec, 3913- 3918.
[98] Karer, G., Mušič, G., Škrjanc, I., Zupančič, B., (2007). Hybrid fuzzy model-based predictive control of temperature in a batch reactor. Computers & Chemical Engineering 31 (12), 1552-1564.
[99] Karer, G., Skrjanc, I., Zupancic, B. (2008). Self-adaptive predictive functional control of the temperature una an axothermic batch reactor. Chemical Engineering and Processing 47, 2379-2385.
[100] Kadirkamanathan, V., Selvarajah, K., Fleming, P. (2006). Stability Analysis of the Particle Dynamics in Particle Swarm Optimizer. IEEE Trans. On Evolutionary Computation, 10 (3): 245-255.
[101] Kennedy, J., Eberhart, R. (1997). A discrete binary version of the particle swarm algorithm. In: Systems, Man, and Cybernetics. IEEE International Conference on Computational Cybernetics and Simulation, 5, 4104-4108.
[102] Kleywegt, A., Papastavrou, J. (1998). The Dynamic and Stochastic Knapsack Problem. Operations Research 46, 17-35.
[103] Kennedy, J., and Eberhart, R. (2001). Swarm Intelligence. Morgan Kaufmann Publishers, Inc., San Francisco, CA.
[104] Lampinen, J., Zelinka, I. (1999a). Mixed integer-discrete-continuous optimization by differential evolution. Part 1: the optimization method. In Pavel, O. (Eds), Proceedings of MENDEL'99, 5th International Mendel Conference on Soft Computing, Brno, Czech Republic, 71-76.
[105] Lampinen, J., Zelinka, I. (1999b). Mixed integer-discrete-continuous optimization by differential evolution. Part 2: a practical example. In Pavel, O. (Eds), Proceedings of MENDEL'99, 5th International Mendel Conference on Soft Computing, Brno, Czech Republic, 77-81.
[106] Larsen, A. (2000). The Dynamic Vehicle Routing Problem. Ph.D. Thesis, Technical University of Denmark.
[107] Lazimy, R. (1985). Improved algorithm for mixed-integer quadratic programs and a computational study. Math. Programming, 32, 100-113.
[108] Lin, S., Kernighan, B. (1973). An Effective Heuristic Algorithm for the Traveling-Salesman Problem. Operations Research 21, 498-516.
[109] Lin, Y., Wang, F., Hwang, K. (1999). A hybrid method of evolutionary algorithms for mixed-integer nonlinear optimization problems. In Proc. IEEE Evol. Comput. Conf., 1999, 2159–2166.

[110] Linkers, D., Mahfonf, M. (1994). Advances in Model-Based Predictive Control. Chap. Generalized Predictive Control in Clinical Anaesthesia. Oxford University Press.
[111] Liu, H., Abraham, A. (2005). Fuzzy Turbulent Particle Swarm Optimization. Proceedings of the 5th International Conference on Hybrid Intelligent Systems, Brazil, IEEE CS Press, USA.
[112] Lobo, F., Lima, C., Michalewicz, Z., Eds. (2007). Parameter Setting in Evolutionary Algorithms. Springer Verlag, 2007.
[113] Lund, H., Miglino, O., Pagliarini, L., Billard, A., Ijspeert, A. (1998). Evolutionary robotics - A childrens game. In Proceedings of the 1998 IEEE International Conference on Evolutionary Computation, 154–158.
[114] Madsen, O., Raven, H., Rygaard, J. (1995). A heuristics algorithm for a dial-a-ride problem with time windows, multiple capacities, and multiple objectives. Annals of Operations Research 60, 193-208.
[115] Michalewicz, Z., Krawczyk, J., Kazemi, M, Janikow, C. (1990). Genetic algorithms and optimal control problems. Proceedings of the 29th IEEE Conference on Decision and Control. Honolulu.
[116] Michalewicz, Z. (1996). Genetic Algorithms + Data Structures = Evolution Program. 3rd Ed., Springer Verlag.
[117] Michalewicz, Z., Schmidt, M. (2002). Evolutionary algorithms and constrained optimization. In R. Sarker, M. Mohammadian, X. Yao, Eds. Evolutionary Optimization. Khiwer Academic Publishers, Boston, Chap. 3, 57-86.
[118] Michalewicz, Z., Nazhiyath., G. (1995). Genocop III a coevolutionary algorithm for numerical optimisation problems with nonlinear constraints. In Proceedings of the 1995 IEEE Conference on Evolutionary Computation, 647-651.
[119] Mitrovic-Minic, S., Krishnamurti, R., Laporte, G. (2004). Double-horizon based heurisitcs for the dynamic pickup and delivery problem with time windows. Transportation Research Part B, 38, 669-685.
[120] Mitrovic-Minic, S., Laporte, G. (2004). Waiting strategies for the dynamic pickup and delivery problem with time windows. Transportation Research Part B 38, 635-655.
[121] Montemanni, R., Gambardella, L., Rizzoli, A., Donati, A. (2005). Ant colony system for a dynamic vehicle routing problem. Journal of Combinatorial Optimization 10(4), 327-343.
[122] Munawar, S., Gudi, R. (2005). A nonlinear transformation based hybrid evolutionary method for MINLP solution. Chemical Engineering Research and Design, 83(A10), 1218-1236.
[123] Na, M., Hwang, I. (2006). Design of PWR power controller using model predictive control optimized by a genetic algorithm. Nuclear Engineering and Technology 38(1), 81-92.
[124] Na, M., Upadhyaya, B., (2006). Application of model predictive control strategy based on fuzzy identification to an SP-100 space reactor. Annals of Nuclear Energy 33, 1467-1478.
[125] Nagar, A., Heragu, S., Haddock, J. (1996). A Combined Branch and Bound and Genetic Algorithm Based for a Flowshop Scheduling Algorithm. *Annals of Operations Research* 63, 397–414.
[126] Nandola, N., Bhartiva, S. (2008). A multiple model approach for predictive control of nonlinear hybrid systems. Journal of Process Control 18, 131-148.
[127] Nedjah, N,. Macedo Mourelle, L. (2006). Swarm Intelligent Systems. Studies in Computational Intelligence, Vol. 26, Springer-Verlag.
[128] Negenborn, R., Beccuti, A., Demiray, T., Leirens, S., Damm, G., De Schutter, B., Morari, M., (2007). Supervisory hybrid model predictive control for voltage stability of power networks. American Control Conference, ACC '07, 5444 – 5449.
[129] van Nimwegen, E., Crutchfield, J., Mitchell, M. (1999). Statistical dynamics of the Royal Road genetic algorithm. Theoretical Computer Science 229, 41-102.
[130] Omran, M., Engelbrecht, A., Salman, A. (2009). Bare bones differential evolution. European Journal of Operational Research 196, 128-139.
[131] Onnen, C., Babuska, R., Kaymak, U., Sousa, J., Verbruggen, H., Isermann, R. (1997). Genetic algorithms for optimization in predictive control. Control Engineering Practice 5(10), 1363–1372.
[132] Osman, M., Abo-Sinna, M., Mousa, A (2005). An effective genetic algorithm approach to multiobjective routing problems (MORPs). Applied Mathematics and Computation 163, 769-781.
[133] Orosz, J., Jacobson, S. (2002). Analysis of static simulated annealing algorithms. Journal of Optimization theory and Applications 115(1), 165-182.
[134] Ozcan, E., Mohan, C. (1999). Particle swarm optimization: surfing the waves. Proc. 1999 Congress on Evolutionary Computation, 1939–1944. Piscataway.
[135] Pang, W., Wang, K., Zhou, C. (2004). Fuzzy discrete particle swarm optimization for solving traveling salesman problem. Proceedings of the 4th International Conference on Computer and Information Technology, 796-800.
[136] Paterlini, S., Krink, T. (2005). Differential Evolution and Particle Swarm Optimization in Partitional Clustering. Computational statistics & data analysis 50, 1220-1247.

[137] Poli, R., Broomhead, D. (2007). Exact Analysis of the Sampling Distribution for the Canonical Particle Swarm Optimiser and its Convergence during Stagnation. In Proceedings of the Genetic and Evolutionary Computation Conference, GECCO 2007, 134-141.
[138] Potocnik, B., Bemporad, A., Torrisi, F., Music, G., Zupancic, B. (2004). Hybrid modeling and optimal control of a multiproduct batch plant. Control Engineering Practice 12, 1127-1137.
[139] Potocnik, B., Music, G., Zupancic, B. (2005). Model predictive control of discrete-time hybrid systems with discrete inputs.**,** ISA Transactions 44, 199–211.
[140] Price, K., Storn, R., Lampinen, J. (2005). Differential Evolution. A Practical Approach to Global Optimization. Springer.
[141] Prugel-Bennet, A., Shapiro, J. (1994). An analysis of genetic algorithms using statistical mechanics. Phys. Review Letters 72(9), 1305-1309.
[142] Prugel-Bennett, A., Rogers, A. (2001). Modelling genetic algorithm dynamics. In: Kallel *et al*. (eds), Theoretical Aspects of Evolutionary Computing, 59 85. Springer.
[143] Prugel-Bennett, A. (2003). Modelling finite populations. In: Potta, *et al*. (eds). Foundations of Genetic Algorithms 7. Morgan Kaufmann, San Francisco, 2003
[144] Psaraftis, H. (1980). A dynamic programming solution to the single many-to-many immediate request dial-a-ride problem. Transportation Science 14(2), 130-154.
[145] Psaraftis, H. (1988). Dynamic vehicle routing problems. Golden y Assad (eds), Vehicle routing methods and studies, 223-248.
[146] Raman, R., Grossmann, I. (1991). Relation between MILP modeling and logical inference for chemical process synthesis. Comput. Chem. Engng 15(2), 73-84.
[147] Rechenberg, I. (1973). Evolutionsstrategie, volume 15 of problemata. Friedrich Forman Verlag (Gunther Holzboog KG).
[148] Rechenberg, I. (1965). Cybernetic solution path of an experimental problem. Library Translation 1122, Royal Aircraft Establishment, Farnborough, UK.
[149] Ren, Y., Cao, G., Zhu, X. (2006) Particle Swarm Optimizaron based predictive control of Proton Exchange Membrana Fuel Cell (PEFMC). Journal of Zhejiang University Science A 7 (3), 458-462.
[150] Richalet, J., Rault, A., Testud, J., Papon, J. (1978). Model Predictive Heuristic Control: Application to Industrial Processes. Automatica 14(2), 413-428.
[151] Richalet, J. (1993). Industrial Applications of Model Based Predictive Control. Automatica 29(5), 1251-1274.
[152] Rudolph, G. (1994). Convergence properties of canonical genetic algorithms. IEEE Transactions on Neural Networks 5(1), 96-101.
[153] G. Rudolph. Convergence of evolutionary algorithms in general search spaces. In ICEC-96 Proceedings of the 1996 IEEE Conference on Evolutionary Computation, 50-54.
[154] Sáez, D., Cortés, C., Núñez, A. (2008). Hybrid adaptive predictive control for the multi-vehicle dynamic pick-up and delivery problem based on genetic algorithms and fuzzy clustering. Computers & Operations Research 35, 3412-3438.
[155] Sandgren, E. (1990). Nonlinear integer and discrete programming in mechanical design optimization. Transactions of the ASME, Journal of Mechanical Design 112(2), 223–229.
[156] Sarimveis, H., Bafas, G. (2003). Fuzzy model predictive control of a non-linear process using genetic algorithms. Fuzzy Sets and Systems 139, 59-80.
[157] Schaffer, J., Caruana, R., Eshelman, L., Das, R. (1989). A study of control parameters affecting online performance of genetic algorithms for function optimisation. In: Schaffer (ed). Proceedings of the 3rd International Conference on Genetic Algorithms, 51-60.
[158] Schwefel, H. (1965). Kybernetische Evolution als Strategie der experimentellen Forschung in der Stromungstechnik Diplomarbeit, Technical University of Berlin.
[159] Schwefel H., (1977). Numerische Optimierung von Computer-Modellen mittels der Evolutionsstrategie (Basel: Birkhauser).
[160] Shi, Y., Eberhart, R. (1998). A Modified Particle Swarm Optimizer. IEEE International Conference on Evolutionary Computation, 69-73.
[161] Shi, Y., Eberhart, R. (1999). Empirical study of particle swarm optimization. In Proceedings of the 1999 Congress .on Evolutionary Computation, 1945-1950.
[162] Shi, Y., Eberhart, R. (2001). Fuzzy adaptive particle swarm optimization. Proceedings of IEEE International Conference on Evolutionary Computation, 101-106.
[163] Shin, S., Park, S. (1998). GA-based predictive control for nonlinear processes. IEEE Electronics Letters 34(20), 1980–1981.
[164] Sirikum, J., Techanitisawad, A. (2006). Power generation expansion planning with emission control: a nonlinear model and a GA-based heuristic approach. International Journal of Energy Research 30, 81-99.

[165] Skrlec, D., Filipec, M., Krajcar, S. (1997). A heuristic modification of genetic algorithm used for solving the single depot capacited vehicle routing problem. Proceedings of Intelligent Information Systems 1997, 184-188.
[166] Slupphaug, O., Foss, B. (1997). Model predictive control for a class of hybrid systems. Proceeding of European Control Conference, Brussels, Belgium.
[167] Slupphaug, O., Vada, J., Foss, B. (1997). MPC in systems with continuous and discrete control inputs. Proceedings of American Control Conference, Alburquerque, NM, USA.
[168] Smit, S., Eiben, A. (2009). Using Entropy for Parameter Analysis of Evolutionary Algorithms. Bartz-Beielstein *et al*. (eds.), Empirical Methods for the Analysis of Optimization Algorithms, Natural Computing Series, Springer, 2009, to appear.
[169] Solis, J., Sáez, D., Estévez, P. (2006). Particle swarm optimization-based fuzzy predictive control strategy. In IEEE world congress on computational intelligence, 8525–8530.
[170] Song, Y., Chen,Z., Yuan, Z. (2007). New chaotic PSO-based neural network predictive control for nonlinear process. IEEE Transactions on Neural Networks 18 (2), 595-601.
[171] Song, Z., Kusiak, A. (2007). Constraint-Based Control of Boiler Efficiency: A Data-Mining Approach. IEEE Transaction on Industrial Informatics, 3(1), 73-83.
[172] Sontag, E.D. (1981). Nonlinear regulation: the piecewise linear approach. IEEE Trans. Autom. Control 26(2), 346–357.
[173] Snyman,J. (2005). Practical Mathematical Optimization: An Introduction to Basic Optimization Theory and Classical and New Gradient-Based Algorithms. Springer Publishing.
[174] Storn, R., Price, K. (1995). Differential evolution - a simple and efficient adaptive scheme for global optimization over continuous spaces. Technical Report TR-95-012, ICSI.
[175] Storn, R., Price, K. (1997). Differential evolution - a simple evolution strategy for fast optimization. Dr. Dobb's Journal 22(4),18-24.
[176] Su, C., Lee, C. (2003). Network Reconfiguration of Distribution Systems Using Improved Mixed-Integer Hybrid Differential Evolution. IEEE Transactions on power delivery 18 (3), 1022-1027.
[177] Sutton, A., Lunacek, M., Whitley, L. (2007). Differential Evolution and Non-separability: Using selective pressure to focus search. In Proceedings of GECCO'07 (ACM Press), 1428 - 1435.
[178] Swihart, M., Papastavrou, J. (1999). A Stochastic and Dynamic Model for the Single-Vehicle Pick-Up and Delivery Problem. European Journal of Operational Research 114, 447-464.
[179] Syswerda, G. (1989). Uniform crossover in genetic algorithms. Proc. 3rd Int. Conf. on Genetic Algorithms, Schaffer (ed), 2–9.
[180] Takagi, T., Sugeno, M. (1985). Fuzzy identification of systems and its applications to modeling and control. IEEE Transactions on Systems, Man and Cybernetics 15, 116-132.
[181] Tarantilis, C. (2005). Solving the vehicle routing problem with adaptive memory programming methodology. Computers & Operations Research 32, 2309-2327.
[182] Thanga Raj, C., Srivastava, S., Agarwal, P. (2008). Differential Evolution based Optimal Control of Induction Motor Serving to Textile Industry**.** IAENG International Journal of Computer Science 35(2), 201-208.
[183] Thomas, B., White III, C. (2004). Anticipatory Route Selection. Transportation Science 38(4), 473-487.
[184] Toth, P., Vigo, D. (2003). The granular Tabu search and its application to the vehicle-routing problem. INFORMS Journal on Computing 15(4), 333-346**.**
[185] Trelea, I. (2003). The particle swarm optimization algorithm: convergence analysis and parameter selection. Information Processing Letters 85(6), 317–325.
[186] Ursem, R., (2003). Models for Evolutionary Algorithms and Their Applications in System Identification and Control Optimization. Ph.D. thesis, EVALife, Dept. of Computer Science, University of Aarhus, Denmark
[187] Ursem, R., Vadstrup, K. (2003). Parameter identification of induction motors using differential evolution. In: Proceedings of CEC 2003, 2, 790- 796.
[188] Uthaichana, K., Bengea, S., DeCarlo, R., Pekarek, S., Zefran, M. (2008). Hybrid model predictive control tracking of a sawtooth driving profile for an HEV**.** American Control Conference, 967 – 974.
[189] van den Bergh, F. (2001). An analysis of particle swarm optimizers. PhD Thesis, University of Pretoria.
[190] van der Schaft, A., Schumacher, J. (1998). Complementarity modelling of hybrid systems. IEEE Trans. Autom. Control 43, 483–490.
[191] Villa, J., Duque, M., Gauthier, A., Rakoto-Ravalontsalama, N. (2004)., A new algorithm for translating MLD systems onto PWA systems: In Proceedings of the 2004 American Control Conference (ACC) 1208–1213.
[192] Von Stryk, O. (1993). Numerical solution of optimal control problems by direct collocation. In: Internat. Ser. Numer. Math, 111, 129-143.
[193] Vose, M. (1999). The Simple Genetic Algorithm. MIT Press, Cambridge, MA, 1999

[194] Wang, X., Xiao, J. (2005). PSO-based model predictive control for nonlinear processes. Lecture Notes in Computer Science 3611, 196–203.
[195] Wei, S., Utaichana, K., Zefran, M., DeCarlo, R. (2007). Applications of numerical optimal control to nonlinear hybrid systems. Nonlinear Analysis: Hybrid Systems 1, 264-279.
[196] Wei, S., Zefran, M., Uthaichana, K., DeCarlo, R. (2007). Hybrid Model Predictive Control for Stabilization of Wheeled Mobile Robots Subject to Wheel Slippage**.** IEEE International Conference on Robotics and Automation, 2373 – 2378.
[197] Whitley, D. (1989). The GENITOR algorithm and selection pressure: why rank-based allocation of reproductive trials is best. Proc. 3rd Int. Conf. on Genetic Algorithms (Fairfax, VA, 1989) Schaffer, J. (ed), San Mateo, CA: Morgan Kaufmann, 116–21.
[198] Woolley, I., Kambhampati, C., Sandoz, D., Warwick, K. (1998). Intelligent control toolkit for an advanced control system. In UKACC IEEE international conference on control, 445–450.
[199] Wu, S., Chow, P. (1995). Genetic algorithms for nonlinear mixed discrete-integer optimization problems via meta-genetic parameter optimization. Engineering Optimization 24(2), 137–159.
[200] Xiangm Z., Chum C., Chenm H. (2008). The study of a dynamic dial a ride problem under time-dependent and stochastic environments. European Journal of Operational Research 185, 534-551.
[201] Yasuda, K., Ide, A., Iwasaki, N. (2003). Adaptive particle swarm optimization. IEEE International Conference on Systems, Man and Cybernetics, 2, 1554–1559.
[202] Yiqing, L., Xigang, Y., Yongjian, L. (2007). An improved PSO algorithm for solving non-convex NLP/MINLP problems with equality constraints. Computers and Chemical Engineering 31, 153-162.
[203] Yoshida, H., Kawata, K., Fukuyama, Y., Takayama, S., Nakanishi, Y. (2000). A Particle Swarm Optimization for Reactive Power and Voltage Control Considering Voltage Security Assessment. IEEE Transactions on Power Systems, 15(4), 1232-1239.
[204] Yucheng, Z., Yannan, Z., Jiaxin, W. (2002). Evolutionary optimal control of free floating spacecraft. In Proceedings of the 4th World Congress on Intelligent Control and Automation, Shanghai, P.R. China.
[205] Yuzgec, U., Becerikli, Y., Turker, M. (2006). Nonlinear predictive control of a drying process using genetic algorithms. Isa Transactions 45 (4), 589-602.
[206] Zafiriou, E. (1990). Robust Model Predictive Control of Processes with Hard Constraints. Computers and Chemical Engineering 14(4/5), 359-371.
[207] Zaharie, D. (2003). Control of population diversity and adaptation in differential evolution algorithms. In: Matousek, R., and Osmera, P. (eds.), Proceedings of 9th International Conference on Soft Computing, Mendel 2003, 41-46.
[208] Zeilinger, M., Jones, C., Morari (2008). Real-time suboptimal Model Predictive Control using a combination of Explicit MPC and Online Optimization. Proceedings of the Conference on Decision and Control, CDC, Cancun, Mexico.
[209] Zhang, C., Wang, H. (1993). Mixed discrete nonlinear optimization with simulated annealing. Engineering Optimization 21, 277-291.
[210] Zhu, Q., Qian, L., Li, Y., Zhu, S. (2006). An Improved Particle Swarm Optimization Algorithm for Vehicle Routing Problem with Time Windows. IEEE Congress on Evolutionary Computation, Vancouver, BC, Canada, 1386-1390.

