UNIVERSIDAD DE MURCIA
FACULTAD DE INFORMÁTICA
MÁSTER EN NUEVAS
TECNOLOGÍAS EN INFORMÁTICA

METAHEURÍSTICAS PARAMETRIZADAS PARALELAS APLICADAS A UN PROBLEMA DE OPTIMIZACIÓN DE COSTES EN LA EXPLOTACIÓN DE RECURSOS HÍDRICOS

MEMORIA DEL TRABAJO FIN DE MÁSTER

ITINERARIO DE SUPERCOMPUTACIÓN

AUTOR:
JOSÉ MATÍAS CUTILLAS LOZANO
JOSEMATIAS.CUTILLAS@UM.ES

TUTOR:
DOMINGO GIMÉNEZ CÁNOVAS
(DPTO. DE INFORMÁTICA Y SISTEMAS, UNIVERSIDAD DE MURCIA)

MURCIA, SEPTIEMBRE DE 2011

RESUMEN

Algunos problemas de optimización se pueden abordar sólo con métodos metaheurísticos, y para obtener una metaheurística satisfactoria es necesario normalmente desarrollar y experimentar con varios métodos y adaptar cada uno de ellos a cada problema particular. En nuestro caso, estudiaremos un problema de optimización de costes en la explotación de recursos hídricos, concretamente, minimizando el coste del consumo de energía eléctrica sujeto a una serie de restricciones.

El uso de un esquema unificado parametrizado de metaheurísticas facilita el desarrollo de metaheurísticas reutilizando las funciones básicas. Dando diferentes valores a los parámetros en cada función, podemos obtener distintas metaheurísticas o combinaciones de ellas. Así, el esquema unificado parametrizado facilita el desarrollo de metaheurísticas y su aplicación. Debido al elevado número de experimentos necesarios para la selección y adaptación de la metaheurística, se puede utilizar paralelismo para reducir el tiempo de ejecución. Para obtener versiones paralelas de las metaheurísticas y adaptarlas a las características del sistema paralelo, se desarrollará un esquema unificado parametrizado en memoria compartida. Dado un sistema computacional particular y fijados los parámetros para la metaheurística secuencial, la selección apropiada de parámetros en el esquema unificado paralelo facilita el desarrollo de metaheurísticas paralelas eficientes.

ÍNDICE

Capítulo 1
INTRODUCCIÓN

1.1 PLANTEAMIENTO

Numerosos problemas de organización, planificación y logística, en los entornos industrial, de servicios y público, pueden ser abordados desde la investigación operativa mediante la construcción de modelos de optimización. La resolución de estos modelos, a veces técnicamente difícil, proporciona una solución a estos problemas complejos que puede ser muy útil como ayuda en la toma de decisiones. La metodología utilizada puede resumirse en varias etapas: en una primera etapa se define el problema y los objetivos, y se efectúa la recopilación de datos. En una segunda etapa se construye un modelo de optimización. En una tercera etapa se resuelve el modelo obtenido mediante un paquete de optimización, si ello es posible, o mediante una solución heurística. Y en la última etapa se analizan los resultados y la robustez de la solución.

En nuestro caso, el problema consiste en la optimización de costes en la explotación de recursos hídricos, concretamente en la minimización del coste eléctrico de bombeo de agua potable sujeta a una serie de restricciones (ver anexo A). Puesto que partimos de la resolución del problema mediante una aproximación inicial secuencial con algoritmos genéticos [8], el presente trabajo comenzará mejorando la tercera etapa antes mencionada. Para la resolución del problema utilizaremos las ventajas que las nuevas tecnologías de computación nos ofrecen en cuanto a potencia y velocidad de cálculo (paralelismo) y procesamiento de datos y resultados. Actualmente la mayoría de los sistemas computacionales paralelos están formados por componentes multinúcleo. Los ordenadores personales y los portátiles son multinúcleo, y los clústeres y supercomputadores están construidos conectando nodos multinúcleo. Así, el desarrollo de versiones multinúcleo eficientes de nuestros algoritmos es obligatorio si queremos usar de manera eficiente los sistemas a los que tenemos acceso. Los sistemas multinúcleo pueden ser programados con el paradigma de memoria compartida, usando OpenMP, el cual usaremos para desarrollar el esquema parametrizado unificado paralelo de memoria compartida a partir de un esquema parametrizado unificado de metaheurísticas [3,5].

Dado que la mayoría de los problemas combinatorios atractivos y de interés, incluyendo el nuestro, son de tipo NP, los métodos exactos no son apropiados excepto para

problemas de tamaño pequeño. Por esta razón se utilizan muchos métodos de aproximación que permiten obtener soluciones de alta calidad en tiempos de ejecución aceptables. En décadas recientes, las metaheurísticas han surgido como técnicas adecuadas para desarrollar algoritmos de aproximación [9,10]. Las metaheurísticas reúnen métodos e ideas de diferentes campos, como inteligencia artificial, matemáticas y biología. El punto más importante es su fácil e inmediata aplicabilidad a problemas complejos.

El uso de un esquema parametrizado unificado para metaheurísticas [5] facilita el desarrollo de metaheurísticas nuevas o híbridas para la experimentación y adaptación a un problema particular. Sin embargo, en el proceso de obtención de una metaheurística bien adaptada a un problema, es necesario experimentar con un elevado número de metaheurísticas y los valores de sus parámetros y, por eso, el tiempo dedicado a los experimentos es muy grande. Para reducir este problema, pueden desarrollarse versiones paralelas de los métodos. Existen algunos estudios de paralelización de metaheurísticas [1]. Cada metaheurística tiene un esquema paralelo diferente, y algunas de ellas pueden tener estructuras diferentes. En nuestro trabajo consideramos el desarrollo común de versiones paralelas empleando un esquema metaheurístico unificado para obtener un esquema unificado paralelo de metaheurísticas [3] que nos permita abordar el problema de optimización de costes en la explotación de recursos hídricos aplicando diferentes metaheurísticas en un entorno paralelo.

Al ser el esquema paralelo parametrizado, se pueden seleccionar los valores de algunos parámetros algorítmicos (número de threads) para optimizar la ejecución de la metaheurística paralela obtenida del esquema secuencial parametrizado de metaheurísticas. Los valores óptimos de los parámetros algorítmicos dependerán tanto de los parámetros de la metaheurística como de las características del sistema computacional.

1.2 OBJETIVOS

En esta tesis proponemos aplicar metaheurísticas secuenciales y paralelas a un problema de optimización de costes en la explotación de recursos hídricos (ver anexo A). Concretamente se plantea el uso de metaheurísticas parametrizadas que facilitan la experimentación con diferentes metaheurísticas y la hibridación/combinación para adaptarlas al problema concreto que nos ocupa [5]. Debido al elevado número de experimentos necesarios para la selección y tuneado de la metaheurística, es

recomendable usar paralelismo para reducir el tiempo de ejecución. Para obtener versiones paralelas de las metaheurísticas y adaptarlas a las características del sistema paralelo, se utilizará un *esquema unificado parametrizado en memoria compartida* [3]. Dado un sistema computacional particular y fijados los parámetros para la metaheurística secuencial, la selección apropiada de parámetros en el esquema unificado paralelo facilita el desarrollo de metaheurísticas paralelas eficientes.

1.3 ESTADO DEL ARTE

En este apartado vamos a ver qué otros trabajos se han realizado anteriormente sobre el empleo de técnicas metaheurísticas en los problemas de optimización de recursos, así como estudios previos de metaheurísticas paralelas parametrizadas.

La optimización en el sentido de encontrar la mejor solución, o al menos una solución lo suficientemente buena, para un cierto problema es un campo de vital importancia en la vida real y en ingeniería. Debido a la gran importancia de los problemas de optimización, a lo largo de la historia de la Informática se han desarrollado múltiples métodos para tratar de resolverlos. Nos centramos en el estudio de las metaheurísticas, puesto que nuestro problema concreto no tiene un algoritmo específico que dé una solución satisfactoria. Podemos destacar ciertas propiedades fundamentales que caracterizan a este tipo de métodos: las metaheurísticas son estrategias o plantillas generales que "guían" el proceso de búsqueda cuyo objetivo es una exploración eficiente del espacio de búsqueda para encontrar soluciones (casi) óptimas. Las metaheurísticas son algoritmos no exactos, y generalmente no deterministas, que pueden incorporar mecanismos para evitar regiones no prometedoras del espacio de búsqueda. El esquema básico de cualquier metaheurística tiene una estructura predefinida y se hace uso del conocimiento del problema que se trata de resolver en forma de heurísticos específicos que son controlados por una estrategia de más alto nivel. Podemos encontrar referencias generales sobre metaheurísticas en [9,10,12].

En cuanto al problema inicial de minimización de costes de energía eléctrica en la explotación de recursos hídricos, tenemos la referencia en el trabajo previo realizado empleando algoritmos genéticos en [8], así como un resumen de aspectos técnicos relevantes en el anexo A.

Debido a la gran amplitud de este campo de estudio, se han revisado algunos trabajos de este tipo en diversas disciplinas científicas, como la asignación óptima de recursos en

sistemas computacionales paralelos [2,6,7], que nos han permitido familiarizarnos con problemas de este tipo. En el sentido de optimización, se han tenido en cuenta trabajos relacionados con econometría y concretamente de obtención de modelos de ecuaciones simultáneas empleando varias metaheurísticas y combinaciones de ellas [4,11]. Aunque el uso de metaheurísticas permite reducir significativamente la complejidad temporal del proceso de búsqueda, este tiempo puede seguir siendo muy elevado en algunos problemas de interés real. El paralelismo puede ayudar no sólo a reducir el tiempo de cómputo, sino a producir también una mejora en la calidad de las soluciones encontradas. Podemos encontrar referencias de metaheurísticas paralelas en [1]. Se han revisado, además, varios trabajos que incluyen algunos primeros ensayos sobre el esquema parametrizado en memoria compartida de metaheurísticas que desarrollaremos a lo largo del presente trabajo [3,5] y que aplicaremos a nuestro problema de optimización de costes.

1.4 METODOLOGÍA

Como ya se ha comentado, no partimos de cero, sino que pretendemos mejorar un problema de minimización de costes ya existente. Se trata de un estudio previo de optimización mediante algoritmos genéticos [8], empleando MATLAB como lenguaje de programación del algoritmo. Así, nos planteamos una serie de objetivos previos al desarrollo principal de la tesis, como son el estudio del trabajo inicial, comprensión del modelo matemático que describe el problema, modificación del código del algoritmo original (en MATLAB) y su traducción a lenguaje C++. Además, adicionalmente a lo anterior, nos planteamos la modificación tanto estructural como procedimental del código original, estructurando las funciones y los datos de forma más compacta y reutilizable y, en definitiva, más adecuada al esquema parametrizado de metaheurísticas inicial que nos servirá como punto de partida para desarrollar el algoritmo paralelo definitivo.

Finalmente, una vez obtenido el código final que implementa el algoritmo de optimización de nuestro problema, realizaremos series de experimentos tanto secuenciales (al principio) como paralelos (después) que nos permitirán obtener resultados sobre el comportamiento del esquema parametrizado [5] sobre nuestro problema concreto: influencia de los parámetros en la bondad de los resultados alcanzados, análisis de las metaheurísticas para encontrar aquélla que mejor se adapta a

nuestro problema, así como resultados sobre la mejor configuración de parámetros de paralelismo que minimicen los tiempos de ejecución.

1.5 EL PROBLEMA DE OPTIMIZACIÓN DE COSTES EN LA EXPLOTACIÓN DE RECURSOS HÍDRICOS. MODELO MATEMÁTICO Y COMPUTACIONAL

Vamos a comenzar nuestro estudio centrándonos en el problema de minimización de costes de energía eléctrica en la explotación de recursos hídricos, empezando por un análisis del problema desde un punto de vista matemático y computacional, para posteriormente aplicar las técnicas vistas en apartados anteriores a la realización de un estudio experimental. La definición y la descripción técnica del problema se puede consultar en el anexo A.

En primer lugar, se presenta la codificación de los elementos y la función de bondad que serán comunes a todas las metaheurísticas, así como las variables específicas del problema que constituyen el cuerpo del mismo.

Disponemos de una serie de bombas de potencia conocida localizadas en sus correspondientes pozos que extraen un caudal de agua determinado. El caudal total es la suma de los caudales aportados por cada pozo. Las bombas pueden estar en funcionamiento o fuera de servicio en un instante dado. Las bombas funcionan con energía eléctrica y ésta tiene un coste diario. Es precisamente la suma de los costes de la energía eléctrica consumida por todas las bombas durante todo un día lo que constituye la función de bondad o función objetivo. Las variables que constituyen el problema son las siguientes:

- Número de *Bombas*, representado por un entero, coincide con el número de pozos de la explotación.
- Número de *RangosHorarios*, representado por un entero, divide la explotación en diferentes tramos horarios.
- Número de *ElementosBomba*, representados por un entero, es el producto de bombas por rangos horarios.
- *CaudalBombas*, vector de números reales de tamaño *Bombas* que contiene el caudal de explotación de cada bomba.
- *Tarifas*, vector de números reales que contiene la estructura tarifaria aplicada en los experimentos (tres tarifas: llano, valle, punta).

- *PotenciaBombas*, vector de números reales de tamaño *Bombas* que contiene la potencia de trabajo en régimen estacionario de cada bomba.
- *ConductividadBombas*, vector de números reales de tamaño *Bombas* que contiene el valor de conductividad del agua explotada por cada bomba.
- *VolumenMaxBombas*, vector de números reales de tamaño *Bombas* que contiene el volumen máximo concedido diario de explotación de cada pozo.
- *ProfundidadNivelDinámico*, vector de números reales de tamaño *Bombas* que contiene la profundidad del nivel dinámico de agua de cada pozo.
- *ProfundidadNivelDinámicoMáxima*, vector de números reales de tamaño *Bombas* que contiene la profundidad del nivel dinámico máxima de cada pozo. Constituye el límite de funcionamiento del pozo, por debajo del cual se considera pozo fuera de servicio y, por tanto, bomba inactiva.
- *MargenNivel*, valor real que representa una cierta tolerancia en el nivel dinámico máximo en cada pozo.
- *EstatusPozos*, vector de números binarios de tamaño *Bombas* que representa la situación de explotación de cada pozo (bomba): 0 significa fuera de servicio, y 1 significa pozo en funcionamiento. Su valor depende de si se ha superado o no el nivel dinámico mínimo permitido.
- *VolumenTotalDiario*, es un real que representa el volumen total a explotar diariamente como suma de las aportaciones de cada pozo.
- *ToleranciaVolumen*, es un margen superior que cierra el intervalo de volumen diario total a alcanzar. De tipo real.
- *CaudalTotalMinimo*, es un número real que representa el caudal acumulado que debe circular como mínimo por las tuberías en todo momento.
- *ConductividadMaximaPermitida,* es un real que representa el valor máximo que no debe superar la media de las conductividades reales aportadas por cada pozo.

Siguiendo la notación comúnmente empleada en algoritmos evolutivos denominaremos *cromosoma* al vector de números binarios que codifica el conjunto de bombas repartidas en los distintos tramos horarios del sistema. Así, el tamaño del *cromosoma* es el resultado de multiplicar el número de bombas por el número de rangos horarios considerado, es decir, el conjunto de bombas se evalúa diariamente tantas veces como tramos horarios hayamos considerado. Si a la información aportada por cada *cromosoma* le adjuntamos su función de bondad, tenemos un *individuo*. A partir de aquí utilizaremos la nomenclatura *individuo* cuando hagamos referencia al vector de *elementos bomba,* al que acompaña, se sobreentiende, su correspondiente función de bondad o *valor*. El conjunto de individuos considerados constituye una *población*.

Tenemos entonces un individuo formado por elementos de tipo binario que representan a bombas en distintos tramos horarios y que pueden estar encendidas (valor uno) o apagadas (valor cero). No todas las posibles combinaciones dan lugar a situaciones factibles, por lo cual cada vez que se genere o modifique un individuo se tendrá que comprobar que se cumplan las restricciones del sistema. Lo que a continuación se presenta es una enumeración detallada de las ecuaciones que gobiernan el problema, como son la función objetivo a minimizar junto con sus restricciones. Además, se incluye una descripción en pseudocódigo de la implementación de dichas funciones dentro del programa que resuelve el problema:

1. *Función Objetivo.*

La función objetivo a minimizar es el coste de energía eléctrica:

$$C_e = \sum_{i=1}^{R} \sum_{j=1}^{B} T_i \cdot P_j \cdot N_i \cdot x_{ij} \qquad (1.1)$$

donde

B = Número de bombas considerado.

R = Número de rangos horarios considerados (hasta 24), con sus tramos de tarificación eléctrica correspondientes (consideramos tres en los experimentos).

C_e = Coste total de la energía eléctrica consumida por la combinación de bombas seleccionada al final del intervalo diario.

T_i = Tarifa energética en el rango horario i.

N_i = Número de horas de funcionamiento de las bombas durante todo el rango horario i.

P_j = Potencia eléctrica consumida por la bomba j, constante durante todo el día.

x_{ij} = Variable binaria de valor 1 para bomba encendida y 0 para bomba apagada.

La implementación de esta función se limitaría a calcular en un bucle, para cada elemento dentro del individuo, el producto de su representación binaria por su potencia, por su tarifa horaria aplicable y por el número de horas del tramo horario. Sumando estos productos para todos los elementos obtendremos el valor del coste eléctrico total.

2. *Restricciones.*

2.1. *R1. Satisfacción de la demanda.*

Esta restricción deriva de la condición de que la suma de los volúmenes aportados en los rangos de horas establecidos se corresponda con la demanda programada al comienzo de cada jornada:

$$H \cdot \sum_{i=1}^{R} \sum_{j=1}^{B} Q_{ij} \cdot x_{ij} = V_{dT} \qquad (1.2)$$

donde

Q_{ij} = Caudal extraído del pozo j en el intervalo horario i.

V_{dT} = Volumen diario total demandado.

H = nº de horas de cada intervalo horario (igual para todos).

Queda implementada según el algoritmo 1.1.

Algoritmo 1.1 Esquema de *RestriccionVolumenTotal*.

1. Si el VolumenDiario calculado es igual al VolumenTotal impuesto para ese día más una cierta Tolerancia entonces se cumple la restricción.
2. El VolumenDiario se calcula acumulando el caudal de todos los elementos (pozos) operativos y multiplicándolo por el número de horas del tramo horario considerado.

2.2. R2. Mantenimiento del caudal mínimo.

Se pretende establecer, en todos los tramos horarios de funcionamiento, un caudal mínimo en la tubería:

$$\sum_{j=1}^{B} Q_j \cdot x_j \geq Q_{\min.} \quad \text{para cada tramo horario.} \qquad (1.3)$$

donde

Q_j = Caudal extraído del pozo j.

$Q_{min.}$ = Caudal mínimo total en tubería para cada tramo horario.

Su implementación se lleva a cabo según el algoritmo 1.2.

Algoritmo 1.2 Esquema de *CaudalTuberiaMinimo*.

1. Recorremos mediante un bucle todo el individuo, comprobando para cada tramo horario si el caudal total acumulado de todos sus elementos es mayor que el mínimo especificado.
2. Si en algún tramo no se cumple la condición anterior, salimos del bucle informando de que no se cumple la restricción.

2.3 R3. Cumplimiento de los volúmenes máximos de explotación de cada pozo j.

En la práctica se actualiza cada día el volumen concedido acumulado para cada pozo y se introduce como parámetro del problema. De manera que cada día debe cumplirse la siguiente restricción para cada pozo *j* explotado:

$$\forall\, j, \quad \frac{24}{R} \cdot Q_j \sum_{i=1}^{R} x_{ij} \leq V_{conc.kj} \tag{1.4}$$

Podemos ver su implementación en el algoritmo 1.3.

Algoritmo 1.3 Esquema de *VolumenConcesionMaximo*.

---

1. Mediante la función *VolumenExplotadoBomba* contamos, con un bucle, las veces que un elemento (bomba) está activo para todos los rangos horarios. Una vez hecho esto, calculamos su volumen de explotación total diario como el producto de su caudal constante por las veces que la bomba está activa en cualquier rango horario y por las horas que tiene dicho intervalo.
2. Recorremos con un bucle todos los elementos del vector de volúmenes máximos de cada pozo comparándolos con los calculados con la función VolumenExplotadoBomba para cada pozo.
3. Si el volumen calculado es menor que el máximo impuesto, el elemento cumple la restricción.

2.4 R4. Mantenimiento de la conductividad media por debajo del límite.

Una vez establecido el valor límite de conductividad para el caudal total del sistema, hemos considerado que su valor real se puede calcular como una media ponderada en proporción a las conductividades y los caudales aportados por cada pozo:

$$\frac{\sum_{j=1}^{B} Q_j \cdot \sigma_j \cdot x_j}{\sum_{j=1}^{B} Q_j \cdot x_j} \leq \sigma_{lím.} \qquad \text{para cada rango horario.} \tag{1.5}$$

siendo

σ_j = Conductividad de cada pozo.

σ_{lim} = Conductividad límite de la mezcla de aguas.

El procedimiento esquemático de cálculo se presenta en el algoritmo 1.4.

Algoritmo 1.4 Esquema de *ConductividadLímite.*

1. Obtenemos la conductividad media. Para ello, vamos calculando en cada pasada del bucle que recorre los elementos de un tramo horario el producto $Q_i \cdot \sigma_i \cdot x_i$ y lo vamos acumulando para todos los elementos, así como el caudal total. Finalmente, para ese intervalo horario calculamos el cociente dado por la ecuación 1.5.
2. Recorremos mediante un bucle todo el individuo, comprobando para cada tramo horario si la conductividad media es menor que la máxima especificada.
3. Si en algún tramo no se cumple la condición anterior, salimos del bucle informando de que no se cumple la restricción.

2.5 R5. Cumplimiento de profundidades máximas de niveles dinámicos.

Aquí se impone la restricción de que cada pozo, para considerarse en funcionamiento, debe tener un cierto nivel mínimo de agua a explotar que, al sobrepasarse, conllevaría la calificación del pozo como fuera de servicio temporalmente hasta su recuperación. Basta con asegurarnos de que cada vez que se genere o modifique un elemento a 1, el estatus del pozo correspondiente está a 1, es decir, operativo con niveles de agua adecuados.

1.6 DEPURACIÓN DEL CÓDIGO Y ANÁLISIS PRELIMINARES

Una vez definido el problema, se han realizado una serie de ensayos previos destinados a depurar el código y comprobar que la solución implementada representa realmente el problema original. Se han elegido problemas relativamente pequeños, lo que permite depurar con facilidad las funciones puesto que el conjunto de resultados a analizar es reducido. Como se ha comentado, se pretende comenzar el análisis del problema empleando un algoritmo genético que, además, es el algoritmo que se utilizó para resolver el problema original con una implementación en MATLAB. Un resumen de los valores de las variables y de los parámetros del algoritmo empleados en un ensayo preliminar tipo para la depuración del código se muestra en la figura 1.1.

```
Bombas: 5
Rangos Horarios: 3
Volumen total acumulado diario (m3): 10000 (+10000)
Caudal total minimo (m3/h): 200
Caudales (m3/h): 258 142 243 247 277
Tarifas (€/(Kw h)): 0.168 0.112 0.056
Potencias (Kw): 103 154 314 132 234
Conductividad maxima permitida (µS/cm): 2500
Conductividades (µS/cm): 1484 1923 1157 1588 2861
Volumenes máximos concedidos diarios (m3): 9245 6721 7455 2274 6462
Profundidades nivel dinámico (m): 152 186 170 270 173
Profundidades nivel dinámico máximas (m): 290 262 260 349 271 (+3)
Estatus pozos: 1 1 1 1 1

1
6 6 0 0 1000 10 6 0 3 0 0 0 0 10 5 3

Solucion: 0 1 1 0 1  0 1 1 0 1  1 1 1 1 0
Ce (€): 1887.42
Volumen total (m3): 17712 Caudal franjas (m3/h): 662 662 890 Volumenes
explotados bombas (m3): 2064 3408 5832 1976 4432 Conductividad franjas (µS/cm):
2034.31 2034.31 1493.62
```

Figura 1.1 Resultados de un experimento tipo para depuración del código del problema.

