UNIVERSIDAD DE GRANADA

E.T.S. DE INGENIERÍA INFORMÁTICA

Departamento de Ciencias de la Computación e Inteligencia Artificial

Estrategias coordinadas paralelas basadas en soft-computing para la solución de problemas de optimización

TESIS DOCTORAL

Carlos A. Cruz Corona

Editor: Editorial de la Universidad de Granada
Autor: Carlos Alberto Cruz Corona
D.L.: Gr. 1803 - 2005
ISBN: 84-338-3609-9

Estrategias coordinadas paralelas basadas en soft-computing para la solución de problemas de optimización

MEMORIA QUE PRESENTA

Carlos A. Cruz Corona

PARA OPTAR AL GRADO DE DOCTOR

Directores:
David Pelta – Alejandro Sancho Royo

Índice general

Indice de Figuras

Indice de Tablas

Resumen

Se presenta en este trabajo una estrategia cooperativa paralela basada en técnicas de la Soft-Computing. Se parte de la idea de disponer de un conjunto de agentes que ejecutan algoritmos de resolución de problemas de optimización combinatoria y ejecutarlos en paralelo de forma coordinada para resolver el problema en cuestión. Un punto relevante es el uso de la teoría de los sistemas y conjuntos difusos aplicados tanto en el nivel de coordinación a través de un conjunto de reglas difusas como en el nivel de los agentes resolvedores al usar la metaheurística difusa FANS. Para validar esta estrategia se muestran los resultados obtenidos en la solución del problema de la mochila y el problema de la p-mediana.

Hay que soñar con cosas;

pero a la vez hay que hacer las cosas con las que se sueña...

Agradecimientos

Al Ministerio de Educación Superior de Cuba y a la Universidad de Holguín por darme la oportunidad de realizar el doctorado.

A la Consejería de Presidencia de la Junta de Andalucía y al Vicerrectorado de Relaciones Internacionales e Institucionales de la Universidad de Granada, que financiaron el Proyecto de Doctorado Cooperado entre la Facultad de Informática y Matemáticas de la Universidad de Holguín, Cuba, y la Escuela Técnica Superior de Informática de la Universidad de Granada; así como al financiamiento aportado por el proyecto TIC2002-4242-C03-C02 para la realización de este trabajo.

A todos los profesores del doctorado por los conocimientos aportados a mi formación, y por los lazos de amistad que desde entonces nos unen.

A los miembros del Departamento de Ciencias de la Computación e Inteligencia Artificial de la Universidad de Granada por los conocimientos, consejos y la ayuda dada en todo momento.

Al Departamento de Arquitectura y Tecnología de Computadores de la Universidad de Granada y su proyecto SINTA-CC-TIN 2004-01411 que me permitió usar el cluster de ordenadores para la ejecución de los experimentos. Gracias a sus profesores y técnicos por toda la ayuda prestada.

A l@s becari@s de "la 16" por compartir ideas, sueños, horas de de trabajo y también las noches de tapas y marcha.

A mis directores y demás miembros del grupo de investigación MODO (con sus respectivas familias), por su colaboración, ayuda y amistad; y en especial a Curro, no solo por ser el padre del Proyecto del Doctorado Cooperado con Cuba gracias al cual ha sido posible que hoy lea mi tesis, sino también por su guía y acertados consejos y en especial por darme la posibilidad de ser su amigo.

A mis amig@s, de aquí, de allá, todos lados, por su apoyo y por hacerme sentir que siempre puedo contar con ellos.

A esta ciudad "...de Manolas" que me acogió y me acompaña, sobre todo en mis largas y frecuentes noches de ronda por las calles del Albaicín, aliviando la profunda nostalgia que me produce estar tan lejos de mi familia y mi Cuba.

Y gracias a la vida, que me ha dado y quitado tanto...

Introducción

El ser humano constantemente se enfrenta a situaciones que lo obligan a tomar decisiones ante una infinidad de problemas en los que no basta el sentido común y entonces hay que acudir a la ayuda de la ciencia. Una categoría importante y compleja de estos problemas son aquellos en que se desea maximizar o minimizar una cantidad específica, que puede ser llamada objetivo, y que depende de un número finito de variables de entrada. Estas variables pueden ser independientes unas de otras o pueden estar relacionadas por una o más restricciones.

Para modelar estos problemas, lo primero es identificar las posibles decisiones que pueden tomarse y esto puede hacerse a través de las variables del problema concreto que son generalmente de carácter cuantitativo, buscándose los valores de las mismas que optimizan el objetivo. Posteriormente se define qué decisiones resultan admisibles lo que conduce a un conjunto de restricciones teniendo presente la naturaleza del problema. Se calcula el coste/beneficio asociado a cada decisión admisible lo que supone determinar una función objetivo que asigna un valor de coste/beneficio a cada conjunto posible de valores para las variables que conforman una decisión. El conjunto de todos estos elementos define un problema de optimización.

La resolución de este tipo de problemas tuvo su florecimiento a partir de la Segunda Guerra Mundial con el surgimiento y desarrollo de la Investigación Operativa como Ciencia. Todo se inicia en Gran Bretaña en 1938 cuando a instancias del Ministerio del Aire un equipo multidisciplinario de científicos comienza a investigar como integrar la información tomada de los radares con la de observadores de tierra para interceptar los bombarderos alemanes. En la década de los años cincuenta con la aparición de los computadores paralelos se comienza a aplicar en gran escala esta ciencia en el mundo empresarial como herramienta para la planificación y administración de negocios.

En este tipo de problemas, en los cuales el dominio es típicamente finito, puede parecer trivial encontrar su solución ya que existe siempre un procedimiento elemental para determinar la solución óptima buscada, que es realizar una exploración exhaustiva del conjunto de soluciones. O sea, basta realizar los siguientes pasos: generar todas las soluciones factibles, calcular el costo asociado a cada una, y finalmente elegir la que haya obtenido el mejor valor. Sin embargo, aunque teóricamente este método siempre llega a la solución buscada, no es eficiente, pues el tiempo de cálculo necesario crece exponencialmente con el número de ítems del problema y por lo tanto procesarlas todas para encontrar la mejor entre ellas resulta prácticamente imposible aún para instancias de tamaño moderado.

Existen problemas combinatorios para los cuales no se conocen algoritmos de resolución, más que aquéllos en los cuales se produce una explosión exponencial del tiempo de cálculo al aumentar el tamaño del problema. Son problemas computacionalmente difíciles de tratar. Por el contrario, para otros problemas combinatorios existen algoritmos que sólo crecen de forma polinomial en el

tiempo con el tamaño del problema.

Matemáticamente se han caracterizado a ambos. Aquellos, para los cuales se conocen algoritmos que necesitan un tiempo polinomial para ofrecer la solución óptima, se dice que pertenecen a la clase P, y se considera que son resolubles eficientemente. Sin embargo, se comprobó que la mayoría de los principales problemas de optimización pertenecen a un superconjunto de P, la clase denominada NP, en la cual se incluyen aquellos problemas para los que no se conoce un algoritmo polinomial de resolución.

Estos problemas difíciles tienen la peculiaridad de que si se pudiera obtener una solución en tiempo polinomial para uno de ellos, se podría lograr para todos los de la clase NP. Esta propiedad ha sido usada para definir una subclase separada en NP: la de los problemas NP-hard. Se dice que un problema es NP-hard si cualquier problema en NP es polinomialmente transformable en él, aunque el problema en sí no pertenezca a NP. Si el problema además pertenece a NP, entonces se le denomina NP-completo.

Es deseable que para cada problema de este tipo se pudiera encontrar la solución óptima a través de algoritmos exactos que requieran solamente una cantidad moderada de recursos. Lamentablemente, los problemas NP-completos no admiten este tipo de enfoque, por tanto, ha sido necesario el desarrollo de técnicas que, aunque puedan llevar a la obtención de soluciones no óptimas, pueden resolver estos problemas en términos de satisfacción para el usuario o decisor y con un gasto aceptable de recursos computacionales. Estas técnicas se conocen como heurísticas.

El término **heurística** proviene de una palabra griega, *heuriskein*, cuyo significado está relacionado con el concepto de encontrar y se vincula a la famosa y supuesta exclamación *¡eureka!* de Arquímedes.

Así, se ha desarrollado un número alto de procedimientos heurísticos para la resolución de problemas de optimización específicos con mucho éxito, de los que se intenta extraer lo mejor de ellos y emplearlo en otros problemas o en contextos más extensos. Esto ha contribuido al desarrollo científico de este campo de investigación y a extender la aplicación de sus resultados. Surgen así las denominadas metaheurísticas, término que apareció por primera vez en un artículo de Fred Glover en 1986 [1].

El término **metaheurística** deriva de la composición de dos palabras griegas: *heurística* ya explicada anteriormente y el sufijo *meta* que significa más allá o de nivel superior. Aunque no existe una definición formal de qué es una metaheurística, las dos siguientes propuestas dan una representación clara de la noción general del término.

- Osman y Laporte [2] la definen así: *una metaheurística se define formalmente como un proceso iterativo que guía una heurística subordinada, combinando de forma inteligente diferentes conceptos para explorar y explotar el espacio de búsqueda.*
- según Voss et al. [3]: *una metaheurística es un proceso maestro iterativo que guía y modifica las operaciones de heurísticas subordinadas para producir, de forma eficiente, soluciones de alta calidad. En cada iteración, puede manipular una solución (completa o incompleta) o un conjunto de soluciones. Las heurísticas subordinadas pueden ser procedimientos de alto o bajo nivel, o simplemente una búsqueda local o método constructivo.*
- Y Melián et al. [4]: *las metaheurísticas son estrategias inteligentes para diseñar o mejorar procedimientos heurísticos muy generales con un alto rendimiento.*

Resumiendo, las características fundamentales de las metaheurísticas son:

- Son estrategias que *guían* el proceso de búsqueda.
- Su objetivo es explorar eficientemente el espacio de búsqueda para encontrar las soluciones sub-óptimas.
- Las técnicas que la forman van desde algoritmos de búsqueda simple a complejos procesos de aprendizaje.
- Son algoritmos aproximados y generalmente no determinísticos.
- Incorporan mecanismos para escapar cuando están confinados en áreas del espacio de búsqueda.
- Sus conceptos básicos permiten una descripción de nivel abstracto.
- No son específicas del problema.
- Pueden hacer uso del conocimiento específico del dominio y/o de la experiencia de la búsqueda para influenciar la búsqueda.
- Generalmente se basan en la experiencia y en conocimientos empíricos.

Es importante tener en cuenta que estas estrategias deben ser escogidas de manera tal que haya un balance dinámico entre la explotación de la experiencia acumulada en la búsqueda de las soluciones (intensificación) y la exploración del espacio de búsqueda (diversificación). Esto es necesario, por un lado, para identificar rápidamente las regiones en el espacio de búsqueda que tienen soluciones de alta calidad; y por otro, no derrochar demasiado tiempo en regiones del espacio de búsqueda que ya hayan sido explorados o no brinden soluciones de alta calidad.

Aunque las metaheurísticas brindan estrategias efectivas para encontrar soluciones aproximadas a los problemas de optimización, los tiempos de ejecución asociados con la exploración del espacio de soluciones pueden resultar muy grandes. Además, existen dificultades al momento de utilizar este tipo de herramientas, las cuales surgen por diferentes motivos:

- Dado un problema nuevo, es prácticamente imposible determinar cuál es la mejor estrategia para resolverlo. El usuario debe decidir entre algoritmos que se comporten bien en problemas de características similares, o utilizar uno que ya conozca.
- Aunque conozcamos un buen algoritmo para la clase de problemas con la que estamos tratando, se puede dar el caso que la instancia o tipo de instancias que necesitamos resolver resulten complejas para el algoritmo en cuestión.
- El comportamiento de las metaheurísticas es una función de los parámetros que se han utilizado en su definición, y por lo general, el ajuste de los mismos se realiza empíricamente. No existe ninguna metodología práctica que permita encontrar el mejor conjunto de parámetros para resolver una instancia particular.

Con el fin de abordar algunos de los inconvenientes mencionados anteriormente se presenta como objetivo final en este trabajo una estrategia paralela cooperativa para resolver problemas de optimización combinatoria.

Con la proliferación de la computación paralela, las estaciones de trabajo poderosas y redes de comunicaciones veloces las implementaciones paralelas de las metaheurísticas aparecen como algo natural de su desarrollo, además de una alternativa para aumentar la velocidad de la búsqueda de las soluciones. Se han propuesto y se han aplicado varias estrategias mostrando que éstas también permiten resolver problemas muy grandes y encontrar soluciones mejores, con respecto a sus contrapartes secuenciales, debido a la división del espacio de búsqueda y/o mejorando la intensificación y la diversificación. Por ello, el paralelismo no sólo es una vía para reducir los tiempos de ejecución de las metaheurísticas sino que también mejoran su efectividad y robustez.

La robustez se obtiene por el uso de diferentes combinaciones de estrategias que cooperan entre sí y ajustes de parámetros en cada traza de ejecución, alcanzando soluciones de muy alta calidad para diferentes clases de instancias del mismo problema sin mucho esfuerzo en el ajuste de parámetros.

La idea básica propuesta en este trabajo es disponer de un conjunto de agentes resolvedores, cuya función básica es ser algoritmos de solución de problemas de optimización combinatoria, y ejecutarlos de forma cooperativa a través de un agente coordinador para resolver el problema en cuestión teniendo como premisa fundamental la generalidad basada en un conocimiento mínimo del problema.

Cada agente resolvedor actúa de forma autónoma y se comunica solamente con un agente coordinador para enviarle las soluciones que va encontrando y para recibir de este las directivas que le indiquen cómo seguir actuando. El agente coordinador recibe las soluciones al problema encontradas por cada agente resolvedor y siguiendo una base de reglas difusas que modelan su comportamiento crea las directivas que envía a éstos, llevando de esta manera todo el control de la estrategia.

Como algoritmo de solución usado en el agente resolvedor se eligió la metaheurística FANS (Fuzzy Adaptive Neighborhood Search), que se define como un método de búsqueda por entornos, adaptable y difuso. Es de búsqueda por entornos porque el algoritmo realiza transiciones desde una solución de referencia a otra de su entorno, produciendo algo así como trayectorias o caminos. Es adaptable porque su comportamiento varía en función del estado de la búsqueda. Es difuso porque las soluciones son evaluadas, además de la función objetivo, mediante una valoración difusa que representa algún concepto subjetivo o abstracto, y gracias a esto el método es capaz de capturar el comportamiento cualitativo de otras metaheurísticas basadas en búsqueda por entornos, con lo cual, es posible tener un herramienta genérica de metaheurísticas simples de búsqueda local.

La estrategia se implementó en el lenguaje C++ y como marco de computación paralela se usó la biblioteca de dominio público PVM, que brinda un ambiente de programación concurrente, distribuido y de propósito general. Todo esto soportado en un cluster de 8 computadores.

Para resolver las tareas que conllevan a alcanzar los objetivos de la investigación, esta memoria se estructuró en cinco capítulos según los temas revisados y estudiados:

En el capítulo 1 se hace una recopilación de algunas de las metaheurísticas mas conocidas, con una explicación breve de cada una, y siguiendo una clasificación entre métodos de trayectoria y métodos de población. No es objetivo de este trabajo hacer un estudio exhaustivo de cada una de ellas, aquí solo se describen sus definiciones básicas. Para profundizar acerca de éstas hay una amplia bibliografía referenciada que puede servir de guía al interesado.

En el capítulo 2 se recogen conceptos básicos acerca de la computación paralela tales como: taxonomías de los computadores paralelos, modelos y métodos de la programación paralela, al-

goritmos paralelos, así como algunos de los ambientes de programación más conocidos, haciendo énfasis en la PVM, por ser la herramienta usada en este trabajo. Se abordaron las metaheurísticas paralelas explicando conceptos, estrategias, taxonomías y se hace una revisión del estado del arte.

En el capítulo 3 se describieron los conceptos básicos de la Soft-Computing y su utilidad al campo de las metaheurísticas. Se explica FANS, describiendo los elementos básicos del algoritmo y su esquema general. Luego, se aborda nuestra propuesta de estrategia mostrando los detalles de su implementación a través del pseudocódigo de sus componentes y explicando la base de reglas que definen sus diferentes esquemas de trabajo.

En los capítulos 4 y 5 se aborda la verificación experimental de la estrategia propuesta como herramienta de optimización. Dado su carácter heurístico, se evaluará su comportamiento en forma empírica a partir de experimentos computacionales sobre un conjunto de instancias del Problema de la Mochila y el Problema de la P-Mediana.

Finalmente se presentan las conclusiones obtenidas, se establecen las posibles líneas de investigación y se recoge la bibliografía usada para alcanzar los conocimientos básicos necesarios sobre estos temas que ha usado el autor y que han sido referenciadas en la memoria.

Capítulo 1

Metaheurísticas

En este capítulo se introducirán algunas definiciones básicas relativas a la optimización combinatoria para unificar términos que se utilizarán en la memoria. Luego se presentarán algunos ejemplos de metaheurísticas exitosas agrupados en métodos de trayectorias y en métodos de poblaciones, en los cuales se describirán las características esenciales de las mismas, y se proveen referencias que pueden servir de guía para el lector interesado. Al final se presentarán las conclusiones del capítulo.

1.1. Conceptos Básicos

Una instancia de un problema de optimización combinatoria P se define como $(\mathcal{S}, f)$ y está compuesta por

- un conjunto de variables $X = \{x_1, \ldots, x_n\}$
- dominios de las variables $D_1, \ldots, D_n$
- un conjunto de restricciones entre variables
- una función objetivo $f : D_1 \times \ldots \times D_n \to R^+$

El conjunto de las asignaciones factibles es $\mathcal{S} = \{s = (x_1, v_1), ..., (x_n, v_n)\}$ con $v_i \in D_i, i = 1, \ldots, n$ y tal que s satisface las restricciones. $\mathcal{S}$ recibe el nombre de espacio de búsqueda.

Para resolver el problema es necesario encontrar la solución que minimice/maximice el valor de la función objetivo. Es decir, asumiendo un problema de minimización, encontrar un $s^* \in \mathcal{S}$ tal que $f(s^*) \leq f(w)$ para todo $w \in \mathcal{S}$. La solución s^* se denomina el óptimo de $(\mathcal{S}, f)$.

Ahora, sea $(\mathcal{S}, f)$ una instancia de un problema de optimización. Se define la función o estructura de vecindario $\mathcal{N}$ como una aplicación $\mathcal{N} : \mathcal{S} \to 2^{\mathcal{S}}$, el cual determina para cada solución $s \in \mathcal{S}$ el conjunto $\mathcal{N}(s) \subseteq \mathcal{S}$ de soluciones "cercanas" (en algún sentido) a s.

El conjunto $\mathcal{N}(s)$ se denomina el vecindario de la solución s. Ejemplos de vecindarios surgen naturalmente por ejemplo, si se utiliza una función de distancia:

$$dist : \mathcal{S} \times \mathcal{S} \to R$$

Se puede definir el vecindario de una solución s como:

$$\mathcal{N}(s) = \{y \in \mathcal{S} \mid dist(s, y) \leq \epsilon\}$$

También se podría utilizar la siguiente definición:

$$N(s) = \{y \in \mathcal{S} \mid \rho(y, s) = True\}$$

donde $\rho(a, b)$ es algún predicado booleano.