10 Apéndice A: Representación de expresiones lógicas en sistemas MLD

En los sistemas MLD las expresiones lógicas de la parte discreta de un sistema híbrido se pueden expresar como restricciones.
Es decir, dadas dos declaraciones L_1 y L_2 que pueden ser verdaderas o falsas, por medio de variables lógicas δ_1 y δ_2 que pueden tomar valores 0 o 1 y que a su vez puede tener las siguientes equivalencias:

$$\begin{aligned} &L_1 \wedge L_2 \text{ es equivalente a } \delta_1 + \delta_2 \geq 1 \\ &L_1 \vee L_2 \text{ es equivalente a } \delta_1 = 1, \delta_2 = 1 \\ &\sim L_1 \quad \text{es equivalente a } \delta_1 = 0 \\ &L_1 \Rightarrow L_2 \text{ es equivalente a } \delta_1 - \delta_2 \leq 0 \\ &L_1 \Leftrightarrow L_2 \text{ es equivalente a } \delta_1 - \delta_2 = 0 \\ &L_1 \oplus L_2 \quad \text{es equivalente a } \delta_1 + \delta_2 = 1 \end{aligned} \tag{9.1}$$

Otras declaraciones posibles son $L = [f(x) \leq 0]$ con $f : \Re n \rightarrow \Re$ es lineal y se asume que $x \in \aleph$, donde $\aleph$es un conjunto acotado y definiendo:

$$M = \max_{x \in \aleph} f(x) \quad \text{y} \quad m = \min_{x \in \aleph} f(x) \tag{9.2}$$

Entonces es fácil verificar que

$$\begin{aligned} [f(x) \leq 0] \wedge [\delta = 1] &\text{ es verdadero si sólo si } f(x) - \delta \leq -1 + m(1 - \delta) \\ [f(x) \leq 0] \vee [\delta = 1] &\text{ es verdadero si sólo si } f(x) \leq \delta M \\ \sim [f(x) \leq 0] &\text{ es verdadero si sólo si } f(x) \geq \varepsilon \end{aligned} \tag{9.3}$$

donde ε es una pequeña tolerancia (típicamente la precisión de la máquina), más allá de la cual la restricción es considerada violada. Con lo cual también se puede demostrar que:

$$\begin{aligned} &[f(x) \leq 0] \Rightarrow [\delta = 1] \text{ es verdadero ssi } f(x) \geq \varepsilon + (m - \varepsilon)\delta \\ &[f(x) \leq 0] \vee [\delta = 1] \text{ es verdadero ssi } f(x) \leq \delta M \\ &\sim [f(x) \leq 0] \text{ es verdadero ssi } f(x) \geq \varepsilon \end{aligned} \tag{9.4}$$

Además la multiplicación de dos variables $\delta_1 \delta_2$ expresada por una tercera $\delta_3 \equiv \delta_1 \delta_2$ se tiene que es equivalente a $[\delta_3 = 1] \Leftrightarrow [\delta_1 = 1] \wedge [\delta_2 = 1]$, luego:

$$\delta_3 = \delta_1\delta_2 \text{ es equivalente a } \begin{cases} -\delta_1 + \delta_3 \le 0 \\ -\delta_2 + \delta_3 \le 0 \\ \delta_1 + \delta_2 - \delta_3 \le 1 \end{cases} \tag{9.5}$$

Por último la expresión $y = \delta f(x)$ es equivalente a:

$$\begin{aligned} & y \le M\delta \\ & y \ge m\delta \\ & y \le f(x) - m(1-\delta) \\ & y \ge f(x) - M(1-\delta) \end{aligned} \tag{9.6}$$

11 Apéndice B: *Branch and Bound* para programación entera mixta no lineal

Se ha visto en la Sección 3.3.5 que el problema de control predictivo híbrido no lineal genérico se puede formular con un problema de programación entera mixta no lineal (*Mixed integer non-linear programing*, MINLP). Se presenta entonces a continuación un método basado en *branch and bound* (ramificar y acotar) para resolver problemas MINLP basado en la formulación genérica.

Formulación

Un problema de programación entera mixta no lineal se puede formular del siguiente modo (Grossman, 2002):

(P1)

$$\begin{aligned} &\min Z = f(x,y) \\ s.a.\quad &g_j(x,y) \le 0 \quad j \in K \\ &x \in X,\ y \in Y \end{aligned} \tag{9.7}$$

donde $f(\cdot), g(\cdot)$ son funciones diferenciables convexas, J es el conjunto de índices de las desigualdades , x e y son las variables continuas y discretas, respectivamente. El conjunto X generalmente se asume convexo y compacto $X = \left\{ x \middle| x \in \mathbb{R}, Dx \le d, x^L \le x \le x^U \right\}$; el conjunto discreto Y corresponde a un conjunto poliédrico de puntos enteros, $Y = \left\{ y \middle| y \in \mathbb{Z}^m, Ay \le a \right\}$, que en la mayor parte de las aplicaciones se restringe a valores 0-1, $y \in \{0,1\}^m$. En la mayor parte de las aplicaciones de interés, las funciones objetivo y de restricciones $f(\cdot), g(\cdot)$ son lineales en y (cargos de costos fijos y restricciones de lógicas mezcladas): $f(x,y) = c^T y + r(x)$, $g(x,y) = By + h(x)$.

Subproblemas

El método de solución basado en *branch and bound* para problemas MINLP se basa en la resolución sistemática del siguiente sub-problema:

1. Relajación NLP:

(NLP1)

$$\begin{aligned} &\min Z_{LB}^k = f(x,y) \\ s.a.\ &g_j(x,y) \le 0 \quad j \in K \\ &x \in X,\ y \in Y_R \\ &y_i \le \alpha_i^k,\ i \in I_{FL}^k \\ &y_i \ge \beta_i^k,\ i \in I_{FU}^k \end{aligned} \tag{9.8}$$

donde Y_R es la relajación continua del conjunto Y, y I_{FL}^k, I_{FU}^k son subconjuntos de índices de las variables enteras $y_i, i \in I$, las cuales están restringidas a cotas inferiores y superiores, α_i^k, β_i^k, en el k-ésimo paso de una enumeración de *branch and bound*. Se debe notar que $\alpha_i^k = \lfloor y_i^l \rfloor, \beta_i^k = \lceil y_i^l \rceil, l < k, m < k$, donde y_i^l, y_i^m son valores no enteros en pasos anteriores.

Se debe notar también que si $I_{FL}^k = I_{FU}^k = \varnothing \ (k = 0)$, (NLP1) corresponde a la relajación continua de (P1). Salvo en unos pocos casos, la solución a este problema entrega un vector no entero para las variables discreta. El problema (NLP1) además corresponde al k-ésimo paso en una búsqueda de *branch and bound*. La función objetivo Z_{LB}^0 entrega una cota inferior absoluta a (P1); para $m \geq k$, la cota sólo es válida para $I_{FL}^k \subset I_{FL}^m, I_{FU}^k \subset I_{FL}^m$.

Branch and Bound

La versión de este método para problemas no lineales (Gupta y Ravidran, 1985; Nabar y Schrge, 1991; Borchers y Mithcell, 1994; Stubbs y Mehrotra, 1999; Leyffer,2001) funciona básicamente del mismo modo que para problema lineales (MILPs) (Dakin, 1965).

Un algoritmo de branch and bound para resolver problemas MINLP requiere las siguientes estructuras de datos. El algoritmo mantiene un lista L de sub-problemas no resueltos. El algoritmo también mantiene un registro de la mejor solución que ha sido encontrada. Esta solución (x^*, y^*), es llamada la solución momentánea. Ésta corresponde a una cota superior Z_U^k para el valor óptimo de la solución del problema MINLP. El algoritmo se resume en el siguiente pseudo-código.

1. Inicializar: Crea una lista *L* donde el problema original es el sub-problema inicial. Si se conoce una buena solución (x^*, y^*), luego su función objetivo es la cota superior Z_U^0. Si no hay ninguna solución conocida, entonces $Z_U^0 = +\infty$.
2. Seleccionar: Escoger un sub-problema no resuelto *S* de la lista *L*. Si L está vacío, Parar: Si hay una solución momentánea, entonces esa solución es la óptima; si no hay solución momentánea, entonces el problema original es infactible.
3. Resolver: Relajar las restricciones de integralidad en *S* y se resuelve la relajación no lineal resultante (NLP1). Se obtienen soluciones $(\hat{x}, \hat{y})$, cuya función objetivo es una cota inferior Z_{LB}^k.
4. Eliminar: si el problema relajado es infactible o si $Z_{LB}^k \geq Z_{UB}^k$, entonces *S* claramente no entregara una mejor solución que la solución momentánea. Luego remover *S* de *L* y volver a 2.
5. Solución entera: Si $\hat{y}$ es entero, entonces se ha obtenido una nueva solución momentánea. Actualizar (x^*, y^*) y Z_{UB}^k con, $(\hat{x}, \hat{y})$ y su función objetivo, respectivamente.
6. Ramificar: al menos una de las variables enteras toma un valor fraccional en la solución del sub-problema actual. Crear dos nuevos sub-problemas, S_1 al añadir la restricción $y_k \leq \lfloor \hat{y}_k \rfloor$, y S_2 al añadir la restricción $y_k \geq \lfloor \hat{y}_k \rfloor$. Remover *S* de *L*, añadir S_1 y S_2 a *L*, y volver a 2.

Este método es en general atractivo sólo si los subproblemas NLP requieren relativamente escaso esfuerzo computacional, o cuando muy pocos de ellos deben ser resueltos. Esto puede suceder por una baja dimensionalidad de las variables discretas o por un pequeño "integrality gap" de la relajación NLP continua.

Los problemas NLP que resultan de la relajación (o fijación) de variables discretas en *branch and bound*, generalmente (y en particular en la aplicación en esta tesis) son resueltos mediante programación secuencial cuadrática (sequential quadratic programming, SQP), que es uno de los métodos más robustos para resolver problemas de programación no lineal. El método se basa en la resolución de una serie de sub-problemas QP, que corresponden a una simplifación cuadrática de la función objetivo y a una simplificación lineal de las restricciones. Si el problema no posee restricciones, el método es equivalente a aplicar el método de Newton a las condiciones de optimalidad de primer orden (condiciones de Karush-Kuhn-Tucker) al problema particular. Para una revisión más en detalle del método de SQP, se sugiere revisar Bertsekas (1999).

12 Apéndice C: Restricciones típicas en control predictivo

En esta sección se presentan las restricciones más comunes (aparte de las ecuaciones del modelo) que se utilizan en la formulación del control predictivo.

Restricciones operativas sobre salida y estado del proceso

Es común que debido a consideraciones de seguridad y/o calidad de los procesos se establezcan cotas mínimas o máximas aceptables para la salida y/o estados del proceso. Este tipo de restricciones se formula como sigue:

$$\left.\begin{array}{l} y_{\min} \le y(t+k) \le y_{\max} \\ x_{\min} \le x(t+k) \le x_{\max} \end{array}\right\} k = 1,...,N_y \tag{9.9}$$

Otro tipo de restricciones sobre la salida se establecen cuando se desea evitar los sobre niveles, por ejemplo, en un manipulador robótico un sobrenivel provocaría un choque con el objeto que se quiere tomar. Esta restricción se formula como:

$$y(t+k) \le y_{ref}, \;\; k = 1,...,N_y \tag{9.10}$$

Otros sistemas pueden exhibir oscilaciones indeseables, y en su lugar se puede exigir que sigan un comportamiento monótono. Por ejemplo, se plantea que la salida sea creciente si se encuentra bajo la referencia, o decreciente si está por sobre la referencia:

$$\left.\begin{array}{l} y(t+k) \le y(t+k-1), \; si \;\; y(t) \ge y_{ref} \\ y(t+k) \ge y(t+k-1), \; si \;\; y(t) \le y_{ref} \end{array}\right\} k = 1,...,N_y \tag{9.11}$$

El último tipo de restricciones que suelen agregarse sobre las salidas se utiliza para evitar el comportamiento de fase no mínima de algunos sistemas. Como este comportamiento consiste en que la salida tiende a moverse en dirección contraria antes de moverse hacia la dirección final cuando se aplica un escalón, la restricción consiste en evitar que se muevan en la dirección opuesta:

$$\left.\begin{array}{l} y(t+k) \le y(t), \; si \;\; y(t) \ge y_{ref} \\ y(t+k) \ge y(t), \; si \;\; y(t) \le y_{ref} \end{array}\right\} k = 1,...,N_y \tag{9.12}$$

Restricciones operativas sobre las acciones de control

Este tipo de restricciones en general se deben a las restricciones físicas de los actuadores. La primera restricción de este tipo corresponde a los límites mínimos y máximos de las acciones que puede aplicar el actuador, y la segunda restricción de esta categoría corresponde a las variaciones

máximas y mínimas en las acciones de control que puede aplicar el actuador entre un instante discreto y el siguiente. Estas restricciones se formulan como sigue:

$$\left.\begin{array}{rcl} u_{\min} & \le & u(t+k-1) \le u_{\max} \\ \Delta u_{\min} & \le & \Delta u(t+k-1) \le \Delta u_{\max} \end{array}\right\} k = 1,...,N_u \tag{9.13}$$

donde $\Delta u(t+k-1) = u(t+k-1) - u(t+k-2)$.