Primeramente, podemos ver la descripción y los valores típicos de las variables y sus unidades para un tamaño de problema de 5 bombas y 3 rangos horarios: potencias, tarifas, volúmenes, conductividades, etc. A continuación, nos encontramos con una primera serie de valores encabezados por un 1, que representan los valores de los parámetros del algoritmo genético número 1 (según la codificación que utilizamos en nuestro programa para referirnos a las distintas metaheurísticas). Aunque analizaremos en detalle los parámetros de las metaheurísticas en capítulos sucesivos, decir que el algoritmo genético que representan está formado por 6 individuos iniciales, de los cuales nos quedamos con los 6 (en el conjunto de referencia), que se seleccionan los 6 mejores para ser combinados como 3 parejas y que se mutará un 10% de los mismos con una intensidad de 5. Finalmente, se constituye de nuevo el conjunto de referencia con los 3 mejores (y el resto, hasta el tamaño del conjunto, con los más dispersos) y todo esto hasta un máximo de 1000 iteraciones o hasta que el valor de la función de bondad no se modifique en 10 iteraciones sucesivas. El último bloque obtenido en la salida de la ejecución, lo constituye el individuo solución (vector binario de elementos bomba), la función objetivo óptima (coste de energía eléctrica) y los valores de las

variables del sistema alcanzados. Podemos comprobar cómo la solución obtenida es factible, puesto que ningún valor sobrepasa los límites establecidos en las restricciones.
La depuración consiste no sólo en comprobar que el resultado obtenido es factible, sino en ir comprobando paso a paso y función a función que cada variable se ajusta a las especificaciones establecidas y que los cambios se realizan de acuerdo a los criterios fijados de antemano en el algoritmo.
Como conclusión de este primer análisis, decir que la implementación del algoritmo en C++ ha sido plenamente satisfactoria, obteniéndose resultados coherentes similares a los obtenidos con la implementación original. La bondad de estos resultados, nos permite dar un paso más en el estudio del problema de optimización, contemplando la posibilidad de adaptar el código a un esquema mucho más amplio de metaheurísticas y poder así optimizarlo para mejorar aún más los resultados iniciales.

Capítulo 2

METAHEURÍSTICAS PARAMETRIZADAS

En este capítulo vamos a mejorar los resultados iniciales obtenidos con el algoritmo genético. Para conseguirlo, aplicaremos un esquema parametrizado de metaheurísticas al estudio de nuestro problema.

2.1 FUNDAMENTOS DE METAHEURÍSTICAS PARAMETRIZADAS

En una primera etapa del proyecto hemos modificado el algoritmo original para ser adaptado al nuevo lenguaje de programación, hemos depurado el código, se ha procedido a la ejecución del problema básico empleando inicialmente un algoritmo genético para su resolución. Éste nos ha servido como punto de partida y para realizar las pruebas iniciales.

Nos encontramos ante un problema complejo, y para el que es necesario determinar metaheurísticas satisfactorias, lo que requiere la realización de experimentos con diversas heurísticas y variantes de cada una de ellas. En este apartado se plantea un marco común para el desarrollo de metaheurísticas para la optimización de recursos hídricos, y se analiza en particular la aplicación de algoritmos genéticos (GA), búsqueda dispersa (SS) y técnica GRASP (GR). Se utilizan para ello, como punto de partida, las ideas de metaheurísticas parametrizadas en [4,5]. Para adaptar una metaheurística a nuestro problema particular, es necesario implementar el conjunto de funciones básicas de la metaheurística, realizando un análisis del problema en cuestión. Obviamente, las diferentes combinaciones proporcionarán diferentes resultados y normalmente se intenta encontrar la mejor opción para el problema. La figura 2.1 muestra el ciclo de desarrollo en el proceso de desarrollar y adaptar metaheurísticas parametrizadas para resolver un problema [5]. Siguiendo su nomenclatura, dado el esquema de metaheurísticas (1) y un problema (2) a resolver por métodos metaheurísticos, se instancian las funciones en el esquema a algunas funciones apropiadas para el problema, y se seleccionan algunos valores de los parámetros en el esquema (3). Se realizan experimentos con diferentes combinaciones de funciones y valores de los parámetros (4), para algunas instancias del problema, y se selecciona y adapta una metaheurística satisfactoria (5) para el problema, seleccionando instancias adecuadas de las funciones y valores de los parámetros,

posiblemente realizando un análisis estadístico.

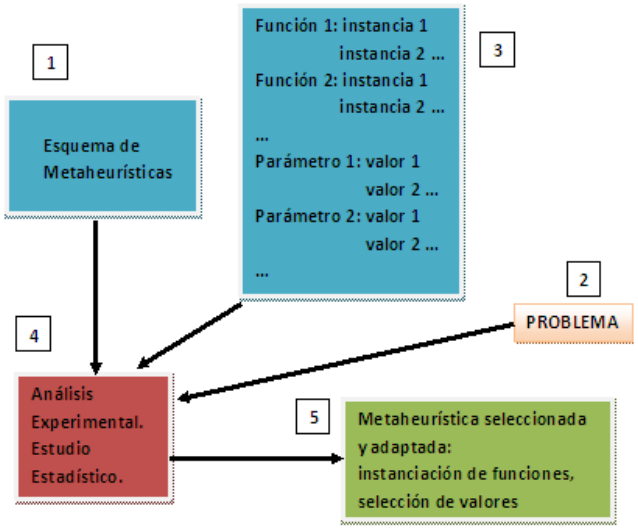

Figura 2.1 Proceso general de búsqueda de la metaheurística parametrizada óptima.

2.2 ESQUEMA PARAMETRIZADO

Como acabamos de mencionar, y ya que nuestra intención es estudiar la aplicación al problema de optimización de recursos de distintas metaheurísticas, una buena opción puede ser utilizar el esquema común a varias técnicas desarrollado a partir del esquema unificado de metaheurísticas de [12]. Un esquema general parametrizado se muestra en el algoritmo 2.1, donde S, SS, SS1 y SS2 representan los conjuntos de elementos generados en cada etapa del algoritmo y ParamX son los parámetros específicos de cada función.

Algoritmo 2.1 Esquema general parametrizado de una metaheurística.

```
Inicializar (S, ParamIni)
mientras (NO CondiciondeFin(S, ParamCondFin))
        SS = Seleccionar (S, ParamSel)
        si (|SS| > 1) SS1 = Combinar (SS, ParamCom)
        sino SS1 = SS
        SS2 = Mejorar (SS1, ParamMej)
        S = Incluir (SS2, ParamInc)
finmientras
```

El uso de un esquema común permite reutilizar las funciones de unas técnicas para otras. Por ejemplo, las funciones Inicializar y CondiciondeFin pueden ser las mismas para un algoritmo genético y para una búsqueda dispersa, y la mejora utilizada en una búsqueda dispersa puede coincidir con la que se use en un método GRASP. Además, para adaptar una metaheurística a un problema particular se consideran varias implementaciones de las funciones del esquema, y esas implementaciones pueden utilizarse en la adaptación de metaheurísticas distintas.

Adicionalmente, para realizar la adaptación, se incluyen parámetros en la metaheurística, y se varían para obtener la combinación que proporciona mejores soluciones. Algunos de estos parámetros son comunes a varias metaheurísticas, como, por ejemplo, el número de elementos del conjunto inicial, el número de iteraciones sin mejorar la mejor solución en la condición de fin...

Como hemos comentado, nos planteamos la utilización de algoritmos genéticos, búsqueda dispersa y métodos GRASP, y el conjunto de parámetros comunes para ellos puede variar si se consideran otras técnicas distintas, pero la metodología de trabajo sería la misma.

Las tres metaheurísticas consideradas corresponden a búsquedas con un conjunto en principio grande de elementos y sin mucha intensificación en la búsqueda a partir de cada elemento (genéticos), búsqueda con conjunto más reducido, cierta intensificación y mayor dispersión en la búsqueda (búsqueda dispersa), y búsqueda con intensificación a partir de ciertos elementos (GRASP). La obtención de parámetros comunes nos permitirá:

- Seleccionar una metaheurística u otra según los valores de los parámetros.
- Obtener múltiples hibridaciones seleccionando en partes distintas del esquema parámetros que den lugar a funciones propias de técnicas distintas.
- Obtener técnicas mixtas (no híbridas) con las que incluir en una metaheurística conceptos propios de otras distintas. Además, si algunos de los parámetros tienen una gradación de valores con valores extremos que corresponden a metaheurísticas distintas, tendremos toda una serie de técnicas mixtas entre esas dos metaheurísticas.

2.3 Aplicación de metaheurísticas para la resolución del problema de optimización

En el capítulo 1 describimos la codificación de los elementos y la función de bondad,

que serán comunes a todas las metaheurísticas. Cada individuo se representa por un vector con N elementos binarios. Estos elementos representan una bomba encendida o apagada en un determinado rango horario mediante codificación binaria. Así, $N = B \cdot R$.
No todas las posibles combinaciones dan lugar a sistemas factibles, pues hay unas condiciones que habrá que evaluar (Algoritmo 2.2), lo que supone un coste adicional. Como vemos en el algoritmo se evalúan cuatro condiciones y no se considera la quinta condición de pozos en funcionamiento o fuera de servicio, la cual se evalúa directamente cada vez que se genera o modifica a uno un elemento dentro del individuo.

Algoritmo 2.2 Pseudocódigo de *ElemetoFactible*.

```
Evaluar las 4 condiciones de factibilidad para cada individuo i:
    RestriccionVolumenTotal(i)
    CaudalTuberiaMinimo(i)
    ConductividadLímite(i)
    VolumenConcesionMaximo(i)
Sólo si se cumplen las 4 entonces
    el elemento considerado es factible
```

Para obtener la función de bondad correspondiente a un individuo es necesario evaluar la función del coste eléctrico (1.1), sujeta a sus correspondientes restricciones vistas en el capítulo 1 (R1 a R5).
Analizamos a continuación cada una de las funciones básicas del esquema, considerando las implementaciones para el problema que nos ocupa, variantes de las funciones y parámetros comunes para las metaheurísticas consideradas, y posibilidad de reutilización de funciones básicas:

- Inicializar

Se crean aleatoriamente elementos válidos para formar el conjunto inicial (algoritmos 2.3 y 2.4). El número de elementos (*NEIIni*) es un parámetro que se puede variar para adaptar la metaheurística al problema.

Algoritmo 2.3 Pseudocódigo de *Inicializar*.

```
GenerarConjuntoInicial(S, NEIIni)
MejorarElementos(S, NEIIni, PEMIni, IMEIni)
```

Algoritmo 2.4 Pseudocódigo de *GenerarConjuntoInicial*.

```
para i:=0 hasta NEIIni hacer
        hacer
            GenerarElemento(i)
        mientras NO ElementoFactible(i)
finpara
```

Si se conocen algunas características que tendrá la mejor combinación de elementos (por ejemplo encender aquellos correspondientes a la tarifa horaria más barata con mayor probabilidad), se pueden insertar algunos de ellos en la población inicial. El porcentaje de aleatoriedad con que se genera el valor de cada posición también puede ser un parámetro a tener en cuenta. Estos parámetros no los consideraremos en los experimentos, pues corresponden a las características del sistema, y habría que tenerlos en cuenta para adaptar las metaheurísticas al problema, pero aquí estamos interesados en parámetros cuyo valor viene determinado por las características de la metaheurística.
Por ejemplo, *NEIIni* normalmente tiene un valor alto en algoritmos genéticos, pero en búsqueda dispersa y GRASP suele ser menor, o bien es alto inicialmente para después seleccionar un subconjunto de dimensión menor (dispersa) o para seleccionar elementos sobre los que realizar mejora (GRASP). Aparece un nuevo parámetro, que será el número de elementos final tras realizar la inicialización, *NEFIni*.
En algunos métodos (dispersa y GRASP) los elementos se mejoran utilizando una búsqueda local o un avance rápido (algoritmos 2.5 y 2.6).

Algoritmo 2.5 Pseudocódigo de *MejorarElementos*.

```
para i:=0 hasta NEIIni hacer
        si nº aleatorio (entre 0 y 100) < PEMIni entonces MejorarElemento(i)
finpara
```

El parámetro *PEMIni* indica el porcentaje de elementos a mejorar del conjunto inicial. Un valor 0 puede corresponder a un genético y un valor 100 a dispersa y GRASP, pero podemos tener una gradación entre 0 y 100. La mejora puede ser más o menos intensa (considerar una vecindad más amplia o un avance rápido en mayor profundidad), lo que representamos por *IMEIni* (intensificación en la mejora de elementos en la inicialización), correspondiendo valores altos a métodos GRASP y valores menores a búsqueda dispersa.

Algoritmo 2.6 Esquema de *MejorarElemento*.

para i:=0 hasta IMEIIni hacer

MejorarElemento1. Consiste, para el individuo, en encender una bomba del rango horario con tarifa eléctrica barata por cada bomba que apaguemos de los otros rangos horarios más caros.

Comprobar si el individuo es factible. Si no lo es, se restaura el individuo original.

MejorarElemento2. Sobre el individuo resultante, dentro de un rango horario seleccionado al azar, se apagará una bomba con potencia elevada y se encenderá otra con potencia más baja.

Comprobar si el individuo es factible. Si no lo es, se restaura el individuo original.

finpara

Para la intensificación utilizamos una búsqueda local, eligiendo aleatoriamente los individuos que van a ser mejorados. El proceso de mejora consiste, en nuestro problema concreto, en dos mejoras dispuestas en serie dentro del bucle principal: en primer lugar, encender elementos bomba de rangos horarios baratos apagando a su vez elementos de rangos caros y, en segundo lugar, elegir un rango horario al azar y apagar una bomba de potencia elevada para encender a su vez otra de potencia reducida. Ambas mejoras actúan favorablemente sobre las variables de la función objetivo (1.1).

Dentro de Inicializar, generamos un conjunto o población inicial de individuos. Cada uno de ellos lo constituye un vector de unos y ceros generado aleatoriamente.

- CondiciondeFin

La condición de fin consiste en un número máximo de iteraciones (*NMFin*) o un número máximo sin mejorar la mejor solución (*NIRFin*). La condición es común para distintas metaheurísticas que trabajan iterativamente y para problemas distintos. Podemos considerar que un valor de cero en alguno de los dos parámetros correspondería a un método GRASP si los avances rápidos reiniciados se han realizado en la parte de inicialización estableciendo valores altos de *PEMIni* y *IMEIni*. Valores intermedios de estos parámetros pueden corresponder a métodos híbridos con GRASP inicial seguido de otra metaheurística.

- Seleccionar

Teniendo en cuenta la forma de seleccionar los elementos en un genético y en la búsqueda dispersa, podemos considerar que se pueden agrupar los elementos en dos conjuntos, los mejores y los peores según la función objetivo. El número de mejores será *NEMSel* y el de peores *NEPSel*, y tendremos normalmente *NEMSel* + *NEPSel* =

NEFIni. En un algoritmo genético *NEMSel* = *NEFIni,* y en la búsqueda dispersa podemos tener *NESel* = *NEPSel* = *NEFIni*/2. Tras seleccionar obtenemos el conjunto SS según el algoritmo 2.1.

- Combinar

Una vez seleccionados los elementos y la forma en que se van a emparejar, la combinación se puede hacer de distintas formas. En cualquier caso el número total de elementos a combinar será 2·(*NMMCom*+*NMPCom*+*NPPCom*), donde los tres parámetros significan respectivamente número de combinaciones de mejores con mejores individuos, número de mejores con peores y número de peores con peores. El hecho de tener diferentes funciones de combinación no implica que tengamos distintas metaheurísticas. En nuestro problema, tenemos dos individuos progenitores, el ascendiente1 y el ascendiente2, que tendrán dos descendientes según el esquema del algoritmo 2.7.

Algoritmo 2.7 Esquema de *CombinarDos* individuos.

1. Se selecciona aleatoriamente un elemento del ascendiente1 (el mismo elemento nos sirve para dividir el ascendiente2 en dos partes).
2. Cortamos ambos ascendientes por el elemento seleccionado y los combinamos de manera que el descendiente1 tenga la primera parte del ascendiente1 y la segunda del ascendiente2.
3. Con el descendiente2 procederíamos de manera similar con la diferencia de coger ahora la primera parte del genoma del ascendiente2 y la segunda del ascendiente1.
4. Si alguno de los descendientes no fuese factible (poco frecuente), lo cambiaríamos por uno de los ascendientes de los que procede. Tras combinar obtendremos el conjunto SS1 según el algoritmo 2.1.

- Mejorar

Podemos ver un esquema general de la función *Mejorar* y de sus subfunciones en los algoritmos 2.8 y 2.9, donde *MejorarElementos* sigue el esquema del algoritmo 2.5. En la búsqueda dispersa la mejora puede consistir en aplicar un avance rápido o una búsqueda local a partir de los elementos generados por combinación, y en los algoritmos genéticos se puede considerar la mutación dentro de la mejora (concretamente se muta un elemento bomba elegido al azar dentro del individuo). Podemos tener intensificación de los elementos generados o de los obtenidos por mutación, y en los dos casos se puede tener un porcentaje de elementos a tratar y un

parámetro de intensificación de la mejora, tal como en *Inicializar*. Además, la función de mejora puede ser la misma de la inicialización. Así, tenemos los parámetros *PEBMej* e *IMBMej* para el porcentaje de elementos y la intensificación de la mejora en la búsqueda local, y *PEMMej* e *IMMMej* para el porcentaje y la intensificación de mutación respectivamente.

Algoritmo 2.8 Pseudocódigo de *Mejorar*.

1. Mejorar los elementos tanto del conjunto de seleccionados (SS) como de combinados (SS1):

   MejorarElementos(SS*, NEFIni*, *PEBMej*, *IMBMej*)

   MejorarElementos(SS1, 2·(*NMMCom*+*NMPCom*+*NPPCom*), *PEBMej, IMBMej*)

2. A continuación, mutar los elementos tanto del conjunto de seleccionados (SS) como de combinados (SS1):

   Mutar(SS, *NEFIni, PEMMej, IMMMej*)

   Mutar(SS1, 2·(*NMMCom*+*NMPCom*+NPPCom*), PEMMej*, *IMMMej*)

Algoritmo 2.9 Pseudocódigo de *Mutar*.

```
para i:=0 hasta NEFIni hacer
    si nº aleatorio (entre 0 y 100) < PEMMej entonces MutarElemento(i)
    si ha mutado, entonces MejorarElemento(i)
finpara
```

En un genético puro *PEBMej* = 0, y en una búsqueda dispersa *PEBMej* = 100 y *PEMMej* = 0, pero se pueden considerar valores intermedios obteniéndose algoritmos meméticos. Por ejemplo, un genético con *PEMMej* de 50% y un valor bajo de *IMMMej* puede considerarse cercano a una búsqueda dispersa, pues la mitad de los elementos se mutan dando lugar a elementos posiblemente alejados (sin usar una función de distancia) de los óptimos. Tras mejorar se obtiene el conjunto SS2 según el algoritmo 2.1.

- Incluir

Hay varias posibilidades para realizar la inclusión, pero consideraremos que en un genético se incluyen los mejores elementos obtenidos tras los pasos anteriores, y en una búsqueda dispersa se incluye un porcentaje de los mejores y el resto los más alejados de los mejores según una función de distancia (algoritmo 2.11).

Algoritmo 2.11 Esquema de *Incluir*.

1. Incluir los *NEMInc* individuos mejores de los conjuntos anteriores en un conjunto auxiliar.
2. Ordenar por mayor dispersión respecto a los mejores según una función distancia.
3. Incluir los elementos más dispersos junto a los mejores en el conjunto referencia S.

Así, disponemos de una serie de parámetros con los que experimentar para hibridizar, mezclar y adaptar las metaheurísticas al problema, para lo que habría que tener además en cuenta otros parámetros propios de cada metaheurística. El número de parámetros se puede disminuir o aumentar, y si se incluyen otras metaheurísticas posiblemente aparecerán parámetros de otros tipos.

2.4 RESULTADOS COMPUTACIONALES

Para comprobar la viabilidad y la conveniencia de aplicar la propuesta de parametrización (de manera que se puedan aplicar metaheurísticas puras y otras formadas por hibridación), aplicamos esta técnica al problema de minimización de costes de consumo de energía eléctrica en la explotación de aguas subterráneas.

Las combinaciones de bombas en cada rango horario usadas en el estudio se generan aleatoriamente. El tamaño de cada individuo quedará establecido por el número de bombas considerado y el número de rangos horarios fijado. La complejidad del problema vendrá también determinada por la severidad de las restricciones impuestas. Así, dependiendo del volumen de agua a alcanzar, el caudal mínimo impuesto, los pozos fuera de servicio, conductividades más o menos elevadas y los requisitos de extracción máximos de cada pozo, tendremos un problema más o menos fácil de inicializar.

Los valores utilizados de cada uno de los parámetros para obtener una determinada combinación de metaheurísticas se muestran en la tabla 2.1, donde se ha omitido las condiciones de finalización, cuyos valores típicos son *NMIFin*=1000, *NIRFin*=10, para las iteraciones máximas y las iteraciones máximas sin repetición respectivamente.

		GRASP	GA	SS	GR+GA	GA+SS	GR+SS	GR+GA+SS
Inicializar	NEIIni	200	100	100	200	100	200	200
	NEFIni	1	100	20	100	50	20	50
	PEMIni	100	0	100	100	100	100	100
	IMEIni	10	0	10	10	10	10	10
Seleccionar	NEMSel	0	100	10	100	25	10	25
	NEPSel	0	0	10	0	25	25	25
Combinar	NMMCom	0	50	90	50	90	90	90
	NMPCom	0	0	100	0	100	100	100
	NPPCom	0	0	90	0	90	90	90
Mejorar	PEBMej	0	0	100	0	100	100	100
	IMBMej	0	0	5	0	5	5	5
	PEMMej	0	10	0	10	10	0	10
	IMMMej	0	5	0	5	5	0	5
Incluir	NEMInc	0	100	10	100	25	10	25

Tabla 2.1 Valores de los parámetros para las distintas combinaciones de metaheurísticas consideradas en los experimentos.

GRASP, GA y SS se refieren a parámetros que dan lugar a las metaheurísticas GRASP, algoritmos genéticos y búsqueda dispersa. GR+GA, GA+SS y GR+SS corresponden a parámetros que proporcionan en alguna medida una combinación de dos metaheurísticas. GR+GA y GR+SS son métodos GRASP seguidos de genéticos o búsqueda dispersa, con lo que son técnicas híbridas, mientras que GA+SS corresponde a una especie de mestizaje entre algoritmo genético y búsqueda dispersa, con un tamaño de conjunto de referencia intermedio, con mejora de los elementos que se generan, y con mutación. GR+GA+SS representa un GRASP seguido de una combinación genético-búsqueda dispersa del tipo anterior.