En general, las soluciones y, se obtienen a partir de la aplicación de un operador $\mathcal{O}$, al que se suele denominar "Move" o "Pivote", que modifica en algún sentido la solución actual s para obtener nuevas soluciones. Este operador suele incluir algún procedimiento de aleatorización, con lo cual sucesivas aplicaciones de $\mathcal{O}(s)$ permiten obtener soluciones y diferentes.

Tiene sentido entonces hablar del vecindario de s bajo $\mathcal{O}$ y definirlo como:

$$\mathcal{N}_{\mathcal{O}}(s) \ = \{\hat{s}_i \mid \hat{s}_i = \mathcal{O}_i(s)\}$$

donde $\mathcal{O}_i(s)$ representa la i-esima aplicación de $\mathcal{O}$ sobre s.

Una vez definido de alguna manera apropiada el vecindario de una solución s como $\mathcal{N}(s)$, y dada una solución inicial s_0, es posible definir un esquema de mejoramiento iterativo que comience buscando alguna solución $s_1 \in N(s_0)$ tal que s_1 verifique alguna propiedad. Por ejemplo, que mejore el costo asociado.

Posteriormente, s_1 pasa a ser la solución actual y el proceso se repite hasta que se satisfaga algún criterio de parada o no sea posible obtener ninguna solución $s_{k+1} \in N(s_k)$ que satisfaga los requerimientos establecidos. En este momento, se establece que s_k es un óptimo local.

Esta clase de métodos reciben también el nombre de *búsqueda local* [5] y en la Fig. 1.1 (a) se muestra su pseudocódigo elemental. La rutina *mejorar(x)* devuelve, si es posible, una nueva solución y del vecindario tal que y sea mejor que s. En caso contrario, retorna "NO" y se devuelve la solución actual, la cual se corresponde con un mínimo/máximo local. Este esquema básico recibe el nombre de Mejoramiento Iterativo o Hill Climbing.

Existen, al menos, dos estrategias básicas para implementar la rutina *mejorar(x)*: la estrategia *First*, que retorna la primera solución del vecindario que mejore el costo, y la estrategia *Best*, que explora el vecindario de forma exhaustiva y retorna la solución con mejor costo. Ambas estrategias, tambien llamadas *reglas de pivot*, finalizan cuando se alcanza un óptimo local y por lo tanto la calidad final de la solución está fuertemente influenciada por las definiciones de $\mathcal{S}, f$ y $\mathcal{N}$.

Los métodos de búsqueda local se pueden clasificar a su vez en métodos de búsqueda local informada y no informada. Las estrategias de búsqueda por entornos no informadas son aquellas búsquedas locales que sólo prestan atención a la estructura de entornos en el espacio de búsqueda y no utilizan el valor de la función objetivo para la construcción del recorrido de búsqueda en las soluciones encontradas. Las búsquedas informadas son aquellas que, explícita o implícitamente, utilizan información de la evaluación de la función objetivo para conducir la búsqueda. Cuando esto sólo se hace en el entorno de la solución actual son llamadas búsquedas locales informadas.

```
Proc. Busqueda Local:
Begin
  s = solucion-inicial();
  While ( mejorar(s) = SI) ) Do
    s = mejorar(s);
  Od
  retornar(s);
End.
```

(a)

```
Proc. Busqueda Local MultiStart:
Begin
  s = solucion-inicial();
  restart = 0;
  While ( restart<MAX ) Do
    While ( mejorar(s) = SI) ) Do
      s = mejorar(s);
    Od
    If (s<best) Then
      best = s;
    Fi
    s = solucion-inicial();
    restart++;
  Od
  retornar(best);
End.
```

(b)

Figura 1.1: Esquemas básicos de Búsqueda Local

Una clasificación detallada e interesante sobre estos métodos aparece en [4].

A pesar de su simplicidad, estos métodos presentan un inconveniente importante: suelen quedar atrapados en mínimos o máximos locales. Como consecuencia es necesario extender los métodos mediante mecanismos adicionales que permitan hacer frente a esta situación.

La forma más simple de extender el esquema se presenta en la Fig.1.1 (b), donde simplemente se reinicia la búsqueda desde una nueva solución, cuando la actual ya no puede ser mejorada.

Esto se conoce como Arranque Múltiple (Multi-Start) [6, 7] y establece pautas para reiniciar de forma inteligente las búsquedas descendentes. Los procedimientos de búsqueda con arranque múltiple realizan varias búsquedas monótonas partiendo de diferentes soluciones iniciales. Una de sus formas más simples consiste en generar una muestra de soluciones iniciales o de arranque. Esto es equivalente a generar al azar una nueva solución de partida cada vez que la búsqueda quede estancada en el entorno de una solución optima local. Típicamente se puede hablar de dos fases. En la primera, se construye una solución y en la segunda se trata de mejorarla mediante la aplicación de un método de búsqueda, obteniéndose una nueva solución (que eventualmente puede ser la simple generación aleatoria de las soluciones), mientras que en otros ejemplos se emplean sofisticados métodos de construcción que consideran las características del problema de optimización para obtener soluciones iniciales de calidad.

Otro mecanismo para evitar el problema de los óptimos locales, consiste en incluir el esquema básico de búsqueda local en esquemas de un nivel superior, o sea en las metaheurísticas.

Si se mira a las metaheurísticas desde un nivel abstracto y se toman como "puras" quizás se puedan establecer algunas clasificaciones teniendo en cuenta sus mecanismos, sus principios, etc. Pero en la práctica es diferente, ya que al ser éstas desmontables de manera relativamente fácil en sus componentes y además brindar varias vías para reensamblarlas, propicia la hibridación entre ellas. Por ello, se hace difícil una clasificación en la que claramente se puedan insertar a todas.

Hay varias maneras de clasificar a las metaheurísticas, entre las más conocidas están las siguientes [8]: en función del origen del método, se habla de algoritmos "bioinspirados" (algoritmos genéticos, de colonia de hormigas, etc) vs. "no bioinspirados". Otra posibilidad es categorizarlos en función de la utilización o no de memoria, o en función del uso de una función objetivo estática o dinámica. Una clasificación interesante es la que surge al diferenciar los métodos que mantienen una única solución, frente a los que mantienen un conjunto o población de soluciones [9].

Con la intención de aclarar ideas y facilitar el estudio, se intenta aquí seguir una clasificación y así explicar de forma breve algunas de las metaheurísticas más conocidas. Para ello, se ha tomado la clasificación basada en métodos de trayectoria y métodos basados en poblaciones, motivado por el hecho que consideramos que esta brinda una descripción más clara de cada tipo de algoritmo.

No es objetivo de este trabajo hacer una recopilación exhaustiva de éstas, sólo se describen las características esenciales de aquellas que resultan más representativas, y dado que para cada una de estas metaheurísticas existe una bibliografía muy amplia se proveen referencias que pueden servir de guía para el lector interesado.

1.2. Metaheurísticas basadas en Trayectorias

Son aquellas metaheurísticas que establecen estrategias para recorrer el espacio de soluciones del problema transformando de forma iterativa soluciones de partida.

Una búsqueda sobre un espacio consiste en generar una sucesión de puntos del espacio pasando de uno a otro por medio de una serie de transformaciones o movimientos, como si siguiera una trayectoria. Un procedimiento de búsqueda para resolver un problema de optimización realiza recorridos sobre el entorno o espacio de las soluciones alternativas y selecciona la mejor solución encontrada en el recorrido.

El esquema general de un procedimiento de búsqueda basado en trayectorias o por entornos consiste en generar una solución inicial y seleccionar iterativamente un movimiento para modificar la solución hasta que se cumpla el criterio de parada. Las soluciones son evaluadas mientras se recorren y se propone la mejor solución del problema encontrada.

A continuación se presenta un resumen de algunas de las metaheurísticas mas representativas de esta clasificación.

1.2.1. Templado Simulado

El Templado o Enfriamiento Simulado (*Simulated Annealing, SA*), [10] [11] [12] es conocida como la primera de las metaheurísticas y uno de los primeros algoritmos con una estrategia explícita para escapar de óptimos locales.

Es el exponente más importante del tipo de metaheurísticas donde la probabilidad de aceptación es una función exponencial del empeoramiento producido. Un modo de evitar que

la búsqueda local finalice en óptimos locales, hecho que suele ocurrir con los algoritmos tradicionales de búsqueda local, es permitir que algunos movimientos se produzcan hacia soluciones peores. Pero si la búsqueda está avanzando realmente hacia una buena solución, estos movimientos de escape de óptimos locales deben realizarse de un modo controlado. Esto aquí se realiza controlando la frecuencia de los movimientos de escape mediante una función que hará disminuir la probabilidad de estos movimientos hacia soluciones peores conforme avanza la búsqueda, y por tanto se está más cerca, previsiblemente, del óptimo local.

El fundamento de este control se basa en el trabajo de Metropolis en el campo de la termodinámica estadística realizado en 1953, quien modeló el proceso de enfriamiento, simulando los cambios energéticos en un sistema de partículas conforme decrece la temperatura hasta que converge a un estado estable (congelado). Las leyes de la termodinámica dicen que a una temperatura t la probabilidad de un incremento energético de magnitud δE se puede aproximar por:

$$P[\delta E] = exp(-\delta E/k \times t) \tag{1.1}$$

siendo k una constante física denominada constante de Boltzmann.

En el modelo de Metropolis se genera una perturbación aleatoria en el sistema y se calculan los cambios de energía resultantes: si hay una caída energética, el cambio se acepta automáticamente; por el contrario, si se produce un incremento energético, el cambio será aceptado con una probabilidad indicada por la anterior expresión.

El Templado Simulado es un método de búsqueda por entornos caracterizado por un criterio de aceptación de soluciones vecinas que se adapta a lo largo de su ejecución y pertenece a una clase más amplia de algoritmos conocidos como algoritmos de umbral.

```
Proc. Algoritmo de Umbral:
Begin
  s_a = solucion-inicial();
  While ( no-finalizacion ) Do
    s_v = GenerarVecino(s_a);
    If (f(s_v - s_a) < t_k) Then
      s_a = s_v;
    Fi
    k=k+1;
  Od
End.
```

Figura 1.2: Algoritmo de Umbral

El esquema básico de un algoritmo de umbral se presenta en la Fig. 1.2. En cada iteración el algoritmo selecciona una solución vecina de la actual y compara la diferencia entre los costos. Si esta diferencia es menor que cierto umbral, la solución vecina pasa a ser la solución actual y el proceso se repite. La secuencia $(t_k,\ k = 0, 1, 2, \cdots)$ denota los umbrales y se cumple que $t_k = c_k,\ k = 0, 1, 2, \cdots$, donde $c_k \geq 0, c_k \geq c_{k+1}$ y $lim_{k\to\infty} c_k = 0$. El uso de umbrales positivos provoca que soluciones mucho peores sean aceptadas en forma limitada. A medida que el algoritmo progresa, los valores de umbral son reducidos hasta alcanzar 0, momento en el cual únicamente las soluciones que mejoren el costo son aceptadas.

En el Templado Simulado, el valor de t_k es una variable aleatoria que sigue cierta función

de probabilidad. TS utiliza umbrales aleatorios con valores entre 0 e infinito y la aceptación de soluciones peores que la actual esta gobernada por el siguiente criterio:

$$rand(0,1) < exp^{(f(x_{i+1})-f(x_i))/T} \tag{1.2}$$

donde el valor de T representa un parámetro que recibe el nombre de "temperatura" y $rand(0,1)$ es un número aleatorio entre 0 y 1 con distribución uniforme.

La estrategia del TS es comenzar con una temperatura inicial "alta", lo cual proporciona una probabilidad también alta de aceptar movimientos de empeoramiento. En cada iteración se reduce la temperatura y por lo tanto las probabilidades de aceptar soluciones peores también disminuyen. De este modo, se comienza la búsqueda con una etapa de exploración, a la que continúa una etapa de explotación, a medida que la ejecución avanza.

Se considera que el esquema de enfriamiento es uno de los elementos cruciales en la efectividad de TS y bajo ciertas condiciones, está probado que converge al óptimo global con probabilidad tan cercana a 1 como se desee. Desafortunadamente, en la práctica dichas condiciones son imposibles de alcanzar.

TS más que un algoritmo es una estrategia heurística que necesita de varias decisiones para que quede totalmente diseñado y las cuales tienen gran influencia en la calidad de las soluciones generadas.

Estas decisiones pueden diferenciarse en genéricas y específicas:

- *Decisiones genéricas:* Están relacionadas con los parámetros que dirigen el programa de enfriamiento, incluyendo los valores máximo y mínimo que podrá tomar la temperatura, la velocidad a la que se reducirá y las condiciones de parada.
- *Decisiones específicas:* Se refieren a la definición del espacio de soluciones, la estructura de los entornos y la función de coste, así como a la elección de la solución inicial.

Cualquier implementación de búsqueda local puede convertirse en una implementación TS al elegir elementos del entorno de modo aleatorio, aceptar automáticamente todos los movimientos hacia una mejor solución y aceptar los movimientos a una solución peor de acuerdo con una probabilidad dada por la función de probabilidad de Metrópolis.

Díaz et al. [10] plantean que su principal desventaja está en que puede necesitar mucho tiempo de cálculo cuando los problemas son grandes y complejos. Como ello está relacionado directamente con el tamaño del espacio de soluciones, se trata de reducir éste, que podría ser a través de un preprocesamiento mediante un conjunto de reglas de reducción, o reduciendo directamente la proporción del entorno a ser explorado en cada iteración.

Problemas y Aplicaciones:

Se ha aplicado a varios problemas de optimización combinatoria en los que se destacan los problemas de localización, empaquetado, planificación, asignación, y localización entre otros. Una recopilación sobre esto se puede ver en [10, 13].

En el ámbito de las aplicaciones finales se ha usado en sistemas para la reorganización de la escolarización en Portugal, a problemas de telecomunicaciones, de física, de biología, etc.

Actualmente este método se usa más como componente de otras metaheurísticas que por sí solo.

1.2.2. Búsqueda Voraz Adaptable Aleatoria

La metaheurística llamada Procedimiento de Búsqueda Voraz Adaptable Aleatoria (*Greedy Randomized Adaptive Search Procedures, GRASP*) [14] [15][16] de Feo y Resende, es un método multiarranque, en el cual cada iteración consiste en la construcción de una solución factible aleatoria, y con criterios adaptables para la elección de los elementos a incluir en la solución, seguida de una búsqueda local usando la solución construida como el punto inicial de la búsqueda (ver Fig. 1.3). Este procedimiento se repite varias veces y la mejor solución encontrada sobre todas las iteraciones se devuelve como resultado.

```
Proc. GRASP:
Begin
  entrada();
  Repeat Until ( k = k_max ) Do
    solucion = ConstruccionGreedy(semilla);
    solucion = busquedaLocal(solucion);
    actualizaSolucion(solucion, mejorSolucion);
  Od
  retornar(mejorSolucion);
End.
```

Figura 1.3: Algoritmo de Búsqueda Voraz Adaptable Aleatoria

En varias iteraciones se construyen soluciones de alta calidad que son posteriormente procesadas para conseguir otras mejores.

Cada iteración consiste en dos fases:

Fase de construcción: se aplica un procedimiento heurístico constructivo para obtener una buena solución inicial, donde:

- Una solución factible es construida en varios pasos adicionando un elemento por paso.
- En cada paso la elección del próximo elemento para ser añadido a la solución parcial viene determinada por una función *greedy*, esto es, se añade el mejor de los elementos disponibles.
- Esta función mide el beneficio de añadir cada uno de los elementos según la función objetivo

Fase de mejora: La solución se mejora mediante un algoritmo de búsqueda local, para ello puede usarse:

- Optimizar localmente con alguna metaheurística
- Limitar la búsqueda a k-vecindades

Veamos en detalle aspectos relevantes de estas dos fases:

Fase de Construcción: Los mecanismos usados construyen una solución incorporando un elemento cada vez. En cada paso del proceso de construcción, se tiene a la mano una solución

parcial. Un elemento que pueda seleccionarse como parte de una solución parcial se llama elemento candidato.

Para determinar que elemento candidato se va a seleccionar a continuación para incluirse en la solución, generalmente se hace uso de una función *greedy* de evaluación que mide la contribución local incremental en la función de coste de cada elemento a la solución parcial.

La evaluación de los elementos por esta función permite la creación de una lista restringida de candidatos (RCL) [14]. Tal lista contiene un conjunto de elementos candidatos con los mejores valores de la función. El siguiente candidato a ser agregado a la solución se selecciona al azar de la lista restringida de candidatos. Dicha lista puede consistir de un numero fijo de elementos (restricción por cardinalidad) o elementos con los valores de la función dentro de un rango dado.

Se puede también mezclar una construcción al azar con una construcción *greedy* de la siguiente manera: elegir secuencialmente un conjunto parcial de elementos candidato al azar y después completar la solución usando un algoritmo *greedy*.

Hay también una variación sobre el enfoque de RCL basado en el valor. En este procedimiento, llamado función de sesgo [15], en vez de seleccionar el elemento de la RCL al azar con iguales probabilidades asignadas a cada elemento, se asignan diferentes probabilidades, favoreciendo elementos bien evaluados. Los elementos de la RCL se ordenan de acuerdo a los valores de la función *greedy*.

Para obtener un balance entre la convergencia rápida del algoritmo *greedy* y de la gran diversidad del algoritmo aleatorio, se acostumbra usar un valor de umbral, $\alpha \in [0, 1]$ en la función de coste. En el caso de $\alpha = 0$ corresponde a un algoritmo *greedy* puro, mientras que $\alpha = 1$ es equivalente a una construcción aleatoria. El parámetro a controla el grado de voracidad o aleatoriedad del algoritmo. Ya que no se conoce a priori qué valor usar, una estrategia razonable es seleccionar al azar un valor diferente en cada iteración GRASP, esto puede hacerse usando una probabilidad uniforme [15].

Existen diferentes variantes de este esquema entre las que podemos destacar el método heurístico conocido como semi-greedy debido a Hart y Shogan [14]. Este método sigue el esquema multiarranque basado en hacer aleatoria una evaluación *greedy* en la construcción, pero no tiene una fase de búsqueda local o mejora. Actualmente se están implementando versiones de este método denominadas Reactive GRASP en donde el ajuste de los parámetros necesarios (básicamente los que determinan la RCL) se realiza de forma dinámica según el estado de la búsqueda [17].

La principal desventaja del GRASP "puro" es su falta de estructuras de memoria. Las iteraciones de GRASP son independientes y no utilizan las observaciones hechas durante iteraciones previas. Un remedio para esto es combinarla con el método de Reencadenamiento de Trayectorias [18].

Problemas y Aplicaciones:

La primera aplicación de GRASP fue en recubrimientos de conjuntos [15] en 1989, y, a partir de entonces, ha sido aplicado a un amplio rango de tipos de problemas, tales como enrutamiento, lógica, cubrimiento y partición, localización, árbol mínimo de Steiner, optimización en grafos, etc.