Los actuadores además pueden presentar zonas muertas, tanto en la acción de control $(u_{\min}^{dead}, u_{\max}^{dead})$, como en la variación entre instantes de muestreo $(\Delta u_{\min}^{dead}, \Delta u_{\max}^{dead})$. Luego las restricciones son planteadas para evitar estas zonas:

$$\left.\begin{array}{rcl} u_{\min}^{dead} & \ge & u(t+k-1) \ge u_{\max}^{dead} \\ \Delta u_{\min}^{dead} & \ge & \Delta u(t+k-1) \ge \Delta u_{\max}^{dead} \end{array}\right\} k = 1,...,N_u \tag{9.14}$$

Restricciones de estado final para asegurar estabilidad

Existen ciertas formulaciones del control predictivo (Constrained receding-horizon predictive control, Clarke y Scattolini, 1991) en las cuales se establece que para asegurar estabilidad del sistema controlado, se debe asegurar que por algunos instantes futuros la salida sea exactamente igual a la referencia. Estos instantes por lo general son posteriores al horizonte de predicción (lo que de cierto modo redefine el concepto del horizonte de predicción).

$$y(t+N_y+i) = y_{ref},\ i = 1,...,m \tag{9.15}$$

13 Apéndice D: Gráficos capítulo 7

13.1 Gráficos de métodos para el reactor batch con entradas discretas

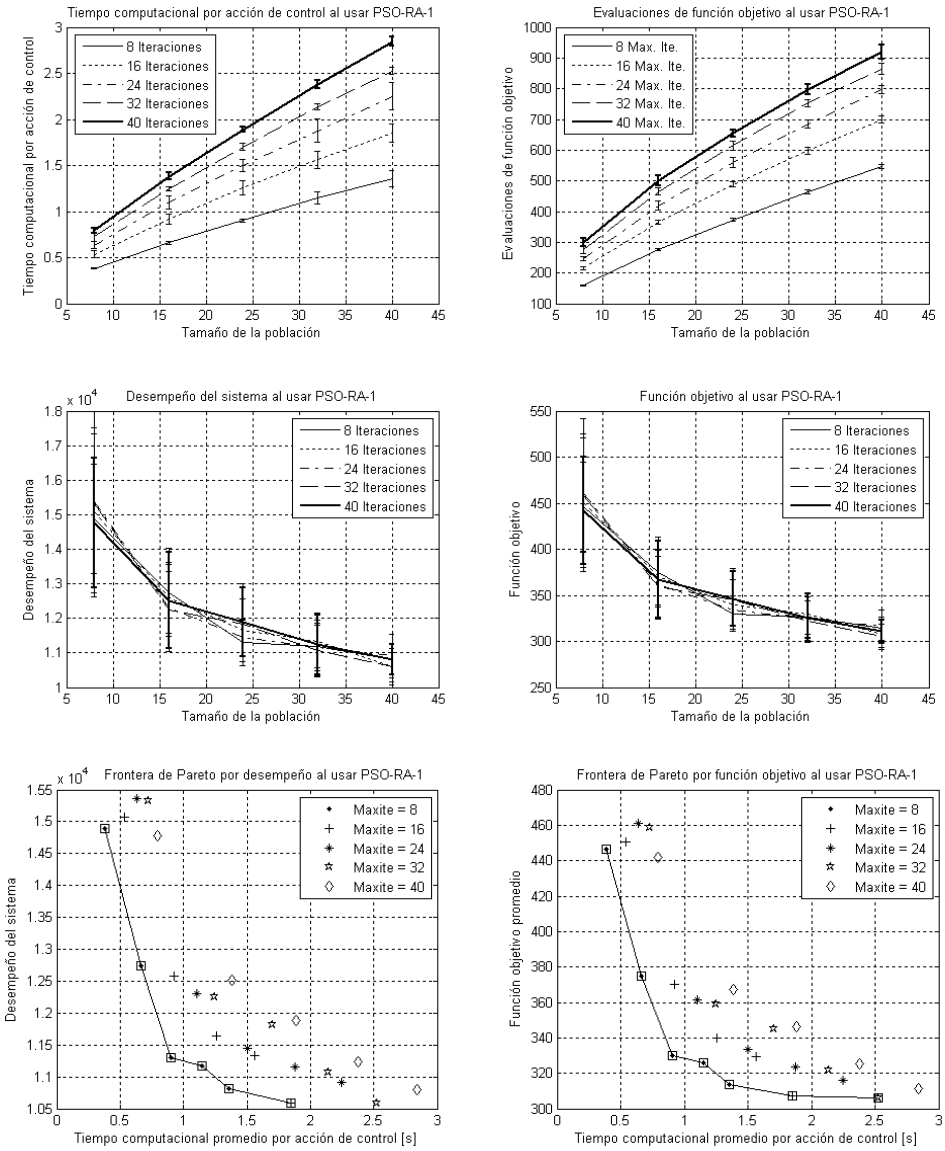


Figura 88: Gráficos de método PSO-RA-1

Tabla 27: Frontera de Pareto para el método PSO-RA-1

Tiempo promedio por acción de control [s]	Desempeño del sistema [ºC^2]	Número de iteraciones	Número de partículas
0,38279	14888	8	8
0,6617	12749	8	16
0,90425	11296	8	24
1,1471	11174	8	32
1,3574	10813	8	40
1,8474	10592	16	40

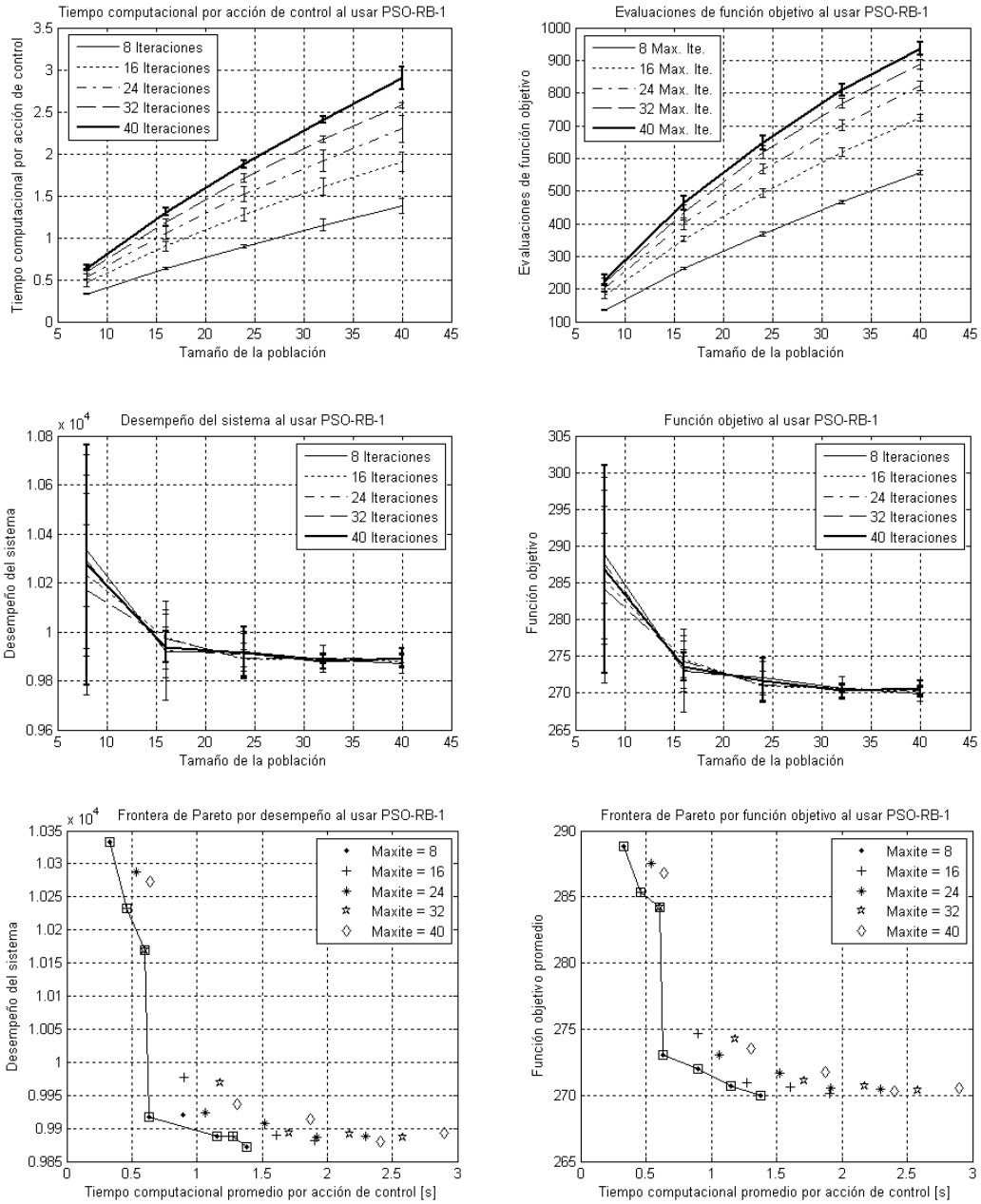


Figura 89: Gráficos de método PSO-RB-1

Tabla 28: Frontera de Pareto para el método PSO-RB-1

Tiempo promedio por acción de control [s]	Desempeño del sistema [ºC^2]	Número de iteraciones	Número de partículas
0,3298	10333	8	8
0,63266	9917,9	8	16
1,1524	9888,7	8	32
1,3806	9872,1	8	40
0,46182	10233	16	8
1,2731	9887,5	16	24
0,6019	10170	32	8

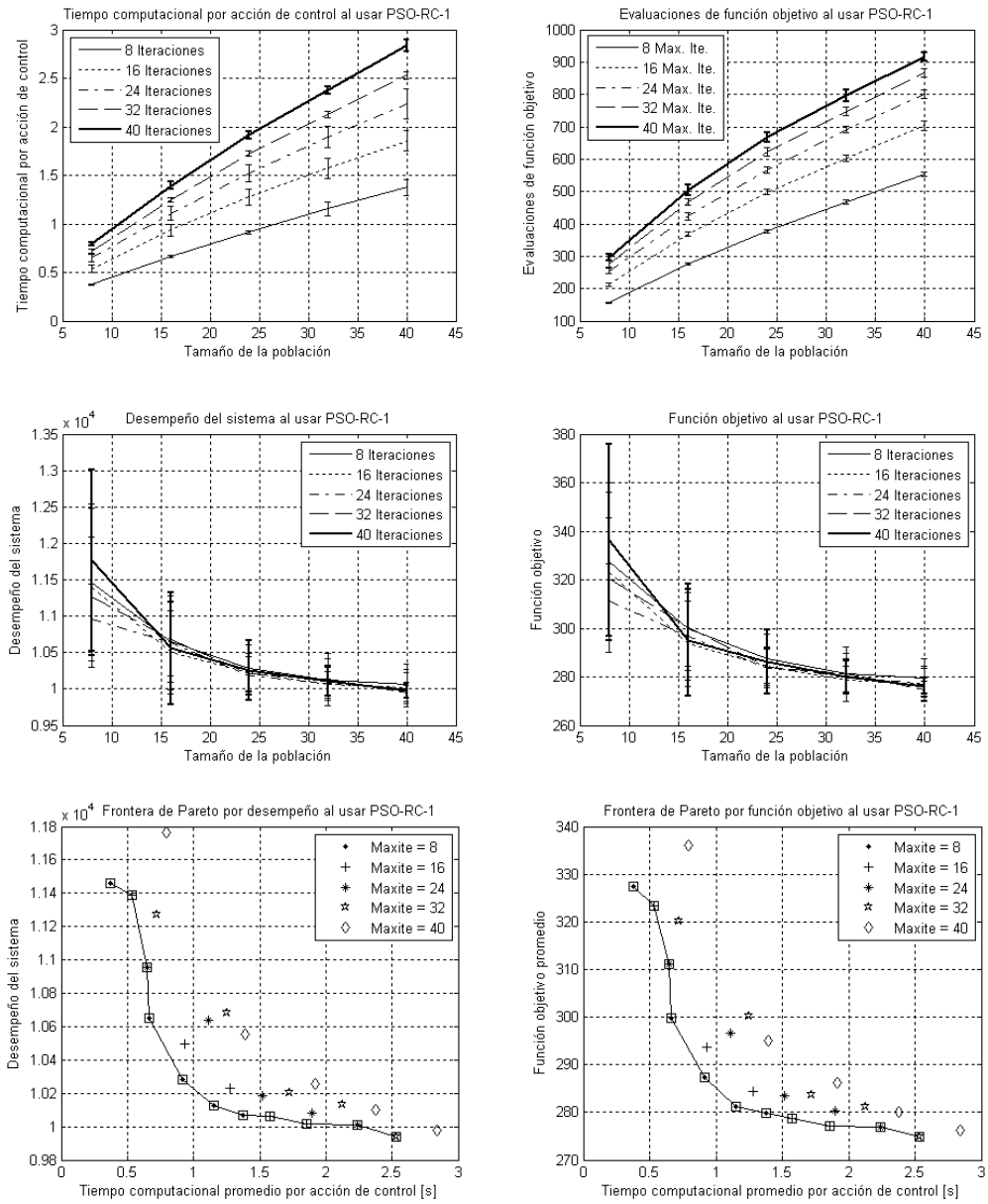


Figura 90: Gráficos de método PSO-RC-1

Tabla 29: Frontera de Pareto para el método PSO-RC-1

Tiempo promedio por acción de control [s]	Desempeño del sistema [ºC^2]	Número de iteraciones	Número de partículas
0,37423	11457	8	8
0,66435	10652	8	16
0,91502	10280	8	24
1,1544	10129	8	32
1,3756	10069	8	40
0,53389	11391	16	8
1,5738	10060	16	32
1,8556	10017	16	40
0,64874	10958	24	8
2,2377	10010	24	40
2,5296	9941,3	32	40

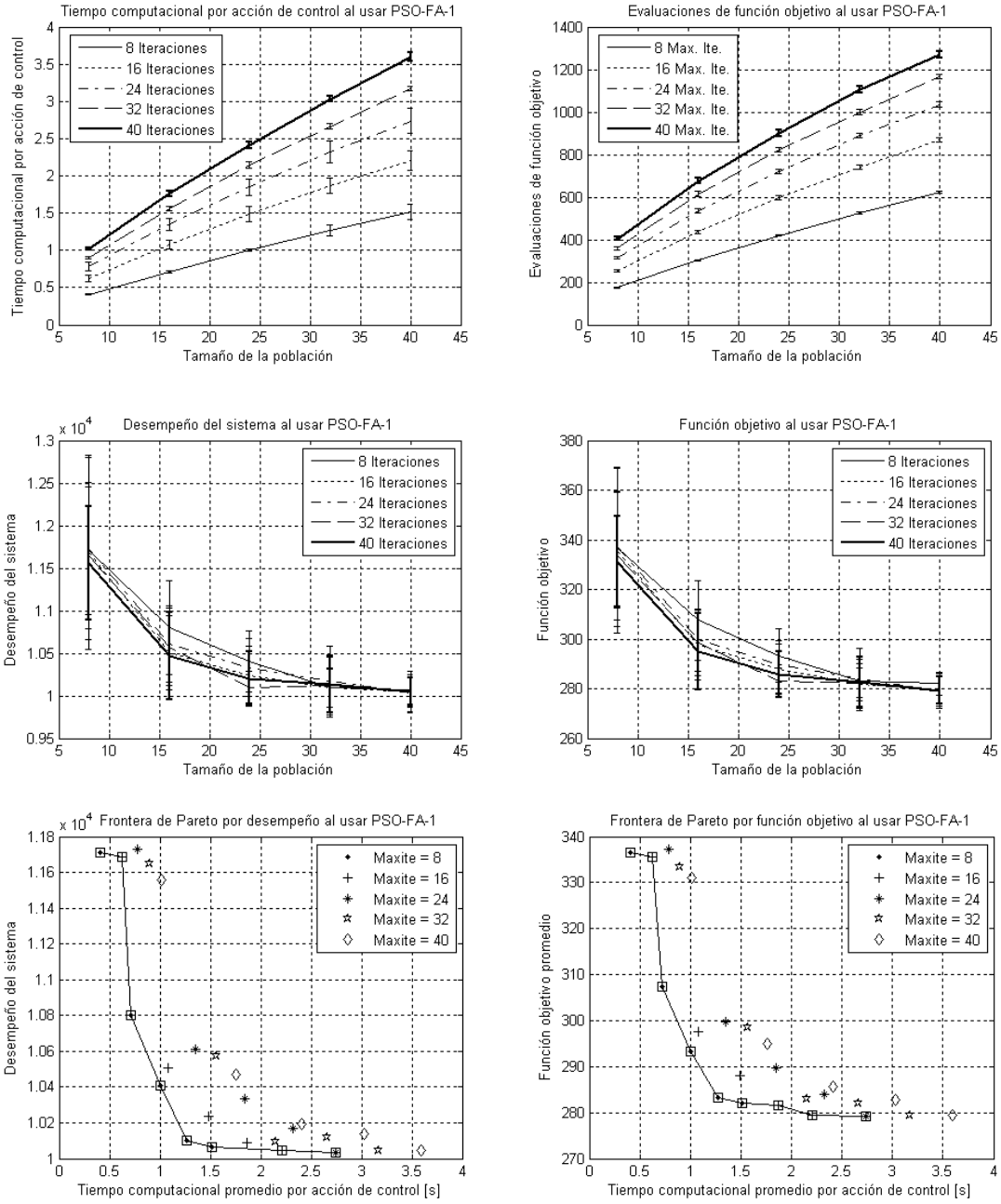


Figura 91: Gráficos de método PSO-FA-1

Tabla 30: Frontera de Pareto para el método PSO-FA-1

Tiempo promedio por acción de control [s]	Desempeño del sistema [ºC^2]	Número de iteraciones	Número de partículas
0,41025	11708	8	8
0,71719	10804	8	16
1,0033	10409	8	24
1,2699	10100	8	32
1,5167	10065	8	40
0,62187	11685	16	8
2,2102	10045	16	40
2,7393	10035	24	40

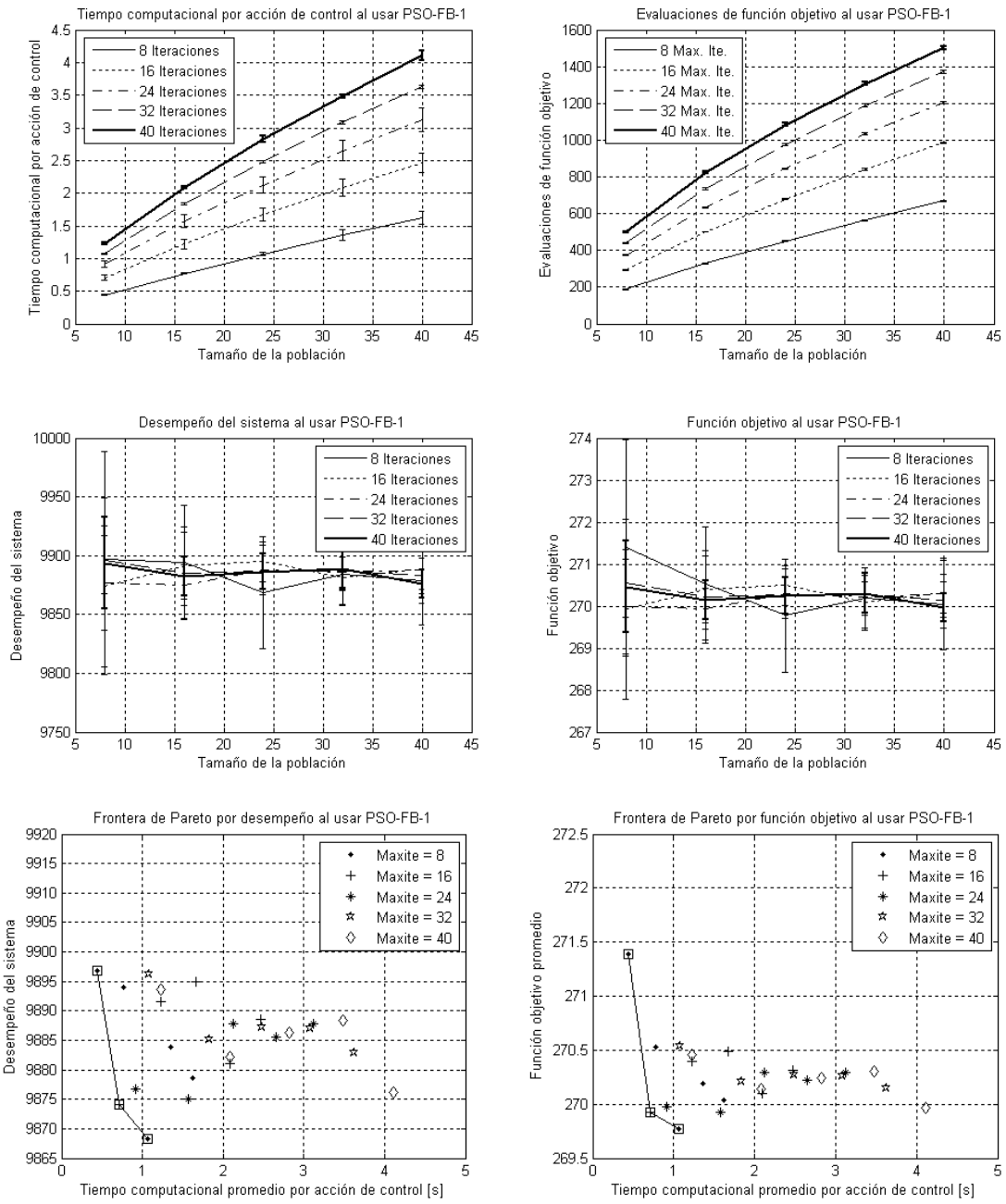


Figura 92: Gráficos de método PSO-FB-1

Tabla 31: Frontera de Pareto para el método PSO-FB-1

Tiempo promedio por acción de control [s]	Desempeño del sistema [ºC^2]	Número de iteraciones	Número de partículas
0,44039	9896,9	8	8
1,0684	9868,3	8	24
0,7072	9874	16	8