Los experimentos que se han llevado a cabo tienen la configuración que se muestra en la tabla 2.2, donde algunas variables del problema se han fijado a valores concretos razonables para el tamaño del sistema considerado y otros se han tomado aleatoriamente entre los extremos de un intervalo también de forma que representen valores reales de funcionamiento del sistema según [8].

En la mayoría de los casos el valor de la función objetivo (1.1) es muy parecido (figura 2.2). Cuanto más bajo es este valor mejor es el resultado obtenido.

	Bombas,Rangos Horarios		
	20,6	**50,3**	**50,6**
VTAD	40000 (+10000)	100000(+20000)	100000(+20000)
CTM	1000	2500	2500
CA	alea[40,300]	alea[40,300]	alea[40,300]
P	alea[30,400]	alea[30,400]	alea[30,400]
CMP	2500	2500	2500
CO	alea[100,3000]	alea[100,3000]	alea[100,3000]
VMCD	alea[1000,10000]	alea[3000,10000]	alea[3000,10000]
PND	alea[150,300]	alea[150,300]	alea[150,300]
PNDM	alea[250,350] (+3)	alea[230,350] (+3)	alea[230,350] (+3)
	Bombas,Rangos Horarios		
	150,3	**200,3**	**200,6**
VTAD	250000 (+50000)	350000 (+50000)	350000 (+50000)
CTM	8000	9000	9000
CA	alea[40,300]	alea[40,300]	alea[40,300]
P	alea[30,400]	alea[30,400]	alea[30,400]
CMP	2500	2500	2500
CO	alea[100,3000]	alea[100,3000]	alea[100,3000]
VMCD	alea[5000,15000]	alea[10000,20000]	alea[10000,20000]
PND	alea[150,300]	alea[150,300]	alea[150,300]
PNDM	alea[230,350] (+3)	alea[230,350] (+3)	alea[230,350] (+3)

VTAD	Volumen total acumulado diario (m3)
CTM	Caudal total mínimo (m3/h)
CA	Caudales (m3/h)
P	Potencias (Kw)
CMP	Conductividad máxima permitida (µS/cm)
CO	Conductividades (µS/cm)
VMCD	Volúmenes máximos concedidos diarios (m3)
PND	Profundidades nivel dinámico (m)
PNDM	Profundidades nivel dinámico máximas (m)

Tarifas eléctricas en intervalo horario (€/(Kw h)) : 0.168 0.112 0.056

donde alea [*x,y*] representa valores aleatorios en el intervalo (*x,y*)
donde *x* (+*y*) representa el valor *x* de una variable con su tolerancia *y*

Tabla 2.2 Valores de las variables de los problemas consideradas en los experimentos.

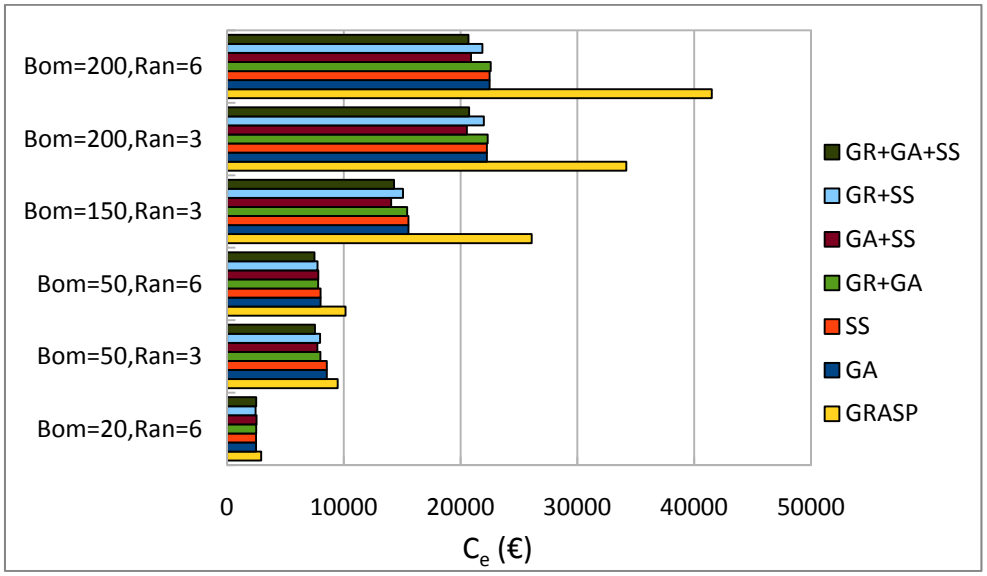

Figura 2.2 Valor de la función C_e (€) obtenido para las distintas combinaciones de metaheurísticas para distintos tamaños de problema.

Con el método GRASP los resultados son peores en todos los casos, pero su hibridación con las otras técnicas produce resultados satisfactorios.

Podemos ver cómo, con esta aproximación, se han mejorado los resultados iniciales obtenidos con algoritmos genéticos, ya que se observa que la mayoría de las metaheurísticas e hibridaciones producen funciones de bondad mejores.

Se han obtenido tiempos de ejecución similares (figura 2.3), sobre todo para tamaños pequeños y medios de problema. Para tamaños muy grandes se observa un tiempo notablemente más elevado en la ejecución híbrida de las tres metaheurísticas, salvo para GR+GA cuyos valores son del orden de los de las metaheurísticas simples. SS origina tiempos de ejecución elevados con tamaños de problema grandes. El uso de esquemas paralelos, a estudiar en los siguientes capítulos, puede contribuir a reducir estos tiempos de ejecución.

La figura 2.4 muestra el valor de la función objetivo en sucesivas iteraciones para cada uno de los métodos, para el problema con 50 bombas y 6 franjas horarias. Vemos que existen dos bloques de resultados similares: GA y GRASP+GA, por un lado, con valores más altos de coste y mayor número de iteraciones hasta el óptimo, y el resto de metaheurísticas e hibridaciones con valores de fitness más bajos y menos iteraciones hasta el óptimo.

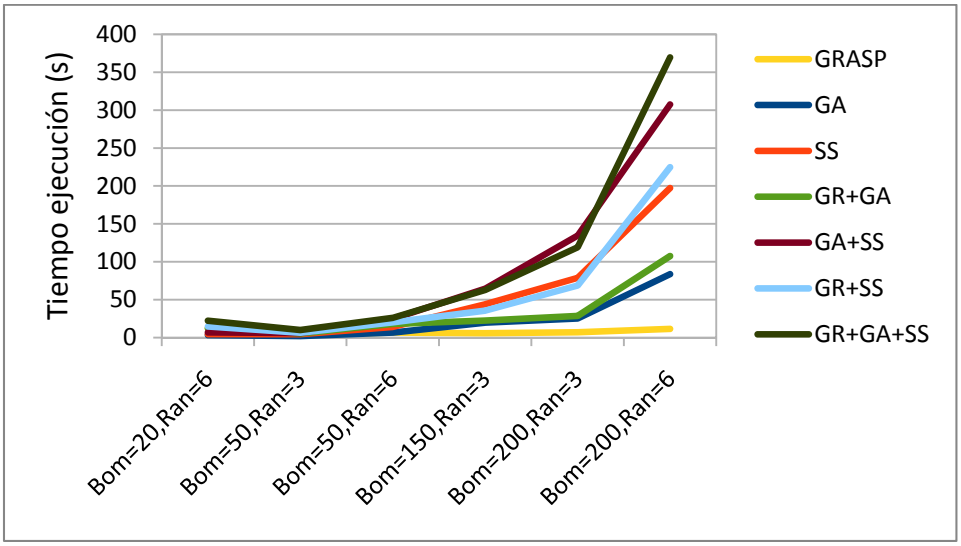

Figura 2.3 Tiempo de ejecución en segundos de las distintas combinaciones de metaheurísticas para distintos tamaños de problema.

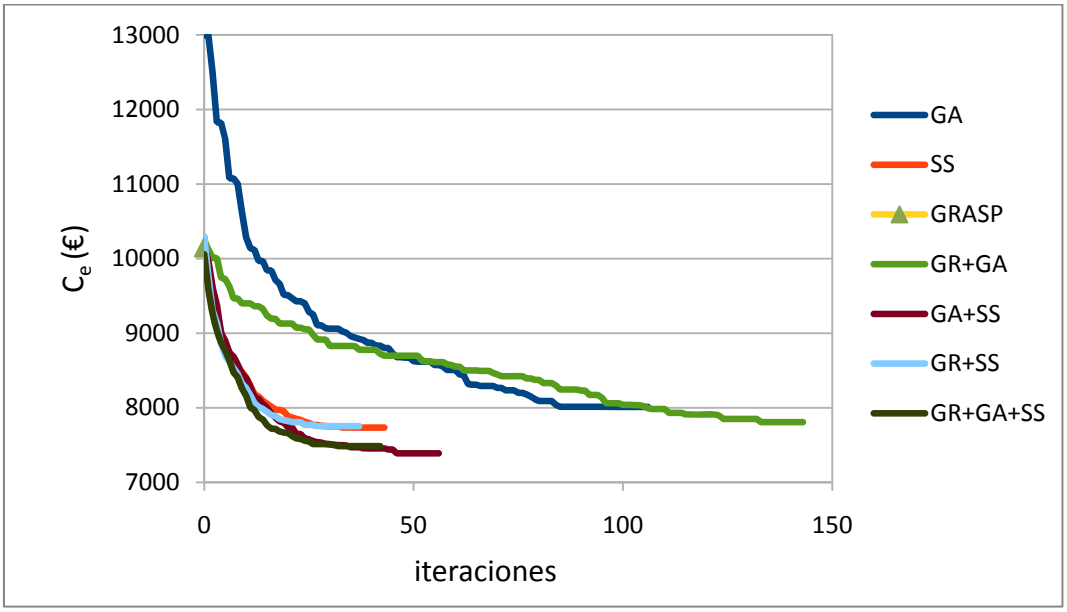

Figura 2.4 Valores óptimos obtenidos por las diferentes metaheurísticas en sucesivas iteraciones, para el problema con 50 bombas y 6 franjas horarias.

Los tiempos en las sucesivas iteraciones para el mismo problema se muestran en la figura 2.5. Los métodos híbridos requieren de un mayor tiempo de ejecución pero producen una mejora en el valor obtenido.

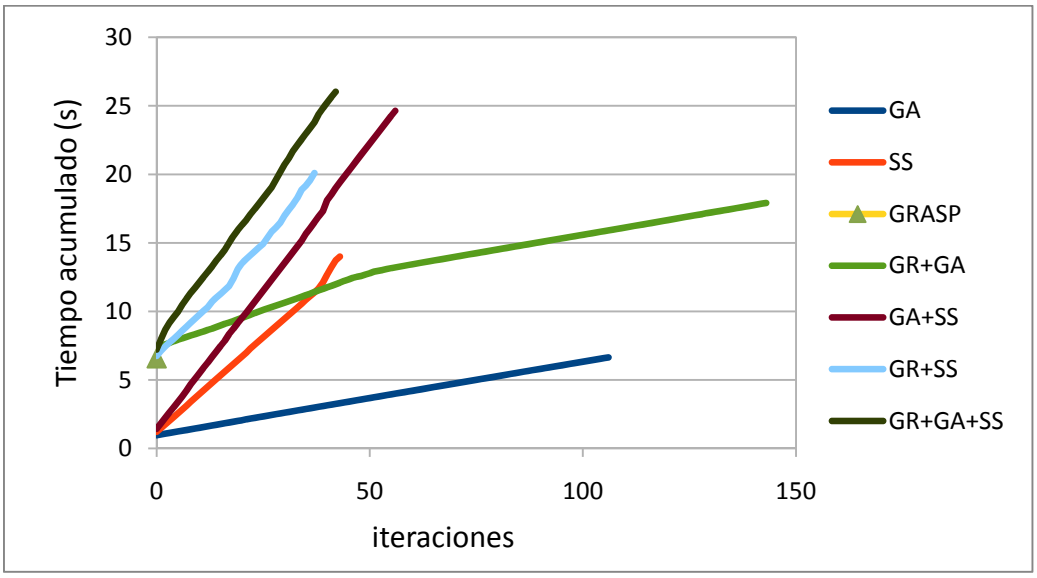

Figura 2.5 Tiempos acumulados en segundos para las diferentes metaheurísticas en sucesivas iteraciones, para el problema con 50 bombas y 6 franjas horarias.

El método que estamos tomando como búsqueda dispersa tiene un tiempo de ejecución reducido y la solución que obtiene está muy cercana a la obtenida con las hibridaciones. Observamos una mejora clara de los resultados respecto a los iniciales obtenidos con el algoritmo genético.

Indicar, para finalizar, que los resultados presentados en las figuras 2.4 y 2.5 son similares a los obtenidos para otros tamaños de problema, pero se ha elegido solo un tamaño por ser representativo del resto.

2.5 CONCLUSIONES

Podemos concluir este capítulo afirmando que el esquema común planteado para la aplicación de metaheurísticas en la optimización de costes de nuestro problema nos ha permitido experimentar satisfactoriamente con distintas metaheurísticas y combinaciones de éstas. Se ha obtenido una mejora clara de los resultados, en cuanto a función de bondad, respecto a los iniciales obtenidos con el algoritmo genético.

Capítulo 3

METAHEURÍSTICAS PARAMETRIZADAS PARALELAS

En este capítulo vamos a utilizar paralelismo para reducir el tiempo de ejecución de los experimentos sobre el esquema de metaheurísticas. Para obtener versiones paralelas de las mismas y adaptarlas a las características del sistema paralelo, se desarrollará un esquema unificado parametrizado en memoria compartida. Se analizarán también las posibles mejoras en la función de bondad al introducir paralelismo.

3.1. DESCRIPCIÓN GENERAL DEL PARALELISMO

En el capítulo 2 hemos visto que el uso de un esquema general de metaheurísticas nos permitía experimentar y desarrollar fácilmente diferentes metaheurísticas para decidir qué metaheurística, combinación/hibridación de metaheurísticas y parámetros de adaptación eran los más adecuados para resolver nuestro problema. Con este esquema, algunas de las funciones se han podido reutilizar por distintos métodos, facilitando así el desarrollo de las metaheurísticas.

Algoritmo 3.1 Esquema general parametrizado paralelo de una metaheurística.

```
Inicializar (S, NEIIni, PEMIni, IMEIni, NEFIni, ThreadsIni, Threads1Ini, Threads2Ini)
mientras (NO CondiciondeFin(S, NMFin, NIRFin))
        SS = Seleccionar (S, NEMSel, NEPSel)
        si (|SS| > 1) SS1 = Combinar (SS, NMMCom, NMPCom, NPPCom, ThreadsCom)
        sino SS1 = SS
        SS2 = Mejorar (SS1, PEBMej, IMBMej, PEMMej, IMMMej,
              Threads1Mej, Threads2Mej, Threads1Mut, Threads2Mut)
        S = Incluir (SS2, NEMInc, ThreadsInc)
finmientras
```

Cada función básica en este esquema unificado de metaheurísticas se ha podido parametrizar (algoritmo 2.1) por lo que diferentes valores de los parámetros han originado diferentes metaheurísticas, hibridación/combinación o diferentes versiones de una metaheurística particular.

El esquema parametrizado de metaheurísticas se puede usar para desarrollar el correspondiente esquema unificado parametrizado en memoria compartida (algoritmo 3.1). Para hacerlo, las funciones del esquema se paralelizan independientemente, y se identifican diferentes patrones de paralelismo en las funciones básicas del esquema que se adapta a nuestro problema de optimización de costes en la explotación de recursos hídricos. Se pueden identificar dos esquemas paralelos básicos para las funciones del algoritmo 2.1. En el primer esquema (algoritmo 3.2) los elementos de un conjunto se tratan independientemente. Para realizar nuestros experimentos, establecemos un número de hilos a usar en el bucle paralelo (threads-un-nivel). Este esquema puede ser usado, por ejemplo, cuando se combinan elementos en un algoritmo genético o cuando se genera aleatoriamente un conjunto inicial de elementos.

Algoritmo 3.2 Esquema paralelo para tratamiento independiente de elementos (esquema 1)

```
un-nivel ():
    omp_set_num_threads ( threads-un-nivel )
    #pragma omp parallel for
        bucle en elementos
            tratar elemento
```

El segundo esquema posee dos niveles de paralelismo (algoritmo 3.3), y se debe establecer un número de hilos para cada nivel. Este tipo de paralelismo aparece en las funciones de mejora o mutación, cuando se selecciona un número de elementos para mejorarlos (que implica un bucle con el número de elementos a mejorar) y cada elemento es mejorado analizando algunos elementos de su vecindario (bucle en el segundo nivel). Puede aparecer en otras partes, por ejemplo cuando se combinan elementos si la función de cruce no es simple y su paralelización puede contribuir a reducir el tiempo de ejecución.

A continuación vamos a describir el paralelismo de cada función específicamente. Para ello se ampliarán los esquemas desarrollados en el capítulo 2 para cada función introduciendo las líneas de paralelismo vistas en general en este capítulo.

Algoritmo 3.3 Esquema paralelo de dos niveles (esquema 2)

```
dos-niveles () :
      omp_set_num_threads ( threads-primer-nivel )
      #pragma omp parallel for
            bucle en elementos
                  segundo-nivel ( threads-primer-nivel )
segundo-nivel ( threads-primer-nivel ) :
     omp_set_num_threads ( threads-segundo-nivel (threads−primer−nivel ) )
     #pragma omp parallel for
           bucle en elementos
                tratar elemento
```

La nomenclatura utilizada para indicar el número de hilos en cada llamada a código paralelo es la siguiente: *THREADSIni* para los hilos en la función *GenerarConjuntoInicial*, *THREADS1Ini* y *THREADS2Ini* para el número de hilos de primer y segundo nivel de paralelismo dentro de la función *MejorarElementos* en la inicialización, *THREADSCom* para la función de combinación, *THREADS1Mej* y *THREADS2Mej* para el paralelismo anidado en la mejora, *THREADS1Mut* y *THREADS2Mut* para el paralelismo de dos niveles en la función de mutación y *THREADSInc* para los hilos paralelos en *Incluir*. Aunque algunas funciones (como las de mejora) se pueden reutilizar, es conveniente llamarlas con un número de hilos específico en cada caso debido a que la cantidad de elementos a tratar en cada función de la metaheurística puede ser diferente. Se asegura así que el paralelismo es el óptimo para cada función en cada llamada.
Solamente presentaremos las funciones sobre las que se ha escrito el código paralelo (algoritmos 3.4 a 3.9).

Algoritmo 3.4 Pseudocódigo paralelo de *GenerarConjuntoInicial*.

```
omp_set_num_threads (THREADSIni)
#pragma omp parallel for
para i:=0 hasta NEIIni hacer
     hacer
          GenerarElemento(i)
     mientras NO ElementoFactible(i)
finpara
```

Algoritmo 3.5 Pseudocódigo paralelo de *MejorarElementos*.

```
para i:=0 hasta NEIIni hacer
        si nº aleatorio (entre 0 y 100) < PEBMej entonces individuo[i]=1 (se mejorará)
        sino individuo[i]=0 (no se mejorará)
finpara
omp_set_nested(1)
omp_set_num_threads(THREADS1Ini)
#pragma omp parallel for
para i:=0 hasta NEIIni hacer
        si individuo[i]=1 entonces MejorarElemento(i)
finpara
```

Algoritmo 3.6 Esquema paralelo de *MejorarElemento*.

```
omp_set_num_threads(THREADS2Ini)
#pragma omp parallel
para i:=0 hasta IMEIIni hacer
        MejorarElemento1 (código primera mejora)
        MejorarElemento2 (código segunda mejora)
        Comprobar si individuo factible (si no lo es se restauran valores originales).
finpara
```

Algoritmo 3.7 Pseudocódigo paralelo de *Combinar*.

```
omp_set_num_threads (THREADSCom)
#pragma omp parallel for
para i:=0 hasta (NMMCom+NMPCom+NPPCom)  hacer
      CombinarDos()
finpara
```

En las funciones de mejora anidadas se ha puesto como ejemplo la llamada desde *Inicializar*, siendo los parámetros de paralelismo los correspondientes a esta función: *THREADS1Ini* y *THREADS2Ini*. Cuando la llamada a la mejora se realice desde las funciones *Mejorar o Mutar* se les pasarán como parámetros las variables que representan los hilos óptimos para estos casos. Es necesario incluir la línea *omp_set_nested(1)* para habilitar el paralelismo anidado.

Algoritmo 3.8 Pseudocódigo paralelo de *Mutar.*

```
para i:=0 hasta NEFIni hacer
        si nº aleatorio (entre 0 y 100) < PEMMej entonces individuo[i]=1 (se mutará)
        sino individuo[i]=0 (no se mutará)
finpara
omp_set_nested(1)
omp_set_num_threads(THREADS1Mut)
#pragma omp parallel for
para i:=0 hasta NEFIni hacer
        si individuo[i]=1 entonces MutarElemento(i)
        si ha mutado (y es factible), entonces MejorarElemento(i)
finpara
```

Algoritmo 3.9 Esquema paralelo de *Incluir*.

```
1. Incluir los NEMInc individuos mejores de los conjuntos anteriores en un conjunto
auxiliar.
omp_set_num_threads (THREADSInc)
#pragma omp parallel for
para i:=0 hasta (nº de elementos restantes de conjuntos referencia o combinación)
hacer
   2. Copiar el resto de individuos del conjunto referencia o combinación en otro
   conjunto auxiliar  calculando su dispersión
finpara
3. Ordenar por mayor dispersión respecto a los mejores según una función distancia.
omp_set_num_threads (THREADSInc)
#pragma omp parallel for
para i:=0 hasta (nº de elementos  mejores (o más dispersos) de conjuntos auxiliares)
hacer
   4. Incluir los elementos más dispersos  junto a los mejores en el conjunto referencia
finpara
```

Finalmente, en la función *Incluir*, se han incluido dentro de las regiones paralelas aquellos pasos que pueden ser paralelizados tanto para el conjunto de referencia como para el de elementos combinados (normalmente copias de elementos entre conjuntos o

cálculos defunciones de funciones de dispersión).
Una vez paralelizado el código se han llevado a cabo algunos experimentos para analizar el comportamiento del código paralelo y compararlo con los resultados secuenciales.

3.2. RESULTADOS COMPUTACIONALES CON EL ESQUEMA PARALELO

Los experimentos se han realizado sobre el supercomputador *BenArabí* del Centro de Supercomputación de la Fundación Parque Científico de Murcia. Está formado por dos arquitecturas claramente diferenciadas, por un lado un nodo central *HP Integrity Superdome SX2000* con 128 núcleos del procesador *Intel Itanium-2 dual-core Montvale* (1.6Ghz, 18 MB de caché L3) y 1,5 TB de memoria compartida, llamado *Ben*. Por otra parte, un clúster formado por 102 nodos de cálculo, que ofrecen un total de 816 núcleos del procesador *Intel Xeon Quad-Core E5450* (3 GHz y 6 MB de caché L2,) y 1072 GB de memoria distribuida, llamado *Arabí*. Podemos ver un resumen de ambas arquitecturas en la tabla 3.1.