También se aplica en áreas de la manufactura, transporte, sistemas de potencia, telecomunicaciones, dibujo de grafos y mapas, lenguaje, estadística, biología, programación matemática, empaquetado, etc. Un estudio pormenorizado de GRASP y sus aplicaciones puede encontrarse en [19].

1.2.3. Búsqueda Tabú

La Búsqueda Tabú (*Tabu Search, TS*) [20] [21], es una de las estrategias que tratan de utilizar la memoria del proceso de búsqueda para mejorar su rendimiento evitando que la búsqueda se concentre en una misma zona del espacio.

Está fundamentada en las ideas expuestas por F. Glover en prohibir temporalmente soluciones muy parecidas a las últimas soluciones del recorrido. En el origen del método el propósito era sólo evitar la reiteración en una misma zona de búsqueda recordando las últimas soluciones recorridas. Sin embargo, posteriormente se han realizado diversas propuestas para rentabilizar la memoria a medio o largo plazo.

BT es el origen del enfoque basado en memoria y estrategia intensiva en la literatura de las metaheurísticas. También es responsable de enfatizar el uso de los diseños estructurados para explotar los patrones históricos de la búsqueda, de forma opuesta a los procesos que confían casi exclusivamente en la aleatoriedad.

Hace menos énfasis a la aleatoriedad, y generalmente se usa en un modo altamente restringido, con el supuesto de que la búsqueda inteligente debería estar basada en formas más sistemáticas de dirección. Por consiguiente, muchas de sus implementaciones son en gran parte deterministas [20].

La Búsqueda Tabú se caracteriza por dos aspectos principales:

- Presenta un mecanismo de generación de vecinos modificado que evita la exploración de zonas del espacio de búsqueda que ya han sido visitadas: generación de entornos tabú restringidos.
- Emplea mecanismos para mejorar la capacidad del algoritmo para la exploración-explotación del espacio de búsqueda.

Para realizar las dos tareas anteriores, hace uso de unas estructuras de memoria adaptables:

- Memoria de corto plazo o Lista tabú, que permite guiar la búsqueda de forma inmediata, desde el comienzo del procedimiento. (Generación de entornos tabú restringidos).
- Memoria de largo plazo, que guarda información que permite guiar la búsqueda a posteriori, después de una primera etapa en la que se ha realizado una o varias ejecuciones del algoritmo en las que se ha aplicado la memoria a corto plazo.

La información guardada en esta memoria se usa para dos tareas distintas:

- Intensificar la búsqueda, consiste en regresar a buenas regiones ya exploradas para estudiarlas más a fondo. Para ello se favorece la aparición de aquellos atributos asociados a buenas soluciones encontradas.
- Diversificar la búsqueda, consiste en visitar nuevas áreas no exploradas del espacio de soluciones. Para ello se modifican las reglas de elección para incorporar a las soluciones atributos que no han sido usados frecuentemente.

El método (ver Fig. 1.4) comienza desde una solución inicial *sa*, construye un entorno *N(sa)* y selecciona la mejor solución *sb* que pertenece a *N(s)*. De la misma manera que un movimiento

```
Proc B. Tabu:
Begin
  s_a = solucion-inicial();
  ListaTabu = {};
  While ( no-finalizacion ) Do
    s_b = Mejorar(s_a, N(s_a), ListaTabu);
    s_a = s_b;
    Actualizar(ListaTabu);
  Od
  retornar(s);
End.
```

Figura 1.4: Algoritmo de la Búsqueda Tabu

m permitió transformar $sa \leftarrow sb$, existirá el movimiento inverso m^{-1} para transformar $sb \leftarrow sa$. Con el objetivo de evitar ciclos, el movimiento m^{-1} se agrega a la lista tabú. El procedimiento continua desde sb hasta que cierta condición de parada se verifique.

Los movimientos permanecen como tabú sólo durante un cierto número de iteraciones, aunque hay excepciones. Cuando un movimiento tabú proporciona una solución mejor que cualquier otra previamente encontrada, su clasificación tabú puede eliminarse. La condición que permite dicha eliminación se denomina criterio de aspiración.

Las restricciones tabú y el criterio de aspiración juegan un papel dual en la restricción y guía del proceso de búsqueda. Las restricciones tabú, permiten que un movimiento sea admisible si no está clasificado como tabú, mientras que si el criterio de aspiración se satisface, permite que un movimiento sea admisible aunque esté clasificado como tabú.

La longitud de la lista tabú es un parámetro. Su definición no es trivial ya que si es demasiado pequeño pueden ocurrir ciclos, mientras que si es demasiado grande, puede restringir en exceso la búsqueda. Esta lista recibe también el nombre de memoria de corto plazo. En ocasiones, y como base para las estrategias de exploración/explotación, también se utilizan memorias de término intermedio y largo plazo.

Existen otros elementos más sofisticados que han dado muy buenos resultados en algunos problemas. Entre ellos se pueden destacar [21]:

- *Movimientos de Influencia:* Son aquellos movimientos que producen un cambio importante en la estructura de las soluciones. Generalmente, en un procedimiento de búsqueda local, la búsqueda es dirigida mediante la evaluación de la función objetivo. Sin embargo, puede ser muy útil el encontrar o diseñar otros evaluadores que guíen a ésta en determinadas ocasiones. Los movimientos de influencia proporcionan una evaluación alternativa de la bondad de los movimientos al margen de la función objetivo. Su utilidad principal es la determinación de estructuras subyacentes en las soluciones. Esto permite que sean la base para procesos de intensificación y diversificación a largo plazo.
- *Oscilación Estratégica:* Opera orientando los movimientos en relación a una cierta frontera en donde el método se detendría normalmente. Sin embargo, en vez de detenerse, las reglas para la elección de los movimientos se modifican para permitir que la región al otro lado de la frontera sea alcanzada. Posteriormente se fuerza al procedimiento a regresar a la zona inicial. El proceso de aproximarse, traspasar y volver sobre una determinada frontera crea un patrón de oscilación que da nombre a esta técnica. Una implementación sencilla consiste en considerar la barrera de la *factibilidad/no factibilidad* de un problema dado. Implementa-

ciones más complejas pueden crearse identificando determinadas estructuras de soluciones que no son visitadas por el algoritmo y considerando procesos de *construcción/destrucción* asociados a éstas. La oscilación estratégica proporciona un medio adicional para lograr una interacción muy efectiva entre intensificación y diversificación.

- *Elecciones Probabilísticas:* Normalmente la Búsqueda Tabú se basa en reglas sistemáticas en lugar de decisiones al azar. Sin embargo, en ocasiones se recomienda permitir el azar en algunos procesos para facilitar la elección de buenos candidatos o cuando no está clara la estrategia a seguir (quizá por tener criterios de selección enfrentados). La selección aleatoria puede ser uniforme o seguir una distribución de probabilidad construida empíricamente a partir de la evaluación asociada a cada movimiento.

- *Umbrales Tabú:* El procedimiento conocido como Tabú Thresholding (TT) se propone para aunar ideas que provienen de la Oscilación Estratégica y de las Estrategias de Listas de Candidatos en un marco sencillo que facilite su implementación. El uso de la memoria es implícito en el sentido que no hay una lista tabú en donde anotar el status de los movimientos, pero la estrategia de elección de los mismos evita dar vueltas indefinidamente. TT utiliza elecciones probabilísticas y umbrales en las listas de candidatos para implementar los principios de la Búsqueda Tabú.

- *Reencadenamiento de Trayectorias:* Este método [18] trata de volver a unir dos buenas soluciones mediante un nuevo camino. Así, si en el proceso de búsqueda se ha encontrado dos soluciones x e y con un buen valor de la función objetivo, se puede considerar el tomar x como solución inicial e y como solución final e iniciar un nuevo camino desde x hasta y. Para seleccionar los movimientos no se considera la función objetivo o el criterio que se haya estado utilizando sino que van incorporando a x los atributos de y hasta llegar a ésta. En algunas implementaciones se ha considerado el explorar el entorno de las soluciones intermedias para dar más posibilidad al descubrimiento de buenas soluciones.

Problemas y Aplicaciones:

De sus áreas de aplicación tradicionales como: problemas de teoría de grafos, localización y asignación, planificación y enrutamiento se ha movido a otras nuevas: optimización continua, optimización multi-criterio, programación estocástica, programación entera mixta, problemas de decisión en tiempo real, etc.

También se usa en solucionar problemas de transporte, diseño, optimización de grafos, telecomunicaciones, lógica e Inteligencia Artificial, optimización de estructuras, producción, inventario e inversión. En el libro de Glover y Laguna [21] aparece una recopilación de varias aplicaciones. También en [22] aparece una lista actualizada y pormenorizada de aplicaciones del método.

1.2.4. Búsqueda por Entornos Variables

Las metaheurísticas de entorno variable modifican de forma sistemática el tipo de movimiento con el objeto de evitar que la búsqueda se quede atrapada en óptimos locales debido a una estructura de entornos rígida.

La Búsqueda por Entornos Variables, *(Variable Neighborhood Variable, VNS)* [23, 24, 25, 26, 27], es una metaheurística que está basada en un principio simple: cambiar sistemáticamente de estructura de entornos dentro de la búsqueda para escapar de los mínimos locales.

El VNS básico obtiene una solución del entorno de la solución actual, ejecuta una búsqueda monótona local desde ella hasta alcanzar un óptimo local, que reemplaza a la solución actual si ha ocurrido una mejora y modifica la estructura de entorno en caso contrario. Un pseudocódigo de este esquema se encuentra en la Figura 1.5.

```
Proc. VNS:
Begin
  /* N_k, k = 1, ..., k_max, estructuras de vecindarios */
  /* s_a: solucion actual */
  /* s_p: solucion vecina de s_a */
  /* s_ol: solucion localmente optima */
  Repeat Until ( finalizacion ) Do
    k=1;
    Repeat Until ( k = k_max ) Do
      /* generar vecino s_p del k-esimo vecindario de s_a (s_p ∈ N_k(s_a)) */
      s_p = ObtenerVecino(s_a,N_k);
      s_ol = BusquedaLocal(s_p);
      If (s_ol es mejor que s_a) Then
        s_a = s_ol;
        Else
          k=k+1;
        Fi
      Od
    Od
  End.
```

Figura 1.5: Algoritmo de Búsqueda por Entornos Variable

La VNS está basada en tres hechos simples:

- Un mínimo local con una estructura de entornos no lo es necesariamente con otra.
- Un mínimo global es mínimo local con todas las posibles estructuras de entornos.
- Para muchos problemas, los mínimos locales con la misma o distinta estructura de entornos están relativamente cerca.

Esta última observación, que es empírica, implica que los óptimos locales proporcionan información acerca del óptimo global. Por ejemplo, puede ser que ambas soluciones tengan características comunes. Sin embargo, generalmente no se conoce cuáles son esas características. Es procedente, por tanto, realizar un estudio organizado en las proximidades de este óptimo local, hasta que se encuentre uno mejor.

Estos hechos sugieren el empleo de varias estructuras de entornos en las búsquedas locales para abordar un problema de optimización. El cambio de estructura de entornos se puede realizar de forma determinística, estocástica, o determinística y estocástica a la vez.

La Búsqueda Descendente por Entornos Variables *(Variable Neighbourhood Descent, VND)*[25], cuyo algoritmo aparece en la Figura 1.6, aplica una búsqueda monótona, o sea de mejora por entornos, cambiando de forma sistemática la estructura de entornos cada vez que se alcanza un mínimo local.

La solución final proporcionada por este algoritmo es un mínimo local con respecto a todas las k_{max} estructuras de entornos, y por tanto la probabilidad de alcanzar un mínimo global es mayor que usando una sola estructura.

La búsqueda local de la VNS puede ser sustituida por la VND y forma lo que se conoce como Búsqueda de Entorno Variable General *(General Variable Neighbourhood Search, GVNS)* que ha

```
Proc VNDS:
Begin
  seleccionaVecindario(N_k);
  solucionInicial(s_init));
  While ( no − haya − mejora ) Do
    k = k + 1;
    Repeat Until ( k = k_max ) Do
      exploraVecindario(s_mejor);
      If (s_mejor es mejor que s_init) Then
        s_init = s_mejor;
        k = 1;
        Else
          k = k + 1;
        Fi
      Od
    Od
  End.
```

Figura 1.6: Algoritmo de Búsqueda Descendente por Entornos Variables

dado lugar a muchas aplicaciones exitosas aparecidas recientemente y referenciadas en [25].

También se le han hecho extensiones que constituyen mejoras prácticas de la VNS que han permitido resolver con éxito problemas muy grandes: la Búsqueda de Entorno Variable Sesgada *(Skewed Variable Neighbourhood Search, SVNS)* y la Búsqueda de Entorno Variable Paralela *(Parallel Variable Neighborhood Search, PVNS)*, así como hibridación con otras metaheurísticas como Tabú, Multi-arranque y GRASP.

Problemas y Aplicaciones:

La VNS se ha aplicado a diversos problemas clásicos de la Inteligencia Artificial, en [25] aparecen referenciados muchos de ellos, y algunos se muestran a continuación. Entre ellos destacan los problemas de satisfacción, el aprendizaje en redes bayesianas, y los de clasificación y planificación.

También su aplicación a multitud de problemas de optimización combinatoria, entre los que se encuentran los problemas de empaquetado, localización, p-mediana, y de rutas. También se proponen y prueban diversas VNS paralelas para este problema. Se aplica la VNS al problema del p-centro, al problema múltiple de Weber y al problema de asignación cuadrática. También encontramos la aplicación de la VNS Reducida y la VNDS para el problema simple de localización de plantas.

Otros problemas de optimización combinatoria importantes que tambien han sido abordados, sobre todo en el contexto de la planificación logística, son los problemas de rutas. Tanto las versiones clásicas del problema del viajante de comercio, del problema de ruta de vehículo y del problema de ruta de arcos.

Existe una propuesta de combinación de la VNS y la Búsqueda Tabú que usa varias estructuras de entornos para el problema del ciclo mediano.

Dentro de los problemas de optimización continua se ha probado la VNS en el problema de programación bilineal (en particular para una aplicación al problema del refinado en la industria del petróleo) y en el problema de optimización minimax continuo del diseño de un código polifásico [24].

1.2.5. Búsqueda Local Iterativa

La Búsqueda Local Iterativa *(Iterative Local Search, ILS)*[28] [29] es una metaheurística que propone un esquema en el que se incluye una heurística base para mejorar los resultados de la repetición de dicha heurística. Esta idea ha sido propuesta en la literatura con distintos denominaciones [29], como descenso iterado, grandes pasos con cadenas de Markov, Lin-Kerningan iterado, búsqueda perturbada o ruidosa o la búsqueda de entorno variable con agitación donde la solución aportada por una heurística de búsqueda por entornos es agitada para producir una solución de partida para la heurística de búsqueda.

Esta estrategia actúa de la siguiente forma: dada una solución obtenida por la aplicación de la heurística base, se aplica un cambio o alteración que da lugar a una solución intermedia. La aplicación de la heurística base a esta nueva solución aporta una nueva solución que, si supera un test de aceptación, pasa a ser la nueva solución alterada. Aunque la heurística base incluida suele ser una búsqueda local, se ha propuesto aplicar cualquier otra metaheurística, determinística o no. De esta forma, el proceso se convierte en una búsqueda estocástica por entornos donde los entornos no se explicitan sino que vienen determinados por la heurística base.

La aplicación de la ILS necesita de la definición de cuatro componentes:

- Una solución inicial (usualmente, aleatoria).
- Un procedimiento de modificación que aplica un cambio brusco sobre la solución actual para obtener una solución intermedia.
- Un procedimiento de búsqueda local.
- Un criterio de aceptación que decide a qué solución se aplica el procedimiento de modificación.

```
Proc. ILS:
Begin
  /* s_a: solucion actual */
  /* s_p: solucion intermedia o perturbada */
  /* s_ol: solucion localmente optima */
  /* H: historia o memoria */
  s_1 = solucion-inicial();
  s_a = BusquedaLocal(s_1);
  Repeat Until ( finalizacion ) Do
    s_p = Perturbar(s_a,H);
    s_ol = BusquedaLocal(s_p);
    s_a = Aceptar?(s_a,s_ol,H);
  Od
End.
```

Figura 1.7: Algoritmo de Búsqueda Local Iterativa

Tal como muestra el algoritmo en la Figura 1.7 en la generación de la solución inicial la búsqueda aplicada a la solución inicial s_1 da el punto de arranque s_a del camino en el conjunto S. Comenzar con un buen s_a puede ser importante si se necesita lograr soluciones de alta calidad tan rápido como sea posible.

Las elecciones estándares para s_a son bien una solución inicial aleatoria, bien una solución obtenida de una heurística constructiva *greedy*. Una solución inicial *greedy* toma, en la media, menos pasos de mejora y por tanto, menos tiempo del procesador que las soluciones aleatorias

y cuando se combina con una búsqueda local éstas casi siempre resultan ser soluciones de mejor calidad. Se ha probado que la solución inicial tiene una influencia significativa sobre la calidad para ejecuciones cortas y medianas. Por ello, se recomienda soluciones *greedy* al inicio cuando se necesitan rápidamente soluciones de bajo costo; y para ejecuciones de instancias grandes la solución inicial es menos relevante, se puede elegir la que resulte mas fácil de poner en práctica.

ILS escapa de mínimos locales aplicando perturbaciones al mínimo local actual. La fuerza de la perturbación se refiere al número de componentes de la solución que son modificados.

El mecanismo de modificación deberá ser lo suficientemente fuerte para permitir salir del mínimo local actual y habilitar la búsqueda local para encontrar el nuevo y posible mejor mínimo local. Pequeñas modificaciones permiten al algoritmo de búsqueda ejecutarse rápido, requiriendo solo pocos pasos para alcanzar el próximo óptimo local y la diversificación del espacio de búsqueda estará muy limitado. En aquellas modificaciones demasiado grandes el efecto será similar a arrancar desde una nueva solución generada aleatoriamente y las mejores soluciones solo serán encontradas con una probabilidad muy baja.

Se obtendrán mejores resultados si las perturbaciones tienen en cuenta propiedades del problema y son bien acoplados al algoritmo de la búsqueda local. La perturbación aplicada a la solución actual también puede hacerse dependiente de la historia de la búsqueda, inspirado en el contexto de la Búsqueda Tabú.

El criterio de aceptación se usa para decidir cual solución de la próxima perturbación debería ser aplicado. Un aspecto importante de éste es que permite lograr un balance entre la intensificación y la diversificación de la búsqueda.

En la práctica la complejidad funcional de ILS está en su dependencia de la historia de la búsqueda; por ello, muchos de los trabajos relacionados al método no la tienen en cuenta [30]; aunque estudios recientes [31] han demostrado que su uso mejora su rendimiento.

La fuerza potencial de ILS consiste en la manera en que hace el muestreo no en el espacio total de soluciones sino en un conjunto más pequeño de óptimos locales. La eficiencia de este muestreo depende de las perturbaciones y del criterio de aceptación, que aun con las implementaciones más simples de éstas, ILS es mucho mejor que un re-arranque aleatorio. Y será mucho mejor si los módulos de ILS son optimizados. Primero, el criterio de aceptación puede ser ajustado empíricamente como un Templado Simulado sin conocer nada acerca del problema. Y segundo, la perturbación puede incorporar tanta información específica del problema como el diseñador esté deseoso de poner en éste. En la práctica la siguiente regla sirve como guía: una buena perturbación transforma una excelente solución en un excelente punto de arranque para una búsqueda local.