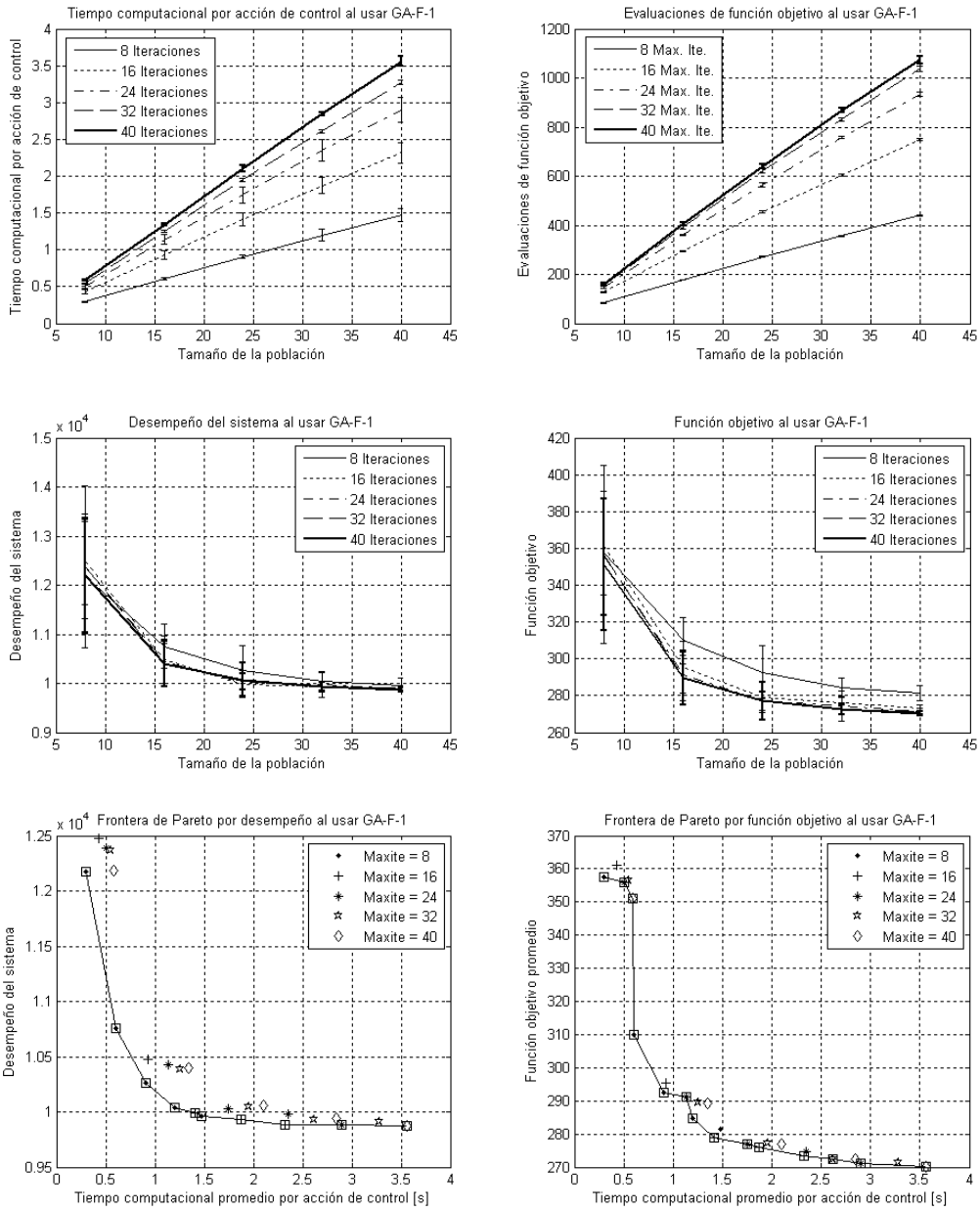


Figura 93: Gráficos de método GA-F-1

Tabla 32: Frontera de Pareto para el método GA-F-1

Tiempo promedio por acción de control [s]	Desempeño del sistema [ºC^2]	Número de iteraciones	Número de partículas
0,29747	12175	8	8
0,60564	10758	8	16
0,90561	10259	8	24
1,1985	10037	8	32
1,4746	9964,5	8	40
1,4127	9986,1	16	24
1,8746	9935,3	16	32
2,3204	9887,9	16	40
2,9005	9886,5	24	40
3,5576	9874,4	40	40

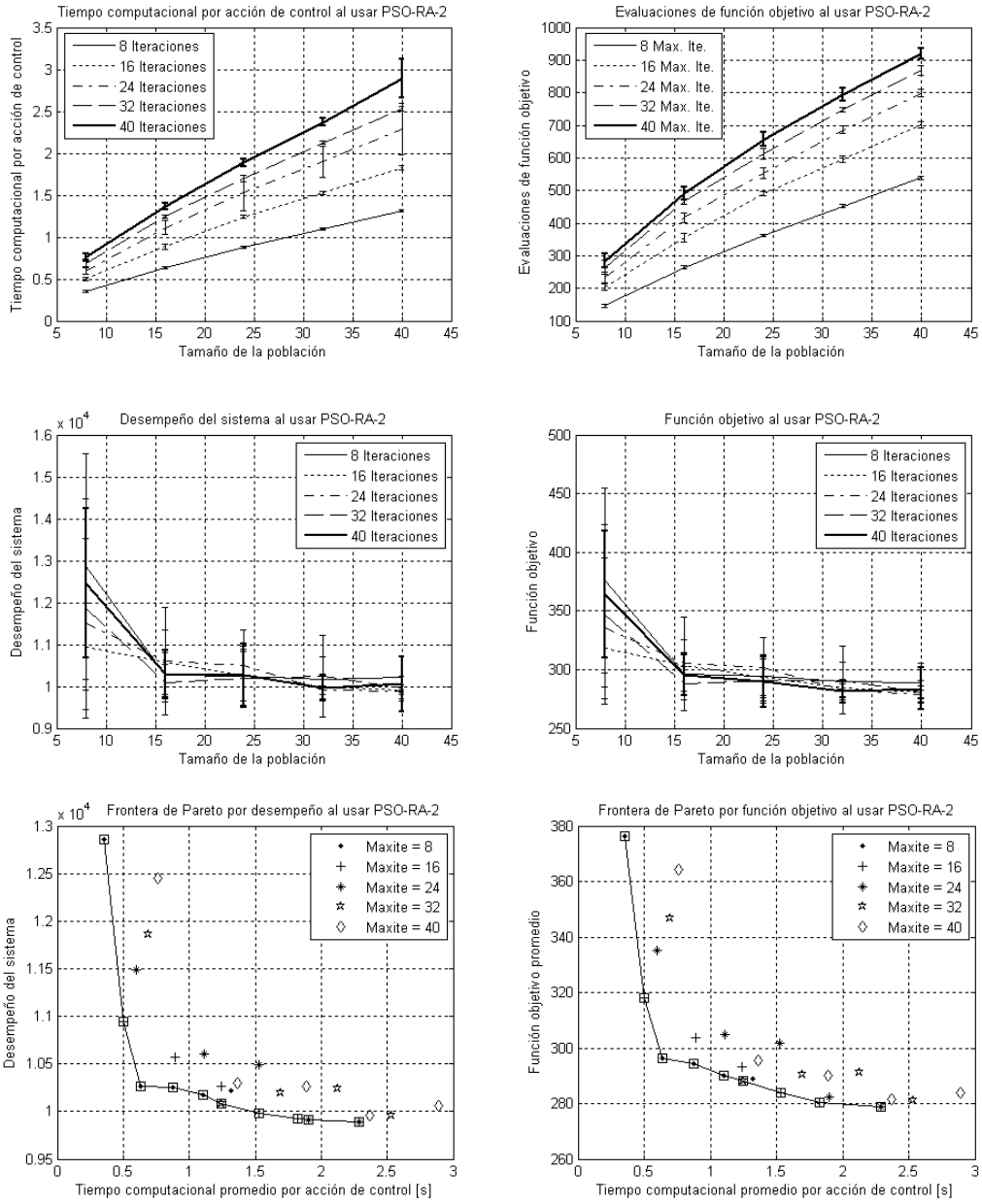


Figura 94: Gráficos de método PSO-RA-2

Tabla 33: Frontera de Pareto para el método PSO-RA-2

Tiempo promedio por acción de control [s]	Desempeño del sistema [ºC^2]	Número de iteraciones	Número de partículas
0,35186	12856	8	8
0,63518	10262	8	16
0,87713	10249	8	24
1,1008	10172	8	32
0,50264	10945	16	8
1,53	9977,9	16	32
1,8257	9920,1	16	40
1,902	9915,3	24	32
2,2848	9886,5	24	40
1,2461	10080	32	16

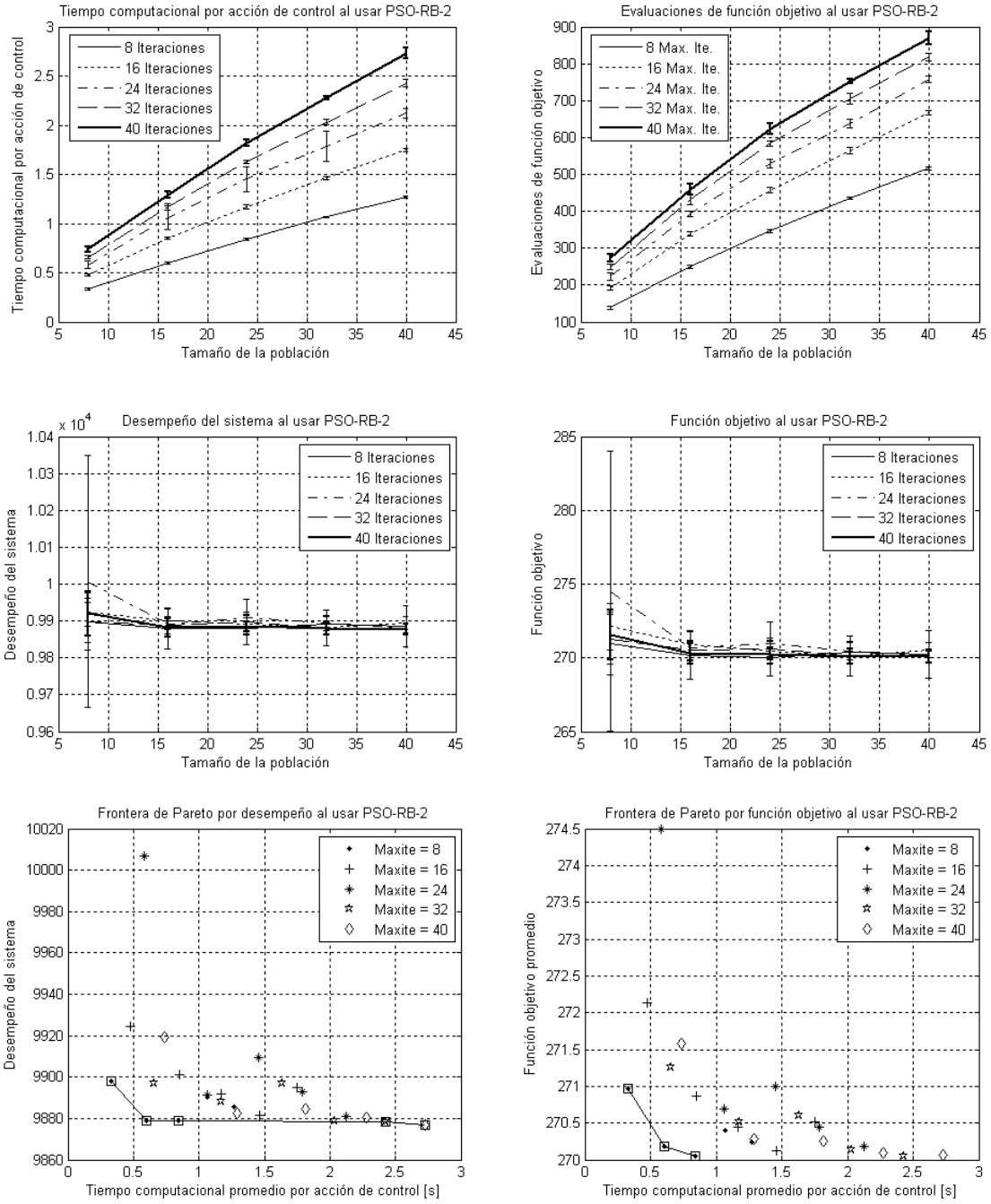


Figura 95: Gráficos de método PSO-RB-2

Tabla 34: Frontera de Pareto para el método PSO-RB-2

Tiempo promedio por acción de control [s]	Desempeño del sistema [ºC^2]	Número de iteraciones	Número de partículas
0,33301	9897,8	8	8
0,60311	9878,9	8	16
0,84177	9878,8	8	24
2,4284	9878,4	32	40
2,7283	9876,7	40	40

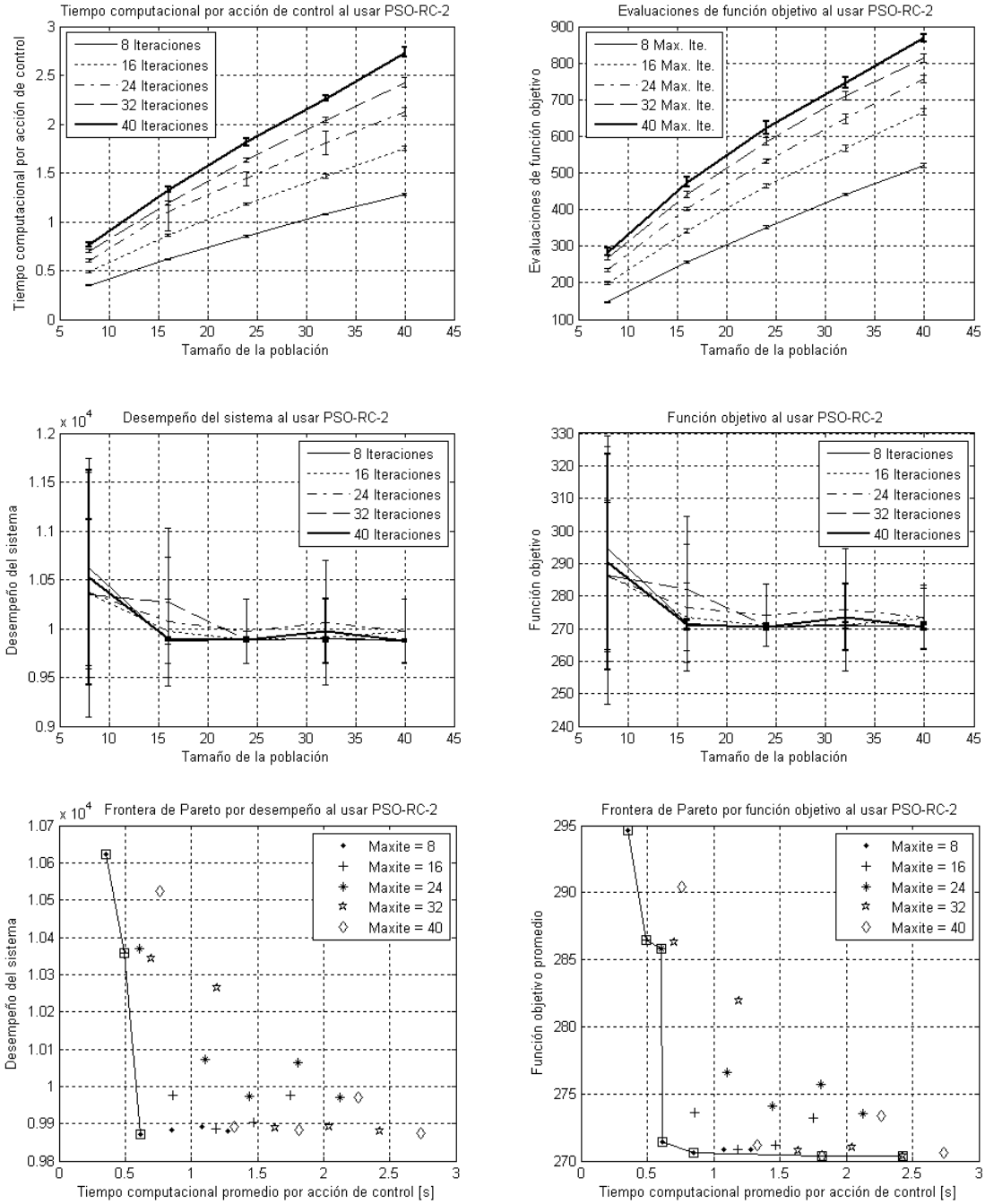


Figura 96: Gráficos de método PSO-RC-2

Tabla 35: Frontera de Pareto para el método PSO-RC-2

Tiempo promedio por acción de control [s]	Desempeño del sistema [ºC^2]	Número de iteraciones	Número de partículas
0,35092	10621	8	8
0,61655	9871,3	8	16
0,49218	10358	16	8