Ben	
Capacidad	819 Gflops
Procesador	Intel Itanium-2 Dual-Core Montvale
Nº de núcleos	128
Memoria Compartida	1,5 TB DDR-2
Memoria Caché	18 MB L3
Frecuencia de reloj	1,6 Ghz
Discos de trabajo temporal	40 x 146 GB SAS = 5,84 TB
Arabí	
Capacidad	9,72 Tflops
Procesador	Intel Xeon Quad-Core E5450
Número de nodos	102
Nº de núcleos	816
Memoria/Nodo	32 nodos de 16 GB y 70 de 8 GB
Memoria/Core	3 MB (6 MB compartidos entre 2 núcleos)
Frecuencia de reloj	3 Ghz

Tabla 3.1 Características del Supercomputador *BenArabí*.

También se han realizado ensayos en el computador *Saturno* del Grupo de Computación

Científica y Programación Paralela de la Universidad de Murcia, cuya arquitectura podemos ver en la figura 3.1 (nodo 0). *Saturno* está formado por cuatro nodos NUMA idénticos con 8183 MB de memoria compartida y tres niveles de memoria caché: L1 (32 KB), L2 (256 KB) y L3 (12 MB). Cada nodo tiene seis cores con dos unidades de procesamiento cada uno (lo que permite la ejecución de hyperthreading).

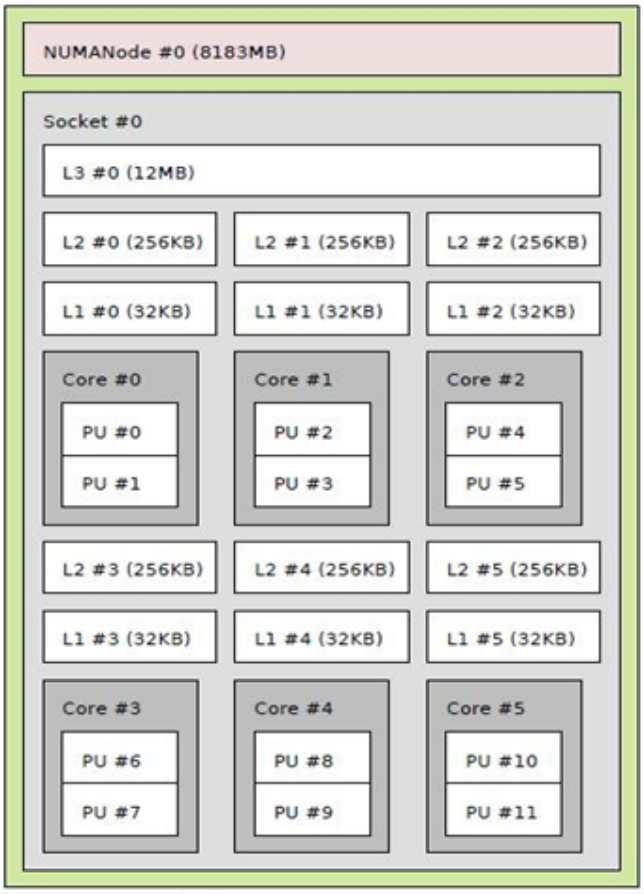

Figura 3.1 Arquitectura de un hexacore de *Saturno*.

Los experimentos persiguen exponer de manera clara el comportamiento del algoritmo paralelo y permiten sacar conclusiones acerca de la bondad de la programación realizada. Un primer grupo de gráficas (3.2 a 3.4) muestran los resultados de la ejecución en *Ben* de cada función paralela por separado representando tiempo frente a número de hilos *p*. Los resultados se han obtenido aplicando la combinación de parámetros de la metaheurística que aparece en la tabla 3.2 (se ha elegido valores intermedios de los parámetros), y para un problema con un tamaño 200,24 cuya nomenclatura y valores de variables del problema podemos encontrar en la tabla 2.2.

NEIIni	*NEFIni*	*PEMIni*	*IMEIni*	*NEMSel*	*NEPSel*	*NMMCom*
100	**50**	**50**	**10**	**25**	**25**	**90**
NMPCom	*NPPCom*	*PEBMej*	*IMBMej*	*PEMMej*	*IMMMej*	*NEMInc*
100	**90**	**50**	**10**	**50**	**5**	**25**

Tabla 3.2 Valores de los parámetros para la metaheurística considerada en los experimentos.

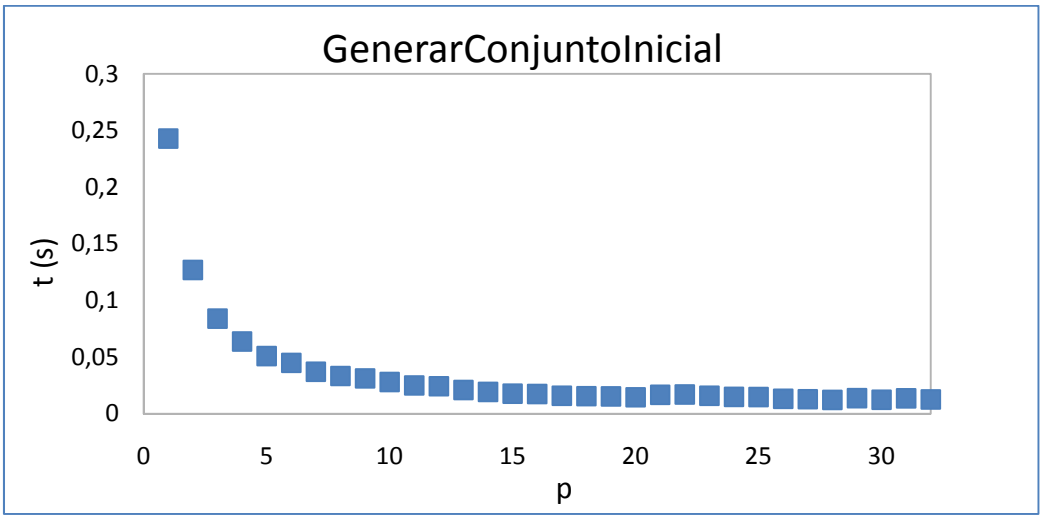

Figura 3.2 Tiempo de ejecución en segundos frente a número de hilos para la función *GenerarConjuntoInicial*.

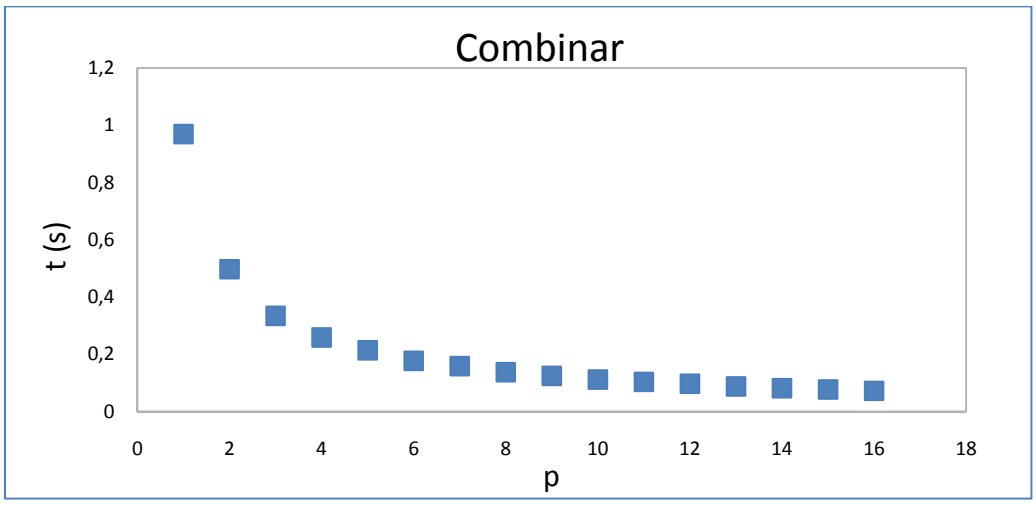

Figura 3.3 Tiempo de ejecución en segundos frente a número de hilos para la función *Combinar*.

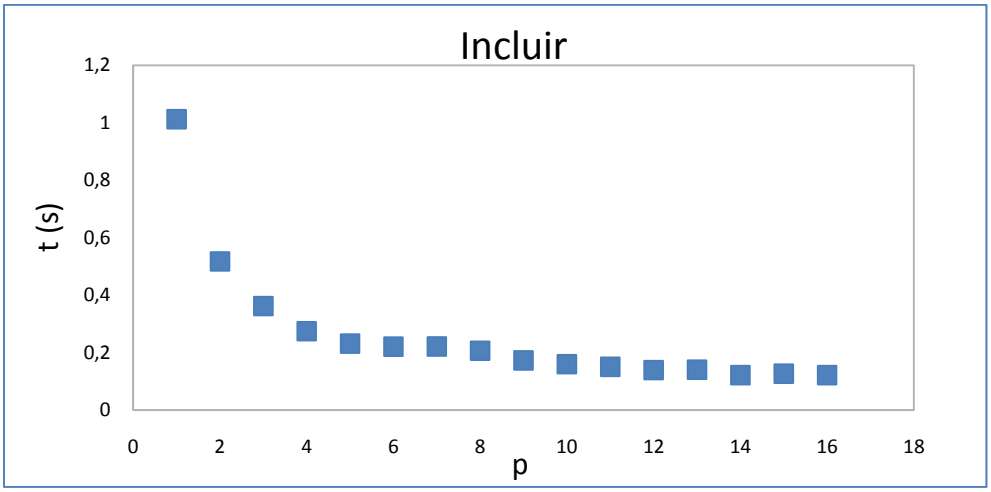

Figura 3.4 Tiempo de ejecución en segundos frente a número de hilos para la función *Incluir*.

El número de hilos máximo quedó limitado en los experimentos a las restricciones de uso del supercomputador, aunque la tendencia se aprecia igualmente en todas las gráficas. Vemos que el comportamiento de las funciones con un solo nivel de paralelismo (*GenerarConjuntoInicial*, *Combinar*, *Incluir*) es similar, con una reducción del tiempo de ejecución en función del número de hilos que sigue una curva del tipo f(x)= 1/x en todos los casos.

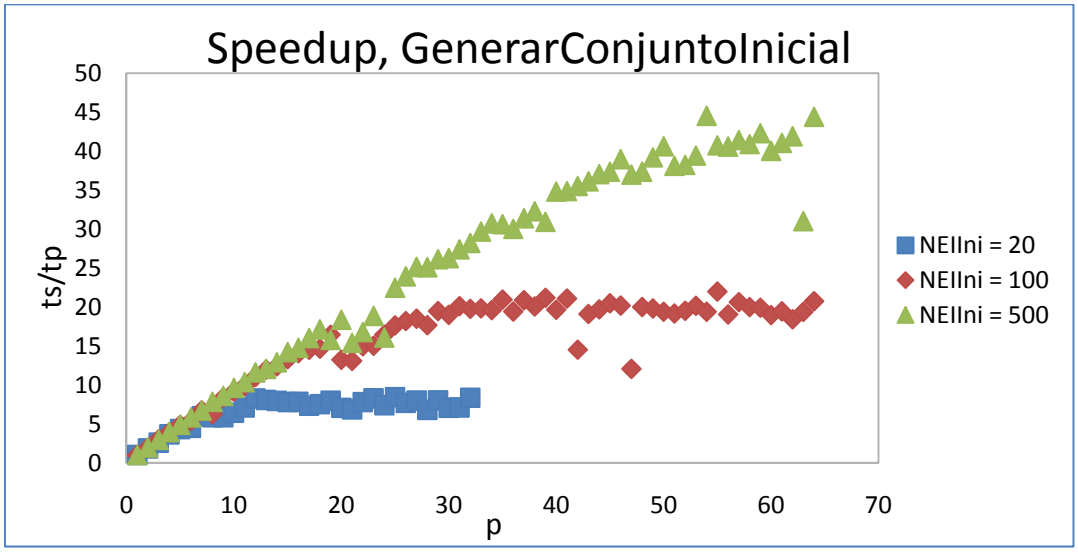

Figura 3.5 Speedup para varios tamaños de población inicial para la función *GenerarConjuntoInicial*.

Una vez visto la similitud de las funciones, vamos a comprobar la necesidad de modelarlas. Estudiamos sólo la función *GenerarConjuntoInicial,* pero las conclusiones serán las mismas para el resto de funciones. Necesitamos constatar que existe una variación del número de hilos óptimo con el tamaño de la población inicial *NEIIni*. En la figura 3.5 se presenta la variación del speedup (S_p) frente al número de hilos (p) en la función *GenerarConjuntoInicial* para tres tamaños de población inicial diferentes.

Se aprecia que existe un óptimo para el tamaño *NEIIni*=20 en torno a 12 hilos, para *NEIIni*=100 alrededor de 30 hilos y para *NEIIni*=500 se observa que el óptimo está incluso por encima de 64 hilos (no se ha podido experimentar con más hilos por limitaciones de uso del supercomputador). En cualquier caso, vemos que el número de hilos necesario para optimizar la ejecución del algoritmo es diferente para cada tamaño inicial de población. Estos resultados nos llevarán, en el capítulo 4, a modelar las funciones obteniendo los valores característicos de los parámetros que definen cada ecuación del modelo.

En cuanto a las funciones con dos niveles de paralelismo (*MejorarElementos* y *Mutar*) se han lanzado experimentos en *Ben,* con un problema de tamaño 200,6 según nomenclatura de la tabla 2.2, en los que se varían por separado los hilos del primer nivel (p_1) en series donde se mantiene constante los hilos de segundo nivel (p_2) y viceversa. Lo vemos en las figuras 3.6 a 3.9. El comportamiento de ambas funciones es similar y se analizará en detalle en el capítulo 4. En la función mutar se han tomado menos puntos experimentales debido a restricciones de uso en el supercomputador.

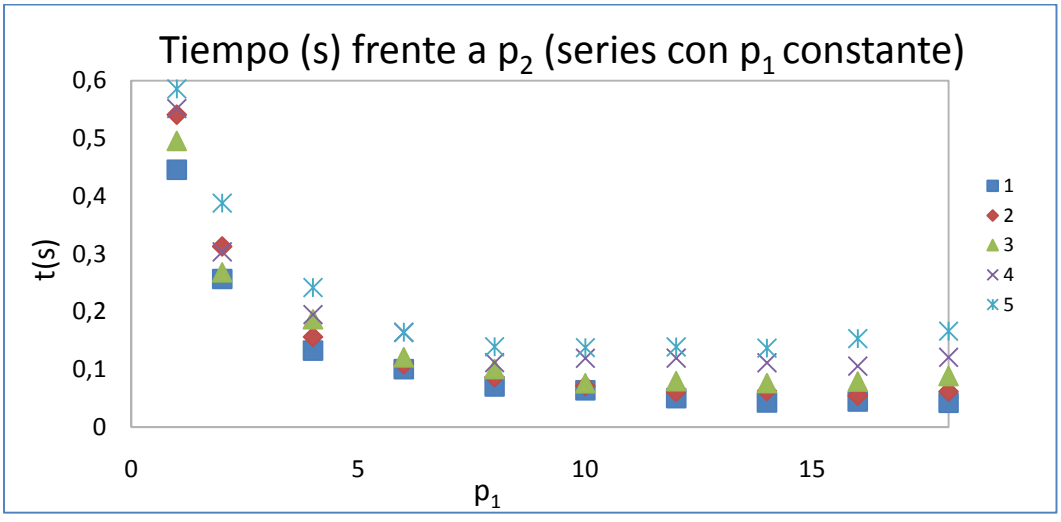

Figura 3.6 Tiempo de ejecución en segundos frente a número de hilos de primer nivel (p_1) para diferentes valores constantes de p_2 para la función *MejorarElementos*.

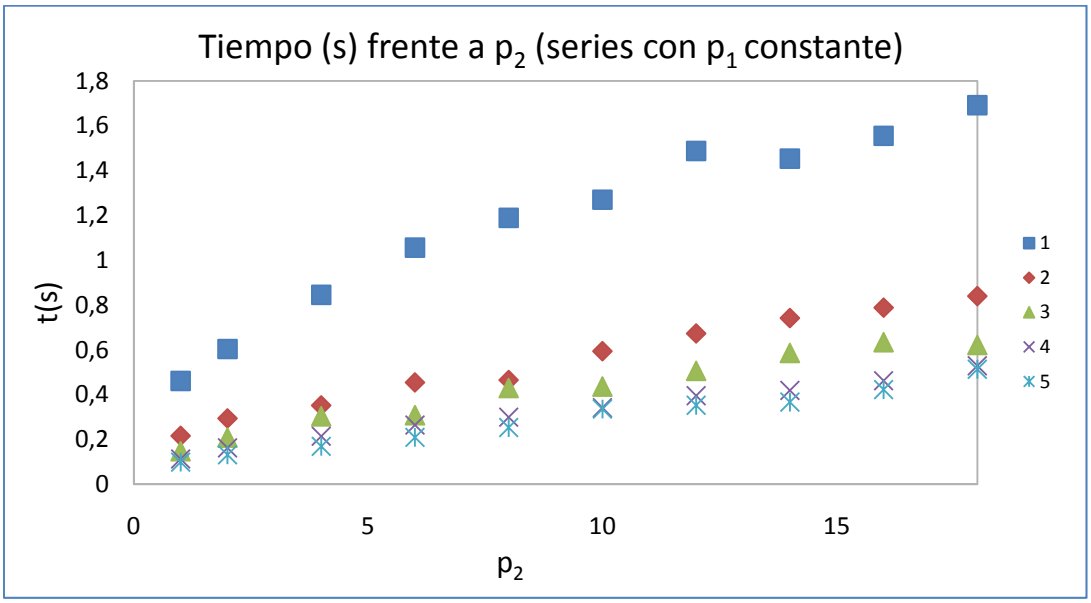

Figura 3.7 Tiempo de ejecución en segundos frente a número de hilos de segundo nivel (p_2) para diferentes valores constantes de p_1 para la función *MejorarElementos*.

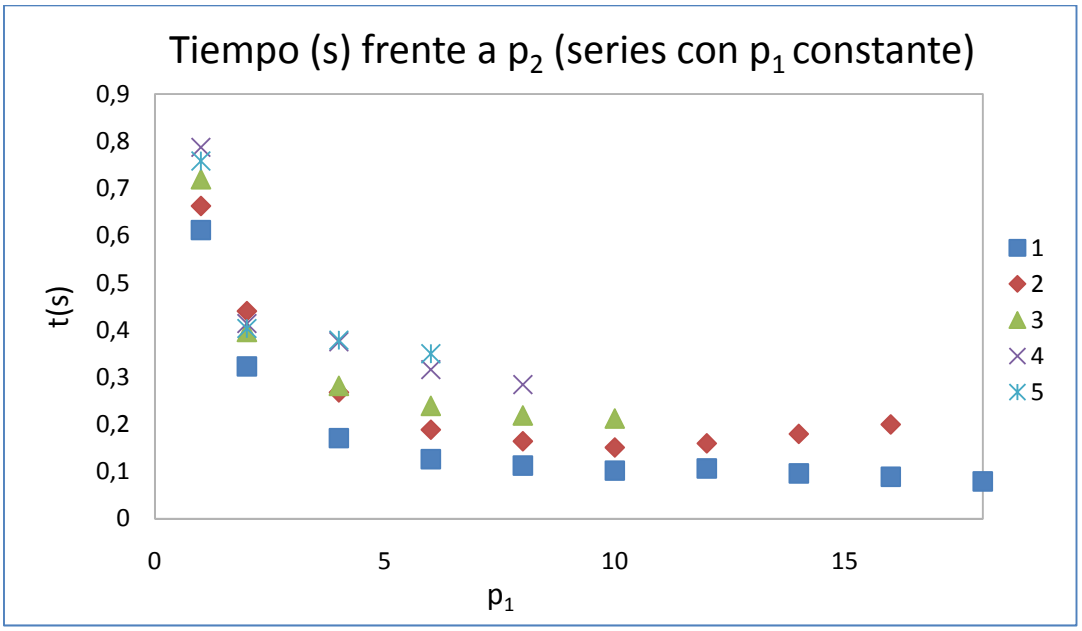

Figura 3.8 Tiempo de ejecución en segundos frente a número de hilos de primer nivel (p_1) para diferentes valores constantes de p_2 para la función *Mutar*.

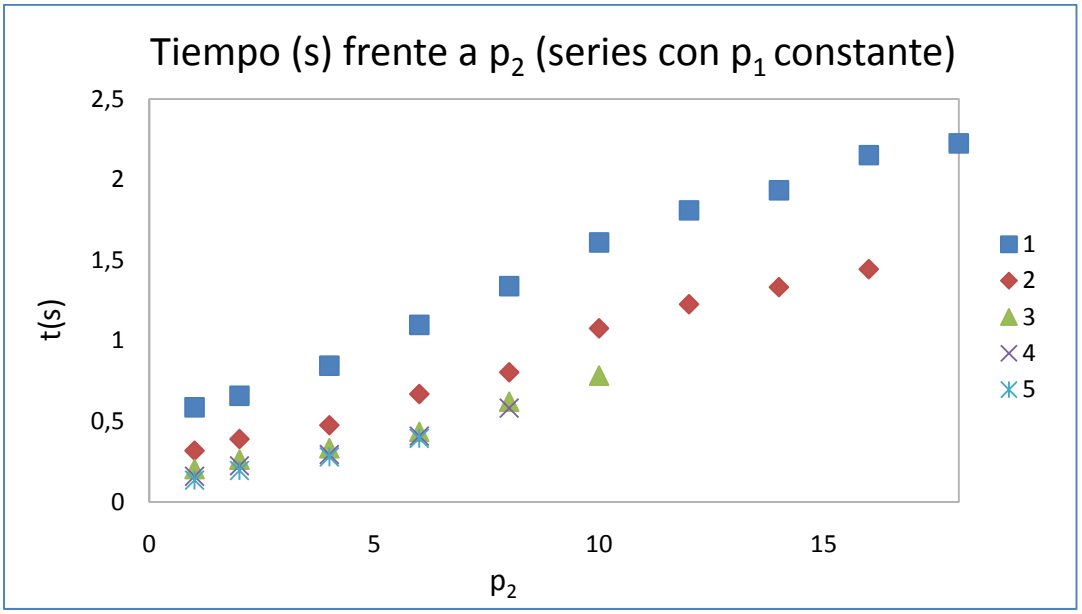

Figura 3.9 Tiempo de ejecución en segundos frente a número de hilos de segundo nivel (p_2) para diferentes valores constantes de p_1 para la función *Mutar*.

Vemos que, al variar p_1 manteniendo p_2 constante, aparece en todas las series la curva típica de la función $f(x)=A/x + B \cdot x + C$.

Por otro lado, al variar p_1 manteniendo p_2 constante, se obtienen líneas rectas en todas las series. Esto muestra que el paralelismo de segundo nivel no reduce el tiempo de ejecución sino que, tal y como se diseñó el algoritmo de mejora y mutación, solo influye linealmente con el coste de puesta en marcha de los hilos anidados del segundo nivel.