Su otra ventaja es la velocidad, se pueden realizar k búsquedas locales dentro de un ILS mucho mas rápido que si las k búsquedas locales son ejecutadas dentro de un arranque aleatorio. Además es simple, fácil de implementar, robusto y altamente efectivo.

Problemas y Aplicaciones:

ILS ha sido aplicado con éxito a una variedad de problemas de optimización combinacional y promete que pueda resolver problemas complejos reales en la industria, los servicios, en áreas que van desde las finanzas a la producción, administración, y la logística.

Los algoritmos ILS para TSP definen el estado del arte en la solución de este problema. Igual ocurre con los problemas de planificación, donde se han obtenido excelentes resultados [31]. Se

ha probado satisfactoriamente [28] en asignación cuadrática (QAP), particionamiento de grafos, máxima satisfacción (MAX-SAT), y en el problema del árbol de Steiner restringido a grafos [29].

También se está estudiando [28] aplicar a problemas donde las restricciones son muy estrictas, en problemas multiobjetivos y en problemas dinámicos o de tiempo real donde los datos del problema varían durante la solución del proceso.

1.2.6. Búsqueda Local Guiada

La metaheurística de Búsqueda Local Guiada *(Guided Local Search, GLS)* [32],[33] consiste básicamente en una secuencia de procedimientos de búsqueda local; al analizar cada uno de ellos se modifica la función objetivo penalizando determinados elementos que aparecen en el óptimo local obtenido en el último paso, estimulando de esta forma la diversificación de la búsqueda.

GLS, desarrollada por Voudoris [34] surge como una extension de GENET, una red neuronal para la satisfacción de restricciones. Mientras GENET intenta encontrar cualquier solución que satisfaga a todas las restricciones, GLS intenta encontrar soluciones óptimas acorde a una función dada.

El principio básico de GLS es ayudar a la búsqueda moviéndose gradualmente afuera del óptimo local, cambiando el escenario de búsqueda. El conjunto de soluciones y la estructura del vecindario son mantenidos fijos, mientras que la función objetivo es cambiada dinámicamente con el objetivo de hacer "menos deseable" el óptimo local actual.

El mecanismo usado por GLS se basa en soluciones características, las cuales tienen propiedades o características que pueden ser usadas para discriminar entre las soluciones. Por ejemplo, en el problema del viajante de comercio (TSP) podrían ser los arcos entre pares de ciudades. Una función $I_i(s)$ indica si la característica i está presente en la solución s:

$$I_i(s) = \begin{cases} 1 & : i \text{ presente en la solución } s \\ 0 & : i \text{ no está presente en la solución } s \end{cases}$$

La función objetivo se modifica para producir una nueva función f' añadiendo un término que depende de las m características:

$$f'(s) = f(s) + \lambda \sum_{i=1}^{m} p_i * I_i(s)$$

Donde:

p_i, son los parámetros de penalidad, que ponderan la importancia de las características: conforme mayor es p_i, mayor es la importancia de la característica i, y por tanto, mayor el costo que tiene la característica en la solución.

λ, es el parámetro de regularización, que balancea la relevancia de las características con respecto a la función objetivo original.

Como se ve en la Figura 1.8 el algoritmo arranca de una solución inicial y se aplica una búsqueda local hasta que se alcanza un mínimo local. Entonces el vector $p = (p_1, ...p_m)$ de penalidades se actualiza incrementando alguna de las penalidades y la búsqueda local se comienza

```
Proc. GLS:
Begin
  s = GxSolucionInic();
  While ( no-finalizacion ) Do
    s = BusqLocal(s, f');
    For (i con max Util(s, i)) Do
      p_i = p_i + 1;
    Od
    Actualizar(f', p);
  Od
End.
```

Figura 1.8: Algoritmo de Búsqueda Local Guiada

de nuevo. Las características se penalizan acorde a una función de utilidad:

$$Util(s,i) = I_i(s) * \frac{c_i}{1+p_i}$$

Donde c_i, se refiere a los costos asignados a cada característica i dando una evaluación heurística de la importancia relativa de las características respecto a las otras. A mayor costo, mayor la utilidad de la característica. No obstante, el costo es ajustado por el parámetro de penalidad para prevenir que el algoritmo esté siendo totalmente polarizado hacia el costo y hacer éste sensitivo a la historia de la búsqueda. La política implementada es que las características sean penalizadas con una frecuencia proporcional a su costo.

El procedimiento para actualizar las penalidades puede ser modificado añadiendo una regla multiplicativa a la regla incremental simple (ésta se aplica en cada iteración). Tiene la forma:

$$p_i \leftarrow p_i \times \alpha$$

donde,

$$\alpha \in]0,1]$$

Problemas y Aplicaciones:

GLS ha sido aplicada [32] exitosamente al problema del viajante de comercio (TSP), asignación cuadrática (QAP), satisfacción (SAT) y máxima satisfacción (MAX-SAT), problemas de asignación de frecuencias de radioenlace (RLPA), asignación generalizada (GAP), enrutamiento de vehículos (VRP), coloración de grafos (GCP), etc.

Se recomienda revisar en internet un sitio web [33] dedicado a GLS, desarrollado por sus creadores y principales investigadores.

1.3. Metaheurísticas basadas en Poblaciones

Las metaheurísticas basadas en poblaciones son las que conducen la evolución en el espacio de búsqueda de conjuntos de soluciones, usualmente llamados poblaciones, con la intención de

acercarse a la solución óptima con sus elementos. El aspecto fundamental de estas heurísticas consiste en la interacción entre los miembros de la población frente a las búsquedas que se guían por la información de soluciones individuales.

Se distinguen por la forma en que combinan la información proporcionada por los elementos de la población para hacerla evolucionar mediante la obtención de nuevas soluciones. En una búsqueda en grupo o basada en poblaciones se sustituye la solución actual que recorre el espacio de soluciones, por un conjunto de soluciones que lo recorren conjuntamente interactuando entre ellas. Además de los movimientos aplicables a las soluciones que forman parte de este conjunto, denominado grupo o población de búsqueda, se contemplan otros operadores para generar nuevas soluciones a partir de las ya existentes.

Como ejemplos de este tipo de estrategia se describirán los siguientes métodos:

1.3.1. Algoritmos Genéticos

Los Algoritmos Genéticos *(Genetic Algorithms, GA)* [35, 36, 37] son una familia de modelos computacionales inspirados en los mecanismos de herencia y evolución natural.

La primera mención del término, y la primera publicación sobre una aplicación del mismo, se deben a una disertación de Bagley en 1967, que diseñó algoritmos genéticos para buscar conjuntos de parámetros en funciones de evaluación de juegos, y los comparó con los algoritmos de correlación, procedimientos de aprendizaje modelizados después de los algoritmos de pesos variantes de ese período. Pero es otro científico al que se considera creador de los Algoritmos Genéticos, John Holland, [38] que los desarrolló, junto a sus alumnos y colegas, durante las décadas de 1960 y 1970.

En contraste con las estrategias evolutivas y la programación evolutiva, el propósito original de Holland no era diseñar algoritmos para resolver problemas concretos, sino estudiar de un modo formal el fenómeno de la adaptación tal y como ocurre en la naturaleza, y desarrollar vías de extrapolar esos mecanismos de adaptación natural a los sistemas computacionales.

Estas técnicas mantienen un conjunto de soluciones potenciales, tienen algún proceso de selección basado en la calidad de las mismas e incorporan operadores que permiten modificar o generar nuevas soluciones.

Mediante una imitación del mecanismo genético de los organismos biológicos, los algoritmos genéticos exploran el espacio de soluciones asociado a un determinado problema. Se establece una codificación apropiada de las soluciones del espacio de búsqueda y una forma de evaluar la función objetivo para cada una de estas codificaciones. Las soluciones se identifican con individuos que pueden formar parte de la población de búsqueda. La codificación de una solución se interpreta como el cromosoma del individuo compuesto de un cierto número de genes a los que les corresponden ciertos alelos. El gen es la característica del problema; alelo, el valor de la característica; genotipo, la estructura y fenotipo, la estructura sometida al problema. Cada cromosoma es una estructura de datos que representa una de las posibles soluciones del espacio de búsqueda del problema. Los cromosomas son sometidos a un proceso de evolución que envuelve evaluación, selección, recombinación sexual o cruce y mutación. Después de varios ciclos de evolución la población deberá contener individuos más aptos.

Se consideran dos operaciones básicas: la mutación y el cruce. La mutación de un individuo consiste en modificar uno o varios genes cambiando al azar el alelo correspondiente. El cruce de dos

```
Proc AG:
Begin
  t = 0;
  inicializar(P(t));
  evaluar(P(t));
  While ( no-finalizacion ) Do
    t = t + 1;
    Seleccionar P(t) desde P(t-1);
    Hacer-Cruzamientos(P(t));
    Aplicar-Mutaciones(P(t));
  Od
  retornar mejor solucion;
End.
```

Figura 1.9: Esquema básico de un Algoritmo Genético

individuos, llamados padres, produce un individuo hijo tomando un numero k (elegido al azar) de genes de uno de los padres y los $t-k$ del otro. La población evoluciona de acuerdo a las estrategias de selección de individuos, tanto para las operaciones como para la supervivencia. La selección se puede hacer simulando una lucha entre los individuos de la población con un procedimiento que, dados dos individuos selecciona uno de ellos teniendo en cuenta su valoración (la función objetivo) y la adaptación al ambiente y a la población (criterios de diversidad, representatividad).

Mediante una imitación de este mecanismo, los algoritmos genéticos exploran el espacio de soluciones asociado a un determinado problema. En la Fig. 1.9 se presenta el esquema básico del algoritmo. En cada iteración t, se mantiene una *población* de *individuos* $P(\mathrm{t}) = \{x_1^t, \ldots, x_n^t\}$ donde cada individuo representa una potencial solución del problema a resolver. Cada solución x_i^t se evalúa para obtener cierta medida de su calidad o *fitness*. Luego, se genera una nueva población $P(t+1)$ tomando como base a los mejores individuos; algunos de los cuales sufren transformaciones a partir de la aplicación de operadores "genéticos".

Típicamente, estos operadores son: *cruzamiento* o *crossover* (nuevos individuos son generados a partir de la combinación de dos o más individuos), y *mutación* (nuevos individuos son generados a partir de pequeños cambios en un individuo).

La idea intuitiva del operador de cruce es permitir el intercambio de información entre individuos de la población. La mutación, en cambio, permite incorporar nueva información que no se puede generar a partir de la clausura transitiva del cruce sobre la población.

Se puede entender el mecanismo de selección si se plantea cada iteración del Algoritmo Genético como un proceso en dos etapas. En la primera, se aplica algún mecanismo de selección sobre la "población actual" para crear una "población intermedia"; y en la segunda etapa, a partir de esta población, se aplica cruce y mutación para generar una "nueva población".

El resultado del proceso de selección es un conjunto de entrecruzamiento formado por los mejores individuos de la población. En base a ellos, la etapa de cruce elegirá los padres y generará los hijos; sobre la salida de esta etapa, se ejecutará la etapa de mutación. La nueva población puede estar formada solamente por los hijos, o por padres e hijos en alguna proporción adecuada.

Para construir un AG se necesita:

- Una representación de las soluciones del problema.
- Que haya correspondencia entre genotipo y fenotipo.

- Una forma de crear una población inicial de soluciones.
- Una función de evaluación en términos de conveniencia o adaptación de la solución evaluada.
- Operadores genéticos.
- Una forma para seleccionar los individuos para ser padres.
- Una forma de cómo reemplazar a los individuos.
- Una condición de parada.

Para la representación se debe disponer de un mecanismo para codificar un individuo como un genotipo. Una vez elegida una representación, hay que tener en mente cómo los genotipos (codificación) serán evaluados y qué operadores genéticos habrá que utilizar.

El paso más costoso para una aplicación real es la evaluación de la función de adaptación. Puede ser una subrutina, un simulador, o cualquier proceso externo. Se pueden utilizar funciones aproximadas para reducir el costo de evaluación. Cuando hay restricciones, estas se pueden introducir en el costo como penalización y con múltiples objetivos se busca una solución de compromiso.

Para representar la mutación pueden tenerse uno o más operadores. Hay que tener en cuenta que debe permitir alcanzar cualquier parte del espacio de búsqueda, su tamaño debe ser controlado y debe producir cromosomas válidos, o sea, acordes al problema. En el caso del cruce también se pueden tener uno o más operadores. También debe tenerse en cuenta que los hijos deberían heredar algunas características de cada padre.

En la estrategia de selección se debe garantizar que los mejores individuos tengan una mayor posibilidad de ser padres (reproducirse) frente a los individuos menos buenos, pero también ser cuidadosos para dar una posibilidad de reproducirse a los individuos menos buenos. Estos pueden incluir material genético útil en el proceso de reproducción. Esta idea define la presión selectiva que conducirá la reproducción como la selección fuerte de los mejores.

En la estrategia de reemplazamiento la presión selectiva se ve también afectada por la forma en que los cromosomas de la población son reemplazados por los nuevos descendientes. Se usan métodos de reemplazamiento aleatorios, o determinísticos. También se puede decidir no reemplazar al mejor(es) cromosoma(s) de la población (Elitismo).

Los criterios de parada pueden ser varios: cuando haya un valor de la solución suficientemente adecuado, o fijar el máximo número de evaluaciones en dependencia de los recursos limitados de CPU, o después de algunas iteraciones sin mejora.

Problemas y Aplicaciones:

Se aplica en problemas complejos con diversos parámetros o características que precisan ser combinadas en busca de la mejor solución; problemas con muchas restricciones o condiciones que no pueden ser representadas matemáticamente y problemas con grandes espacios de búsqueda.

Los Algoritmos Genéticos han sido aplicados en diversos problemas de optimización, tales como: optimización de funciones matemáticas, optimización combinatoria, optimización de planeamiento, problema del viajante, problema de optimización de rutas, optimización de diseño de circuitos, optimización de distribución, optimización en negocios, síntesis de circuitos electrónicos, etc. Se recomienda revisar en [39, 13, 40] donde aparecen muchos ejemplos de sus aplicaciones.

```
Proc AM:
Begin
  pob =inicPoblacion(sizePob);
  Repeat Until ( terminacion ) Do
    pob =pasoGeneracion(pob, ops);
    If (Convergencia(pob)) Then
      pob =reInicPoblacion(pob);
    Fi
  Od
  retornar(iesimoMejor(pob, 1));
End.
```

Figura 1.10: Esquema básico de un Algoritmo Memético

1.3.2. Algoritmos Meméticos

Entre las metaheurísticas derivadas de los algoritmos genéticos destacan los Algoritmos Meméticos *(Memetic Algorithms, MA)* [41, 42, 17], desarrollados por Moscato en 1989, y que surgen de combinar los algoritmos genéticos con búsquedas locales. Por esto último se considera un algoritmo híbrido, aunque en esta memoria se ha puesto dentro de las metaheurísticas basadas en poblaciones.

La denominación 'memético' surge del término inglés 'meme', acuñado por R. Dawkins (1976) como el análogo del gen en el contexto de la evolución cultural.

Un MA mantiene en todo momento una población de diversas soluciones al problema considerado, que se denominan agentes. Esta denominación es una extensión del término individuo, y permite capturar elementos distintivos de los Algoritmos Meméticos (por ejemplo, un agente puede contener mas de una solución al problema [42]). Estos agentes se interrelacionan entre sí en un marco de competición y de cooperación, de manera muy semejante a lo que ocurre en la Naturaleza entre los individuos de una misma especie (los seres humanos, sin ir mas lejos).

Cuando se considera la población de agentes en su conjunto, esta interacción puede ser estructurada en una sucesión de grandes pasos temporales denominados generaciones. Cada generación consiste en la actualización de la población de agentes, usando para tal fin una nueva población obtenida mediante la recombinación de las características de algunos agentes seleccionados. Precisamente, este componente de selección es junto con el paso final de actualización, el responsable de forzar la competición entre agentes. Ver su algoritmo básico en la Figura 1.10.

Más concretamente, la selección se encarga de elegir una muestra de los mejores agentes contenidos en la población actual. Esto se realiza mediante el empleo de una función guía Fg encargada de cuantificar cuán bueno es cada uno de los agentes en la resolución del problema abordado. Por su parte, el reemplazo o actualización incide en el aspecto competitivo, encargándose de la importante tarea de limitar el tamaño de la población, esto es, eliminar algunos agentes para permitir la entrada de otros nuevos y así enfocar la tarea de búsqueda. En este proceso también puede emplearse la información proporcionada por la función guía para seleccionar los agentes que se eliminarían.

Tanto la selección como el reemplazo son procesos puramente competitivos en los que únicamente varía la distribución de agentes existentes, esto es, no se crean nuevos agentes. Esto es responsabilidad de la fase de reproducción. Dicha reproducción tiene lugar mediante la aplicación de cierto número de operadores reproductivos. No obstante, lo más típico es emplear únicamente

dos operadores: recombinación y mutación. El primero es el responsable de llevar a cabo los procesos de cooperación entre agentes. En relación con los operadores de mutación, es posible definir un metaoperador basado en la aplicación iterativa de un operador de mutación arbitrario sobre un agente. El empleo de estos metaoperadores es uno de los rasgos más distintivos de los Algoritmos Meméticos. Concretamente, dichos metaoperadores iteran la aplicación del operador de mutación, conservando los cambios que llevan a una mejora en la bondad del agente, motivo por el cual son denominados optimizadores locales.

La iteración en la aplicación del operador de mutación y la subsiguiente conservación de los cambios favorables se realiza hasta que se alcanza un cierto criterio de terminación (un número de iteraciones fijo, un cierto número de iteraciones sin mejora, haber alcanzado una mejora suficiente, etc.).

Estos optimizadores locales pueden considerarse como un operador más, y como tales pueden emplearse en diferentes fases de la reproducción. Por ejemplo, pueden usarse tras la aplicación de otros operadores simples de recombinación y mutación, aplicarse solo a un subconjunto de los agentes o únicamente ser aplicados al final del ciclo reproductivo.

La generación de la población inicial puede acometerse de diferentes formas. Por ejemplo, pueden crearse agentes al azar, o emplear las soluciones proporcionadas por heurísticas existentes. Una posibilidad mas sofisticada es el empleo de optimizadores locales para tal fin.

La función para la reiniciación de la población es otro de los componentes fundamentales del Algoritmo Memético, ya que de ella dependerá que se haga un uso apropiado de los recursos computacionales del sistema, o por el contrario éstos se malgasten en la explotación de una población de agentes que haya alcanzado un estado degenerado, o sea, con una gran similitud entre todos los agentes de la población. Esto es lo que se conoce como convergencia del Algoritmo Memético, y es algo que puede ser cuantificado empleando por ejemplo medidas clásicas de Teoría de la Información como la entropía de Shannon [41]. Una vez se ha detectado la convergencia del Algoritmo Memético (cuando la entropía ha caído por debajo de un cierto valor límite por ejemplo), la población de agentes se reinicia, conservando una porción de la misma, y generando nuevos agentes para completarla.