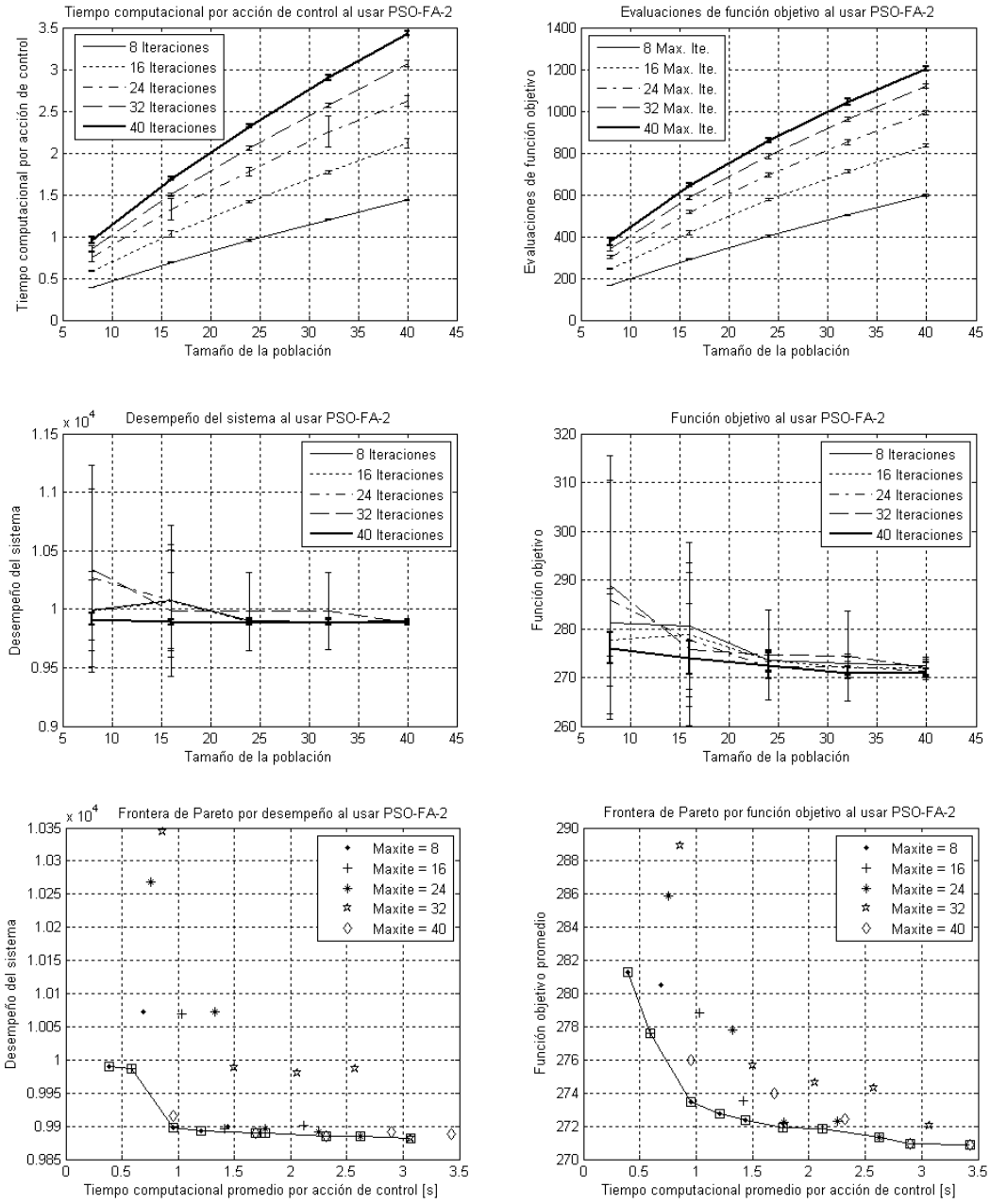


Figura 97: Gráficos de método PSO-FA-2

Tabla 36: Frontera de Pareto para el método PSO-FA-2

Tiempo promedio por acción de control [s]	Desempeño del sistema [ºC^2]	Número de iteraciones	Número de partículas
0,38911	9990,2	8	8
0,95572	9898,5	8	24
1,2048	9892,7	8	32
0,5881	9986,8	16	8
1,7725	9889,7	16	32
2,6215	9884,4	24	40
3,0675	9881,3	32	40
1,6906	9889,8	40	16
2,3183	9885,4	40	24

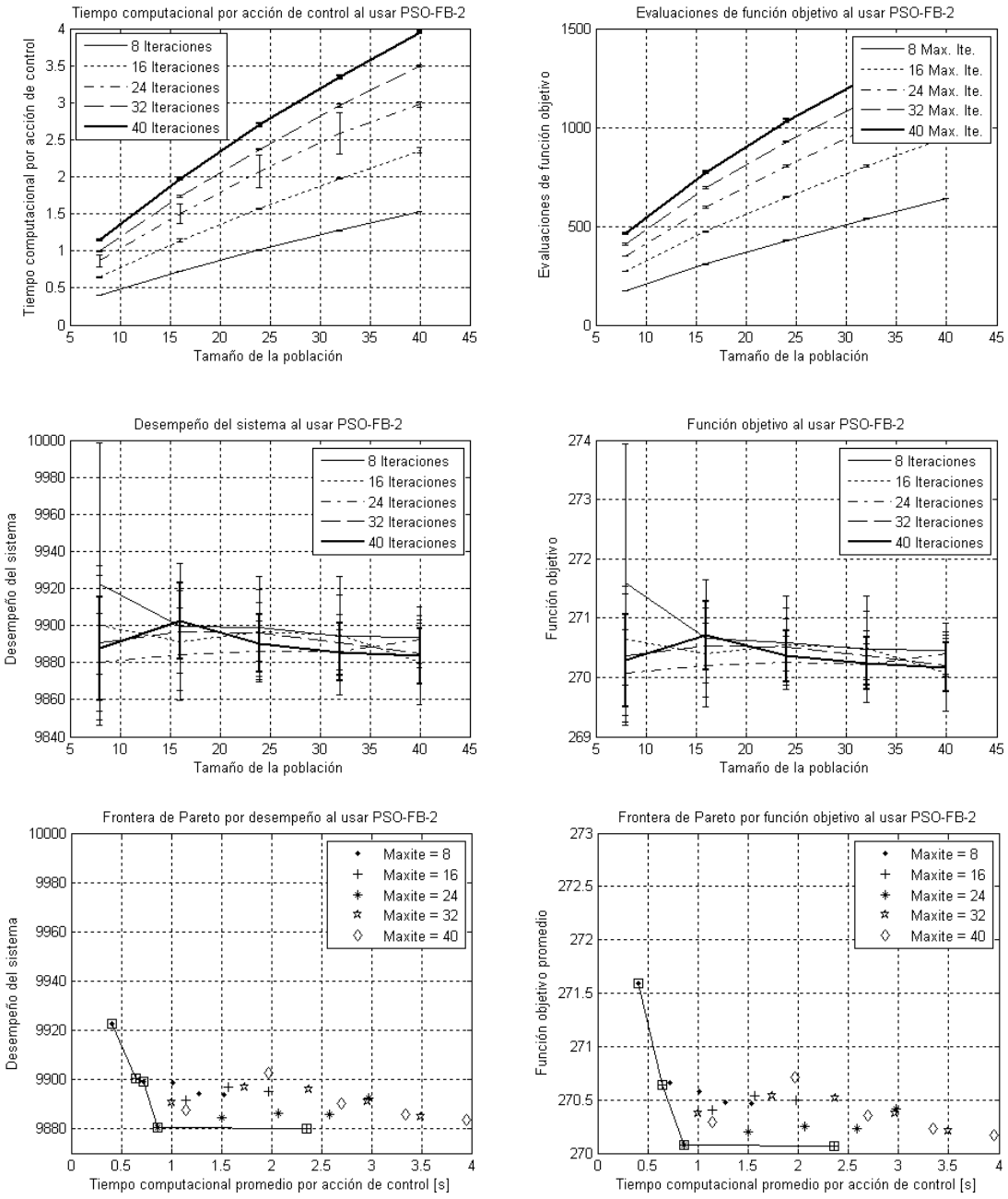


Figura 98: Gráficos de método PSO-FB-2

Tabla 37: Frontera de Pareto para el método PSO-FB-2

Tiempo promedio por acción de control [s]	Desempeño del sistema [ºC^2]	Número de iteraciones	Número de partículas
0,40581	9922,4	8	8
0,72463	9899,3	8	16
0,64595	9900,4	16	8
2,3582	9880,1	16	40
0,8649	9880,2	24	8

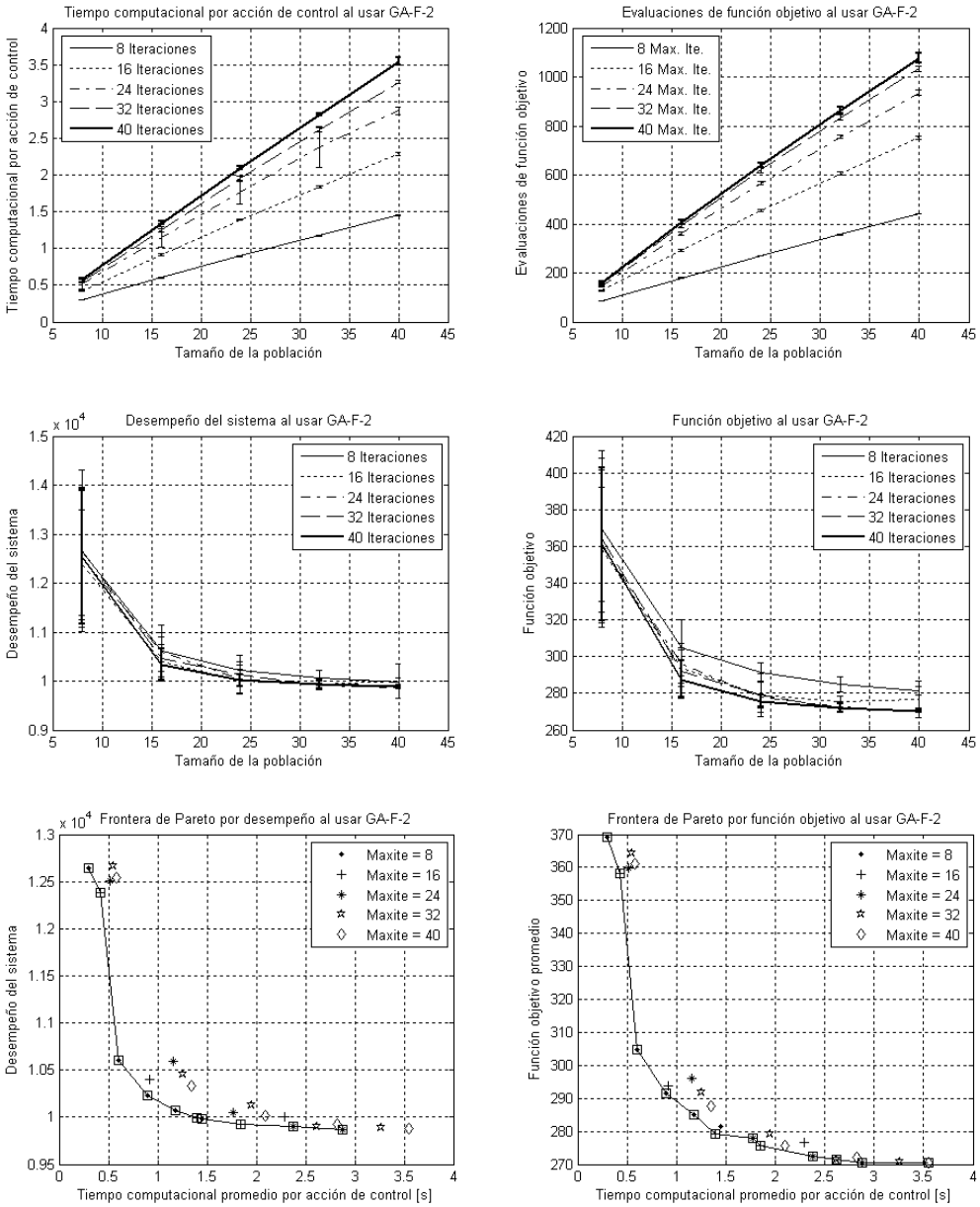


Figura 99: Gráficos de método GA-F-2

Tabla 38: Frontera de Pareto para el método GA-F-2

Tiempo promedio por acción de control [s]	Desempeño del sistema [ºC^2]	Número de iteraciones	Número de partículas
0,29795	12648	8	8
0,60187	10608	8	16
0,89619	10227	8	24
1,176	10071	8	32
1,4519	9975,9	8	40
0,42286	12377	16	8
1,3913	9995,2	16	24
1,8431	9925,7	16	32
2,3782	9901,4	24	32
2,8754	9864,8	24	40

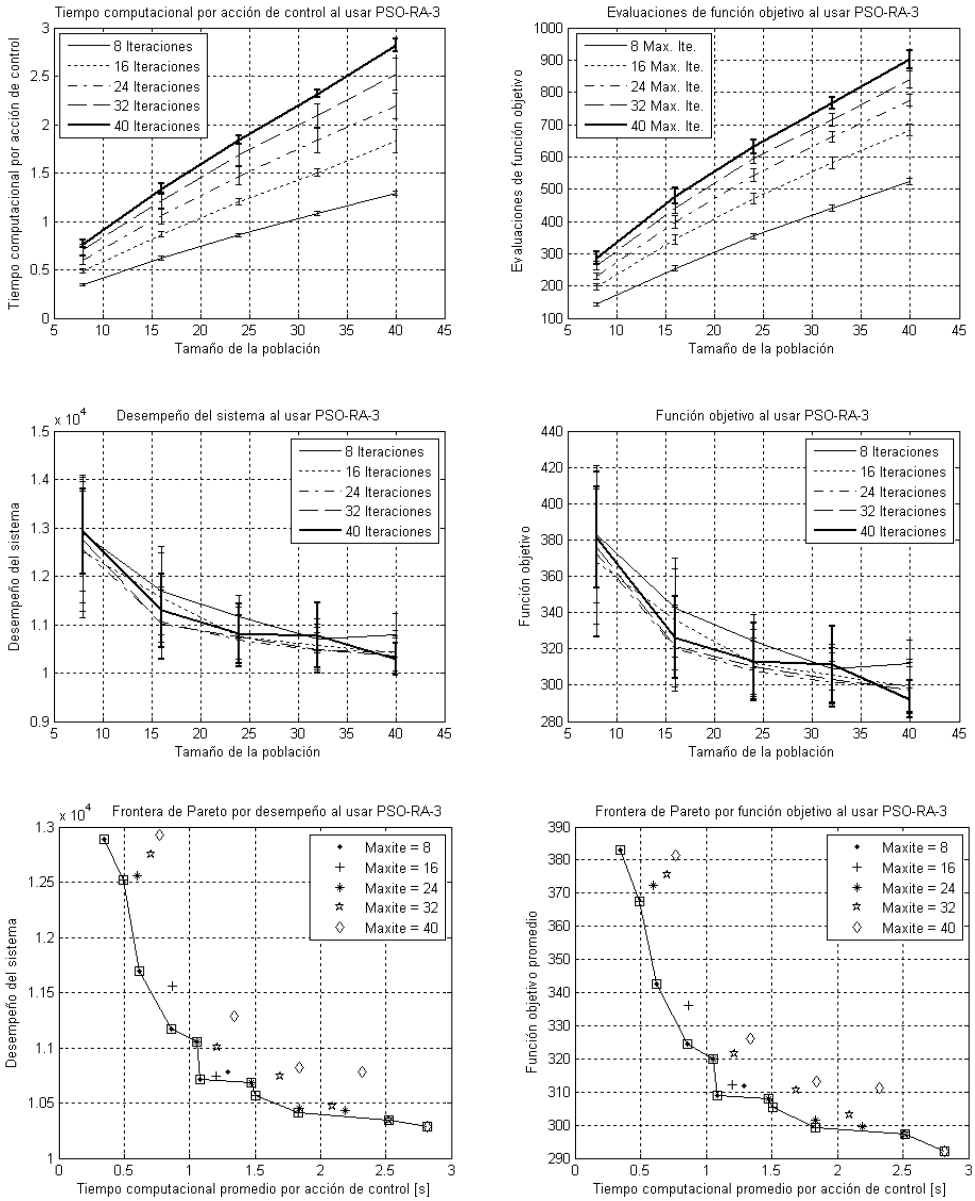


Figura 100: Gráficos de método PSO-RA-3

Tabla 39: Frontera de Pareto para el método PSO-RA-3

Tiempo promedio por acción de control [s]	Desempeño del sistema [ºC^2]	Número de iteraciones	Número de partículas
0,34525	12889	8	8
0,61814	11698	8	16
0,85998	11169	8	24
1,0822	10711	8	32
0,491	12516	16	8
1,5054	10568	16	32
1,8323	10415	16	40
1,0556	11049	24	16
1,4725	10680	24	24
2,5201	10349	32	40
2,8201	10287	40	40

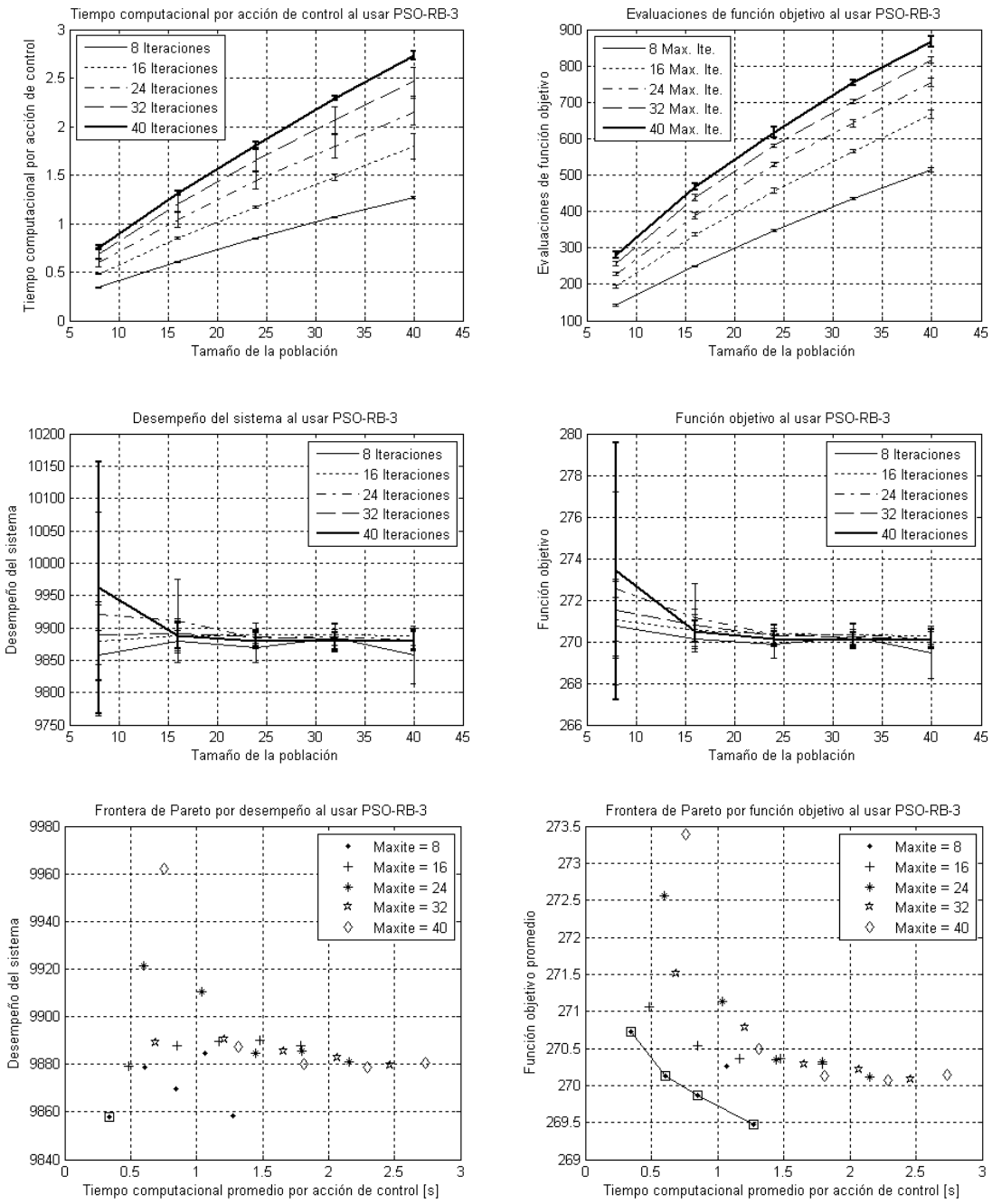


Figura 101: Gráficos de método PSO-RB-3

Tabla 40: Frontera de Pareto para el método PSO-RB-3

Tiempo promedio por acción de control [s]	Desempeño del sistema [ºC^2]	Número de iteraciones	Número de partículas
0,34151	9857,8	8	8