El último grupo de gráficas, evaluadas en *Ben*, recogen el tiempo de ejecución total y la función de bondad frente al número de hilos de primer y segundo nivel de paralelismo para todas las metaheurísticas y combinaciones estudiadas en el capítulo 2 para un tamaño de problema de 50,6, según nomenclatura de la tabla 2.2. Vamos a aclarar el significado de las tres componentes de las abscisas (p_1, p_1, p_2) en las gráficas 3.10 a 3.12: por ejemplo, si tenemos (8,4,2), indica que lanzamos 8 hilos en funciones con un solo nivel de paralelismo, y 4 y 2 hilos de primer y segundo nivel en funciones con dos niveles de paralelismo. Cuando sólo existe un nivel de paralelismo, no existe el problema de decidir cómo distribuir los hilos puestos en marcha, puesto que todos están en el mismo nivel. Sin embargo, las posibles combinaciones de hilos, cuando ponemos en marcha un segundo nivel anidado, puede ser grande. Para elegir el óptimo en este último caso, se realizan experimentos variando las combinaciones de hilos de primer y segundo nivel de paralelismo, manteniendo constante el número total de hilos de

ejecución.

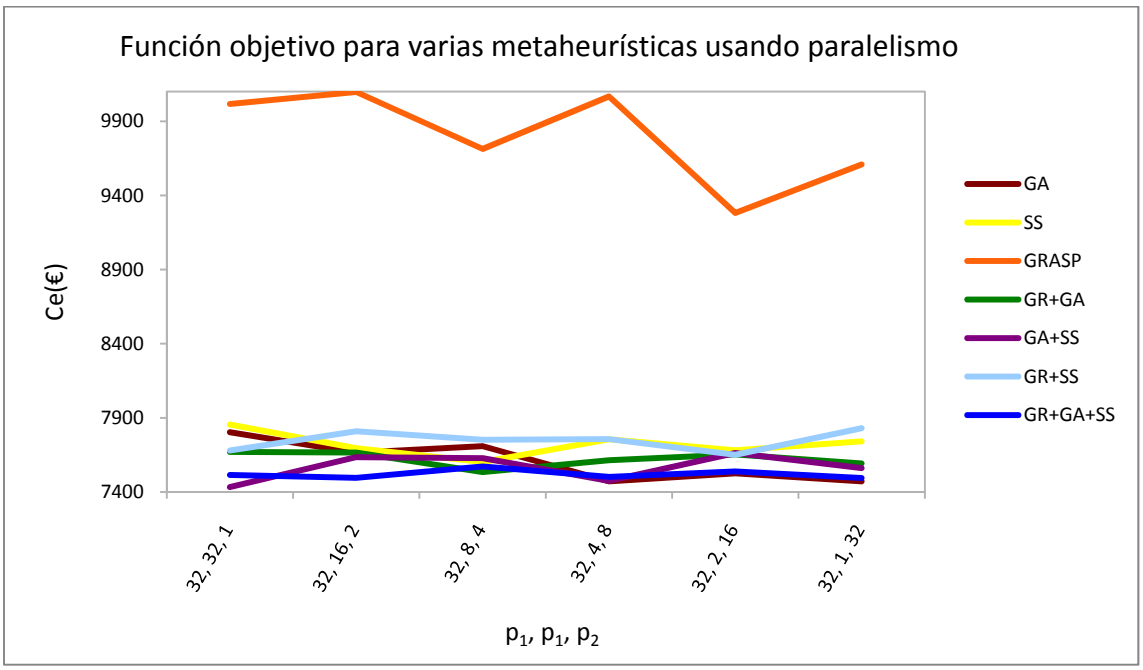

Figura 3.10 Función objetivo, $C_e(€)$, para las distintas metaheurísticas lanzando distintas combinaciones de hilos de primer y segundo nivel de paralelismo hasta un total de 32 hilos.

En la figura 3.10, podemos observar los resultados obtenidos para un número de hilos total de 32. En todas las metaheurísticas estudiadas no existe una dependencia clara entre la función de bondad y las distintas combinaciones de hilos, manteniéndose prácticamente constante en todos los casos.

Una vez visto que no existe influencia importante del paralelismo de segundo nivel sobre la función de bondad, establecemos como variable a minimizar el tiempo de ejecución. Por lo tanto, elegiremos la combinación de hilos que nos proporcione un tiempo menor, es decir, pondremos en marcha siempre un hilo de ejecución de segundo nivel.

En la figura 3.11 representamos el tiempo de ejecución en segundos frente al número de hilos de ejecución. Observamos, en todos los casos, una disminución del tiempo de ejecución al aumentar el paralelismo.

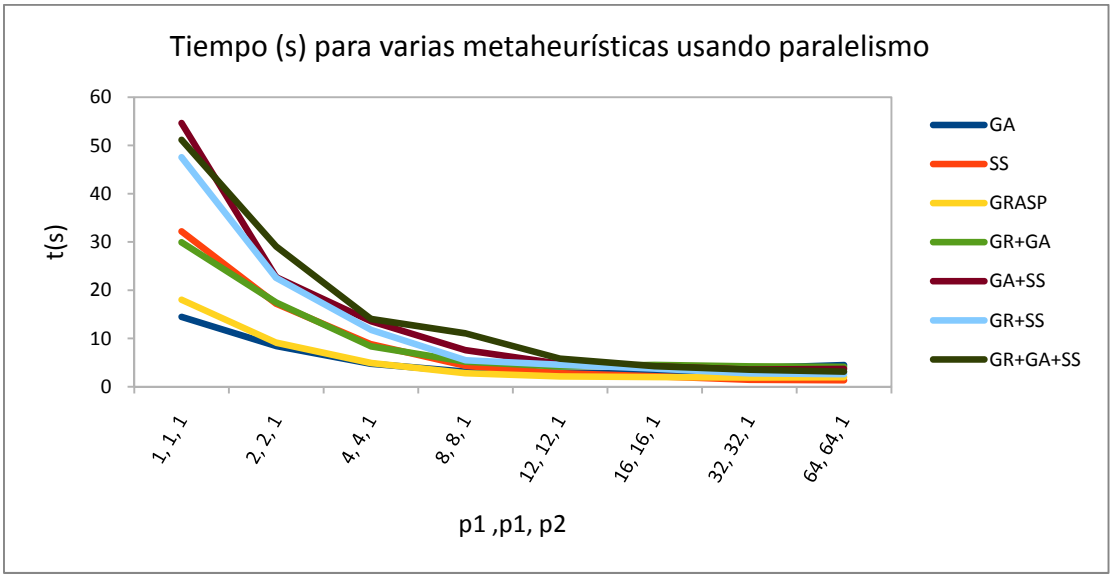

Figura 3.11 Tiempo de ejecución en segundos para cada metaheurística lanzando distinto número de hilos.

Para ver mejor el comportamiento del paralelismo, presentamos los resultados anteriores en forma de speedup en la figura 3.12. Observamos que GA, GR+GA, GA+SS y GRASP alcanzan el máximo en el intervalo de 16 a 32 hilos, mientras que SS, GR+SS y GR+GA+SS tienen el máximo por encima de 64 hilos.

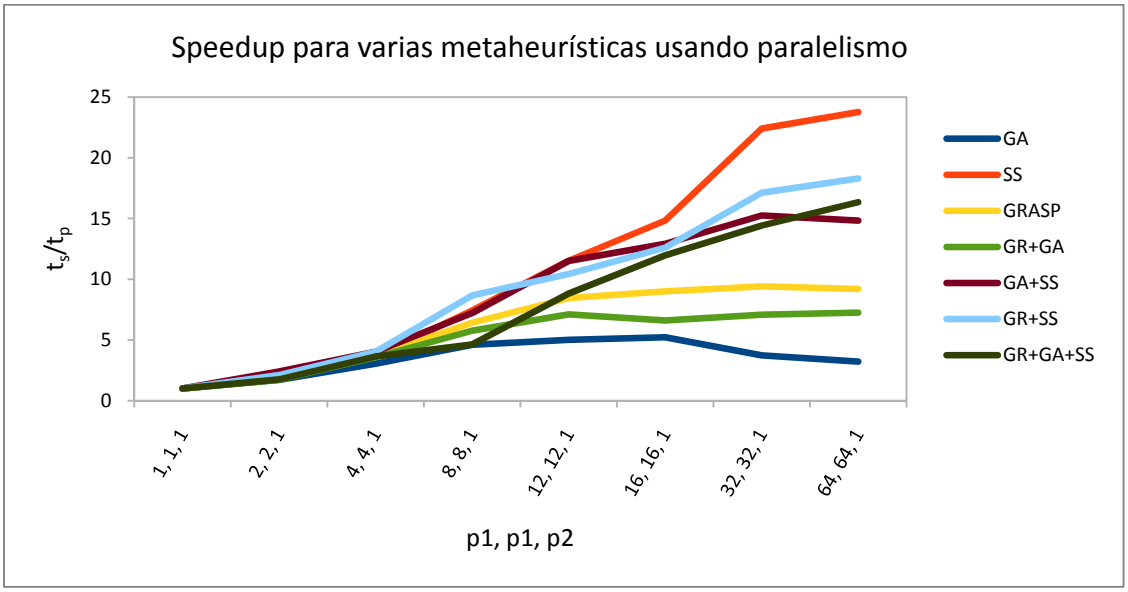

Figura 3.12 Speedup para cada metaheurística lanzando distinto número de hilos.

3.3 CONCLUSIONES

A la vista de los resultados alcanzados podemos concluir que se ha paralelizado correctamente el código secuencial de la metaheurística, mejorando así los resultados en cuanto a tiempo de ejecución respecto a la primera versión secuencial. Las mejoras de la función de bondad no son significativas al introducir paralelismo. La dependencia del número de hilos necesario para optimizar la ejecución del algoritmo con el tamaño inicial de población nos llevará, en el capítulo 4, a modelar las funciones obteniendo los valores característicos de los parámetros que definen cada ecuación del modelo.

Capítulo 4

MODELADO Y AUTOOPTIMIZACIÓN

En este capítulo vamos a modelar algunas de las funciones paralelizadas en el capítulo 3. Para ello realizaremos experimentos variando el número de hilos de primer y segundo nivel obteniendo, para cada combinación de parámetros de la metaheurística, los tiempos de ejecución alcanzados. Esto nos permitirá elegir el número de hilos óptimo como función de los parámetros de las metaheurísticas, optimizando así el esquema paralelo.

4.1 MODELADO Y AUTOOPTIMIZACIÓN EN EL ESQUEMA PARALELO

En el capítulo 3 hemos visto que el uso de un esquema unificado parametrizado en memoria compartida nos permite realizar una primera paralelización del código, analizando los tiempos de ejecución paralelos del conjunto del problema y de las funciones individualmente. Se trataba de una primera aproximación donde el número de hilos a usar en cada función se establecía sin un criterio definido. Para hacerlo, las funciones del esquema se paralelizaban independientemente, y se identificaban patrones de paralelismo en las funciones básicas del esquema.

En este capítulo vamos a dar un paso más en el estudio del sistema paralelo estableciendo el número de hilos óptimo de cada método como una función de los parámetros de la metaheurística, lo que denominamos autooptimización del esquema paralelo. Como ya se comentó en el capítulo anterior, se pueden identificar dos esquemas paralelos básicos para las funciones del algoritmo 2.1. En el primer esquema (algoritmo 4.1) los elementos de un conjunto se tratan independientemente. En este caso, se pasa a la función el conjunto de los parámetros de la metaheurística (ParamMetaheur), y se obtiene el número de hilos a usar en el bucle paralelo (threads-un-nivel) como una función de los valores de los parámetros de la metaheurística. Este esquema se puede usar, por ejemplo, cuando se combinan elementos en un algoritmo genético o cuando se genera aleatoriamente un conjunto inicial de elementos. Para las diferentes funciones, el valor óptimo de threads-un-nivel depende de los valores de los

parámetros de la metaheurística y también del coste de la función de procesamiento (el coste de la función de combinación, de la función de generación aleatoria…), que depende de la metaheurística y del sistema computacional.

Algoritmo 4.1 Esquema paralelo para tratamiento independiente de elementos (esquema 1)

```
un-nivel ( ParamMetaheur ):
    omp_set_num_threads ( threads-un-nivel ( ParamMetaheur ) )
    #pragma omp parallel for
        bucle en elementos
            tratar elementos
```

El segundo esquema posee dos niveles de paralelismo (algoritmo 4.2), y se debe determinar un número de hilos para cada nivel. El número de hilos que trabajan en el primer nivel (threads-primer-nivel) se obtiene ahora como una función de los parámetros de la metaheurística (también de sus funciones, y consecuentemente del coste de éstas en el sistema computacional). Una vez que se ha determinado este número de hilos, el número de hilos que trabajarán en el segundo nivel (threads-segundo-nivel) se obtiene como una función de los parámetros de la metaheurística y del número de hilos que trabajan en el primer nivel.

Algoritmo 4.2 Esquema paralelo de dos niveles (esquema 2)

```
dos-niveles ( ParamMetaheur ) :
     omp_set_num_threads ( threads-primer-nivel ( ParamMetaheur ) )
     #pragma omp parallel for
          bucle en elementos
               segundo-nivel ( ParamMetaheur, threads-primer-nivel )
segundo-nivel ( ParamMetaheur , threads-primer-nivel ) :
    omp_set_num_threads ( threads-segundo-nivel ( ParamMetaheur , threads −
        primer−nivel ) )
    #pragma omp parallel for
        bucle en elementos
            tratar elementos
```

Por supuesto, el primer esquema es un caso particular de este segundo esquema cuando el número de hilos en el segundo nivel se fija a uno, pero es mejor considerar dos esquemas diferentes porque el número de parámetros a obtener y como son obtenidos es diferente.

4.2 ESTUDIO DEL TIEMPO DE EJECUCIÓN DE LOS ESQUEMAS PARALELOS BÁSICOS

Para determinar los parámetros óptimos de paralelismo, se han realizado algunos experimentos sobre el supercomputador *BenArabí* y en *Saturno*.

Como se ha comentado en el apartado anterior, vamos a tener funciones con un solo nivel de paralelismo y otras con dos niveles de paralelismo anidados. Puesto que el número de funciones a paralelizar es elevado, hemos decidido, por cuestión de simplicidad y claridad de resultados, centrarnos únicamente en dos funciones tipo que representen los esquemas de paralelismo mencionados en el apartado 4.1. Concretamente, tenemos la función *GenerarConjuntoInicial* (dentro de *Inicializar*) y *MejorarElementos* como representantes de los esquemas 1 y 2 respectivamente. El resto de funciones requerirían de un estudio similar, por lo que su análisis seguiría el procedimiento que describimos para las dos funciones tipo.

En primer lugar, hemos realizado ensayos con la función *GenerarConjuntoInicial*. Presentamos los resultados obtenidos para un problema con un tamaño 200,24 cuya nomenclatura y valores de variables del problema coinciden con los del tamaño 200,6 de la tabla 2.2. Se ha elegido este tamaño por ser relativamente grande, lo que nos permite obtener resultados poco dispersos y representativos del comportamiento del sistema. Un estudio teórico del problema nos lleva a plantear que existe una relación entre el tiempo de ejecución, el número de hilos y el tamaño del conjunto inicial de individuos a generar que viene dada por:

$$t_{gen} = \frac{(k_g \cdot NEIIni)}{p} + k_p \cdot p \qquad (4.1)$$

donde t_{gen} es el tiempo de ejecución de la función *GenerarConjuntoInicial* dentro de *Inicializar*, k_g y k_p son constantes a determinar que dependen del sistema computacional donde se realizan los experimentos, *NEIIni* es el parámetro de la metaheurística que representa el número de individuos a generar aleatoriamente en la inicialización, y p es el número de hilos de ejecución. La determinación de los parámetros de la ecuación 4.1,

nos permitirá predecir el tiempo de ejecución para cualquier número de hilos lanzados y, por tanto, nos ayudará a predecir el paralelismo óptimo para un determinado tamaño inicial *NEIIni*.

Se ha abordado el problema en dos partes. Primeramente, hemos determinado el valor de k_g secuencialmente utilizando:

$$t_{gen} = k_g \cdot NEIIni \tag{4.2}$$

donde hemos considerado que la k_g aquí obtenida es válida para el caso paralelo por ser característica del sistema, independientemente de los hilos que estemos ejecutando. Primeramente, realizamos las ejecuciones en *Ben*. Como vemos en la figura 4.1, representando valores del tiempo de ejecución secuencial frente a *NEIIni*, obtenemos el valor de la pendiente de la recta que coincide con k_g. La recta es la obtenida al realizar el ajuste de mínimos cuadrados de los datos experimentales obtenidos y que arroja un valor de coeficiente de regresión bueno cercano a uno. A partir de la pendiente, tenemos, $k_g = 2.38 \cdot 10^{-3}$ s.

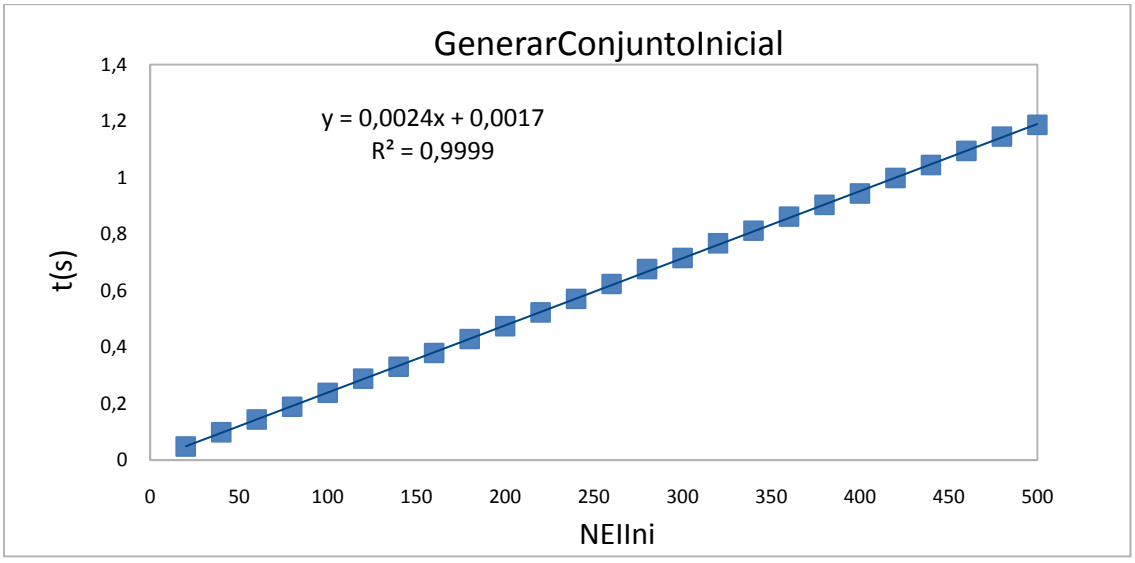

Figura 4.1 Tiempo de ejecución en segundos secuencial para distintos valores del parámetro *NEIIni*.

Una vez determinado el valor de k_g, nos disponemos a estimar el valor de la otra constante característica del sistema, k_p. Para ello, al igual que hemos hecho anteriormente, vamos a realizar un ajuste por mínimos cuadrados de los datos

experimentales, en este caso analizando el tiempo de ejecución frente a número de hilos. Además, tendremos que realizar series de experimentos paralelos fijando el valor de *NEIIni* y usando el valor de k_g obtenido en la versión secuencial:

$$k_p = \frac{\sum_{i=1}^{n} \left(p_i \cdot t_{gen,i} - k_g \cdot NEIIni \right)}{\sum_{i=1}^{n} p_i^2} \tag{4.3}$$

Presentamos a continuación los resultados experimentales de speedup (*Sp*), calculado como tiempo secuencial dividido entre tiempo paralelo, frente a número de hilos para valores de *NEIIni* de 20, 100 y 500 (figuras 4.2 a 4.4). Presentamos en la misma gráfica los valores teóricos, una vez fijadas las constantes k_g, k_p y *NEIIni* de la ecuación 4.1.

Al comienzo de la sección hemos planteado los esquemas paralelos de la metaheurística y hemos considerado que debía existir una relación entre el número de hilos de ejecución y los parámetros característicos de la función considerada en la metaheurística. Concretamente estamos analizando la función *GeneralConjuntoInicial* y el único parámetro *NEIIni* que nos da el tamaño de la población inicial. Hemos incluido en la misma gráfica, junto a los resultados de cada *NEIIni*, los speedup calculados usando los valores de k_p obtenidos para los otros dos tamaños considerados. De esta manera se puede comprobar si el modelo propuesto predice el comportamiento para cualquier tamaño de la población.

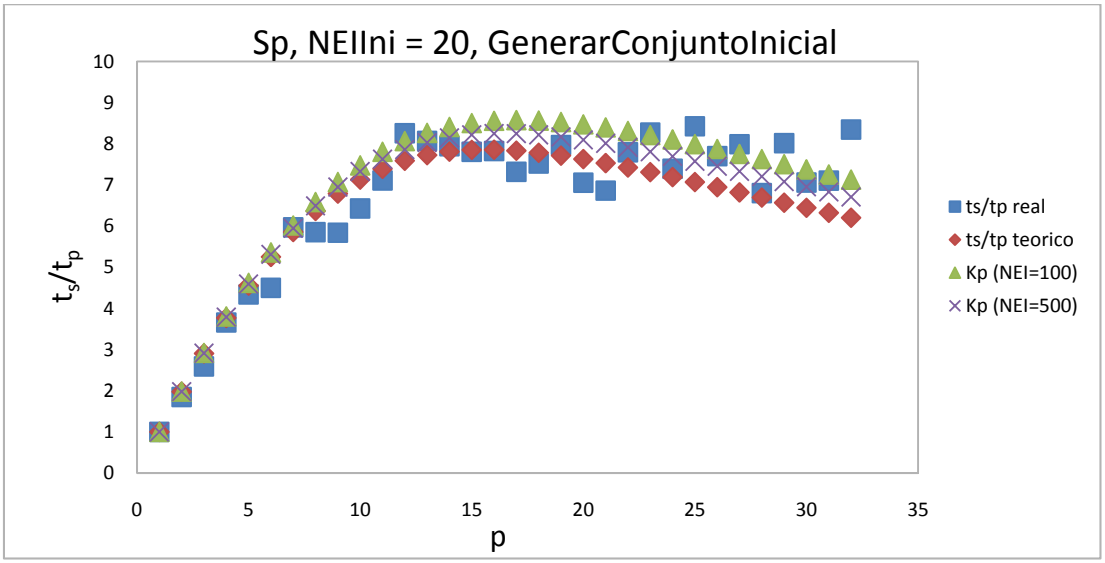

Figura 4.2 Speedup real y teórico frente a número de hilos para *NEIIni*=20. Adjunto, Speedups calculados con k_p correspondientes a otros *NEIIni*.