Como resumen puede decirse que el algoritmo se caracteriza como una colección de agentes que realizan exploraciones autónomas del espacio de búsqueda, cooperando ocasionalmente a través de la recombinación, y compitiendo continuamente por los recursos computacionales a través de la selección y el reemplazo.

Problemas y Aplicaciones:

Existen varias aplicaciones de los Algoritmos Meméticos en el ámbito de la optimización. Se destacan las siguientes: problemas de partición en grafos, partición de números, conjunto independiente de cardinalidad máxima, empaquetado, coloreado de grafos, recubrimiento de conjuntos, problemas de asignación generalizados, problema de la mochila multidimensional, programación entera no-lineal, asignación cuadrática, partición de conjuntos, isomorfismos en grafos y el problema del viajante de comercio, predicción de las estructuras secundarias de las proteínas.

También se menciona su uso para: diseño de trayectorias óptimas para naves espaciales, problemas de emplazamiento, problemas de transporte, asignación de tareas, problemas de bi-conexión de vértices, agrupamiento, identificación de sistemas no-lineales, programación de tareas de mantenimiento, en diferentes problemas de planificación tales como: planificación de tareas en una máquina con tiempos de “setup” y fechas de entrega, trayectorias, tareas en varias máquinas,

proyectos, almacén, producción, turnos, juegos deportivos y exámenes, y en confección de horarios. Otras aplicaciones de estas técnicas pueden encontrarse en: telecomunicaciones, electrónica, ingeniería, medicina, economía, oceanografía, matemáticas, procesamiento de imágenes y de voz, biología molecular, etc.

Recomendamos revisar a [41, 42] donde aparecen referencias a muchas de ellas.

1.3.3. Algoritmos de Estimación de Distribuciones

Los Algoritmos de Estimación de Distribuciones *(Estimation of Distribution Algorithms, EDA)* [43, 44] son algoritmos evolutivos que usan una colección de soluciones candidatas para realizar trayectorias de búsqueda evitando mínimos locales. Fueron introducidos en la computación evolutiva por Mühlenbein y Paaß [45] en 1996 .

Estos algoritmos no requieren de operadores de cruce, ni de mutación; y en lugar de manipular directamente a los individuos que representan soluciones del problema usan la estimación y simulación de la distribución de probabilidad conjunta como un mecanismo de evolución, la cual es estimada a partir de una base de datos conteniendo los individuos seleccionados en la generación anterior.

Mientras que en el resto de los algoritmos evolutivos las interrelaciones entre las variables (genes o posiciones de los individuos) se mantienen implícitas, en los EDA dichas interrelaciones se expresan de manera explícita a través de la distribución de probabilidad conjunta asociada a los individuos seleccionados en cada generación.

Un algoritmo EDA (ver Figura 1.11) comienza generando aleatoriamente una población de individuos. Se realizan iterativamente tres tipos de operaciones sobre la población. El primer tipo de operación consiste en la generación de un subconjunto de los mejores individuos de la población. En segundo lugar se realiza un proceso de aprendizaje de un modelo de distribución de probabilidad a partir de los individuos seleccionados. En tercer lugar se generan nuevos individuos simulando el modelo de distribución obtenido. El algoritmo se detiene cuando se alcanza un cierto número de generaciones o cuando el rendimiento de la población deja de mejorar significativamente.

Al comienzo se generan M individuos al azar, por ejemplo a partir de una distribución uniforme para cada variable. Estos M individuos constituyen la población inicial, D_0, y cada uno de ellos es evaluado. En un primer paso, un numero N $(N = M)$ de individuos es seleccionado (normalmente aquellos con mejores valores de función objetivo).

```
Proc EDA:
Begin
  D_0 =gxIndivAzar(M);
  Repeat Until ( terminacion ) Do
    D_sel =selecIndiv(N ≤ M);
    pl(x) = p(x|D_sel) =estimarDistProb();
    Dl =muestrearIndiv(M, pl(x));
  Od
End.
```

Figura 1.11: Algoritmo de Estimación de Distribuciones

A continuación, se efectúa la inducción del modelo probabilístico n-dimensional que mejor

refleja las interdependencias entre las variables. En un tercer paso, se obtienen M nuevos individuos (la nueva población) a partir de la simulación de la distribución de probabilidad que se ha aprendido en el paso previo. Los tres pasos anteriores se repiten hasta que se verifique la condición de parada. Ejemplos de condiciones de parada son: cuando un número previsto de poblaciones o de diferentes individuos se obtiene, al lograr uniformidad en la población generada, cuando no hay mejoramiento en la estimación del mejor individuo obtenido en la generación anterior.

El problema principal con los EDA radica en cómo estimar la distribución de probabilidad $pl(x)$ [44]. Obviamente, la computación de todos los parámetros necesarios para especificar la distribución de probabilidad n-dimensional conjunta no es práctica. La manera de abordar el problema es considerar que la distribución de probabilidad se factoriza de acuerdo a un modelo de probabilidad simplificado.

Problemas y Aplicaciones:

Se aplica en optimización en dominios continuos, donde aparece una serie de los algoritmos anteriores adaptados a este tema, como UMDA continuo, PBIL continuo, MIMIC continuo, etc. que aparecen muy bien explicados en [43].

Por ser un método relativamente nuevo existen muchos campos abiertos donde llevar a cabo tareas de investigación. Se podrían destacar [44]: modelado matemático de EDA, optimización multiobjetivo, algoritmos paralelos, en funciones multimodales, utilización de diferentes métodos a la hora de codificar la distribución de probabilidad, etc.

1.3.4. Búsqueda Dispersa

El enfoque de la metaheurística de Búsqueda Dispersa *(Scatter Search, SS)* [18, 46, 47] se basa en el principio de que la información sobre la calidad o el atractivo de un conjunto de reglas, restricciones o soluciones pueden ser utilizados mediante la combinación de éstas en lugar de usarlas aisladamente.

Contempla el uso de un conjunto de soluciones, denominado conjunto de referencia, de buenas soluciones dispersas que sirve, tanto para conducir la búsqueda mejorando las herramientas para combinarlas adecuadamente para crear nuevas soluciones que mejoren las anteriores, como para mantener un grado satisfactorio de diversidad. Por ello, se clasifica como evolutivo; pero no está fundamentado en la selección aleatoria sobre un conjunto relativamente grande de soluciones sino en elecciones sistemáticas y estratégicas sobre un conjunto pequeño.

Integra la combinación de soluciones con la búsqueda local, que aunque puede tener una estructura de memoria, en la mayoría de los casos se usa una búsqueda lineal convencional.

La propuesta inicial fue introducida por Glover en 1977 como heurística para programación entera y se basó en estrategias para crear reglas de decisión compuestas [46].

Su esquema consta de 5 elementos o métodos:

- *Generador de soluciones diversas:* Se basa en generar un conjunto P de soluciones diversas del que se extrae un subconjunto pequeño, el conjunto de referencia.
- *Método de mejora:* es un método de búsqueda local para mejorar las soluciones tanto para el subconjunto de referencia como para las combinadas antes de analizar su inclusión en el conjunto de referencia.

- *Método para crear y actualizar el conjunto de referencia:* A partir del conjunto de soluciones diversas P se extrae el conjunto de referencia según el criterio de contener soluciones de calidad y diferentes entre sí.
- *Método de generación de subconjunto de referencia:* Genera los subconjuntos de referencia a los que se aplicará el método de combinación. Especifica la forma en que se seleccionan los subconjuntos.
- *Método de combinación:* Se basa en combinar todas las soluciones del conjunto de referencia generadas por el método anterior. Las soluciones obtenidas pueden introducirse inmediatamente en el conjunto de referencia (actualización dinámica) o almacenadas temporalmente en una lista hasta terminar de realizar todas las combinaciones y después de ver que soluciones entran en éste (actualización estática).

```
Proc SS:
Begin
  p = {};
  Repeat Until ( tamañoPrefijado ) Do
    gxSolution(x0);
    mejoraSolution(x0, x);
    If (x not ∈ P) Then
      add(x,P);
    Fi
  Od
  obtSetRef(RefSet);
  evalOrdenaSol(RefSet);
  NuevaSolucion=TRUE;
  While ( NuevaSolucion ) Do
    NuevaSolucion=FALSE;
    gxSubset(RefSet, subSet);
    While ( exam(subSet) ) Do
      select(subSet);
      combina(subSet, solucion);
      mejoraSolution(solucion, x);
      gxSolution(x0);
      If (((f(x) < f(x')) and (x not ∈ RefSet))) Then
        x' = x;
        ordena(RefSet);
      Fi
    Od
  Od
End.
```

Figura 1.12: Algoritmo de Búsqueda Dispersa

El algoritmo (ver Fig. 1.12) comienza con la creación de un conjunto inicial de referencia de soluciones *RefSet*. El método generador de soluciones diversas construye un gran conjunto **P** de soluciones diversas. El conjunto inicial de referencia se construye acorde al método de crear y actualizar el conjunto de referencia. Por ejemplo, podría consistir de la selección de b con soluciones distintas y máximamente diversas desde **P**, o alternativamente que las mejores soluciones de **P**, medidas por el valor de la función objetivo, sean seleccionadas como conjunto de referencia.

Las soluciones en *RefSet* son ordenadas de acuerdo a su calidad, donde la mejor solución es la primera en la lista. La búsqueda se inicia entonces asignando **TRUE** a *NuevaSolucion*. Se construye el nuevo subconjunto y *NuevaSolucion* se iguala a **FALSE**. La forma más simple del método de generación de subconjuntos consiste en la generación de todos los pares de las soluciones de referencia. Los pares en el nuevo subconjunto son seleccionados uno a la vez en

orden lexicográfico y se les aplica el método de combinación para generar una o más soluciones a las cuales se les aplica el método de mejora.

El algoritmo mostrado se detiene cuando al tratar de combinar no hay nuevos elementos en el conjunto de referencia $NuevaSolucion = 0$. Puede ser anidado en un esquema global que permita reconstruir el conjunto de referencia cuando ya ha sido utilizado. Lo habitual es regenerar el conjunto de referencia dejando las $b/2$ mejores y eliminando la otra mitad. Entonces se genera un conjunto P como al inicio del algoritmo, del que se extraen solo las $b/2$ soluciones más diversas con las ya existentes en *RefSet*. Así, se obtiene un nuevo conjunto de referencia en el que se mantienen las soluciones de calidad y se renuevan las debidas a la diversidad. Luego se vuelve a combinar como antes sobre este conjunto de referencia obteniéndose un esquema cíclico indefinido al que hay que ponerle una variable de control para detenerlo, que puede estar en función del tiempo o del número de iteraciones. De los cinco métodos en la metodología, sólo cuatro son requeridos estrictamente. El método de mejora es usado generalmente si se desean salidas de muy alta calidad, pero puede ser implementada sin éste. Los avances en esta metaheurística son relativos a la manera en que estos cinco métodos se desarrollan, o sea la sofisticación viene más de la implementación de sus métodos que de la decisión de incluir o excluir alguno de ellos. Búsqueda Dispersa es un método evolutivo que se encuentra en desarrollo y en la actualidad no hay un esquema único para aplicarlo.

Problemas y Aplicaciones:

En el libro de Martí y Laguna [47] aparecen una serie de aplicaciones entre las que están: la solución del problema del cartero chino capacitado, encaminamiento de vehículos, coloración de grafos, asignación cuadrática, etc.

También se ha usado en el entrenamiento de redes neuronales en el contexto de simulaciones de optimización. Así como en la planificación de horarios, planificación de flujos, planificación de grupos de trabajo, trazado de grafos, optimización no restringida, asignación multiobjetivo, problemas de árboles, minimización de cruces en grafos, en modelos de programación lineal con mezcla de variables binarias y enteras.

También muestra aplicaciones finales en la solución de problemas de encaminamiento multiobjetivo de autobuses escolares, en logística en una fábrica de partes de coches, etc.

1.3.5. Colonia de Hormigas

La metaheurística de Sistema o Colonia de Hormigas para Optimización *(Ants Colony Optimization, ACO)* [48, 49, 50] emplea estrategias inspiradas en el comportamiento de las colonias de hormigas para descubrir fuentes de alimentación, al establecer el camino más corto entre éstas y el hormiguero y transmitir esta información al resto de sus compañeras.

Cada hormiga construye una solución, o un componente de esta, comenzando en un estado inicial seleccionado de acuerdo a criterios dependientes del problema. Mientras construye su propia solución colecciona información sobre las características del problema y sobre su actuación y usa esta información para modificar la representación del problema (ver Fig. 1.13).

Las hormigas pueden actuar de forma concurrente e independiente mostrando un comportamiento cooperativo, pero sin una comunicación directa.

Una aproximación constructiva incremental es usada por éstas para buscar la posible solución,

y una solución se expresa como un camino de costo mínimo a través de los estados del problema y de acuerdo a sus restricciones.

Una simple hormiga debe ser capaz por sí sola de encontrar una solución (probablemente de baja calidad). Las soluciones de alta calidad son sólo encontradas como resultado de la cooperación global entre todos los agentes de la colonia trabajando concurrentemente diferentes soluciones.

De acuerdo a la noción asignada de vecindario (dependiente del problema) cada hormiga construye una solución por movimientos a través de una secuencia finita de estados vecinos. Los movimientos son seleccionados al aplicar una búsqueda local estocástica dirigida por:

- Información privada de la hormiga: memoria o estado interno de la hormiga.
- El carril público de feromona y una información previa local del problema específico.

El estado interno de las hormigas almacena información de la historia pasada de éstas. Éste puede portar información útil para computar el valor/bondad de la solución generada y/o la contribución de cada movimiento ejecutado.

La información local pública contiene alguna información heurística específica del problema y el conocimiento, codificado en el carril de feromonas, acumulado por todas las hormigas desde el comienzo del proceso de búsqueda.

```
Proc ACO:
Begin
  While ( terminacion ) Do
    PlanifActiv;
    gxActivHormigas();
    evapFeromonas();
    actuaDemonio();
    FinPlanifActiv;
  Od
End.
```

Figura 1.13: Algoritmo de Colonia de Hormigas

Las decisiones acerca de cuándo las hormigas deberían liberar feromona sobre el ambiente y cuánta feromona debería depositarse dependen de las características del problema y del diseño de la implementación. Las hormigas pueden liberar feromona mientras construyen la solución, o después de que la solución ha sido construida o ambos.

La metaheurística basada en colonias de hormigas posee tres componentes, dos de ellos que actúan desde una perspectiva local: la generación y actividad de hormigas, y la evaporación de la feromona. Además tiene componentes extras que usan información global que, por ejemplo, pueden ser usados para observar el comportamiento de las hormigas y coleccionar información global para depositar feromona de información adicional.

Estas tres actividades básicas del algoritmo necesitan algún tipo de sincronización. En general una planificación estrictamente secuencial de las actividades es usada para problemas no distribuidos, donde el conocimiento global es fácilmente accesible en cualquier instante y las operaciones pueden ser explotadas convenientemente. Por el contrario, algún tipo de paralelismo puede ser explotado eficiente y fácilmente en problemas distribuidos. No es aconsejable en problemas en los cuales cada estado tiene un vecindario demasiado grande. Una hormiga que visita un

estado con un vecindario de gran tamaño tiene un número muy grande de posibles movimientos entre los cuales elegir; por tanto, la probabilidad que muchas hormigas visiten el mismo estado es muy pequeño y consecuentemente hay muy poca diferencia entre usar o no carriles de feromona.

Problemas y Aplicaciones:

En [48, 49] aparecen varias de sus aplicaciones a la solución de diferentes problemas, tales como:

Problemas de optimización estática combinatoria; Viajante de comercio, asignación cuadrática, planificación de horarios de trabajo, encaminamiento de vehículos, ordenamiento secuencial, coloración de grafos, etc.

Problemas de optimización combinatoria dinámica; encaminamiento de redes orientadas a conexión, encaminamiento de redes sin conexión, etc.

También en [51] se referencia un sitio de internet donde aparece una recopilación de aplicaciones de este método, además de información, artículos, publicaciones, etc.

1.3.6. Optimización con enjambre de partículas

Optimización con enjambre de partículas *(Particle Swarm Optimization, PSO)* es una metaheurística evolutiva inspirada en el comportamiento social de las bandadas de pájaros o bancos de peces desarrollada por Eberhart y Kennedy en 1995 [52].

Comparte similaridades con técnicas de la computación evolutiva, como los Algoritmos Genéticos. Aquí el sistema es inicializado con una población de soluciones aleatorias y hace búsquedas del óptimo actualizando las generaciones. Sin embargo no tiene operadores de cruce ni mutación. En PSO las soluciones, llamadas partículas, se "echan a volar" en el espacio de búsqueda guiadas por la partícula que mejor solución ha encontrado hasta el momento y que hace de líder de la bandada. Cada partícula evoluciona teniendo en cuenta la mejor solución encontrada en su recorrido y al líder.

El procedimiento también tiene en cuenta el mejor valor alcanzado por alguna de las partículas en su entorno. En cada iteración, las partículas modifican su velocidad hacia la mejor solución de su entorno teniendo en cuenta la información del líder.

En cada iteración, cada partícula es actualizada por dos valores (ver Fig. 1.14). El primero es la mejor solución que esta partícula ha logrado, este valor se llama *pbest*. El otro mejor valor que es seguido por el optimizador es el mejor obtenido por cualquier partícula en toda la población. Este mejor valor es global y es llamado *gbest*. Cuando una partícula toma parte de su población como su vecindario topológico, el mejor valor es un mejor local y es llamado *lbest*, o sea, se crea un vecindario para cada partícula conteniendo el suyo propio y los k vecinos más cercanos en la población. Por ejemplo, si $k = 2$, entonces cada partícula será influenciado por la mejor actuación del grupo formado por las partículas $i-1$, i, e $i+1$. Su efecto es que cada partícula es influenciada por la mejor actuación de cualquier miembro de la población entera.

El concepto de PSO consiste en ir cambiando la velocidad (aceleración) de cada partícula hacia su *pbest* y *lbest*. Después de encontrar los dos mejores valores, la partícula actualiza su velocidad y posición con las siguientes ecuaciones:

```
Proc PSO:
Begin
  For (cada partic) Do
    inicPartic();
  Od
  While ( terminacion ) Do
    For (cada partic) Do
      calcFitness();
      If (fitn es mejor que pBest) Then
        pBest = fitn;
      Fi
    Od
    gBest =particMejor(pBest);
    For (cada partic) Do
      calcVeloc(equat(a));
      actualizaPosit(equat(b));
    Od
  Od
End.
```

Figura 1.14: Algoritmo de Enjambre de Partículas

$$v[.] = v[.] + c1 * rand(.) * (pbest[.] - present[.]) + c2 * rand(.) * (gbest[.] - present[.])$$

$$present[.] = present[.] + v[.]$$

Donde,

$v[.]$ es la velocidad de la partícula,

$present[.]$ es la partícula actual

$pbest[.]$ y $gbest[.]$ fueron definidos arriba.