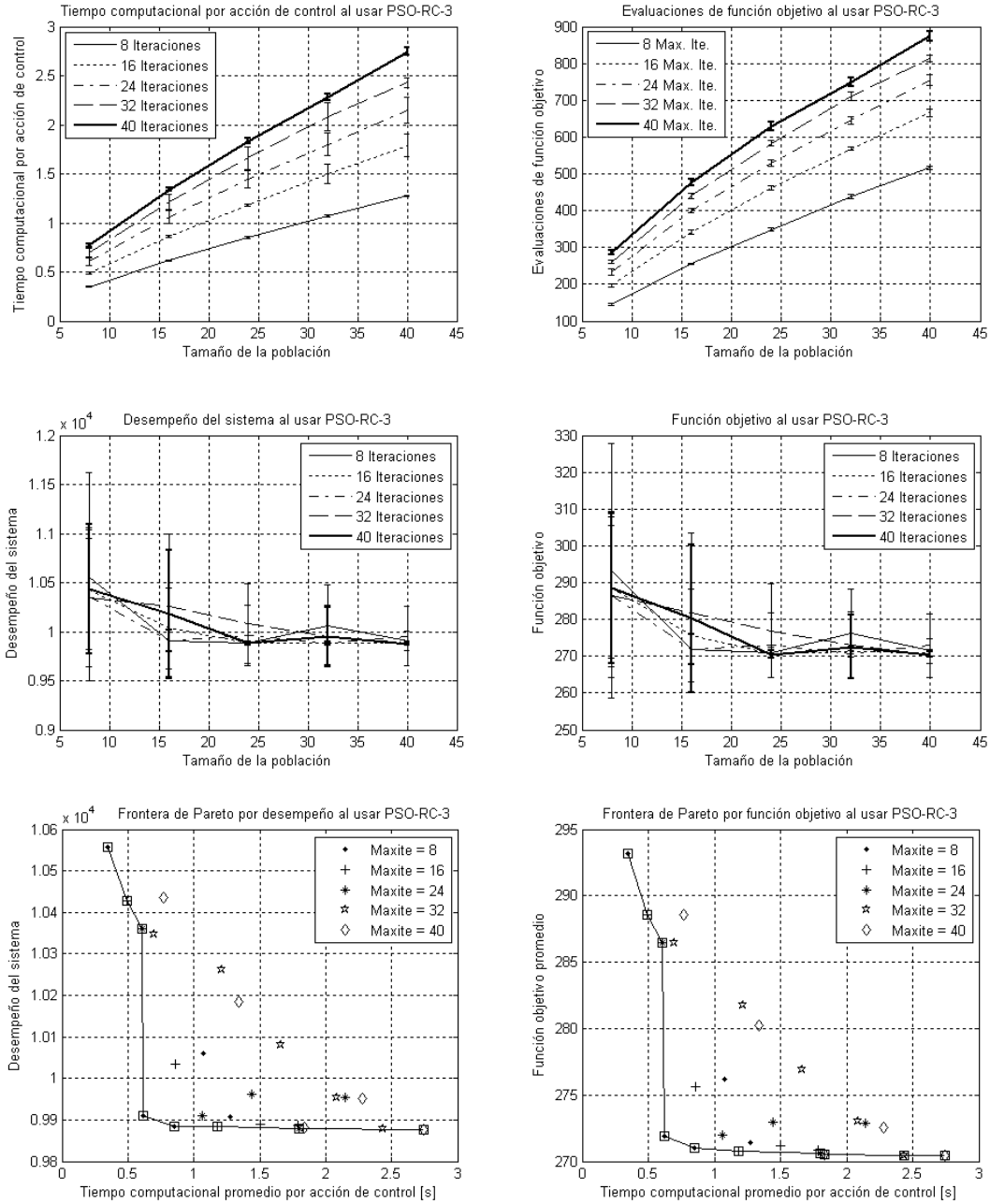


Figura 102: Gráficos de método PSO-RC-3

Tabla 41: Frontera de Pareto para el método PSO-RC-3

Tiempo promedio por acción de control [s]	Desempeño del sistema [ºC^2]	Número de iteraciones	Número de partículas
0,34917	10558	8	8
0,61935	9910,3	8	16
0,85077	9884	8	24
0,48943	10428	16	8
1,1811	9883,2	16	24
0,60511	10361	24	8
1,8011	9879,2	24	32
2,7445	9876,6	40	40

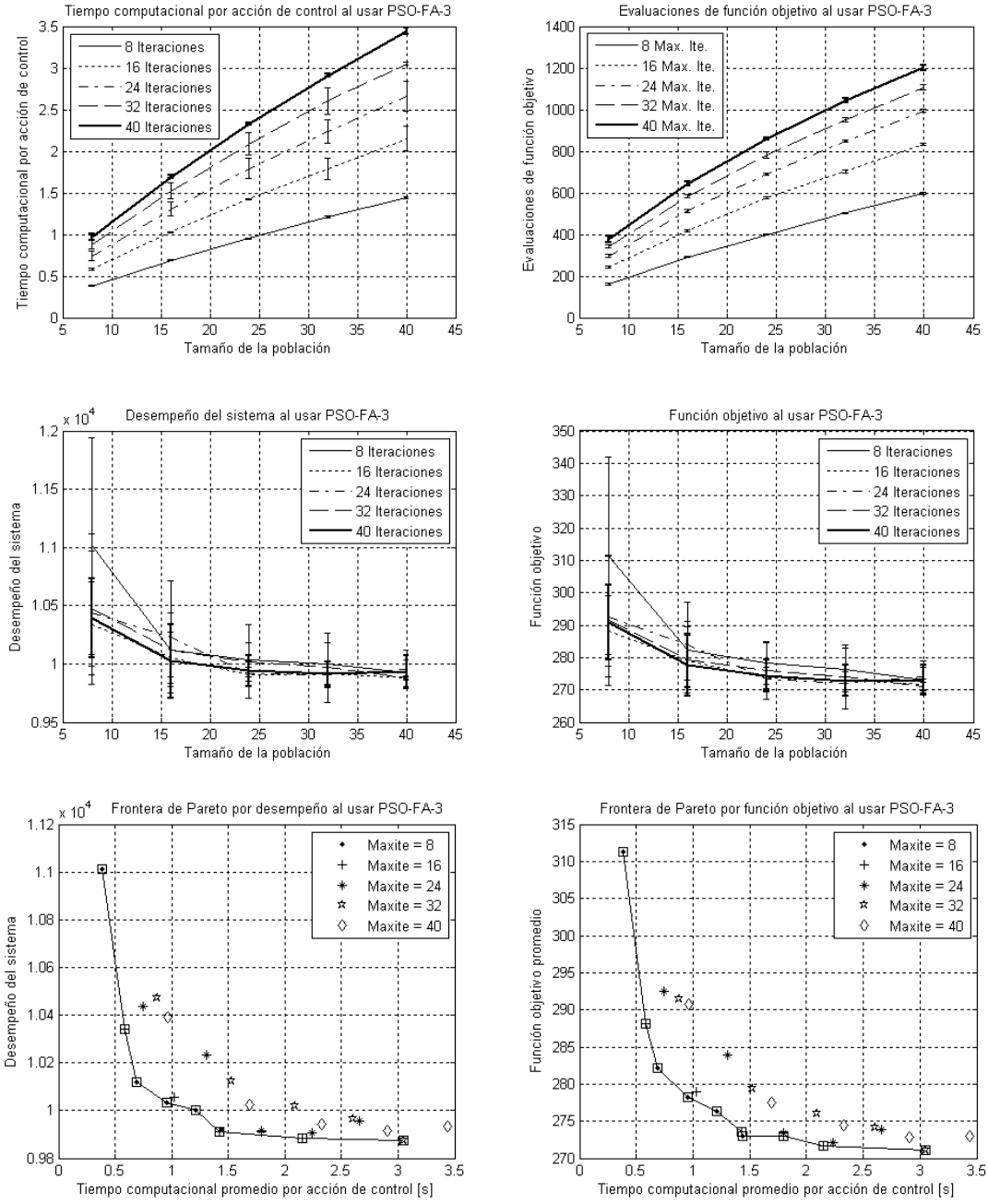


Figura 103: Gráficos de método PSO-FA-3

Tabla 42: Frontera de Pareto para el método PSO-FA-3

Tiempo promedio por acción de control [s]	Desempeño del sistema [ºC^2]	Número de iteraciones	Número de partículas
0,38125	11011	8	8
0,6899	10118	8	16
0,9542	10033	8	24
1,2098	9999,4	8	32
0,58571	10340	16	8
1,4237	9909,7	16	24
2,1523	9883,6	16	40
3,0513	9876,1	32	40

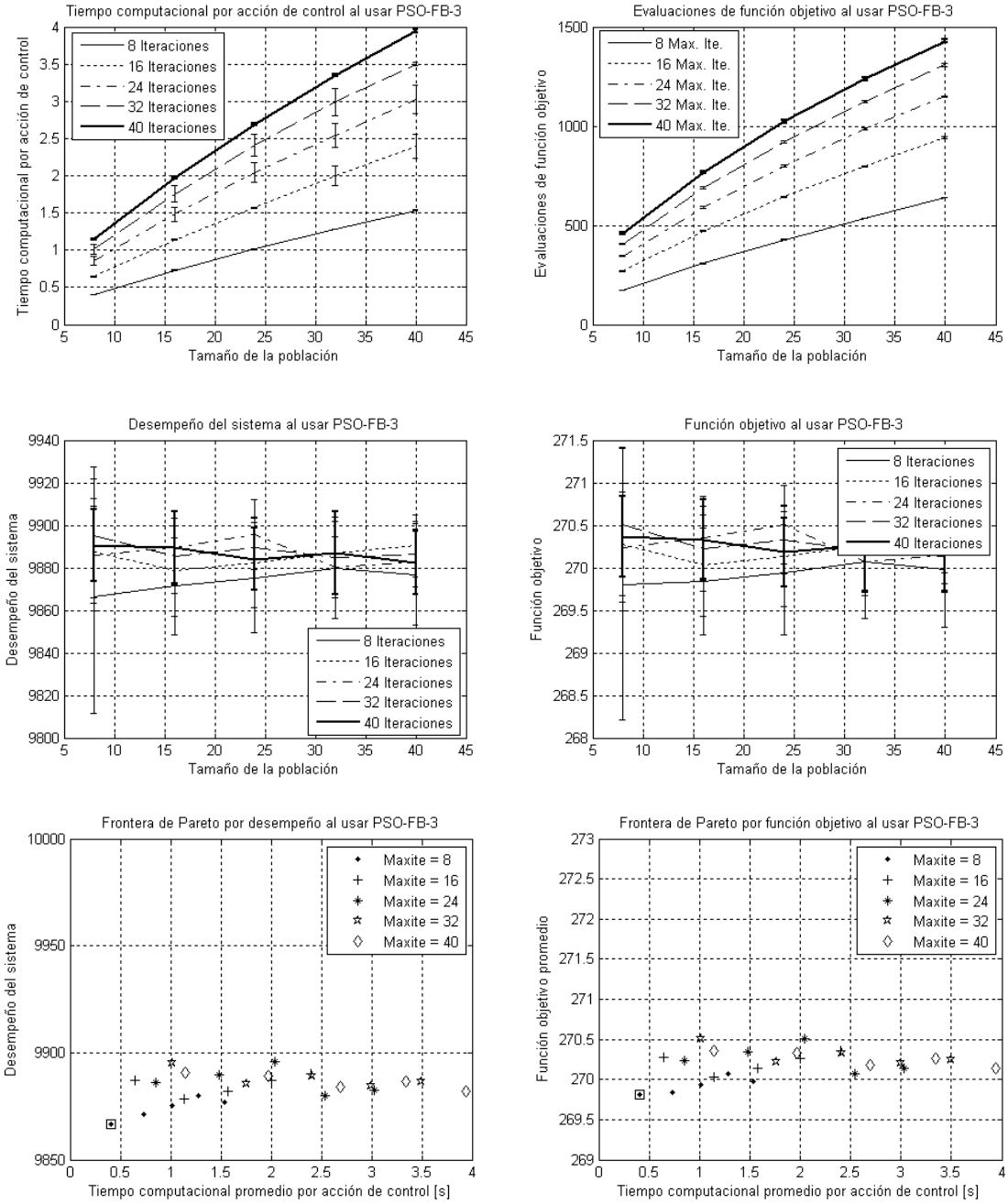


Figura 104: Gráficos de método PSO-FB-3

Tabla 43: Frontera de Pareto para el método PSO-FB-3

Tiempo promedio por acción de control [s]	Desempeño del sistema [ºC^2]	Número de iteraciones	Número de partículas
0,40599	9866,5	8	8

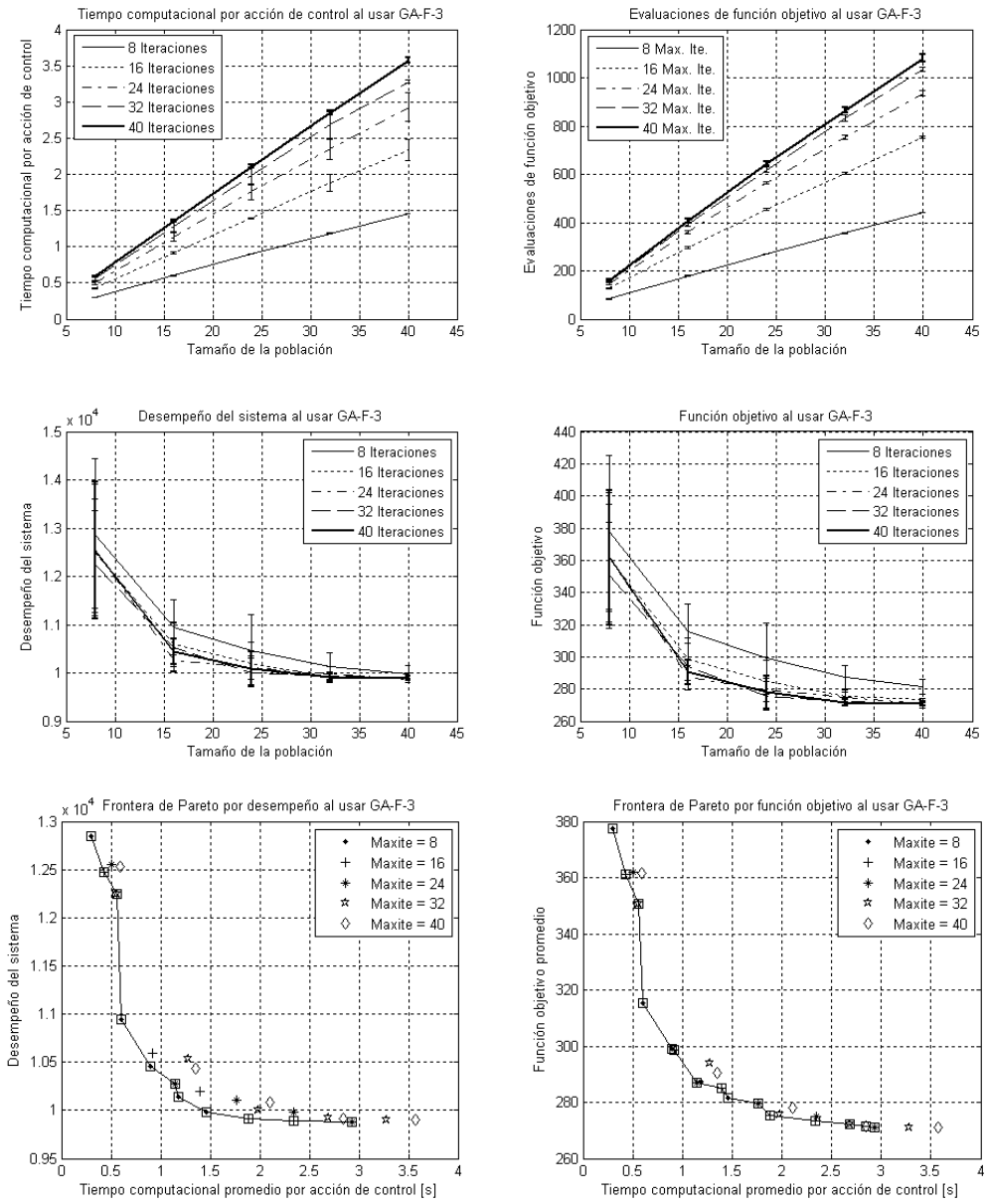


Figura 105: Gráficos de método GA-F-3

Tabla 44: Frontera de Pareto para el método GA-F-3

Tiempo promedio por acción de control [s]	Desempeño del sistema [ºC^2]	Número de iteraciones	Número de partículas
0,29855	12851	8	8
0,60437	10948	8	16
0,89908	10460	8	24
1,1817	10141	8	32
1,4572	9977,9	8	40
0,4268	12478	16	8
1,8812	9917,4	16	32
2,3386	9892,9	16	40
1,1418	10274	24	16
2,9303	9876,6	24	40
0,55814	12251	32	8