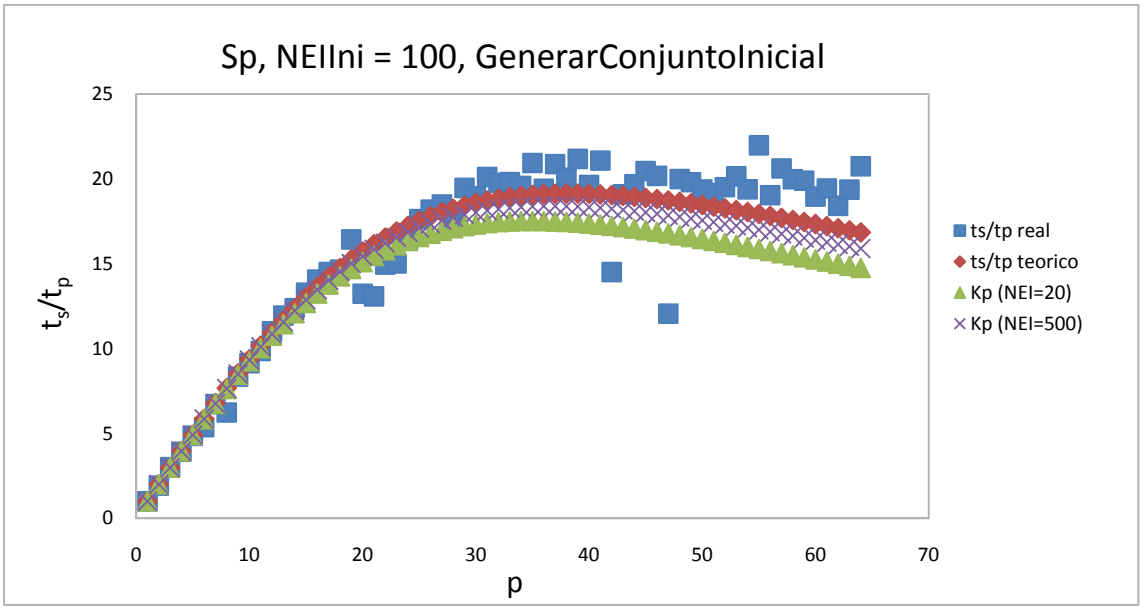

Figura 4.3 Speedup real y teórico frente a número de hilos para *NEIIni*=100. Adjunto, Speedups calculados con k_p correspondientes a otros *NEIIni*.

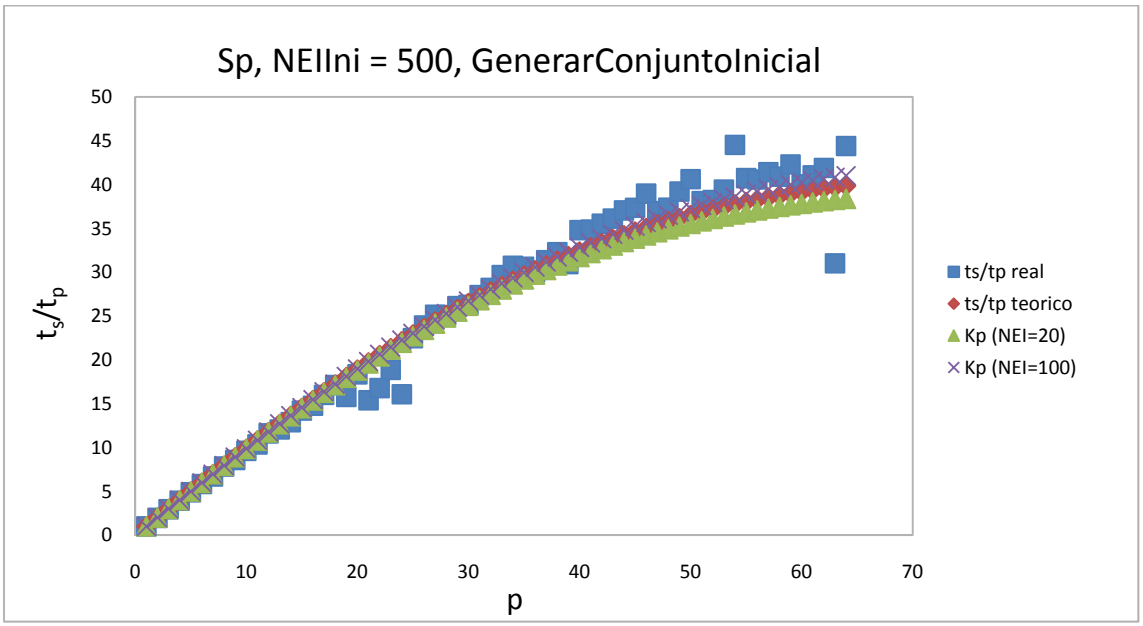

Figura 4.4 Speedup real y teórico frente a número de hilos para *NEIIni*=500. Adjunto, Speedups calculados con k_p correspondientes a otros *NEIIni*.

Se aprecia que el ajuste teórico es aceptable en todos los casos obteniéndose valores de k_p muy parecidos (tabla 4.3).

NEIIni	k_p
20	1,94E-04
100	1,63E-04
500	1,76E-04

Tabla 4.3 Valores de k_p para distintos tamaños iniciales de población *NEIIni*.

Podemos ver que los ajustes realizados con constantes k_p obtenidas para tamaños *NEIIni* distintos se aproximan bastante a los obtenidos con el valor propio del tamaño considerado. En la tabla 4.4 tenemos datos de speedup ($Sp_{opt.}$) y número de hilos óptimo ($p_{opt.}$) obtenidos de los experimentos y los ajustes. Los valores teóricos se dividen a su vez en tres filas correspondientes a datos calculados utilizando constantes k_p obtenidas de los ajustes hechos para cada tamaño *NEIIni*.

		NEIIni		
		20	100	500
$Sp_{opt.}$ (real)		**8,4**	**22**	**44**
$Sp_{opt.}$ (teórico)	20	**7,9**	**18**	**38**
	100	**8,6**	**19**	**41**
	500	**8,2**	**18**	**40**
$p_{opt.}$ (real)		**25**	**55**	**64**
$p_{opt.}$ (teórico)	20	**16**	**35**	**64**
	100	**17**	**38**	**64**
	500	**16**	**37**	**64**

Tabla 4.4 Speedup y número de hilos óptimos para distintos tamaños iniciales de población *NEIIni*. Valores experimentales (reales) y obtenidos del ajuste (teóricos).

Para un tamaño *NEIIni* = 20, se observa que el número óptimo de hilos teórico está en torno a 16 alejándose moderadamente del óptimo real de 25. Sin embargo, los Speedups predichos teóricamente son muy cercanos al real (alrededor de 8,4). Para un tamaño inicial de 100, ocurre algo similar, obteniéndose un p_{opt} *(teórico)* de 35 a 38, algo alejado del valor real de 55. No obstante, los Speedups reales y teóricos están muy próximos entorno a 20. Finalmente, decir que el análisis para un tamaño de población de 500 quedó limitado a 64 hilos por problemas de disponibilidad del supercomputador.

A pesar de ello, se observa que para el número máximo de hilos de ejecución, los speedup son parecidos con valores en torno a 42.
Podemos modelar el sistema utilizando un valor medio de k_p obteniendo el óptimo de hilos de ejecución para cualquier tamaño según la ecuación siguiente:

$$p_{opt.} = \sqrt{\frac{k_g}{k_p} \cdot NEIIni} \qquad (4.4)$$

Tenemos:
$k_g = 2.38 \cdot 10^{-3}$ s. y $k_p(medio) = 1{,}77 \cdot 10^{-4}$ s.

por tanto, $p_{opt.} = 3{,}66 \cdot \sqrt{NEIIni}$

Dando valores al parámetro *NEIIni*, obtenemos los valores teóricos de la tabla 4.5.

	NEIIni		
	20	100	500
$p_{opt.}$	**16**	**37**	**82**
$Sp_{opt.}$	**8,6**	**19**	**41**

Tabla 4.5 Número de hilos y speedup óptimos predichos por el modelo para la función *GenerarConjuntoInicial*.

Podemos confirmar que los valores óptimos obtenidos por el modelo predicen bien el comportamiento real del sistema y son similares a los teóricos obtenidos en la tabla 4.4. Para un tamaño de 500, el número 82 de hilos predicho por el modelo indica, como era de esperar, que no es suficiente con 64 hilos para alcanzar el óptimo.
Finalmente, como aplicación de los resultados alcanzados en esta parte, podríamos diseñar y programar una función que asigne automáticamente el número de hilos a lanzar en función del tamaño generado de población inicial. Esta autooptimización del código es ampliable al resto de funciones tanto de un nivel como de dos niveles de paralelismo.
Adicionalmente, se ha representado la eficiencia del sistema, definida como tiempo secuencial (t_s) dividido entre tiempo paralelo por número de hilos ($t_p \cdot p$), para los distintos tamaños iniciales (figura 4.5).

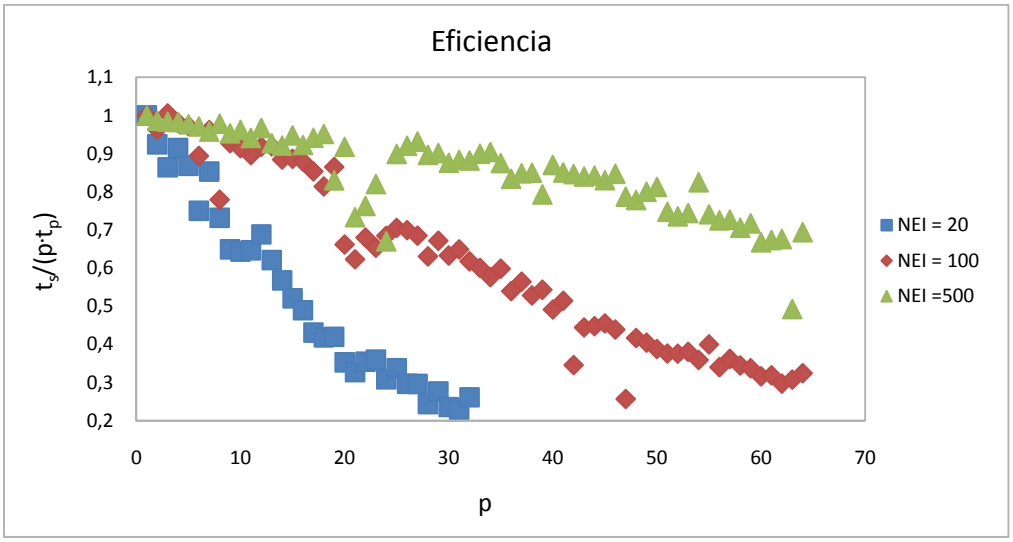

Figura 4.5 Eficiencia frente a número de hilos.

Hasta aquí, los resultados obtenidos en *Ben* para la función *GenerarConjuntoInicial*. Vamos a presentar, a modo de comparación, los resultados obtenidos en *Arabí* y en *Saturno*. No vamos a entrar en el modelado en estos dos sistemas.

Primero, los resultados obtenidos en *Arabí*. Como vemos en la figura 4.6, representando valores del tiempo de ejecución secuencial frente a *NEIIni*, obtenemos el valor de la pendiente de la recta que coincide con $k_g = 2.81 \cdot 10^{-4}$ s.

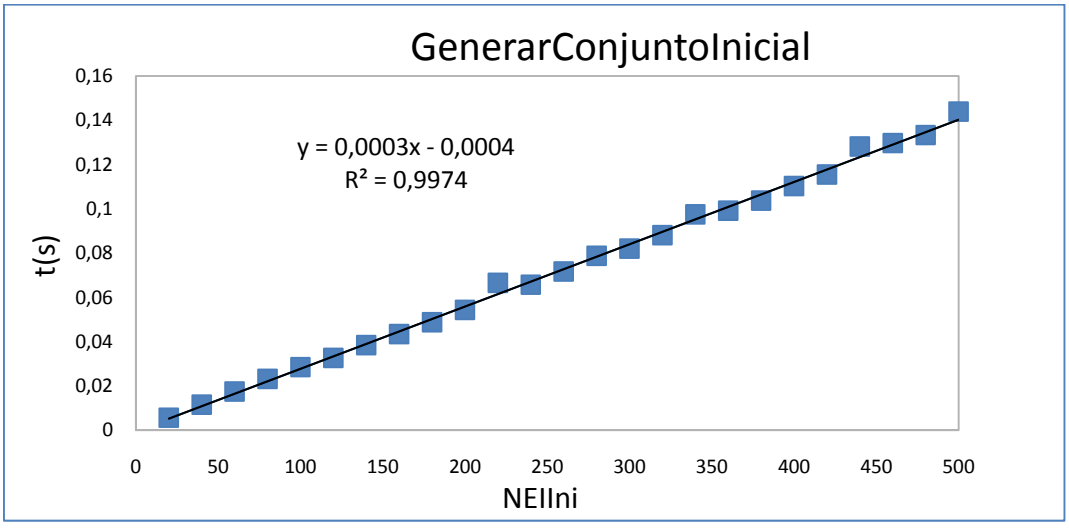

Figura 4.6 Tiempo de ejecución en segundos secuencial para distintos valores del parámetro *NEIIni* en *Arabí*.

En la figura 4.7 presentamos resultados de tiempo de ejecución frente al número de hilos para un *NEIIni* =20. Se observa un salto pronunciado en la tendencia de la curva al sobrepasar p = 8 hilos. Esto es debido a la arquitectura de *Arabí* formada por nodos de 8 cores cada uno. Cuando ejecutamos más de 8 hilos la carga se balancea entre los diferentes nodos, con lo que el tiempo de ejecución disminuye bruscamente. Debido a esta restricción en la arquitectura, se modelará en *Arabí* con un máximo de 8 hilos en paralelo.

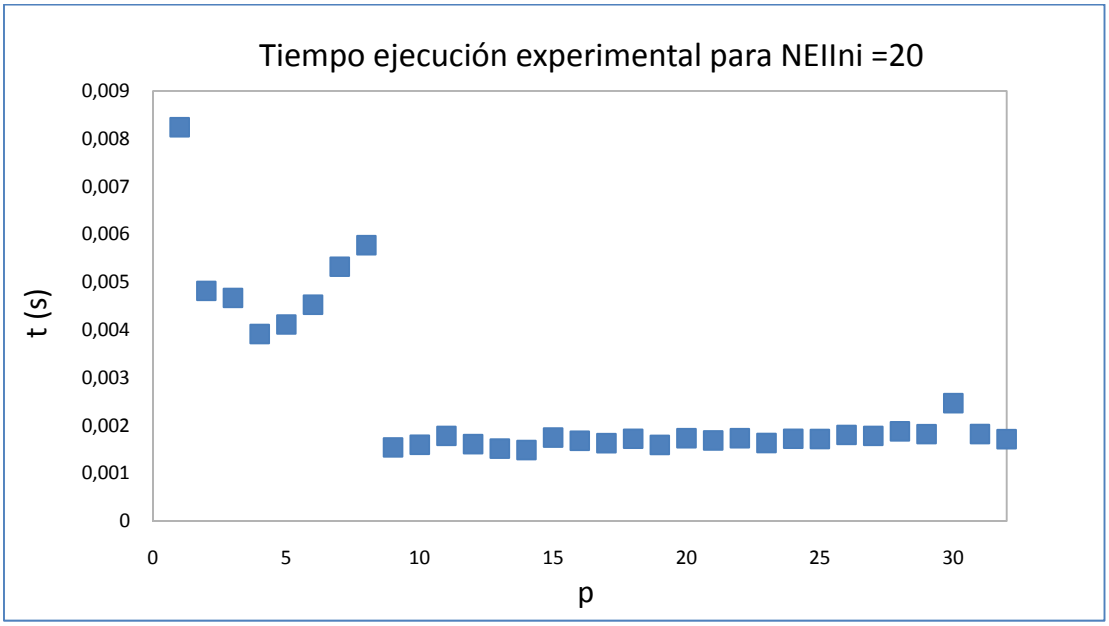

Figura 4.7 Tiempo de ejecución en segundos para *NEIIni* = 20 en *Arabí*.

Presentamos en las figuras 4.8 a 4.10 los resultados experimentales y teóricos del speedup frente a número de hilos para valores de *NEIIni* de 20, 100 y 500 con un máximo de 8 hilos en *Arabí*. Y el resumen de los valores de las constantes alcanzados lo vemos en la tabla 4.6.

NEIIni	k_p
20	6,55E-04
100	1,98E-04
500	2,73E-03

Tabla 4.6 Valores de k_p para distintos tamaños iniciales de población *NEIIni*.

En este caso los valores de k_p se diferencian bastante entre sí. No sucedía esto en *Ben* donde eran prácticamente iguales para los tres tamaños estudiados. En este caso el modelado no es tan bueno puesto que el valor de k_p medio obtenido no permitiría ajustar el comportamiento para otros tamaños con precisión.

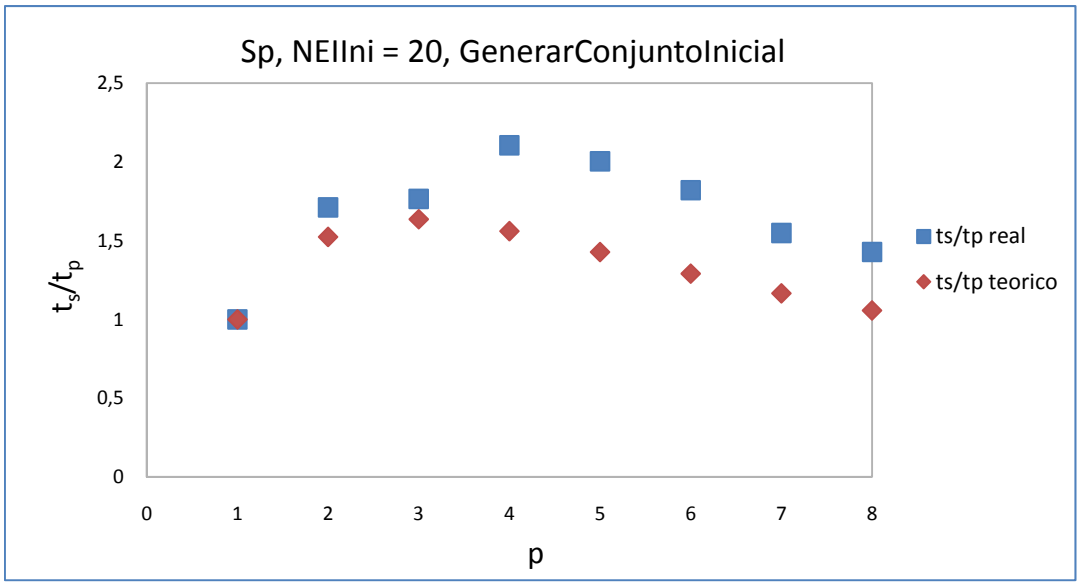

Figura 4.8 Speedup real y teórico frente a número de hilos para *NEIIni*=20.

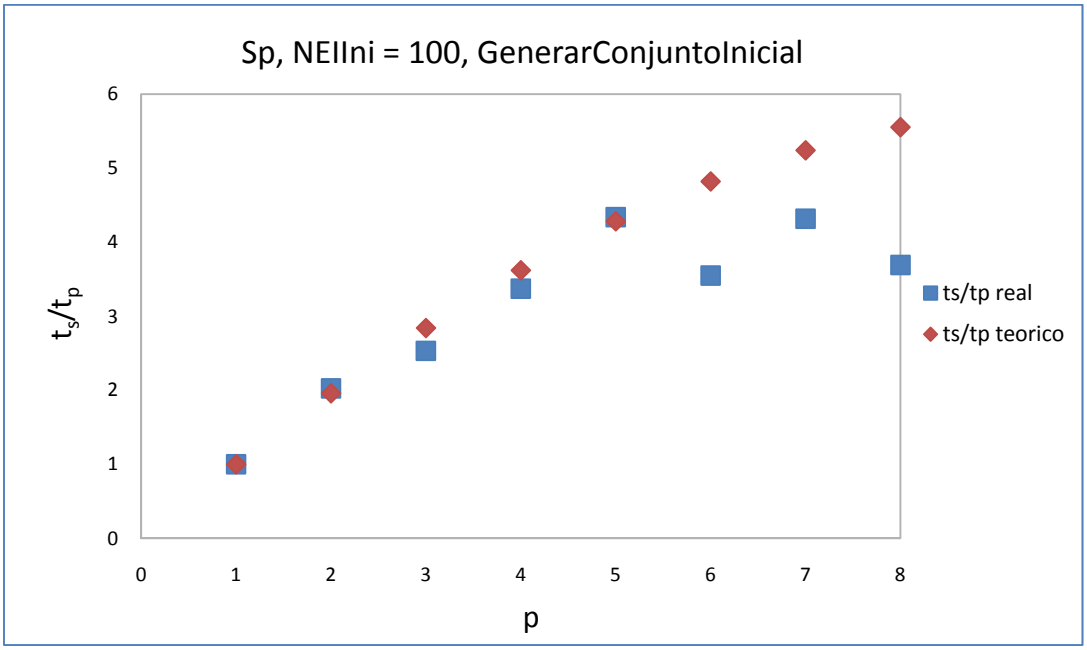

Figura 4.9 Speedup real y teórico frente a número de hilos para *NEIIni*=100.

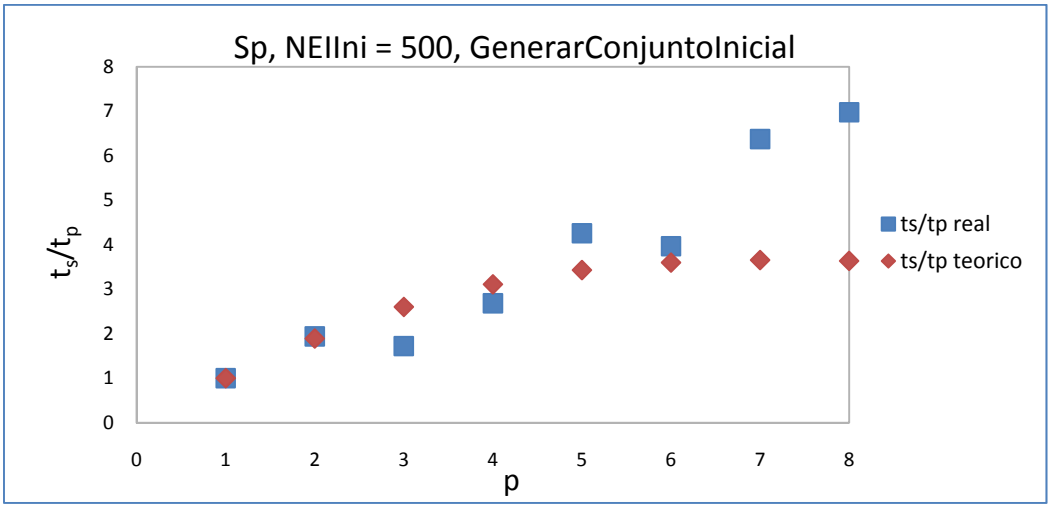

Figura 4.10 Speedup real y teórico frente a número de hilos para *NEIIni*=500.

Por último presentamos los resultados obtenidos en *Saturno* (figuras 4.11 a 4.15). Primero los experimentos en secuencial, al igual que en las otras arquitecturas, para obtener k_g. A partir de la pendiente, tenemos, $k_g = 2.65 \cdot 10^{-4}$.