$rand(.)$ es un número aleatorio entre (0,1)

$c1, c2$ son factores de aprendizaje,

La suma de las aceleraciones podría causar que la velocidad en la dimensión exceda $Vmax$, el cual es un parámetro especificado por el usuario, por ello la velocidad en la dimensión es limitada a $Vmax$.

PSO obtiene muy rápido sus resultados y además tiene pocos parámetros para ajustar por lo que resulta atractivo para muchas áreas de aplicación e investigación tales como: la optimización, entrenamiento de redes neuronales, sistemas de control difusos y otras áreas donde los Algoritmos Genéticos pueden ser aplicados.

PSO tiene muchos puntos en común con los Algoritmos Genéticos. Ambos algoritmos comienzan con un grupo de la población generada aleatoriamente, y tienen valores de ajuste para evaluar la población. Ambos actualizan la población y buscan el óptimo con técnicas aleatorias y ninguno garantiza el éxito. Pero PSO no tiene operadores genéticos como cruce y mutación. Las partículas se actualizan ellas mismas con la velocidad interna, y poseen memoria. También difiere en su mecanismo de compartir la información. En los Algoritmos Genéticos los cromosomas comparten información unos con otros, de manera que toda la población se mueve como un grupo hacia el área optima. En PSO, solo *gbest* (o *lbest*) brinda la información a los otros. Es más bien

un mecanismo de información compartida. La evolución solo busca la mejor solución. Todas las partículas convergen a la mejor solución rápidamente aun en la versión local en la mayoría de los casos.

Problemas y Aplicaciones:

Se conocen aplicaciones [52] en el entrenamiento de redes neuronales reemplazando al algoritmo de aprendizaje e incluyendo no solo la ponderación de la red sino también indirectamente su estructura. Esto trabaja para cualquier topología de red y cualquier algoritmo de entrenamiento.

En medicina en el entrenamiento de una red neuronal que clasifica el temblor en las personas (la enfermedad de Parkinson o temblor elemental) vs. sujetos normales. En la optimización en máquinas de control numérico que implica el uso de redes neuronales para el proceso de simulación y PSO en la optimización multimodal. Esto se está extendiendo a otros procesos de maquinado y a la predicción y optimización de un mayor conjunto integral de parámetros del proceso.

En la optimización de mezcla de ingredientes en un entorno industrial, PSO brindó una mezcla optimizada que resultó ser mucho más adecuada que la mezcla encontrada por métodos tradicionales.

En el control de fuentes reactivas y de voltaje. Se usó para determinar una estrategia de control con variables de control discretas y continuas. Una de las razones de su elección fue la capacidad de PSO de ser expandido a problemas de optimización no lineal. En la estimación del estado de carga de un grupo de baterías de vehículos en combinación con un algoritmo de retropropagación para entrenar una red neuronal de tres capas como estimador.

En Internet, por ejemplo en [53, 54] se puede encontrar información sobre el método y una gran cantidad de referencias a enlaces y bibliografía actualizada.

1.4. Conclusiones

En este capítulo se dieron algunas definiciones básicas relativas a la optimización combinatoria y las metaheurísticas. Dado que entre los objetivos de este trabajo se encuentra el diseño de estrategias de búsqueda robustas, en este capítulo nos hemos propuesto repasar algunas de las metaheurísticas más exitosas. Como clasificación para su estudio hemos decidido agruparlas en dos grandes clases: métodos de trayectorias y métodos de poblaciones. Así se abordaron las metaheurísticas siguientes: Templado Simulado, GRASP, Búsqueda Tabú, VNS, ILS, GLS, Algoritmos Genéticos, Meméticos, Algoritmos de Estimación de Distribución, Búsqueda Dispersa, Colonias de Hormigas y Enjambres de Partículas. Hemos señalado las referencias más importantes de cada uno de estos métodos, sus elementos más destacados, así como los problemas y aplicaciones que se han abordado con éxito a través de ellos.

Capítulo 2

Metaheurísticas Paralelas

Aunque las metaheurísticas brindan estrategias efectivas para resolver los problemas de optimización combinatoria al ser capaces de encontrar soluciones cercanas al óptimo, los tiempos de computación asociados a la exploración del espacio de soluciones pueden ser muy grandes. La computación paralela y su aplicación en la implementación de las metaheurísticas aparece de manera natural en el desarrollo de éstas como una alternativa para mejorar el factor de aceleración en la búsqueda de las soluciones. Posteriormente [55] se ha podido constatar que pueden lograrse aplicaciones robustas que pueden alcanzar soluciones de más calidad que sus correspondientes algoritmos secuenciales.

Para tratar esto, el capítulo se estructura en dos partes. La primera, dedicada a los conceptos básicos acerca de la computación paralela, donde se presentan las arquitecturas de máquinas computadoras más usadas, se habla de la programación paralela, se presentan conceptos básicos y métodos de diseño para los algoritmos paralelos, métodos de la programación paralela y ambientes de programación en el cual se muestran diferentes paquetes de software. Uno de ellos, PVM, usado en este trabajo se presenta explicando sus principales características. La segunda parte está dedicada a las metaheurísticas paralelas y se explican conceptos, estrategias de paralelización, taxonomías y se hace una revisión del estado del arte. Para finalizar, se dan las conclusiones del capítulo.

2.1. Conceptos básicos sobre Computación Paralela

Durante los últimos 50 años la computación secuencial ha sido el modo habitual de computación. Su éxito espectacular radica sobre todo en la adopción del modelo básico y ya universal propuesto por Von Neuman, cuya estabilidad ha sido la base para el desarrollo de aplicaciones de alto nivel con la garantía de que el software desarrollado no iba a quedar obsoleto en el siguiente cambio tecnológico. Una computadora según Von Neuman (ver Fig. 2.1) consiste de una unidad central de procesamiento (CPU) conectada a una unidad de almacenamiento (memoria) y cuyos principios básicos son los siguientes:

- La memoria se usa para almacenar instrucciones de programa y datos.
- Las instrucciones de programa son datos codificados que le dicen al procesador lo que debe hacer.

- Los datos son simplemente informaciones a ser usadas por el programa.

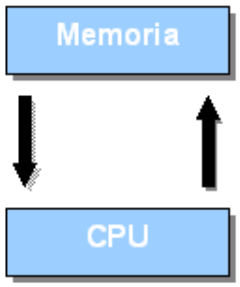


Figura 2.1: Modelo de Von Newman

Aunque presenta dos limitaciones básicas [56] como son: la velocidad de ejecución de las instrucciones y la velocidad de transferencia entre CPU y memoria, la robustez de este modelo está precisamente en su simplicidad.

Entiéndase por Computación Paralela aquella en la que el cómputo se realiza en varias unidades de procesamiento, buscando el mejor aprovechamiento de un mayor número de recursos para la reducción del tiempo invertido en un proceso informático. Por tanto, una computadora paralela es un conjunto de procesadores capaces de trabajar cooperativamente para resolver un problema. Esta definición lo mismo incluye a supercomputadores paralelos que tienen cientos o miles de procesadores, que redes de estaciones de trabajo, estaciones de trabajo de multiprocesamiento y sistemas integrados o embebidos.

El computador paralelo no es idea nueva, desde hace varios años algunos investigadores ya presentaban ideas al respecto. En 1958 Gill [57] escribe acerca de la programación paralela, Holland [58] en 1959 habla acerca de "*un computador que fuera capaz de ejecutar un número arbitrario de subprogramas simultáneamente*". Conway [59] en 1963 describe el diseño de un computador paralelo y su programación.

Ya en 1981 se presenta Butterfly, que se conoce como el primer sistema comercial paralelo. Fue comercializado por la BBN Computers Advanced de Bolt, Beranek y Newman. Este era capaz de distribuir su trabajo entre 256 procesadores Motorola 68000, conectados entre sí por una red multietapa con memorias de 500 Kbytes por procesador. Se vendieron unos 35 computadores de este tipo a universidades y centros de investigación. Flynn y Rudd predicen en 1996 que "*el futuro es el paralelismo*" [60].

La computación paralela surge como una necesidad de lograr un mayor poder computacional en una sociedad tecnológica cada vez mas abrumada por la ingente cantidad de operaciones necesarias para el análisis de la realidad que la envuelve, y en la que no se busca paralelizar como meta en sí, el objetivo es obtener mayores rendimientos que permitan resolver problemas cada vez más grandes en tiempos de computación aceptables.

La computación paralela ha añadido la gestión de los recursos en el espacio, a la del tiempo, y ofrece el potencial para concentrar sobre el problema en cuestión tanto los procesadores, como memoria y amplios anchos de banda de entrada/salida de datos.

La computación paralela logra acelerar la ejecución de un programa mediante su descomposición en fragmentos que pueden ejecutarse de manera simultánea, cada uno en su propia unidad

de procesamiento.

Idealmente al utilizar n computadores, se multiplica por n la velocidad computacional de un computador; sin embargo, existen factores de índole computacional y físico que hacen que nunca se obtenga un escalado en el rendimiento que sea igual o superior al escalado en el número de procesadores. Las comunicaciones entre procesadores son lentas en comparación con la velocidad de transmisión de datos y de cómputo dentro de un procesador, y en la práctica los problemas no pueden dividirse perfectamente en partes totalmente independientes y se necesita alguna interacción entre ellas lo que ocasiona una disminución de la velocidad. Sin embargo, el incremento de velocidad se aproxima al número de procesadores que se integran en el sistema en determinados algoritmos con una paralelización intrínseca muy fuerte.

Además, hay gran cantidad de científicos trabajando para acercar el límite práctico al límite teórico, por lo que, a los avances en hardware, se han de sumar los avances que se producen en algoritmia. A todo esto hay que añadir que el factor coste sigue siendo determinante para optar por soluciones paralelas aunque no sean óptimas, ya que, aunque no obtengamos el rendimiento máximo teórico, un incremento lineal en la potencia del procesador de una máquina implica un crecimiento exponencial del precio de una máquina, mientras que, con el incremento del número de procesadores, el incremento del precio es mucho menor.

Las computadoras son cada día más rápidas y potentes. Todavía hoy se cumple la conocida Ley de Moore de que "la velocidad de los microprocesadores se multiplica por cuatro cada 3 años", y esto pudiera sugerir que se llegará a una velocidad tope y que ese apetito o necesidad de poder computacional será saciado. Sin embargo la historia ha demostrado que una tecnología particular satisface las aplicaciones conocidas, y que nuevas aplicaciones demandarán nuevas tecnologías. Un caso conocido de esto fue el reporte preparado para el gobierno británico en la década de 1940, en el cual se concluía que los requerimientos computacionales de toda la Gran Bretaña podían ser resueltos por dos o quizás tres computadores. En aquella época los computadores se usaban principalmente en calcular tablas balísticas y los autores no consideraron otras aplicaciones en ingeniería y ciencias, y mucho menos aplicaciones comerciales. También un estudio inicial que se hizo para la empresa Cray Research predecía un mercado solo para 10 supercomputadoras, sin embargo en poco tiempo cientos de ellas se construyeron y vendieron.

El mayor obstáculo que tienen los sistemas paralelos es el coste humano del desarrollo del código. Los sistemas paralelos son bastante complejos de programar y sobretodo de verificar. Por ello, estos sistemas sólo encuentran su justificación en aquellos casos en los que el incremento de costes en la programación realmente está justificado porque esa potencia de cálculo es necesaria. Para estas aplicaciones, sin embargo, los sistemas paralelos tienen gran éxito, encontrando como únicas barreras para su imposición final: la complejidad del desarrollo, ya que necesita de programadores realmente preparados que conozcan a fondo materias tan complejas como mecanismos de sincronización y división de datos; y la escasez de herramientas, que permitan hacer más cómoda la verificación de los programas.

2.1.1. Arquitecturas de máquinas computadoras

Los años noventa significaron una etapa importante en la diversificación de las arquitecturas de máquinas computadoras paralelas [61]. Teniendo en cuenta la propia definición de computación paralela se pueden construir sistemas de procesamiento paralelo de multitud de formas diferentes. La complejidad de estas posibilidades tan diversas hace que se presenten diferentes taxonomías.

Aquí se presentan dos de las más usadas, una basada en el criterio del flujo de datos y de instrucciones, conocida como taxonomía de Flynn, y otra basada en la organización del espacio de memoria.

2.1.2. Taxonomía de Flynn

Michael Flynn [62, 63, 64] en 1966 clasificó las arquitecturas de computadoras en base a una variedad de características que incluyen el número de procesadores, número de programas que estas ejecutan y la estructura de la memoria. Crea una matriz (ver Figura 2.2) en base a dos dimensiones independientes: Datos e Instrucción, los que en cada dimensión solo pueden tener dos posibles estados: Único o Múltiple.

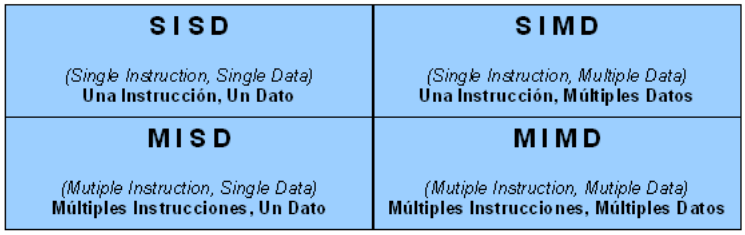


Figura 2.2: Taxonomía de Flynn

SISD, Una Instrucción, Un Dato

Estos computadores solo tienen una CPU que ejecuta una instrucción a la vez, y busca o almacena un solo ítem de dato a la vez. Poseen un único registro, llamado contador de programa, que obliga a la ejecución en serie de las instrucciones, o sea de manera secuencial, tal como se ve en la Figura 2.3.

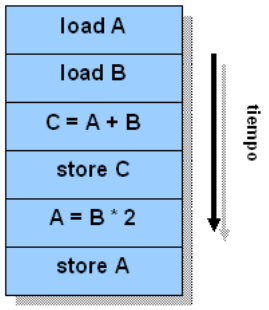


Figura 2.3: SISD

SIMD, Una Instrucción, Múltiples Datos

Estos computadores tienen una sola Unidad de Control que ejecuta una sola instrucción; pero tienen más de un Elemento de Procesamiento. La unidad de control genera las señales de control para todos los elementos de procesamiento, los cuales ejecutan la misma operación sobre datos diferentes, como se observa en la Figura 2.4 donde se ejecutan las mismas instrucciones sobre n

datos diferentes. Tienen una ejecución determinística y sincrónica. Hay dos tipos: las de arreglo de procesadores y las de procesamiento vectorial.

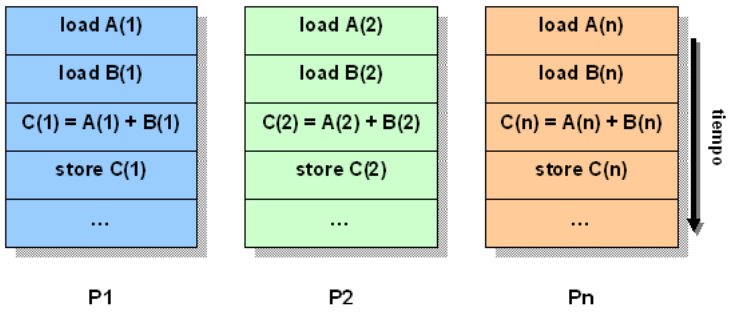


Figura 2.4: SIMD

Estas máquinas son convenientes para resolver problemas caracterizados por un alto grado de regularidad, por ejemplo en el procesamiento de imágenes. Algunos computadores comerciales de este tipo conocidos son:

- Arreglo de Procesadores: Connection Machine CM-2, Maspar MP-1, MP-2.
- Procesamiento Vectorial: IBM 9000, Cray C90, Fujitsu VP, NEC SX-2, Hitachi S820.

MISD, Múltiples Instrucciones, Un Dato

Estos computadores (ver Figura 2.5) pueden ejecutar varios programas diferentes que usan el mismo ítem de dato, o sea, que hay varias instrucciones operando a la vez con el mismo y único dato.

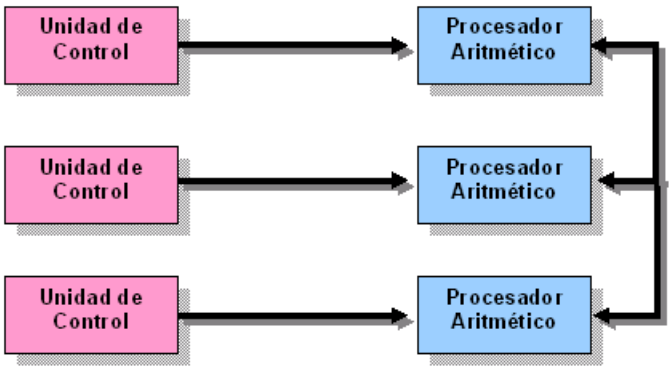


Figura 2.5: MISD

Esta arquitectura puede ser vista en dos categorías:

- Aquellos computadores que requieren distintas unidades de procesamiento que podrían recibir distintas instrucciones que operarían sobre el mismo dato. Esto constituye un reto muy grande para los diseñadores y por ello no se construyen máquinas de este tipo en la actualidad.

- Una clase de máquinas en las cuales el dato fluye a través de una serie de elementos de procesamiento.

Ejemplos del uso de este tipo de arquitectura son:

- Múltiples filtros de frecuencia actuando sobre la misma señal.
- Múltiples algoritmos de decodificación criptográfica intentando violar el mismo código de mensaje.

MIMD, Múltiples Instrucciones, Múltiples Datos

Estos computadores son también llamados multiprocesadores. Tienen más de un procesador y cada uno puede ejecutar un programa diferente que use su propio juego de datos, como se ve en la Figura 2.6 donde se ejecutan n procesos diferentes de forma concurrente.

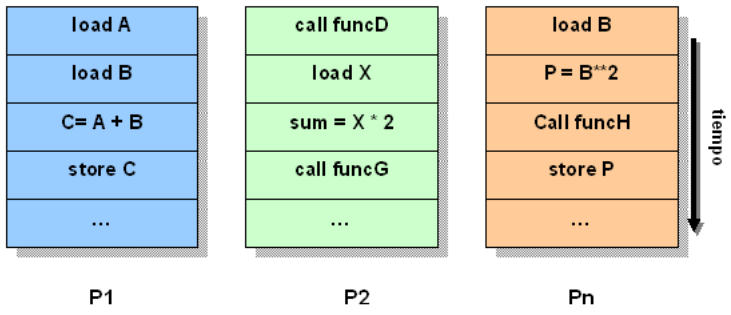


Figura 2.6: MIMD

Para reducir el retraso en la comunicación entre los procesadores, en la mayoría de los sistemas MIMD cada procesador tiene acceso a una memoria global. La ejecución puede ser sincrónica o asincrónica, determinística o probabilística.

Algunos ejemplos comerciarles de esta arquitectura son: la BBN Butterfly, la serie Alliant FX, la serie Intel iPSC, etc. Caen en esta clasificación la mayoría de los supercomputadores actuales, los computadores de multiprocesamiento (SMP, Symmetric MultiProcessors), las redes (grids) de computadores paralelos.