13.2 Gráficos de métodos enteros mixtos para el reactor batch con entradas continuas y discretas

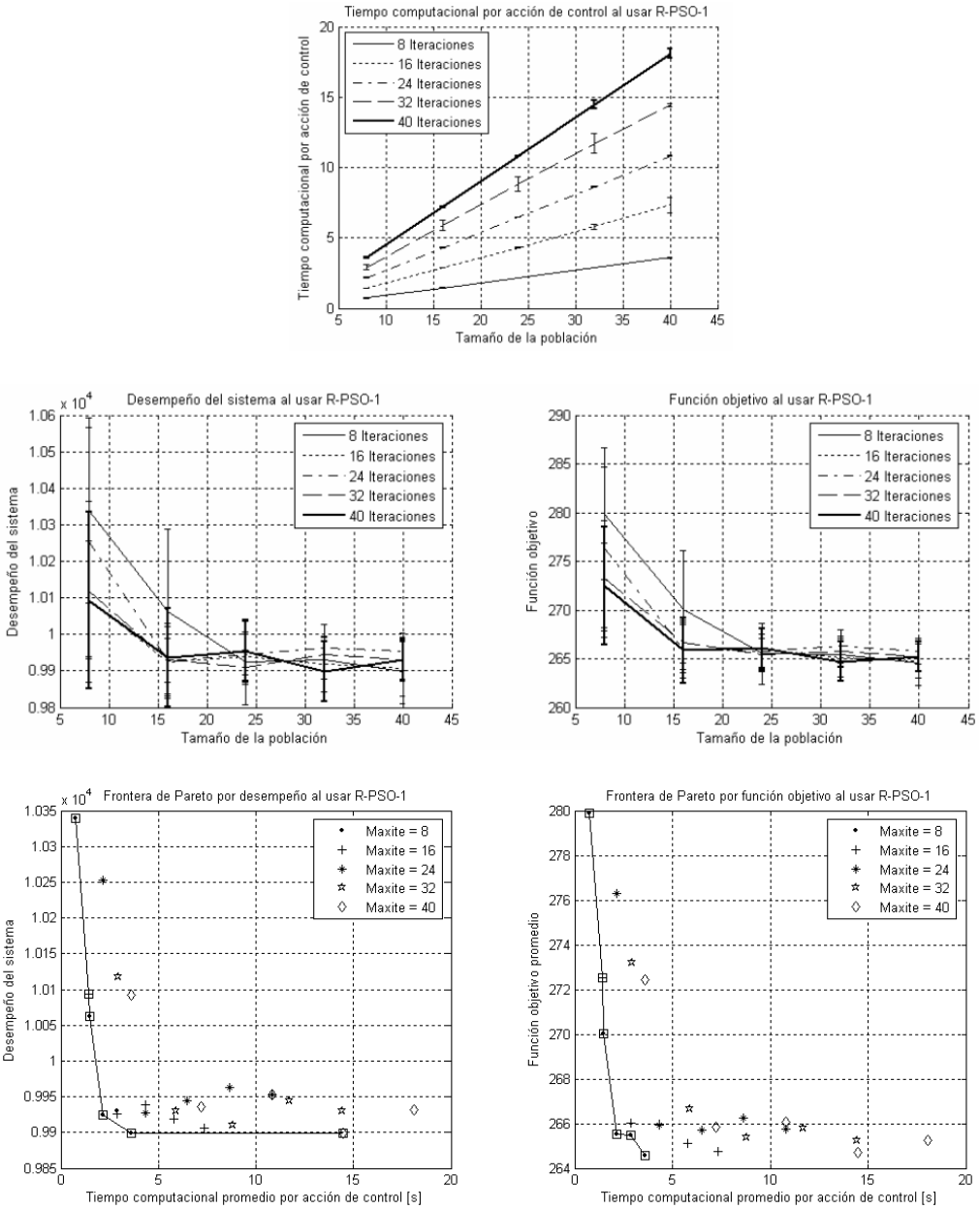


Figura 106: Gráficos de método R-PSO-1

Tabla 45: Frontera de Pareto para el método R-PSO-1

Tiempo promedio por acción de control [s]	Desempeño del sistema [ºC^2]	Número de iteraciones	Número de partículas
0,73444	10339	8	8
1,4552	10062	8	16
2,1722	9925	8	24
3,603	9899,4	8	40
1,4456	10094	16	8
14,475	9898,5	40	32

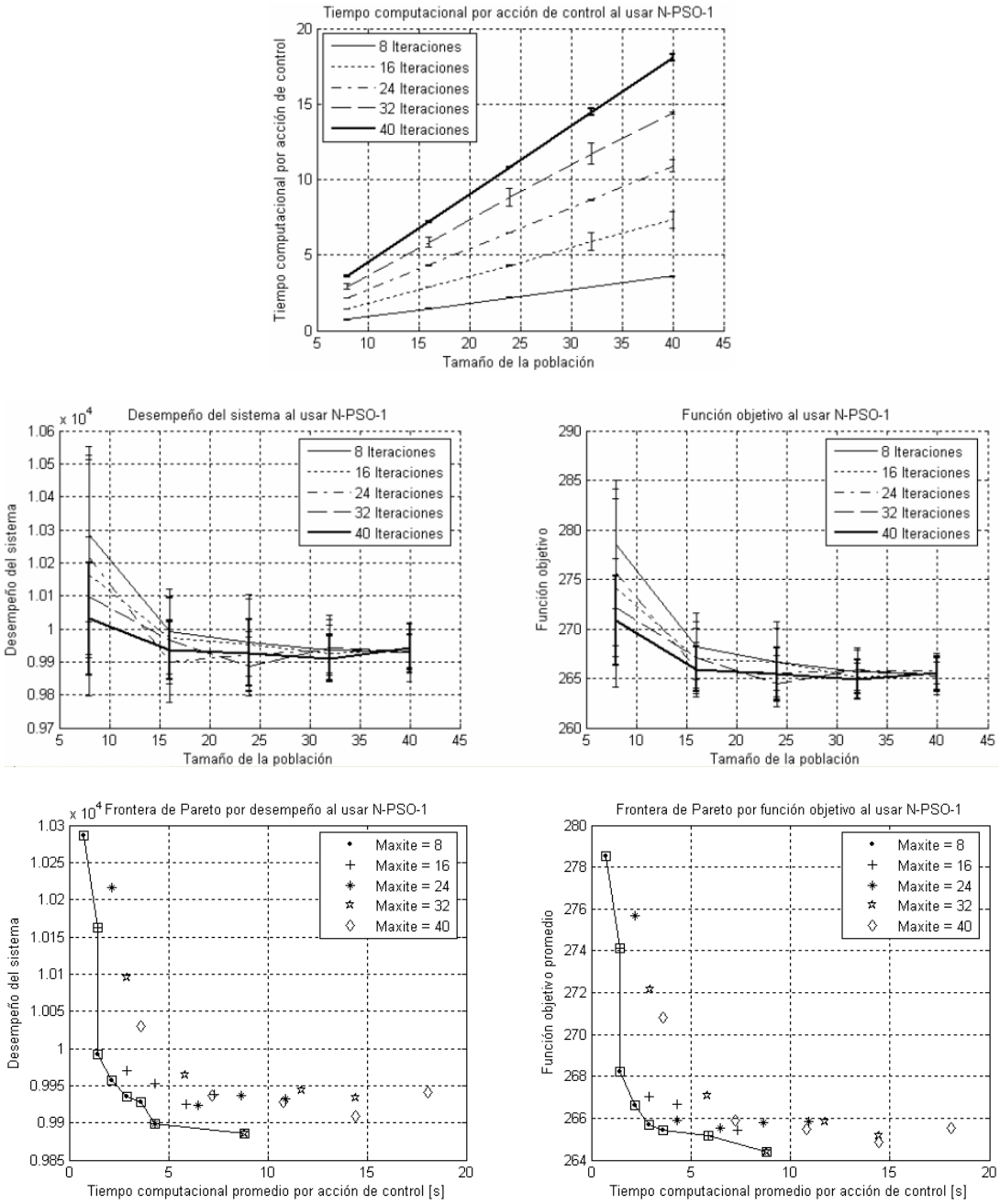


Figura 107: Gráficos de método N-PSO-1

Tabla 46: Frontera de Pareto para el método N-PSO-1

Tiempo promedio por acción de control [s]	Desempeño del sistema [ºC^2]	Número de iteraciones	Número de partículas
0,7339	10286	8	8
1,4528	9992,1	8	16
2,1734	9957,2	8	24
2,8861	9935,7	8	32
3,6063	9928,2	8	40
1,4454	10162	16	8
4,324	9899,3	24	16
8,8102	9885,6	32	24

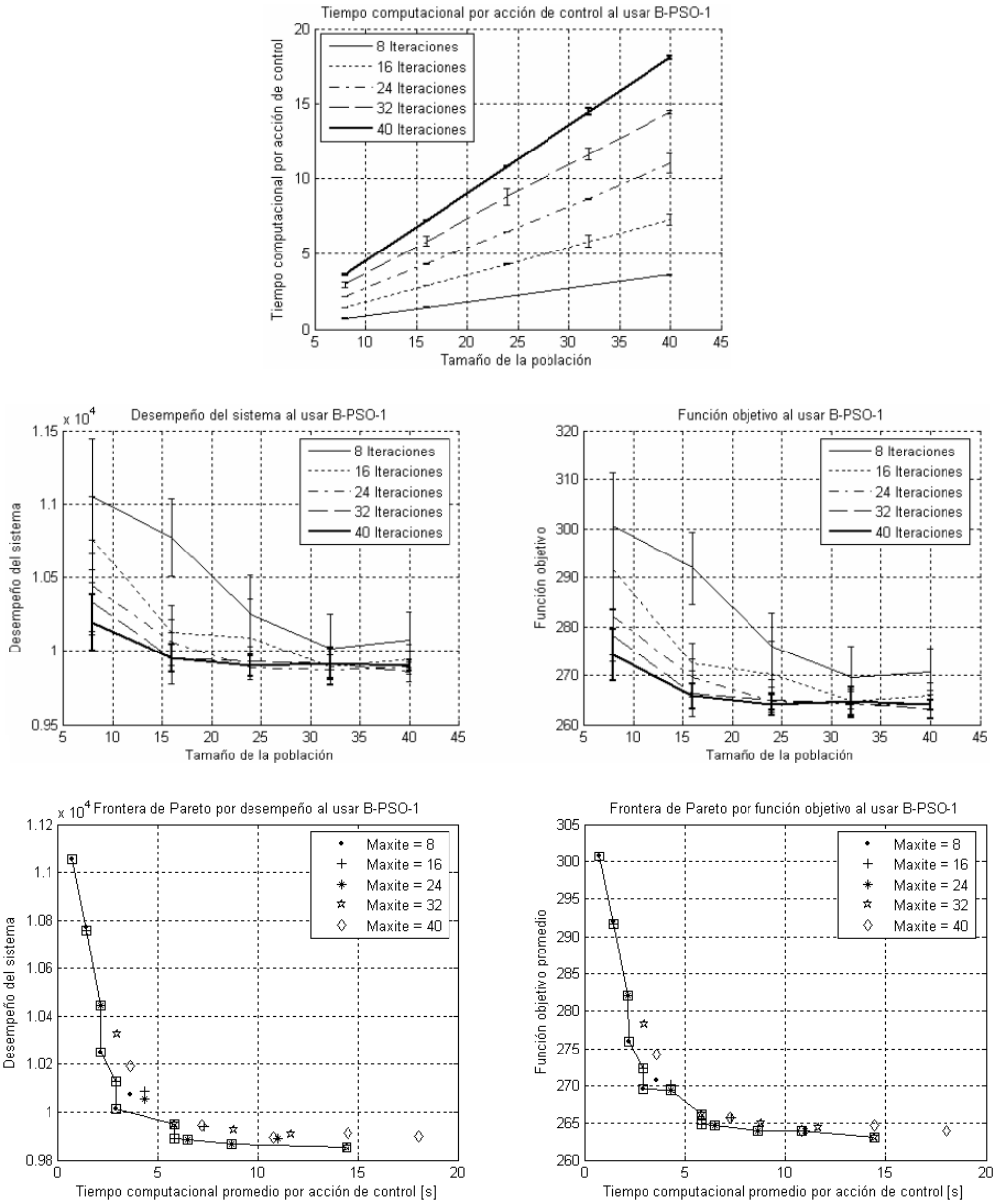


Figura 108: Gráficos de método B-PSO-1

Tabla 47: Frontera de Pareto para el método B-PSO-1

Tiempo promedio por acción de control [s]	Desempeño del sistema [ºC^2]	Número de iteraciones	Número de partículas
0,73157	11052	8	8
2,17	10253	8	24
2,8852	10015	8	32
1,4455	10758	16	8
2,8837	10130	16	16
5,8535	9891,3	16	32
2,1639	10445	24	8
6,4818	9887,7	24	24
8,6542	9871,3	24	32
5,8488	9950,8	32	16
14,442	9857,1	32	40

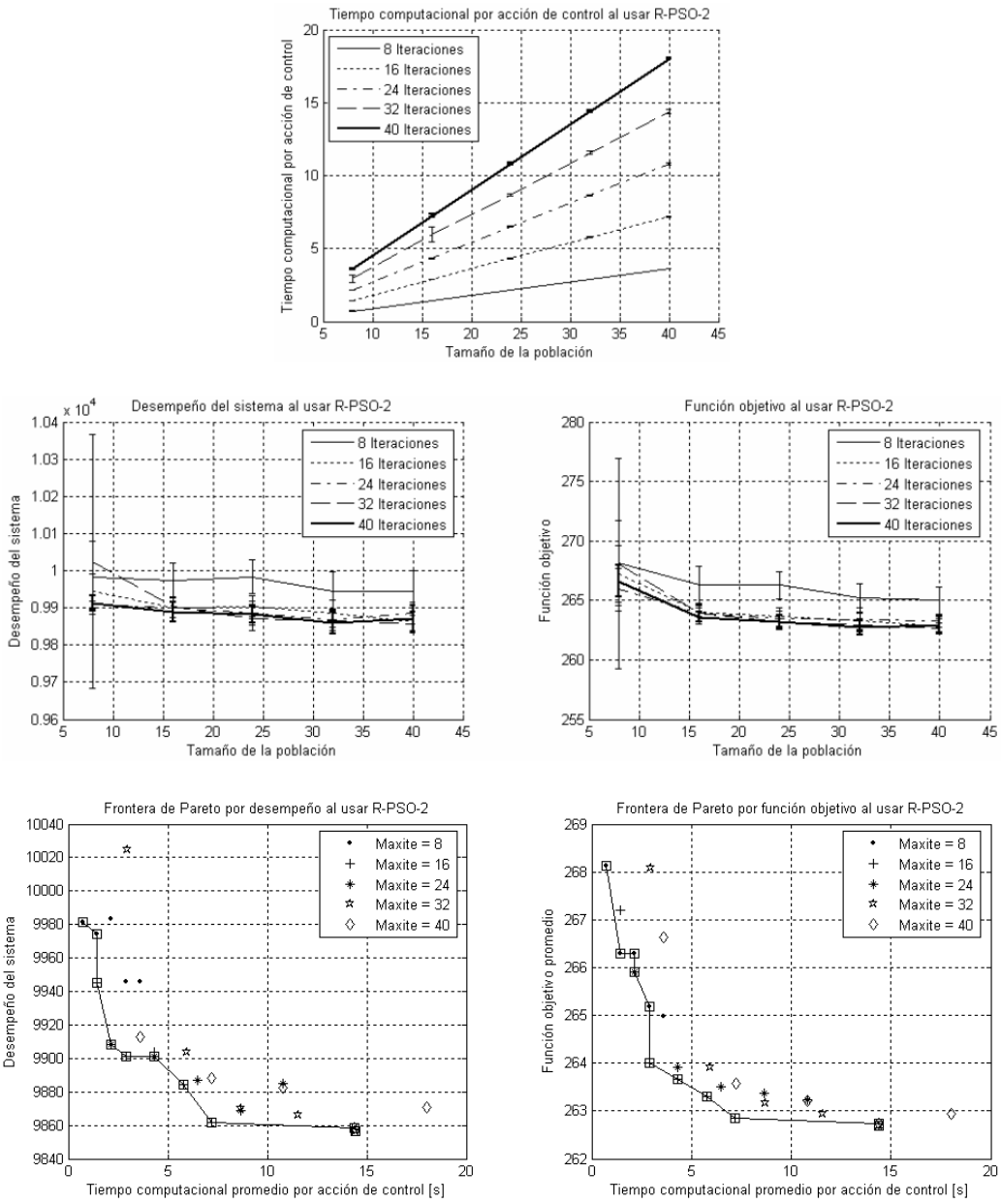


Figura 109: Gráficos de método R-PSO-2

Tabla 48: Frontera de Pareto para el método R-PSO-2

Tiempo promedio por acción de control [s]	Desempeño del sistema [ºC^2]	Número de iteraciones	Número de partículas
0,7314	9981,2	8	8
1,4456	9974	8	16
1,4468	9945,2	16	8
2,8917	9901,4	16	16
5,767	9884,4	16	32
7,2083	9861,7	16	40
2,1679	9908,5	24	8
4,3294	9901,3	24	16
14,407	9856,2	32	40
14,391	9858,4	40	32

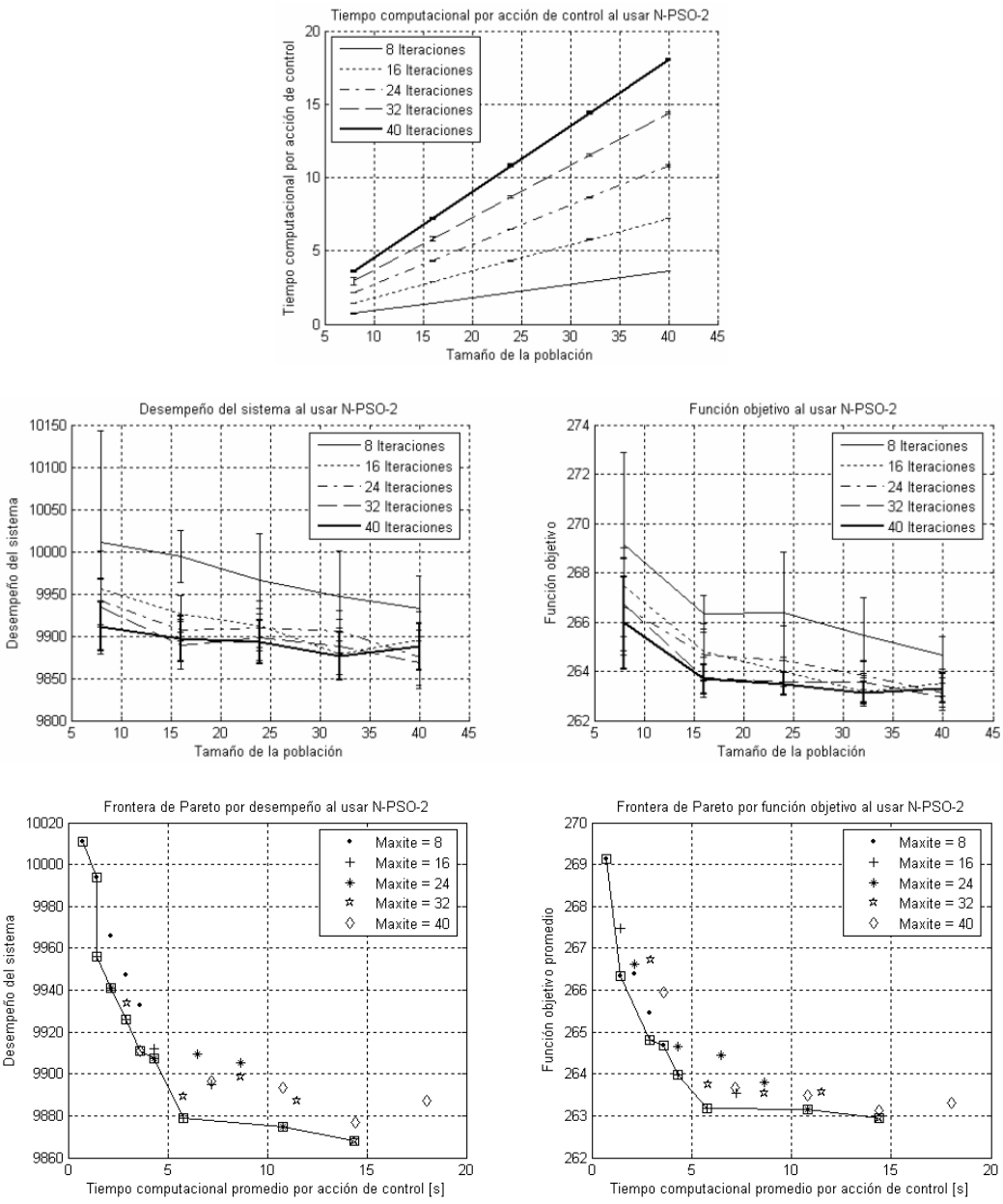


Figura 110: Gráficos de método N-PSO-2

Tabla 49: Frontera de Pareto para el método N-PSO-2

Tiempo promedio por acción de control [s]	Desempeño del sistema [ºC^2]	Número de iteraciones	Número de partículas
0,73292	10011	8	8
1,4458	9994,1	8	16
1,4491	9956	16	8
2,8928	9926,1	16	16
5,7667	9878,8	16	32
2,1668	9941,2	24	8
4,3318	9907,2	24	16
10,809	9874,8	24	40
14,389	9868,2	32	40
3,6045	9911	40	8