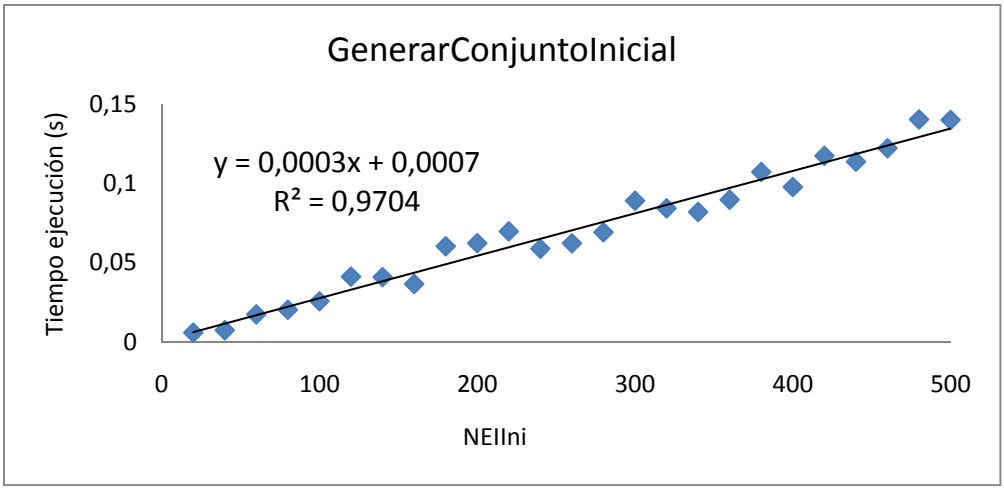

Figura 4.11 Tiempo de ejecución secuencial en segundos para distintos valores del parámetro *NEIIni*.

Una vez determinado el valor de k_g, nos disponemos a estimar el valor de la otra constante característica del sistema, k_p. Una primera serie de experimentos se realizó para determinar el comportamiento del computador en cuanto a hyperthreading. La figura 4.12 muestra los resultados alcanzados en cuanto a tiempo de ejecución frente a

número de hilos lanzados para un *NEIIni*=500, con un máximo de 48 hilos. Puesto que el computador dispone de 24 núcleos, podemos ver el efecto del hyperthreading.
Se observa que al aumentar el número de hilos por encima de 24, el hyperthreading no se comporta bien en esta rutina observándose fluctuaciones debido a los tiempos de ejecución pequeños. Este resultado nos lleva a imponer un límite superior de 24 núcleos para realizar los experimentos paralelos.

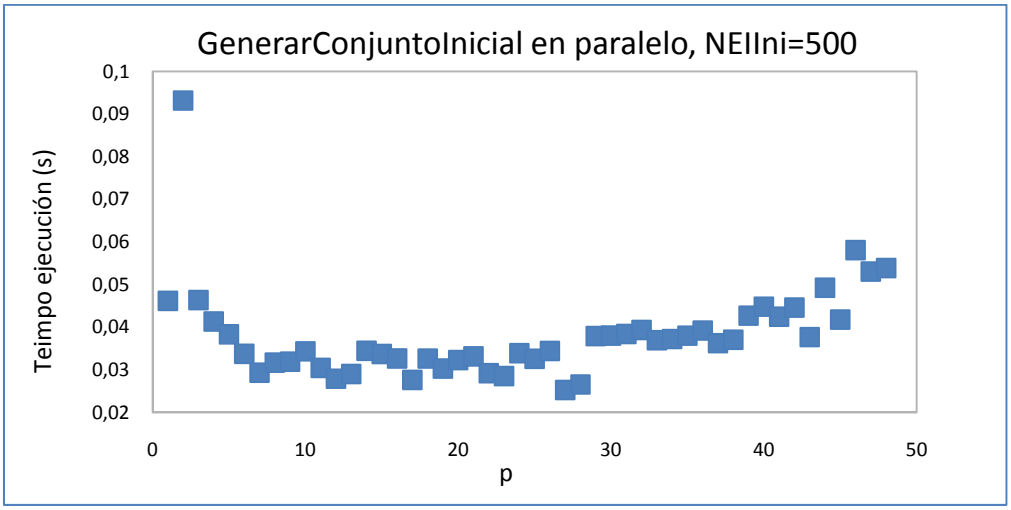

Figura 4.12 Tiempo de ejecución en segundos frente a número de hilos para *NEIIni*=500 (hasta un máximo de 48 hilos).

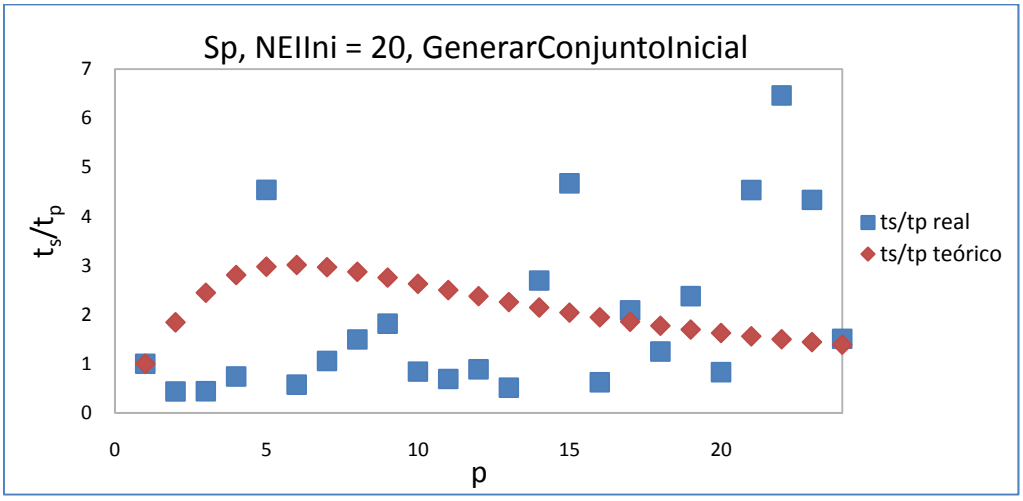

Figura 4.13 Speedup real y teórico frente a número de hilos para *NEIIni*=20.

En las figuras 4.13 a 4.15 presentamos los resultados experimentales de speedup frente a número de hilos para valores de *NEIIni* de 20, 100 y 500. Vemos que existe una gran dispersión en los resultados experimentales, especialmente para tamaños más reducidos. Esto parece normal si tenemos en cuenta el reducido tamaño de la población inicial generada, que conlleva tiempos de ejecución pequeños, pudiendo existir interferencias del sistema durante la ejecución de los experimentos.

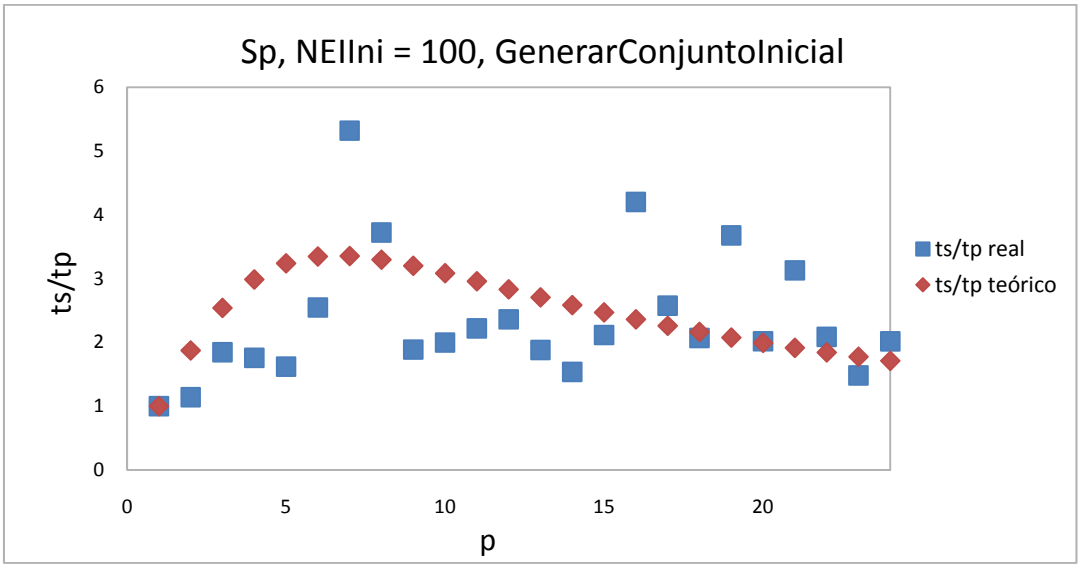

Figura 4.14 Speedup real y teórico frente a número de hilos para *NEIIni*=100.

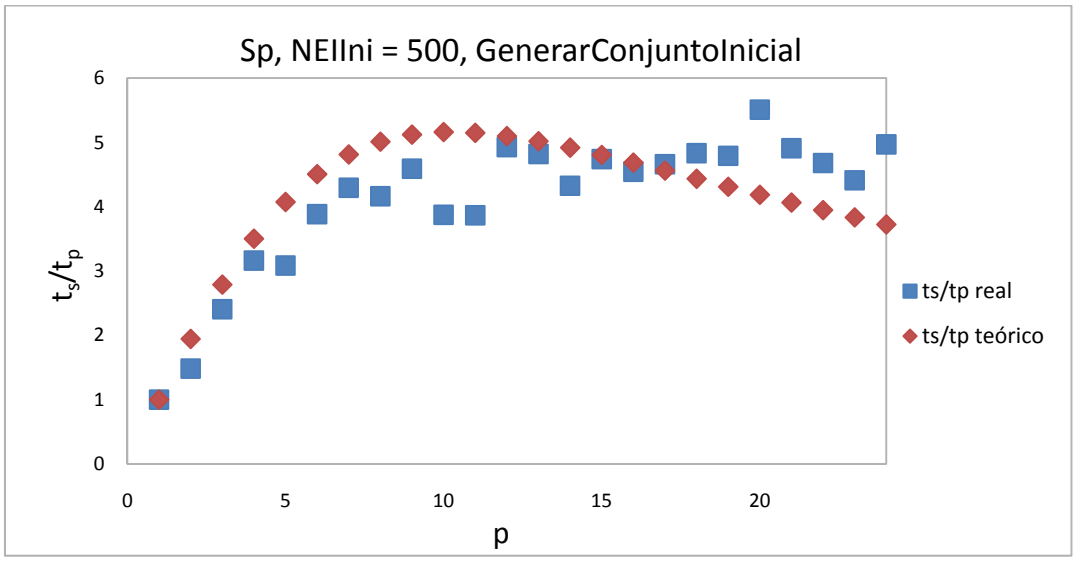

Figura 4.15 Speedup real y teórico frente a número de hilos para *NEIIni*=500.

Se observa que, conforme aumentamos el tamaño de la población inicial, se va reduciendo la dispersión de los resultados experimentales, aproximándose cada vez más la tendencia a la teórica.
Vemos un resumen de los valores, bastante diferentes, de k_p obtenidos de los ajustes para distintos valores de *NEIIni* en la tabla 4.7. Al igual que en *Arabí*, el modelado en *Saturno* no sería tan preciso como en *Ben* debido a la disparidad de valores de k_p obtenidos para los diferentes tamaños ensayados.

NEIIni	k_p
20	1,54E-04
100	6,14E-04
500	1,27E-03

Tabla 4.7 Valores de k_p para distintos tamaños iniciales de población *NEIIni*.

La segunda parte del estudio de paralelismo, se centra en el análisis del paralelismo anidado de dos niveles que presenta la función de la metaheurística *MejorarElementos*. La descripción del algoritmo secuencial de la función la podemos encontrar en el capítulo 2. Podemos reconocer el paralelismo de dos niveles en la estructura de la función *MejorarElementos*. Ésta correspondería al primer nivel y la llamada desde aquí a la función *MejorarElemento* introduciría el segundo nivel de paralelismo anidado. De este modo, la mejora de los elementos individuales seleccionados en el primer nivel, se realiza en paralelo, teniendo tantas mejoras como hilos de segundo nivel hayamos puesto en marcha. En definitiva, la reducción del tiempo de ejecución sólo la produciría el paralelismo de primer nivel y el coste computacional de mejorar cada elemento sería similar al secuencial más el coste de poner en marcha los hilos de segundo nivel.
Presentamos los resultados obtenidos al ejecutar en *Ben* un problema con un tamaño 200,6 cuya nomenclatura y valores de variables del problema podemos encontrar en la tabla 2.2. El estudio teórico del problema nos lleva a plantear que existe una relación entre el tiempo de ejecución, el número de hilos y los parámetros *NEIIni*, *PEMIni* e *IMEIni* que viene dada por:

$$t_{mej} = \frac{k_m \cdot \frac{NEIIni \cdot PEMIni \cdot IMEIni}{100}}{p_1} + k_{p,1} \cdot p_1 + k_{p,2} \cdot p_2 \qquad (4.4)$$

donde t_{mej} es el tiempo de ejecución de la función *MejorarElementos* dentro de *Inicializar*, k_m, $k_{p,1}$ y $k_{p,2}$ son constantes a determinar que dependen del sistema computacional donde se realizan los experimentos, *NEIIni* la población inicial, *PEMIni* es el porcentaje de elementos a mejorar, *IMEIni* es la intensificación de dicha mejora, y p_1 y p_2 son el número de hilos de ejecución de primer y segundo nivel de paralelismo.

Para comprobar que la ecuación teórica propuesta es la que describe nuestro sistema, se han realizado una serie de experimentos iniciales para un tamaño medio de *NEIIni* =100, *PEMIni*=50 e *IMEIni*=10. En ellos, se varían por separado los hilos del primer nivel (p_1) en series donde se mantiene constante los hilos de segundo nivel (p_2) y viceversa. Lo veíamos en las figuras 3.6 y 3.7.

Veíamos que, al variar p_1 manteniendo p_2 constante, aparece en todas las series la curva típica de la función $f(x)=A/x +B\cdot x +C$ que nos indica que los resultados se ajustan a la ecuación 4.4 propuesta teóricamente. Podemos localizar, en la figura 3.6, un mínimo de tiempo de ejecución en el intervalo p_1[10-14].

Observábamos, en la figura 3.7, que obtienen líneas rectas en todas las series. Esto demostraba que el paralelismo de segundo nivel no reduce el tiempo de ejecución sino que, tal como se predijo y como se diseñó el algoritmo de mejora, solo influye linealmente con el coste de puesta en marcha de los hilos anidados del segundo nivel. También observamos que las pendientes de las rectas son similares, lo que nos asegura que la constante $k_{p,2}$ en la ecuación 4.4 solo va multiplicada por p_2.

Podemos deducir, como hemos dicho, que sólo existe reducción en el tiempo de ejecución de la función *MejorarElementos* utilizando paralelismo de primer nivel (existe un óptimo como hemos visto). El uso de un segundo nivel anidado de paralelismo sólo se justifica desde el punto de vista de la mejora de la función de bondad al explorar más soluciones en paralelo y por tanto enriquecer el proceso de búsqueda.

Una vez establecida la ecuación que representa el sistema, la determinación de los parámetros de la ecuación 4.4, nos permitirá predecir el tiempo de ejecución para cualquier número de hilos lanzados y, por tanto, nos ayudará a predecir el paralelismo óptimo para una determinada combinación de parámetros.

Se ha abordado el problema en dos partes. Primeramente, vamos a determinar el valor de k_m secuencialmente planteando:

$$t_{mej} = k_m \cdot \frac{NEIIni \cdot PEMIni \cdot IMEIni}{100} \tag{4.5}$$

donde hemos considerado que la k_m aquí obtenida es válida para el caso paralelo por ser característica del sistema, independientemente de los hilos que estemos ejecutando. Para ello, realizamos un ajuste por mínimos cuadrados de los datos experimentales, en este caso analizando el tiempo de ejecución frente a diferentes valores de los parámetros de la función dentro de un rango razonable*: NEIIni* [20,100,500], *PEMIni* [5,50,100] y *IMEIni* [5,10,20] con un total de 3^3 combinaciones. Despejando k_m:

$$k_m = \frac{\sum_{i=1}^{n}\left(\frac{NEIIni_i \cdot PEMIni_i \cdot IMEIni_i}{100} \cdot t_{mej,i}\right)}{\sum_{i=1}^{n}\left(\frac{NEIIni_i \cdot PEMIni_i \cdot IMEIni_i}{100}\right)^2} \tag{4.6}$$

El ajuste nos arroja un valor de $k_m = 9{,}10 \cdot 10^{-4}$ s.
Podemos hacernos una idea de la bondad del ajuste teniendo en cuenta el error, en tanto por ciento, calculado como:

$$\%\ error = \frac{\left|t_{real} - t_{teórico}\right|}{t_{real}} \cdot 100 \tag{4.7}$$

Para tamaños pequeños de problema y bajas intensificaciones de mejora no sobrepasa de media el 15%, disminuyendo a 5% para tamaños medianos y con niveles inferiores al 1% a tamaños e intensificaciones pequeñas.
Una vez determinado el valor de k_m, nos disponemos a estimar el valor de las otras constantes características del sistema, k_{p1} y k_{p2}. Para ello, al igual que hemos hecho anteriormente, vamos a realizar un ajuste por mínimos cuadrados de los datos experimentales, en este caso analizando el tiempo de ejecución frente a número de hilos. Tendremos que realizar series de experimentos paralelos fijando los valores de *NEIIni*, *PEMIni* y *IMEIni*, y usando el valor de k_m obtenido en la versión secuencial. Despejando los valores de las constantes del ajuste de mínimos cuadrados obtenemos:

$$k_{p,1} = \frac{\sum_{i=1}^{n} p_{2,i} \cdot t_{mej,i} - \sum_{i=1}^{n} k_m \cdot \frac{NEIIni_i \cdot PEMIni_i \cdot IMEIni_i}{100 \cdot p_{1,i}} \cdot p_{2,i} - k_{p,2} \cdot \sum_{i=1}^{n} p_{2,i}^2}{\sum_{i=1}^{n} p_{1,i} \cdot p_{2,i}} \tag{4.8}$$

$$k_{p,2} = \frac{\sum_{i=1}^{n} p_{1,i}^2 \cdot \left(\sum_{i=1}^{n} p_{2,i} \cdot t_{mej,i} - \sum_{i=1}^{n} k_m \cdot \frac{NEIIni_i \cdot PEMIni_i \cdot IMEIni_i}{100 \cdot p_{1,i}} \cdot p_{2,i} \right)}{\sum_{i=1}^{n} p_{1,i}^2 \cdot \sum_{i=1}^{n} p_{2,i}^2 - \left(\sum_{i=1}^{n} p_{1,i} \cdot p_{2,i} \right)^2} - \frac{\sum_{i=1}^{n} p_{1,i} \cdot p_{2,i} \cdot \left(\sum_{i=1}^{n} p_{1,i} \cdot t_{mej,i} - \sum_{i=1}^{n} k_m \cdot \frac{NEIIni_i \cdot PEMIni_i \cdot IMEIni_i}{100} \right)}{\sum_{i=1}^{n} p_{1,i}^2 \cdot \sum_{i=1}^{n} p_{2,i}^2 - \left(\sum_{i=1}^{n} p_{1,i} \cdot p_{2,i} \right)^2} \tag{4.9}$$

Presentamos, en las figuras 4.16 a 4.18, los resultados experimentales de tiempo de ejecución en segundos frente a número de hilos de segundo nivel (p_2) para las combinaciones de parámetros recogidas en la tabla 4.8 (una gráfica por fila de la tabla). Se han representado en una misma gráfica varias series de resultados fijando el número de hilos de primer nivel de paralelismo (p_1). Presentamos también, en cada gráfica, los valores teóricos una vez fijadas las constantes k_g, $k_{p,1}$, $k_{p,2}$, *NEIIni*, *PEMIni* y *IMEIni* de la ecuación 4.4.

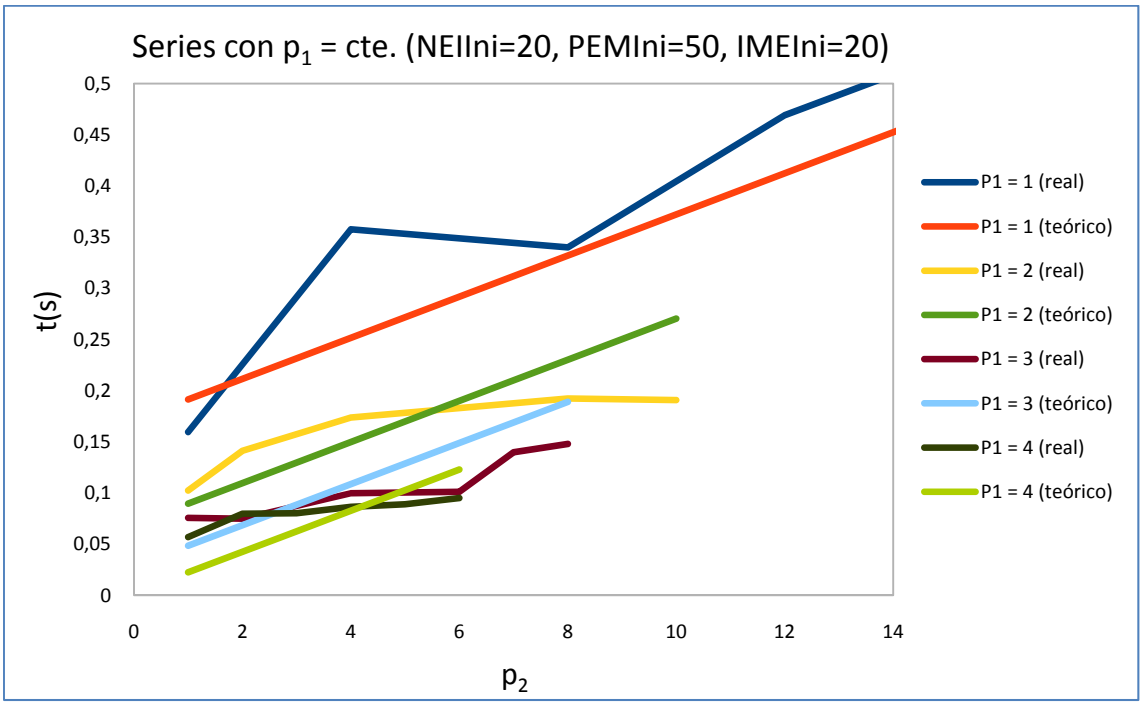

Figura 4.16 Tiempo de ejecución en segundos frente a número de hilos de segundo nivel (p_2) para diferentes valores constantes de p_1. Combinación 1 en tabla 4.8.

Combinación	**NEIIni**	**PEMIni**	**IMEIni**
1	20	50	20
2	100	50	10
3	500	100	5

Tabla 4.8 Valores de parámetros utilizados en los experimentos.