2.1.2.1. Taxonomía basada en el acceso a la memoria

En base a la manera en que se accede a la memoria se reconocen dos tipos de arquitecturas: las máquinas de memoria compartida y las máquinas de memoria distribuida [65].

Máquinas de memoria compartida

En las máquinas de memoria compartida, todos los procesadores son capaces de acceder a un espacio de memoria global (ver Figura 2.7) y la comunicación entre las tareas se hace gracias a operaciones de lectura y escritura sobre esta memoria. Los cambios efectuados en una localización de memoria son visibles a todos los procesadores.

Ventajas

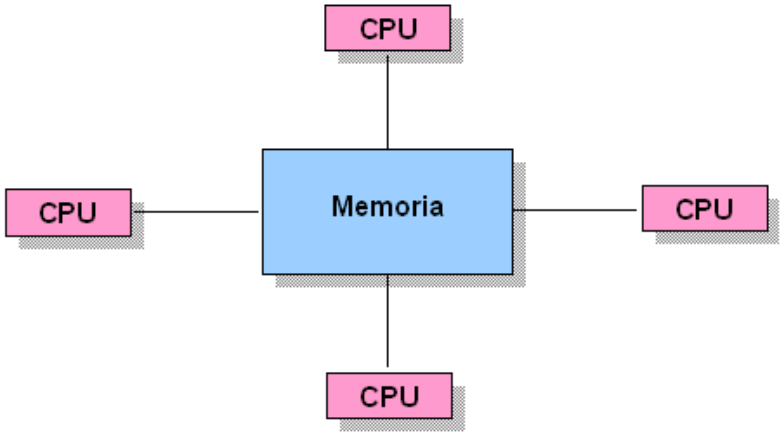


Figura 2.7: Máquinas de memoria compartida

- El espacio global de direcciones brinda una programación amigable desde el punto de vista de la memoria.
- La función de compartir los datos entre las tareas es rápida y uniforme debido a la proximidad de la memoria a las CPUs.

Desventajas

- Adolece de falta de ampliación entre la memoria y los CPUs. Añadir otro CPU a la configuración incrementa el tráfico sobre los buses de la memoria compartida a los CPUs.
- Es responsabilidad del programador construir una sincronización que asegure un acceso correcto a la memoria global.
- El costo y la dificultad en el diseño y la producción se incrementan al aumentar el número de procesadores.

A su vez teniendo en cuenta el modo en que se accede a la memoria global estas se clasifican en dos clases: UMA (Acceso Uniforme a la Memoria) y NUMA (Acceso No Uniforme a la Memoria).

Acceso Uniforme a la Memoria (UMA, Uniform Memory Access):

- Cada procesador tiene un acceso uniforme a memoria.
- Conocidos como multiprocesadores simétricos (SMP, Symmetric MultiProcessors).
- Procesadores idénticos.
- Iguales tiempos de acceso a memoria.

Acceso No Uniforme a la Memoria (NUMA, Non Uniform Memory Access):

- Se construyen enlazando físicamente dos o más SMPs.
- Un SMP puede acceder directamente a la memoria de otro SMP.
- Ningún procesador tiene accesos iguales a todas las memorias.

- El tiempo de acceso depende de la ubicación de los datos.
- Son más fáciles de ampliar que los SMPs.

Máquinas de memoria distribuida

Estas máquinas tienen su memoria distribuida físicamente entre los procesadores y cada uno de ellos sólo puede acceder a su propia memoria (ver Figura 2.8). La comunicación entre los procesos que se ejecutan en diferentes procesadores se realiza a través de mensajes que se pasan a través de una red de comunicación y su sincronización es responsabilidad del programador.

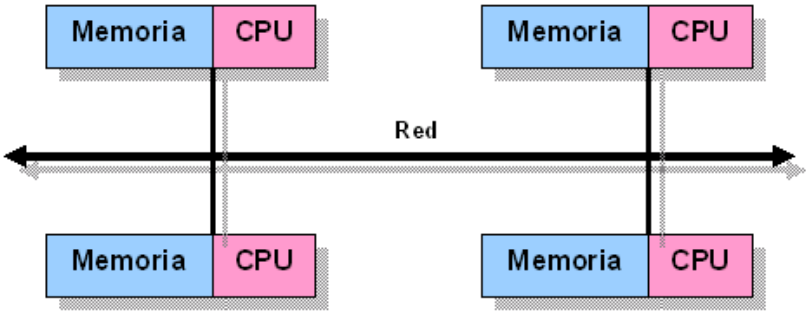


Figura 2.8: Máquinas de memoria distribuida

Ventajas

- La memoria es ampliable con el número de procesadores. Al incrementarse el número de procesadores aumenta proporcionalmente el tamaño de la memoria.
- Cada procesador accede a su propia memoria sin interferencia.

Desventajas

- El programador es responsable de la comunicación de los datos entre los procesadores.
- Es difícil diseñar una estructura de datos en la que se pueda compartir toda la memoria del sistema.
- Los tiempos de acceso no son uniformes.

Modelo Híbrido

Las máquinas computadoras actuales más grandes y más rápidas emplean una arquitectura híbrida entre memoria compartida y distribuida.

El componente de la arquitectura de memoria compartida se usa como una máquina caché SMP. Los procesadores del SMP dado pueden acceder a la memoria global de esta máquina.

El componente de la arquitectura de memoria distribuida se basa en una red de múltiples SMPs (ver Figura 2.9). Cada SMP que forma el sistema sólo conoce acerca de su propia memoria y usa la red de comunicación para mover datos a otro SMP.

Las ventajas y desventajas asociadas a esta arquitectura son las relacionadas a los componentes de memoria compartida y distribuida que posee.

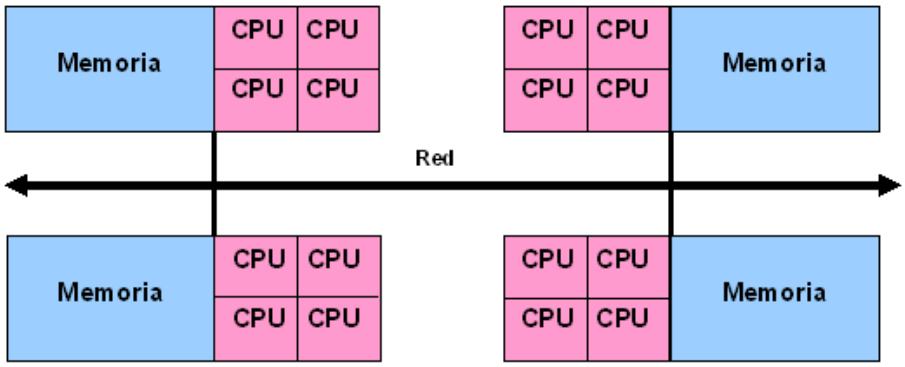


Figura 2.9: Modelo Híbrido

La innovación mas importante en este campo ha sido la conexión de máquinas SMP a través de redes locales rápidas (Ej.: Myrinet, SCI, ATM, GigaEthernet) y la conexión de máquinas paralelas de propósito general a través de redes nacionales o internacionales de muy alta velocidad. Además de mejorar la ampliación y la tolerancia a fallos, estas redes de computadoras o *clusters* tienen una alta relación costo/rendimiento al compararlas con otras máquinas paralelas, lo que las ha convertido en una de las tendencias principales dentro de la computación paralela. Por ello y por ser usada esta configuración en este trabajo se explicará en detalles a continuación.

Cluster

Entiéndase por *cluster* a una colección de computadores personales (PC) o estaciones de trabajo conectados entre sí por una red de computadoras. Las mejoras en rendimiento y fiabilidad de las estaciones de trabajo y la aparición de las redes de alta velocidad han contribuido a ello como una alternativa atractiva, relativamente fácil de montar y de bajo costo para las aplicaciones paralelas.

La literatura refleja varios términos para referirse a un *cluster*, entre los más conocidos están: Redes de estaciones de trabajo (NOWs), Redes de PCs (NOPCs), *Cluster* de estaciones de trabajo (COWs) y *Cluster* de PCs (COPCs) [Roosta].

Desde comienzos de los años 90 del siglo pasado hay una tendencia a alejarse de los supercomputadores especializados paralelos (supercomputadores vectoriales y procesadores masivamente paralelos, MPPs), debido a sus altos costes en hardware, mantenimiento y programación. Los *clusters* han comenzado a ocupar ese espacio, entre otras cosas, debido a:

- Se construyen con un esfuerzo relativamente moderado.
- Bajo coste.
- Usan hardware convencional y accesible en el mercado.
- Generalmente usan un sistema de comunicación basado en una red de área local rápida.
- Utilizan software de libre distribución.
- Son ampliables.
- Cada nodo del *cluster* puede ser un sistema completo utilizable para otros propósitos.

No obstante también presentan algunas desventajas:

- La latencia de las redes ordinarias es alta y su ancho de banda es relativamente bajo comparado con los de un sistema SMP.
- En los sistemas operativos monoprocesador hay poco software para administrar el *cluster* como un único sistema.

Debido a la flexibilidad de este tipo de arquitectura aparecen a diario sistemas cada vez más potentes y con mayor número de procesadores.

Ahora, el éxito de los *clusters* se debe más a los avances en el rendimiento de las redes de comunicación que a la mejora de los procesadores teniendo en cuenta que lo primero es lo que mas afecta en el rendimiento global por no alcanzar aún velocidades comparables a las que logran los procesadores.

Un *cluster* se puede clasificar en dependencia de si cada computador que lo forma está exclusivamente dedicado a él (Beowulf), o no lo está (Redes de estaciones de trabajo).

Otra manera de clasificar un *cluster* es:

- Alta o baja disponibilidad: Depende de si sus recursos están dispuestos de manera que sean redundantes o no, es decir, que exista o no la misma arquitectura duplicada para que si el sistema que brinda el servicio falla, su clon pueda tomar el control y hacerlo él.
- Alto o bajo rendimiento: Si dispone o no del máximo número de flops, que es la unidad de medida para las operaciones de punto flotante realizadas por segundo de tiempo.
- Con o sin balanceo de carga: Depende de si la carga de cómputo puede ser repartida o no. En caso de que si tenga balanceo, este a su vez puede ser clasificado de forma automática o manual, realizado por el programador.

A veces es difícil clasificar un sistema dentro de un solo grupo, aunque prácticamente es imposible que uno pueda considerarse en los tres a la vez.

2.1.2.2. Otras Taxonomías

Atendiendo a otros factores se pueden hacer otras clasificaciones. Así, los sistemas paralelos también se clasifican de acuerdo a la potencia y el número de procesadores que lo forman. Esta taxonomía es conocida por granularidad del procesador. Si son pocos procesadores y muy potentes se denominan computadores de grano grueso, mientras que si son muchos pero menos potentes se denominan de grano fino. Entre ellos están los de grano medio que tienen muchos procesadores del tipo de las estaciones de trabajo. Debido a razones de escala, los computadores de grano grueso son mucho más caros que los de grano medio y fino, ya que su tecnología de fabricación es muy costosa y los procesadores de grano grueso no se fabrican a nivel industrial.

Entre los de grano grueso están los computadores Cray Y-MP, que constan de 8 a 16 procesadores cada uno de varios Gigaflops. Entre los grano medio están los nCUBE 2 y Paragon XP/S y de grano fino los MASPar MP-1 con mas de 16384 procesadores de cuatro bits.

No hay que confundir esta clasificación con el concepto de granularidad del computador paralelo. Este es un factor de medida de eficiencia del ancho de banda de la red y se explicará más adelante.

2.1.2.3. Paralelismo de datos y paralelismo de control

El paralelismo ideal es aquel en el cual un problema puede ser dividido inmediatamente en partes completamente independientes que puedan ser ejecutados simultáneamente.

El paralelismo puede ser logrado de diferentes maneras, a través de:

- Procesamiento concurrente de operaciones de entrada y salida en diferentes procesadores.
- Acceso concurrente a memoria usando sistemas de almacenamiento multipuertos y sistemas de intercalado de memoria.
- Ejecución simultánea de instrucciones usando unidades funcionales yuxtapuestas.
- Decodificación concurrente de instrucciones usando unidades de control yuxtapuestas.
- Ejecución concurrente de instrucciones usando múltiples unidades funcionales.
- Transmisión concurrente de datos entre los dispositivos usando múltiples buses.

Se identifican dos tipos de paralelismo al intentar dividir un problema, el paralelismo de datos y el paralelismo de control o funcional.

El paralelismo de datos se obtiene al asignar elementos de datos a varios procesadores, cada uno de los cuales realiza la misma operación simultáneamente sobre sus datos. Un problema que tenga paralelismo de datos consiste en una secuencia simple de instrucciones o flujo de instrucciones cada una de las cuales se aplica a diferentes elementos de datos.

Este tipo de problemas puede ser ejecutado lo mismo en computadores SIMD como en MIMD, aunque son los computadores SIMD su espacio natural, en los cuales una unidad de control global difunde las instrucciones a los elementos de procesamiento, los cuales contienen los datos y ejecutan las instrucciones de manera sincrónica.

En contraste con el paralelismo de datos, el paralelismo de control se logra realizando diferentes operaciones sobre diferentes elementos de datos simultáneamente. O sea, el paralelismo de control se refiere a la ejecución simultánea de diferentes flujos de instrucciones. Los problemas que poseen paralelismo de control se ejecutan en computadores MIMD. El paralelismo de control en un problema es independiente de su tamaño, mientras que el paralelismo de datos si depende del tamaño del problema.

2.1.2.4. Velocidad computacional

Hay varios factores que influyen directamente en la velocidad computacional de un sistema paralelo. Entre los más usados están:

Granularidad de los procesos:

Para intentar lograr una mejora de la velocidad computacional en una aplicación se divide el cálculo en procesos que se puedan ejecutar de forma simultánea. El tamaño de un proceso se describe por su granularidad.

Un proceso con granularidad gruesa es aquel compuesto por un gran número de instrucciones secuenciales y un proceso con granularidad fina cuando dispone de pocas instrucciones secuenciales. Normalmente se desea incrementar la granularidad para reducir los costos de la creación y

comunicación de procesos; pero eso reduce el numero de procesos concurrentes y la cantidad de paralelismo por lo que hay que encontrar un compromiso entre una cosa y la otra.

En el modelo de paso de mensajes es necesario reducir la sobrecarga en la comunicación, sobre todo en los *clusters* donde la latencia de comunicación puede llegar a ser importante y dominar sobre el tiempo total de ejecución.

Se usa esta razón:

$$G = \frac{Tiempo\,de\,Computacion}{Tiempo\,de\,Comunicacion}$$

como medida de granularidad y se busca maximizarla.

Cuando este cociente es pequeño se aconseja usar el tipo de computadores en cuestión en algoritmos que necesiten comunicaciones frecuentes y en caso del cociente grande serán adecuados para algoritmos que no necesiten muchas comunicaciones [65].

Factor de aceleración (speedup):

Este factor mide el rendimiento relativo entre un sistema multiprocesador y un sistema con único procesador y se define así:

$$A = \frac{Tiempo\,de\,ejecucion\,en\,un\,unico\,procesador}{Tiempo\,de\,ejecucion\,en\,un\,sistema\,paralelo}$$

Para comparar una solución paralela con una solución secuencial se utilizará el algoritmo secuencial más rápido para ejecutarlo en un único procesador.

Hay varios factores que pueden sobrecargar los programas paralelos y que limitan este factor, entre ellos están:

- Los periodos en los que no todos los procesadores están realizando un trabajo útil.
- Los cálculos adicionales que aparecen en el programa paralelo y no están en el secuencial.
- El tiempo de comunicación para enviar mensajes.

Ley de Amdahl [66]

Considera que la parte secuencial de un programa determina la cota inferior para el tiempo de ejecución de un programa que se paraleliza, aún cuando se usen al máximo las técnicas de paralelismo.

Esta expresa que el tiempo de ejecución de la parte paralelizable de un programa es independiente del número de procesadores.

Esto funciona para un problema de tamaño fijo en diferentes números de procesadores y no es así cuando el tamaño del problema crece con el número de procesadores disponibles. Por tanto aumentar el número de procesadores tiene sentido para resolver problemas de gran tamaño y no para intentar resolver mas rápido un problema de tamaño fijo.

Eficiencia

Se obtiene al normalizar la aceleración de un programa paralelo dividiéndolo entre el número de procesadores:

$$E = \frac{Tiempo\ de\ ejecucion\ en\ un\ unico\ procesador}{Tiempo\ de\ ejecucion\ en\ un\ sistema\ paralelo \times\ numero\ de\ procesadores}$$

Esto da la fracción de tiempo que utilizan los procesadores durante la computación, o sea, una medida de cuan eficientemente se han utilizado los procesadores.

Coste

El coste o trabajo se define como:

$$C = Tiempo\ de\ ejecucion \times\ Numero\ de\ procesadores\ usados$$

Así, el coste óptimo de un algoritmo paralelo es aquel proporcional al coste que tiene en un sistema con un único procesador.

Balance de carga

Se considera que el problema a resolver se divide en un número fijo de procesos que pueden ejecutarse en paralelo. Se supone que los procesos se distribuyen entre los computadores sin tener en cuenta el tipo de procesador y su velocidad. La situación ideal es que todos los procesadores trabajen de forma continuada sobre las tareas disponibles para conseguir el mínimo tiempo de ejecución.

La consecución de este objetivo dividiendo las tareas equitativamente entre los procesadores es conocido como balance de carga. Este se puede tratar de forma estática, antes de la ejecución de cualquier proceso, y de forma dinámica, durante la ejecución de procesos.

El balance de carga estático tiene desventajas frente al dinámico:

- Es difícil estimar de forma precisa el tiempo de ejecución de todas las partes en las que se divide un programa.
- Se dificulta añadir una variable de retardo de comunicación, ya que esta puede variar bajo diferentes circunstancias.
- En muchos problemas no se conoce el número de pasos computacionales para alcanzar la solución.

En el balance de carga dinámico la división de la carga depende de las tareas que se están ejecutando y no de la estimación del tiempo que pueden tardar en ejecutarse.

2.1.3. Programación Paralela

Un proceso es una secuencia de instrucciones de programa que puede ser ejecutado en secuencia o en paralelo con otras instrucciones de programa. Entonces, un programa puede ser visto como un número de procesos que son ejecutados secuencial o concurrentemente.

En un ambiente de programación secuencial se procesa un único programa y cada instrucción es ejecutada sin interferencia con las otras instrucciones del programa. Además, su algoritmo se evalúa en términos de su tiempo de ejecución, expresado como una función del tamaño de sus datos de entrada.

En un ambiente de multiprogramación pueden procesarse varios programas a la vez. La unidad de procesamiento es multiplexada de un programa a otro y las instrucciones de uno pueden ser intercaladas con las de otro en determinadas etapas de su ejecución. El tiempo de ejecución de un algoritmo paralelo depende no solo del tamaño de su entrada, sino también de la arquitectura del computador paralelo en el que se ejecuta y del número de procesadores.

Así, en un ambiente de multiprogramación, el programador necesita no solo programa y datos, como ocurre en un ambiente secuencial, sino también herramientas para controlar la sincronización y la interacción entre los procesos.