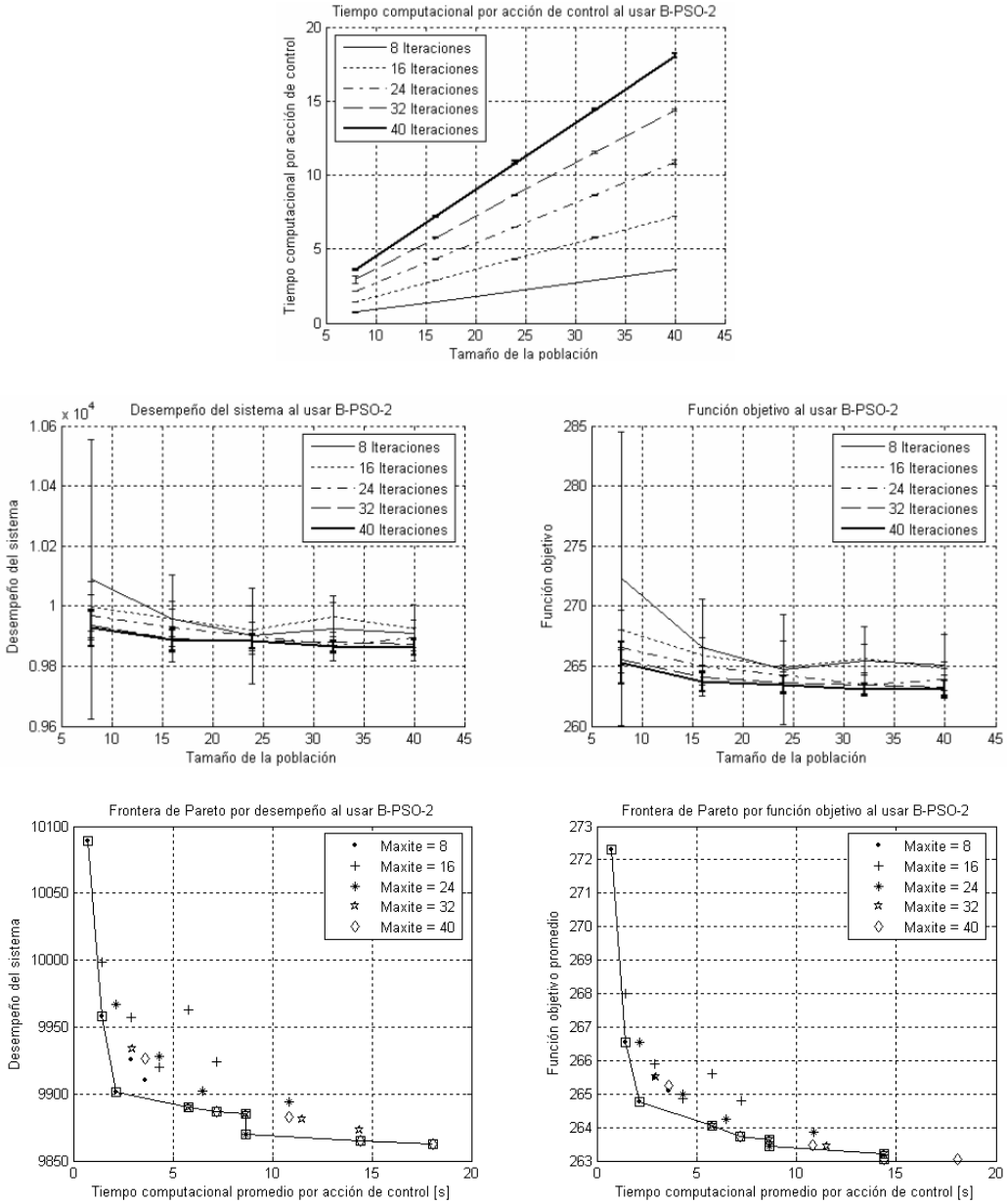


Figura 111: Gráficos de método B-PSO-2

Tabla 50: Frontera de Pareto para el método B-PSO-2

Tiempo promedio por acción de control [s]	Desempeño del sistema [ºC^2]	Número de iteraciones	Número de partículas
0,73609	10089	8	8
1,4459	9958,2	8	16
2,1664	9901,6	8	24
8,6499	9870	24	32
5,7675	9890,4	32	16
8,6497	9885	32	24
7,2019	9887	40	16
14,417	9865	40	32
18,06	9862,3	40	40

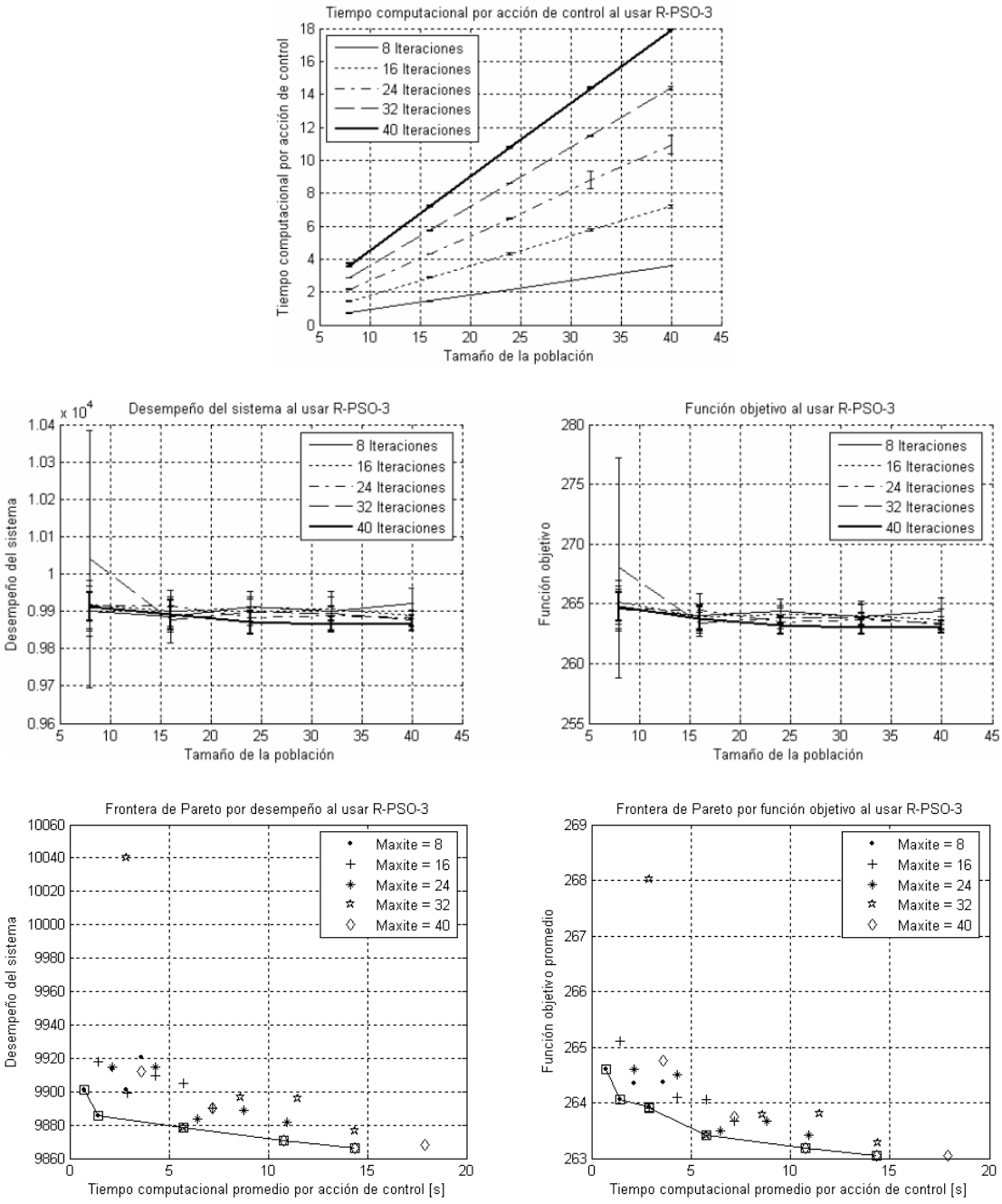


Figura 112: Gráficos de método R-PSO-3

Tabla 51: Frontera de Pareto para el método R-PSO-3

Tiempo promedio por acción de control [s]	Desempeño del sistema [ºC^2]	Número de iteraciones	Número de partículas
0,73123	9901,4	8	8
1,453	9885,8	8	16
5,746	9878,2	32	16
10,766	9870,4	40	24
14,349	9866,4	40	32

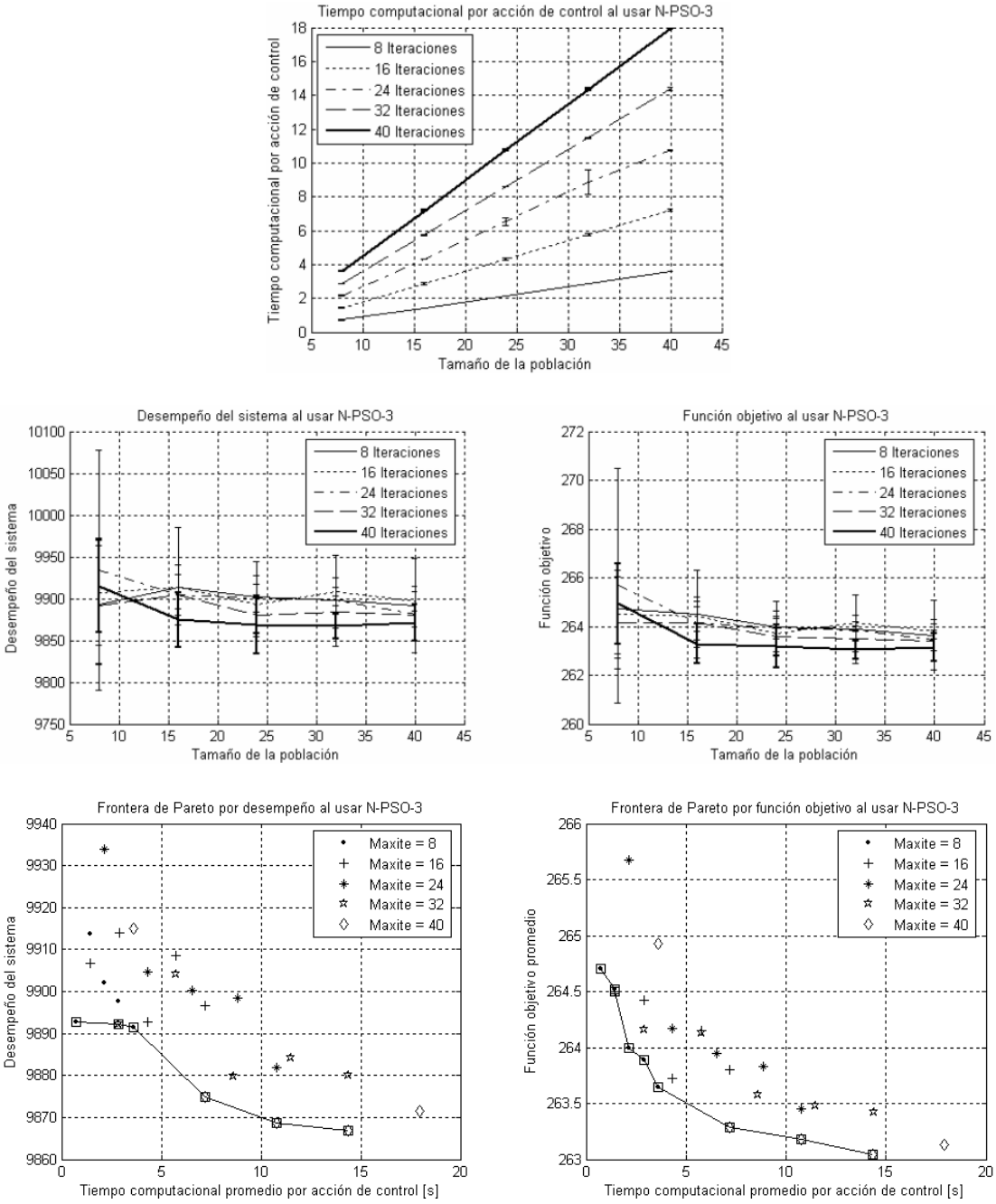


Figura 113: Gráficos de método N-PSO-3

Tabla 52: Frontera de Pareto para el método N-PSO-3

Tiempo promedio por acción de control [s]	Desempeño del sistema [ºC^2]	Número de iteraciones	Número de partículas
0,72958	9892,9	8	8
3,5888	9891,5	8	40
2,8763	9892,2	32	8
7,1777	9874,8	40	16
10,765	9868,7	40	24
14,35	9866,9	40	32

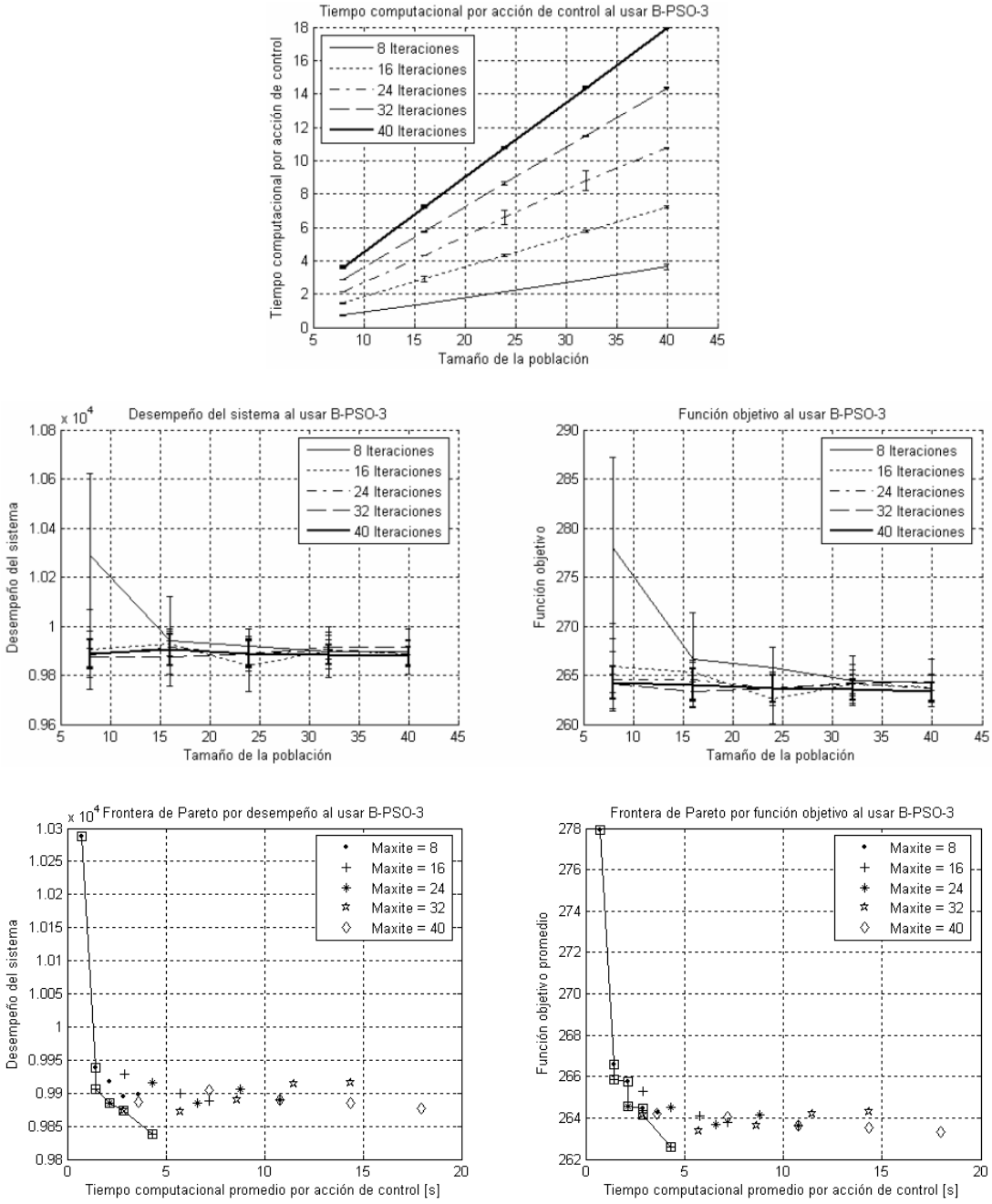


Figura 114: Gráficos de método B-PSO-3

Tabla 53: Frontera de Pareto para el método B-PSO-3

Tiempo promedio por acción de control [s]	Desempeño del sistema [ºC^2]	Número de iteraciones	Número de partículas
0,72792	10287	8	8
1,4433	9938,5	8	16
1,4606	9905,7	16	8
4,3203	9837,5	16	24
2,1581	9884,2	24	8
2,8782	9873,7	32	8