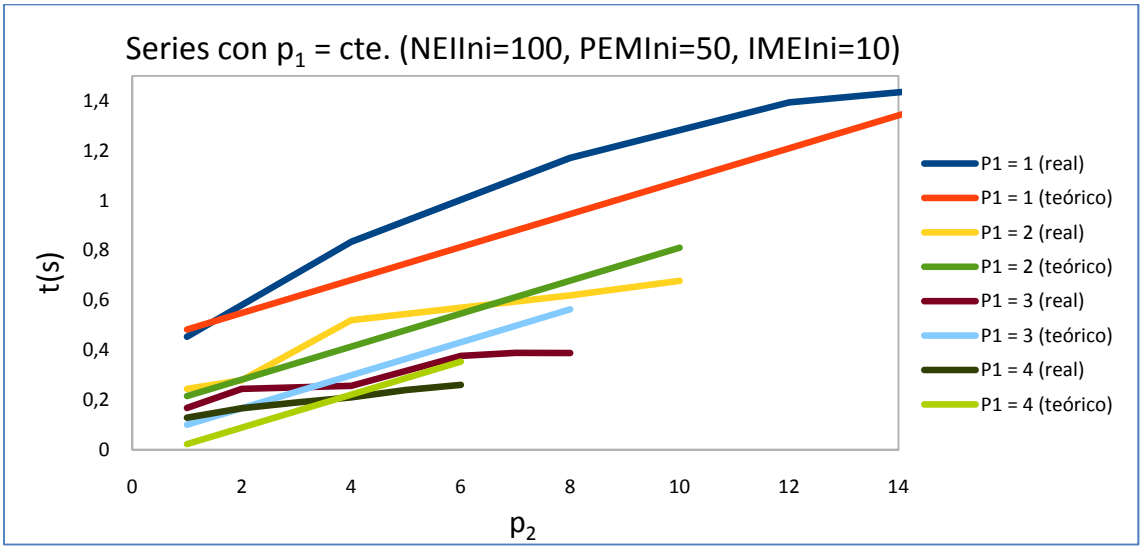

Figura 4.17 Tiempo de ejecución en segundos frente a número de hilos de segundo nivel (p_2) para diferentes valores constantes de p_1. Combinación 2 en tabla 4.8.

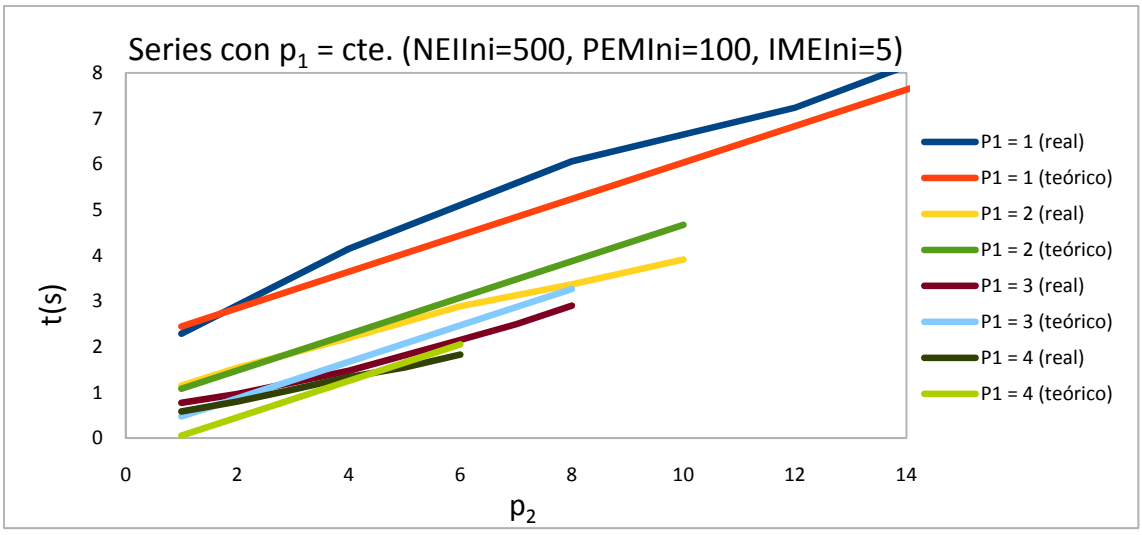

Figura 4.18 Tiempo de ejecución en segundos frente a número de hilos de segundo nivel (p_2) para diferentes valores constantes de p_1. Combinación 3 en tabla 4.8.

Finalmente presentamos un resumen con los valores de las constantes obtenidas para cada combinación de parámetros en la tabla 4.9.

Combinación	***NEIIni***	***PEMIni***	***IMEIni***	$k_{p,1}$	$k_{p,2}$
1	20	50	20	-1,08E-02	2,01E-02
2	100	50	10	-3,95E-02	6,62E-02
3	500	100	5	-2,28E-01	3,99E-01

Tabla 4.9 Valores las constantes $k_{p,1}$ y $k_{p,2}$ para las distintas combinaciones de parámetros.

Podemos ver que los ajustes son mejores al aumentar el tamaño del problema (las curvas teóricas se aproximan más a las experimentales). En todos los caso se observa una tendencia lineal que se corresponde con el tipo de paralelismo definido para el segundo nivel en la mejora. Dentro de una misma gráfica, el ajuste es mejor para las series con p_1 pequeño (cercano a 1).

Los valores de las constantes obtenidos son parecidos en las combinaciones de parámetros 1 y 2 (según tabla 4.9), y se diferencias más con las obtenidas en la combinación 3.

4.3 CONCLUSIONES

En este capítulo hemos constatado que se pueden modelar algunas de las funciones paralelizadas en el capítulo 3. Para ello hemos realizado experimentos variando el número de hilos de primer y segundo nivel, para cada combinación de parámetros de la metaheurística, obteniendo resultados satisfactorios. Esto nos permite elegir el número de hilos óptimo como función de los parámetros de las metaheurísticas, optimizando el esquema paralelo.

Capítulo 5
CONCLUSIONES

Después de lo expuesto en los capítulos anteriores, vamos a extraer algunas conclusiones así como posibles trabajos futuros, y se indicarán los trabajos realizados y en preparación a partir de esta tesis de máster.

5.1 CONCLUSIONES GENERALES

Hemos comprobado que nuestro problema de optimización de costes en la explotación de recursos hídricos sólo se podía abordar con métodos metaheurísticos, y para obtener una metaheurística satisfactoria ha sido necesario desarrollar y experimentar con varios métodos hasta alcanzar la metaheurística óptima para nuestro problema. Hemos reutilizando las funciones básicas de nuestro algoritmo gracias al uso de un esquema unificado parametrizado de metaheurísticas facilitando su desarrollo. Los resultados más significativos obtenidos han sido los siguientes:

- Dando diferentes valores a los parámetros en cada función, hemos obtenido distintas metaheurísticas o combinaciones de ellas.
- En el caso secuencial, los mejores resultados en cuanto a función de bondad se han obtenido combinando metaheurísticas: concretamente con la combinación GA+SS y con GR+GA +SS, mejorando los resultados iniciales con algoritmos genéticos.
- Para obtener versiones paralelas de las metaheurísticas y adaptarlas a las características del sistema paralelo, hemos utilizado un esquema unificado parametrizado en memoria compartida. Se han realizado experimentos paralelos orientados a mejorar los resultados alcanzados en la versión secuencial. No sólo se ha mejorado el tiempo de ejecución, sino también la función de bondad aunque en menor medida.
- Hemos confirmado experimentalmente la dependencia del número de hilos, necesario para minimizar el tiempo de ejecución, con los parámetros de la metaheurística. Se ha modelado algunas funciones representativas del sistema obteniéndose resultados satisfactorios y permitiéndonos elegir el número de hilos más adecuado como función de los parámetros de las metaheurísticas, optimizando el esquema paralelo.

5.2 TRABAJOS FUTUROS

En base a todo lo visto, se plantean los siguientes trabajos futuros:

- A partir del conjunto de parámetros establecido se podrán diseñar hiperheurísticas [5] para obtener la mejor combinación de parámetros para el problema que estemos resolviendo, con lo que se facilitará la obtención de la mejor metaheurística y combinación de parámetros para ella, o de la mejor técnica híbrida o mixta.
- Diseño de un mecanismo de selección automática del número óptimo de hilos en cada función del esquema paralelo, utilizando para ello los modelos teóricos del capítulo 4, y obteniendo el esquema metaheurístico parametrizado paralelo con autooptimización.
- Extensión del esquema parametrizado de memoria compartida a paso de mensajes y GPUs.
- Desarrollo de una interfaz de usuario atractiva.
- Utilización del software desarrollado en un entorno real.

5.3 RESULTADOS

Se realizó una presentación inicial sobre el tema de la tesis en la reunión del grupo de investigación en Computación Científica y Programación Paralela el 15 diciembre 2010 en la Facultad de Informática Universidad de Murcia (http://www.um.es/pcgum/reunion101215.html).

Con los resultados conseguidos se están preparando los siguientes trabajos:

- Un trabajo para el MAEB 2012 que se celebrará en febrero de 2012 (http://congresomaeb2012.uclm.es/).
- Un trabajo para el Evostar 2012 que se celebrará en abril de 2012 (http://www.evostar.org).

REFERENCIAS

[1] Alba, E.: Parallel Metaheuristics: A New Class of Algorithms. Wiley Interscience, 2005.

[2] Almeida, F., Cuenca, J., Giménez, D., Llanes-Castro, A., Martínez-Gallar, J.P.: A framework for the application of metaheuristics to tasks-to-processors assignation problems. Journal of Supercomputing (published online, September 2009).

[3] Almeida, F., Giménez, D., López-Espín, J.J.: A parameterised shared-memory scheme for parameterised metaheuristics. In: Proc. Int. Conf. CMMSE, pp. 57–64 (2010).

[4] Almeida, F., Giménez, D., López-Espín, J.J.: Comparación de metaheurísticas para la obtención de modelos de ecuaciones simultáneas. VII Congreso Español sobre Metaheurísticas, Algoritmos Evolutivos y Bioinspirados, Valencia, 7-10 septiembre 2010.

[5] Almeida, F., Giménez, D., López-Espín, J.J.: Parameterized schemes of metaheuristics: basic ideas and applications. En revisión.

[6] Cuenca, J., Giménez, D., López-Espín, J.J., Martínez-Gallar, J.P.: A proposal of metaheuristics to schedule independent tasks in heterogeneous memory-constrained systems. In: Proc. IEEE Int. Conf. on Cluster Computing. IEEE Computer Society, pp. 422–427 (2007).

[7] Cuenca, J., Giménez, D.: Improving Metaheuristics for Mapping Independent Tasks into Heterogeneous Memory-Constrained Systems. ICCS (1), LNCS 5101, Springer, 2008, 236-245.

[8] Cutillas, L.G.: Metaheurística aplicada a la optimización de los criterios de producción de aguas subterráneas. Proyecto Sondea. Proyecto Final de Carrera, Universidad de Alicante, 2008.

[9] Dréo, J., Pétrowski, A., Siarry, P., Taillard, E.: Metaheuristics for Hard Optimization. Springer, 2005.

[10] Glover, F., Kochenberger, G. A.: Handbook of Metaheuristics. Kluwer, 2003.

[11] López-Espín, J.J., Giménez, D.: Genetic algorithms for simultaneous equation models. In: DCAI, pp. 215–224 (2008).

[12] Raidl, G.R.: A unified view on hybrid metaheuristics. Hybrid Metaheuristics, Third International Workshop, LNCS 4030, 2006, 1-12.

Anexo A.

ANÁLISIS TÉCNICO DEL PROBLEMA DE OPTIMIZACIÓN DE COSTES EN LA EXPLOTACIÓN DE RECURSOS HÍDRICOS

Se presenta a continuación, a modo de especificación técnica, un resumen del proyecto original [8] tomado como problema para el análisis realizado en la presente tesis de máster.

La gestión del suministro de agua a una gran ciudad es una labor compleja que depende fundamentalmente de la actuación de los operadores de la red. Aunque ciertas tareas se realizan de manera automatizada, la mayor parte de las decisiones se toman basándose en la intuición y la experiencia previa de los propios operadores en cada momento. Esto redunda en una falta de definición de la política global de gestión del sistema, que suele estar basada en las decisiones tomadas por distintos operadores para resolver los problemas que se puedan presentar en la red de suministro en cada situación concreta.

La adopción de una estrategia de eficiencia energética elaborada en un proceso de optimización global del régimen de explotación de aguas subterráneas y de la red de abastecimiento y distribución de agua a través de un control centralizado de determinación del esquema de operación de bombas más adecuado, permite minimizar el coste de explotación en la gestión de aguas de la empresa. Así, esta minimización pasará por la reducción de los costes de energía eléctrica utilizada en la explotación de las aguas subterráneas.

Ante la multitud de variables que pueden influir en la elección de los orígenes del agua producida, procedentes de los distintos sondeos disponibles, se automatiza dicho proceso de selección a partir de los requerimientos de demanda y de las condiciones propias de cada sondeo como son: nivel piezométrico, calidad química, volumen máximo de extracción e interacción con otros sondeos en explotación. Adicionalmente, se tienen en cuenta los costes eléctricos de cada pozo y la tarifación horaria, factores fundamentales para la determinación de los costes de explotación.

Se partirá de una demanda cuya variación y distribución en el periodo a optimizar habrá sido ya previamente determinada por métodos apropiados y con la suficiente fiabilidad.

Todos los datos no modelados pueden ser proporcionados en tiempo real por el sistema telemandado de la empresa de aguas, siendo fundamentales como parámetros de entrada diarios.

A.1 EL PROBLEMA DE PROGRAMACIÓN ÓPTIMA DE BOMBEOS

La demanda de agua de una población es variable en el tiempo. La cantidad de agua a suministrar para satisfacer dicha demanda deberá, por lo tanto, ser también variable en el tiempo. En general, el sistema de explotación consta de un conjunto de bombas de diferentes capacidades que extraen agua de diferentes sistemas acuíferos y aportaciones superficiales que contribuyen a la satisfacción de la demanda. Los caudales explotados llegan a una serie de depósitos de distribución donde son distribuidos por los distintos municipios.

En general, el sistema de bombeo cuenta con un conjunto de bombas de diferentes capacidades que bombean agua a uno o más depósitos de regulación. Estas bombas trabajan combinadas para bombear la cantidad de agua necesaria, atendiendo a las restricciones del problema. Por lo tanto, dependiendo del momento, algunas bombas estarán encendidas y otras apagadas.

Programar el bombeo en una estación consiste en establecer la combinación de bombas a utilizar en cada intervalo de tiempo del horizonte de planificación. Luego, una programación de bombeo es el conjunto de todas las combinaciones de bombas a utilizar durante cada intervalo del horizonte de planificación. Una programación óptima de bombeo puede definirse, por ejemplo, como una programación que cumpla con las restricciones del problema (como atender la demanda), pero que además optimice los objetivos establecidos.

A.2. FORMULACIÓN TÉCNICA DEL PROBLEMA

El objetivo del problema original estudiado era el de minimización del coste del consumo de energía eléctrica. Bajo esta premisa, se utiliza una función objetivo básica que será optimizada (minimizada) al objeto de encontrar una solución de mínimo coste. Además, se establecen las restricciones que delimitan las soluciones reales del problema.

A.2.1. FUNCIÓN OBJETIVO: COSTE DE ENERGÍA ELÉCTRICA.

El coste de energía eléctrica consumida se define como el gasto que implica el consumo de energía eléctrica por parte de las bombas empleadas para extraer el agua.

Un factor importante en el coste de la energía eléctrica es la estructura tarifaria de la misma. En muchos sistemas de suministro de electricidad el coste de la energía eléctrica

no es el mismo en todas las horas del día, dado que existen horas de mayor consumo y otras de menor consumo. Sin embargo, las instalaciones deben dimensionarse para el consumo máximo. Esto es conocido como tarifa diferenciada. Normalmente se considera la siguiente estructura tarifaria estándar:

Tarifa de hora valle (T_v): de 0 a 8 hs.

Tarifa de hora punta (T_p): de 8 a 12 hs.

Tarifa de hora llano (T_{ll}): de 12 a 24 hs.

Los rangos horarios considerados para la evaluación del algoritmo podrán tener, ajustándose a esta división horaria, cuatro estructuras, encajando dentro de cada intervalo. Así, podremos tener:

- 24 rangos de 1 hora.
- 12 rangos de 2 horas.
- 6 rangos de 4 horas.
- 3 rangos de 8 horas.

Cabe destacar que la influencia en los experimentos y el programa general de esta diferencia tarifaria dentro de la programación de bombeo es notable, ya que se reducen los costes de consumo de energía eléctrica si la programación de bombeo óptima establece la menor cantidad posible de bombas encendidas durante el horario de punta de carga, es decir en el rango de horas con tarifa punta, cuando la tarifa es mayor, intentando en cualquier caso que se concentre la explotación en las horas de mínimo gasto. Así, se intentará siempre concentrar la explotación en las 8 horas de que consta el rango de tarifa valle. De no ser posible, se intenta continuar la explotación en las correspondientes horas de tarifa llano, evitando en lo posible la explotación en horas punta.

En aquellas horas en las que no se centra la explotación, se ha tenido la precaución de continuar manteniendo un caudal mínimo que nos servirá para mantener siempre en carga las tuberías de abastecimiento, favoreciendo así la explotación desde un punto de vista técnico.

Se ha definido la función coste de energía eléctrica C_e en base a la combinación de bombas adoptada en cada intervalo horario, en base a la energía empleada para elevar el caudal estimado a la altura pertinente y en base a la tarifa horaria aplicable a dicha energía. Con todo ello establecíamos la ecuación (1.1), donde el valor de la potencia será conocido y función del caudal y la altura manométrica, que son datos conocidos extraídos de la curva de demanda y de las características de la explotación respectivamente. Además, debido a la existencia de programas específicos para su cálculo, el dato de potencia es de obtención directa.

A.2.2. DEFINICIÓN TÉCNICA DE LAS RESTRICCIONES

1º) Mantenimiento de la oferta. Satisfacción de la demanda

Esta restricción deriva de la condición de que la suma de los volúmenes aportados en los rangos de horas se corresponda con la demanda programada al comienzo de cada jornada. En la medida de lo posible, la explotación se centrará en horas de tarifa valle (por tanto más baratas), estableciendo el volumen que debe ser aportado en este rango horario. Si no se llegara a aportar el volumen teórico demandado para el rango de las horas de tarifa valle, la diferencia entre éste y el máximo explotado en dicho rango será acumulada a las horas de tarifa llano, evitando que en horas punta se sobrepase el límite inferior del caudal mínimo recomendado.
Así, matemáticamente teníamos la ecuación (1.2) para satisfacción de la demanda y (1.3) para el mantenimiento del caudal mínimo en la tubería para cada franja horaria.

2º) Cumplimiento de los volúmenes máximos de explotación anuales de cada pozo

Como anteriormente se ha comentado, cada pozo posee un volumen máximo de concesión anual, no debiendo éste ser superado al final de cada año de explotación. Como consecuencia, se debe imponer una restricción que limite diariamente el volumen explotado para no superar la parte proporcional de concesión diaria que le corresponda a cada pozo. Así, si un día no se extrae agua de un pozo, o se extrae menos del caudal de concesión acumulado que le correspondiera, este caudal no extraído se sumará al de concesión del día siguiente.
Sumando la diferencia entre los caudales de concesión acumulados y los caudales explotados diariamente, ésta no deberá ser inferior a 0 al final del año.
Así:

$$\sum_{k=1}^{365} \sum_{j=1}^{B} V_{conc\,kj} \geq 0 \qquad \text{(A.1)}$$

siendo

$$V_{conc\,kj} = V_{conc\,k-1,j} - \sum_{i=1}^{R} \sum_{j=1}^{B} V_{ij} \cdot x_{ij} \qquad \text{(A.2)}$$

donde

$V_{conc\ kj}$ = Volumen de concesión del pozo *j* el día *k*.
V_{ij} = Volumen extraído del pozo *j* en la franja horaria *i*.

Con la opción de imponer que el sumatorio sea mayor o igual que 0, se asegura a su vez que exista una diversificación en los pozos explotados, lo que, desde un punto de vista hidrogeológico es beneficioso para los mismos.
En cualquier caso, se ha decidido flexibilizar en la medida de lo posible esta restricción, ya que en muchas ocasiones resulta conveniente sobreexplotar, dentro de ciertos márgenes, determinados pozos con respecto a su volumen de concesión teórico.
En la práctica se actualiza cada día el volumen concedido acumulado para cada pozo y se introduce como parámetro del problema. De manera que cada día debe cumplirse la restricción para cada pozo *j* explotado dada por (1.4).

3º) Cumplimiento de calidad química de las aguas en función de la conductividad

La conductividad nos expresa una medida indirecta de la calidad química de las aguas, ya que proporciona una medida del contenido en sales disueltas en la misma. La conductividad eléctrica, se define como la capacidad que tienen las sales inorgánicas en solución (electrolitos) para conducir la corriente eléctrica.
El agua pura prácticamente no conduce la corriente, sin embargo el agua con sales disueltas conduce la corriente eléctrica. Los iones cargados positiva y negativamente son los que conducen la corriente, y la cantidad conducida dependerá del número de iones presentes y de su movilidad. En la mayoría de las soluciones acuosas, cuanto mayor sea la cantidad de sales disueltas, mayor será la conductividad. Este efecto continúa hasta que la solución está tan llena de iones que se restringe la libertad de movimiento y la conductividad puede disminuir en lugar de aumentar. Las unidades de la conductividad eléctrica son el Siemens/cm.
La conductividad no será mayor que una cierta conductividad límite, 2500 μS/cm a 20ºC, expresada en el REAL DECRETO 140/2003, de 7 de febrero, por el que se establecen los criterios sanitarios de la calidad del agua de consumo humano tal y como se recoge en su Parte C.
Una vez establecido el valor límite de conductividad de la mezcla cabe preguntarse cómo determinar de una manera sencilla dicho valor en un caso real conociendo únicamente las conductividades de las aguas en el origen. Este punto se ha simplificado de manera que el valor de la conductividad de la mezcla se ha obtenido como una media ponderada en proporción a las conductividades y los caudales aportados.

Si bien el valor de conductividad obtenido no va a ser el exacto para la mezcla de aguas, consideramos que sí proporcionará un orden de magnitud suficientemente cercano al propósito perseguido.
Así para cada rango horario o tramo de funcionamiento teníamos (1.5).

4º) Cumplimiento de profundidades máximas de niveles dinámicos. Control diario de la profundidad del nivel dinámico de cada pozo. Cono de descensos

Si nos centramos en el comportamiento del agua subterránea cuando se bombea en un sondeo vertical y suponemos que el bombeo se realiza en un acuífero libre cuya superficie freática inicial es horizontal, observamos que el agua comienza a fluir radialmente hacia el sondeo, y, transcurrido un tiempo, la superficie freática adquiere una forma característica denominada cono de descensos. La forma del cono resulta ser convexa ya que el flujo necesita un gradiente cada vez mayor para circular por secciones cada vez menores.
La explotación de los pozos supone irremediablemente un abatimiento en su nivel dinámico derivado de la extracción de un caudal continuo de los mismos. Dicho abatimiento no puede llegar a suponer un problema para el funcionamiento de las bombas, que pueden sufrir diversas patologías derivadas (cavitación) y, por supuesto, no puede suponer el agotamiento del pozo (a menos que el decisor lo crea oportuno). Es por ello que se establece como restricción la profundidad máxima a la que puede llegar el nivel dinámico de cada pozo. Ahora bien, en lugar de establecer la oportuna formulación para modelar el cono de descensos en cada sondeo, se opta por obtener diariamente un vector donde se recogen las profundidades a las que se encuentra el nivel dinámico de cada pozo y compararlo con otro vector donde se recogen las profundidades máximas a las que éste puede llegar, quedando estas últimas afectadas por las holguras que se consideren oportunas. Los datos que constituyen los vectores mencionados son parámetros de obtención en tiempo real.
En el caso de que el nivel de un pozo alcanzara los márgenes determinados como de riesgo para el correcto funcionamiento hidráulico del mismo o para su bomba asociada, el pozo quedará fuera de servicio automáticamente, hasta que los niveles se vuelvan a estabilizar (presumiblemente en un nivel inferior al nivel estático del intervalo anterior) o en caso de que el decisor lo considere oportuno.