Un buen ambiente de programación paralela debe cumplir determinados objetivos:

- Debe potenciar el lenguaje de programación secuencial que sea más apropiado para el problema en sí.
- Debe soportar los procesos de creación y comunicación entre procesos como una extensión del lenguaje de programación base.
- Debe ser capaz de ejecutarse en cualquier arquitectura paralela o en cualquier conjunto de computadores en redes.
- Debe brindar operaciones simples y fáciles de usar para crear y coordinar los procesos paralelos.

La ejecución de un programa paralelo forma un conjunto de procesos ejecutándose concurrentemente, que se comunican y sincronizan por lecturas y escrituras de variables compartidas. En algunos casos los procesos trabajan con sus propios datos y no interactúan entre ellos y en otros se comunican y sincronizan entre sí cuando intercambian resultados de la ejecución.

Hay dos métodos de sincronización:

- Sincronización por precedencia.
- Sincronización por exclusión mutua.

La sincronización por precedencia garantiza que un evento no comienza hasta que el otro evento que se ejecuta haya finalizado.

La sincronización por exclusión mutua garantiza que solo un único proceso puede acceder a la sección crítica donde los datos son compartidos.

2.1.3.1. Modelos de programación paralela

Los modelos de programación paralela existen como una abstracción por encima de la arquitectura del hardware y la memoria. Por ello no especifican un tipo particular de computador.

Se identifican varios modelos [60, 56]:

- Memoria compartida.
- Modelo de hilos de ejecución o threads.
- Paso de mensajes.

- Paralelismo de Datos.
- Híbridos.

Modelo de Memoria Compartida:

Las tareas comparten un espacio de direccionamiento común, en el cual ellas leen y escriben asincrónicamente y por tanto se necesitan mecanismos para controlar los accesos a la memoria compartida.

Una ventaja de este modelo es que desde el punto de vista del programador no es necesario especificar explícitamente la comunicación de datos entre las tareas permitiendo que el desarrollo de programas pueda ser simplificado. Y su desventaja está en que se hace muy difícil desde el punto de vista de rendimiento el conocimiento y manejo de los datos locales.

Sobre esta plataforma los compiladores nativos traducen las variables de los programas clientes en direcciones de memoria global. Se pueden realizar implementaciones de este modelo aunque la memoria física del sistema esté distribuida.

Modelo de hilos de ejecución (threads):

En este modelo un solo proceso puede tener múltiples vías de ejecución concurrente. Un proceso de este modelo para ejecutar su trabajo crearía varias tareas (hilos) preparadas para ser ejecutadas concurrentemente por el sistema operativo. Cada tarea o hilo tendría sus datos locales; pero también comparte todos los recursos del proceso salvando así el tener que replicar los recursos del proceso a cada tarea. También se beneficia de una vista de la memoria global del proceso. Cada tarea puede ejecutar cualquier subrutina al mismo tiempo que otra tarea.

Los hilos de ejecución están asociados con la arquitectura de memoria compartida.

Desde la perspectiva de programación no son más que una biblioteca de subrutinas que son llamadas desde el código fuente paralelo y/o un conjunto de directivas del compilador dentro del código fuente serie o paralelo. El programador es responsable de diseñar todo el paralelismo.

Modelo de paso de mensajes:

Este modelo se caracteriza por un conjunto de tareas que usan su propia memoria local. Y estas pueden estar todas en el mismo computador o en varios. Las tareas intercambian datos a través de un sistema de comunicación basado en enviar y recibir mensajes que requiere operaciones cooperativas, o sea cada operación de envío debe tener una operación de recepción y viceversa.

Desde el punto de vista de la programación es una biblioteca de subrutinas embebidas en el código fuente y el programador es responsable de determinar todo el paralelismo.

Modelo de Datos Paralelos:

Este modelo se caracteriza por un grupo de tareas que trabajan colectivamente sobre la misma estructura de datos, sin embargo cada tarea trabaja sobre una partición diferente de esa estructura realizando la misma operación.

La programación bajo este modelo no es más que la escritura de un programa con conceptos de datos paralelos. Estos conceptos pueden ser las llamadas a bibliotecas de subrutinas de datos paralelos o directivas del compilador reconocidas por un compilador de datos paralelos que permiten al programador especificar la distribución y ubicación de los datos.

Modelo Híbrido:

En este se combinan dos o más modelos de programación de los mencionados anteriormente. Un ejemplo muy usado es la combinación del modelo de paso de mensajes lo mismo con el modelo de hilos de ejecución o con el modelo de memoria compartida. Otro ejemplo es la combinación del modelo de datos paralelos con el paso de mensajes.

Su clasifican en:

- Único Programa, Múltiples Datos (SPMD, Single Program Multiple Data): Es un modelo de programación híbrido de alto nivel y que se basa en que todas las tareas ejecutan un único programa. Las tareas no tienen que ejecutar necesariamente el programa entero, pueden estar ejecutando las mismas instrucciones o diferentes dentro del mismo programa y pueden usar datos diferentes.
- Múltiples Programas, Múltiples Datos (MPMD, Múltiple Programa Multiple Data): Es un modelo de programación híbrido de alto nivel en el que cada tarea puede estar ejecutando el mismo programa o uno diferente a las demás y además usar datos diferentes.

2.1.4. Algoritmos paralelos

La computación paralela ha traído una nueva dimensión al diseño de algoritmos. Así, se entiende por algoritmo paralelo aquel en cual varias operaciones pueden ser ejecutadas simultáneamente [56]. O sea, es una colección de tareas independientes, algunas de las cuales pueden ser ejecutadas concurrentemente utilizando un computador paralelo.

Hay varios consejos prácticos a tener en cuenta en el diseño de algoritmos paralelos:

- *Principio de Planificación:* Basado en el teorema de Brent, este hace posible reducir el número de procesadores usados en un algoritmo paralelo, sin incrementar de manera proporcional el tiempo total de ejecución. Es decir, si un algoritmo tiene un tiempo de ejecución de O(log n), entonces el tiempo total de ejecución podría incrementarse por un factor constante.
- *Principio de Yuxtaposición (Pipelining):* Este puede usarse en situaciones en las cuales se realizan varias operaciones en secuencia $P_1, ..., P_n$, donde esas operaciones tienen la propiedad de que algunos pasos de P_{i+1} puedan realizarse antes que la operación P_i haya finalizado. Al solaparse estos pasos se decrementa el tiempo total de ejecución. Esta técnica es muchas veces usada en algoritmos MIMD, aunque muchos algoritmos SIMD pueden también tomar ventajas de este principio.
- *Principio Divide y Vencerás:* Es el principio de dividir un problema en pequeños componentes independientes y resolver estos en paralelo.
- *Principio de Dependencia de Grafo:* Se crea un grafo dirigido en el cual los nodos representan bloques de operaciones independientes y los arcos representan situaciones en las cuales un bloque de operaciones depende de la salida de la actuación de otros bloques.
- *Principio de la Condición de Competencia:* Si dos procesos intentan acceder a los mismos datos compartidos, estos pueden interferir unos con otros si la salida de la computación depende completamente de cual proceso accede primero a los datos compartidos.

Hay tres métodos de diseño de algoritmos paralelos:

- Diseñar de manera paralela un algoritmo secuencial existente o modificar uno existente explotando aquellas partes del algoritmo que son paralelizables por naturaleza.
- Diseñar un algoritmo paralelo completamente nuevo que pueda ser adaptado a alguna de las arquitecturas paralelas.
- Diseñar un nuevo algoritmo paralelo a partir de uno ya existente.

Esto indica que un algoritmo puede ser transformado de uno secuencial a uno paralelo o de una forma paralela a otra [56], siempre que mantenga una equivalencia al pasarlo de una forma a otra, y no se convierta en un algoritmo totalmente nuevo.

En la práctica algunas versiones de algoritmos secuenciales pueden ser adaptadas mejor a la estructura de una arquitectura paralela que a otras. Por ejemplo, los computadores SIMD no son capaces de ejecutar algoritmos asincrónicos y los computadores MIMD no tienen suficiente potencial para los algoritmos sincrónicos.

Estos tres métodos deben tener en cuenta la arquitectura particular del computador paralelo. Así, al diseñar algoritmos MIMD se trata de maximizar la cantidad de concurrencia. Se buscan las operaciones que puedan ser ejecutadas simultáneamente y se crean múltiples procesos para manipular estas operaciones.

La solución para evitar las restricciones debidas a la particularización de la arquitectura del computador es usar el modelo híbrido, explicado anteriormente, para ello se intenta definir un marco de trabajo general, y así.

- Al diseñar un algoritmo paralelo para una arquitectura ideal el número de procesadores disponibles se considera ilimitado y estos sistemas pueden tener un alto número de procesadores homogéneos.
- Estos sistemas pueden ser organizados con procesadores accediendo a una memoria global compartida o cada procesador con una memoria local asociada o un híbrido de estos dos métodos.
- Pueden ser divididos en subsistemas independientes de varios tamaños. Esta división puede ser cambiada dinámicamente en tiempo de ejecución.
- Cada partición puede realizar lo mismo operaciones SIMD que MIMD, esta puede cambiar de modo dinámicamente durante la ejecución.
- Poseen una red de interconexión flexible que brinda una amplia variedad de patrones de comunicación en cada partición.

2.1.5. Métodos de programación paralela

Los métodos de programación paralela son desarrollados lo mismo por la introducción de nuevos paradigmas tales como Linda y Occam [60] o por la extensión de lenguajes secuenciales tales como C o Fortran.

Los lenguajes de programación paralela se pueden categorizar en [56]:

- Lenguajes SIMD: tienen un espacio global de direccionamiento que obvia la necesidad de encaminar explícitamente los datos entre los elementos de procesamiento. Los procesadores son sincronizados al nivel de instrucción, teniendo cada uno su propio flujo de datos. Este paradigma es también llamado programación paralela de datos.
- Lenguajes SMPD: Son una clase especial de los programas SIMD pero a nivel de programa mas que al nivel de instrucción, o sea un único programa y múltiples datos, enfatizando el paralelismo de grano medio y la sincronización al nivel de subprograma.
- Lenguajes MIMD: Cada procesador tiene su propio programa a ejecutar.
- Lenguajes MPMD: Es un subconjunto del MIMD a nivel de programa, o sea múltiples programas y múltiples datos.

Existen también otras estrategias para facilitar la programación en ambientes paralelos, entre ellas están:

- *Compiladores que reconocen las partes de código secuencial y las paraleliza:* El sistema de uso de los compiladores paralelizantes es bastante sencillo: se hace el programa secuencial, y el propio compilador busca el paralelismo intrínseco al programa y lo explota. Desde el punto de vista del programador este es el enfoque óptimo. Estos compiladores tienen una escasa disponibilidad. Esta área aún no está completamente desarrollada; son muy difíciles de construir y, hasta los mejores raramente extraen todo el paralelismo inherente al código. Además, solo existen soluciones para máquinas paralelas, y no existen grandes soluciones conocidas de compiladores que generen automáticamente código para sistemas distribuidos. Por último, la totalidad de los compiladores paralelizantes acaba generando código con alto grado de acoplamiento, por lo que necesitan una máquina que sea capaz de tener alto rendimiento con programas paralelos de alto acoplamiento para que el resultado sea razonable. Por estas razones, a pesar de que hoy en día hay muchos investigadores trabajando en el tema, aún no se puede decir que exista una solución satisfactoria.
- *Soporte de multiprocesamiento para el sistema operativo:* Otro mecanismo podría ser emplear un sistema operativo que oculte a los procesos que se estén empleando que se ejecutan en una máquina paralela, de forma que los programas se diseñen de forma secuencial y, cuando varios programas se ejecuten concurrentemente, el sistema operativo asigne a cada proceso concurrente un procesador distinto para su ejecución. Este enfoque tiene el problema de necesitar un computador con más de un procesador, además el incremento de potencia puede que no sea suficiente como para dejar todo el peso del trabajo a los computadores de este tipo que aparecen normalmente en el mercado, por ejemplo los computadores duales. Por ello, si bien podemos aprovechar este hecho instalando el soporte de SMP necesario, se necesita completarlo con otro método para interconectar varios computadores.
- *Bibliotecas concurrentes:* Otro método a usar sobre un lenguaje no concurrente y una máquina monoprocesador son las bibliotecas que nos permitan interconectar varias máquinas. Este mecanismo tiene el inconveniente de que el programador acaba haciendo todo el trabajo de concurrencia; a cambio, nos da libertad para escoger como va a ser la comunicación. De hecho, tiene buen rendimiento, y permite adaptar el grado de acoplamiento del paralelismo según la arquitectura de la máquina y la naturaleza del problema, algo que es más difícil en los lenguajes concurrentes y casi imposible en los compiladores paralelizantes. Estas son las más extendidas, y sobre ellas se han desarrollado varios ambientes de programación que se explican a continuación.

2.1.6. Ambientes de programación

Se han desarrollado diferentes paquetes de software que intentan facilitar el trabajo de los programadores en la computación paralela. Entre los más conocidos están los siguientes:

p4 System

p4 [67, 68] es una biblioteca de macros y subrutinas desarrolladas por Argonne National Laboratory que soporta lo mismo el modelo de memoria compartida (basada en monitores) como el modelo de memoria distribuida (usando paso de mensajes). Para el modelo de memoria compartida brinda un conjunto de monitores útiles y de primitivas desde las cuales se pueden construir los monitores. Para el modelo de memoria distribuida brinda operaciones de envío y recepción, así como creación de procesos. El proceso de creación de procesos es estático y se hace a través de un fichero de configuración. La creación de procesos de manera dinámica es solo posible por un proceso creado estáticamente que invoca una función especial para crear un nuevo proceso.

Express

Express [69] es un conjunto de herramientas desarrollado por ParaSoft Corporation que de manera individual resuelven diferentes aspectos de la computación paralela. Su filosofía se basa en seguir el desarrollo del ciclo de vida de una aplicación secuencial hasta convertirla en una versión paralela.

Linda System

Linda [70] es un modelo desarrollado por la Universidad de Yale. En él, en vez de emplear modelos de memoria compartida tradicional o de paso de mensajes, se emplea como abstracción un espacio de tuplas. La característica básica de Linda es ser un modelo de memoria asociativa compartida, que todos los procesos pueden insertar y extraer tuplas, siempre de forma asociativa, del espacio de tuplas. La memoria subyacente puede ser distribuida, pero esto queda oculto por el propio modelo. El sistema, como tal, es un conjunto de rutinas que oculta el hardware y permite realizar las operaciones de inserción y extracción en el espacio de tuplas. Esta abstracción es muy elegante y fácil de usar, aunque es bastante compleja de adaptar a un problema. Dependiendo del ambiente el mecanismo del espacio de tuplas se implementa usando diferentes técnicas y grados de eficiencia. Una evolución del concepto de Linda es el modelo Pirhana [71], en la que los recursos computacionales se comportan como agentes inteligentes, realizando ellos mismos la búsqueda de las tareas.

Sockets

Es uno de los mejores mecanismos para trabajar con sistemas distribuidos debido a su eficiencia. Son portables, sobradamente documentados y conocidos. Están disponibles libremente para casi todas las computadoras que existen. Sin embargo, tiene algunos problemas. El más importante es que la complejidad para ser programado es muy alta respecto a otras abstracciones, ya que los mecanismos de comunicación entre máquinas son muy complejos usando sockets respecto a las bibliotecas anteriormente citadas. Al usarse en computadores heterogéneos sobre los que se ejecutarían los programas, se deben programar rutinas de codificación y decodificación de datos para todas las arquitecturas posibles, y si aparece un computador nuevo, hay que programar nuevas rutinas de conversión de datos específicas para éste.

MPI

MPI (Message Parsing Interface) [72]. Es el estándar “oficial” de la industria de máquinas

paralela creado por un amplio comité de expertos y usuarios en 1994 con el objetivo de definir una infraestructura común y una semántica y sintaxis específica de interfaz de comunicación. De esta forma los productores de software de procesamiento paralelo pueden implementar su propia versión de MPI siguiendo las especificaciones de este estándar, con lo cual múltiples implementaciones de MPI cambiarían solamente en factores tales como la eficiencia de su implementación y herramientas de desarrollos, pero no en sus principios básicos. Está más orientada a ser un estándar de paso de mensajes para todos los supercomputadores paralelos que un rival para otras bibliotecas.

2.1.7. PVM, Máquina Virtual Paralela

Otro de los ambientes de programación existentes es PVM y que es el que se usa en este trabajo teniendo en cuenta las ventajas que se explicarán a continuación y el hecho de tener el autor alguna experiencia en su uso en el desarrollo de aplicaciones.

PVM (Paralell Virtual Machine) [73, 74] es un conjunto de herramientas software que permiten emular un marco de computación concurrente, distribuido y de propósito general utilizando para ello tanto computadores como redes de interconexión heterogéneas, de forma que trabajen como una única maquina computadora paralela de alto rendimiento.

PVM posibilita crear una máquina virtual paralela como una abstracción empleando los recursos disponibles de todos los computadores que integran su entorno de computación. Permite escribir programas que empleen una serie de recursos capaces de operar independientemente; pero que se coordinan y cooperan para obtener un resultado global que dependa de los cálculos realizados en paralelo por todos ellos. Gracias a ello, ofrece al programador una vista lógica de todo el hardware subyacente de forma que solo ve un conjunto de tareas que se comunican entre sí.

Es decir, se emplean los recursos hardware de dicho paradigma; pero programando el conjunto de máquinas como si se tratara de una sola máquina paralela, que es mucho más cómodo.

2.1.7.1. Características de la PVM

Las características principales de la PVM que muestran sus principios y ventajas son:

- *Computación basada en procesos:* La unidad de concurrencia es el proceso. Permite crear cualquier número de procesos independientemente del número de procesadores asignando los procesos a los procesadores de forma automática o preasignándolos el programador directamente.
- *Modelo basado en paso de mensajes:* Las tareas que componen la aplicación pueden dialogar y cooperar mediante el intercambio explicito de mensajes entre ellas, con independencia de que se haya optado por un paralelismo funcional o a nivel de datos.
- *Heterogeneidad a nivel de computadores, redes y aplicaciones:* Los computadores que forman la máquina virtual pueden ser o no de distintas arquitecturas y poseer cada uno características diferentes. También se abstrae de la topología y tecnologías de la red de interconexión, permitiendo combinaciones de redes locales y redes de área extendida. Las aplicaciones pueden estar escritas en C, C++, Java y/o Fortran.

- *Acceso explícito al hardware:* Los programadores pueden usar la máquina virtual como una colección genérica de procesadores, u optar por asignar explícitamente algunas tareas a aquellos procesadores que sean más adecuados para el fin de la aplicación.
- *Soporte para multiprocesadores:* Los computadores que conforman la máquina virtual pueden ser multiprocesadores de memoria distribuida o compartida. Ante las restantes máquinas, un multiprocesador se puede presentar como una máquina secuencial o como varias.
- *Facilidad de ampliación:* El número de computadores que forman la máquina virtual puede ser alterado, añadiendo o eliminando unidades de forma dinámica durante la ejecución de una aplicación.
- *Tolerancia a fallos:* Como consecuencia derivada del punto anterior, PVM puede detectar de forma automática cuando un computador o una red falla y brinda al programador todas las facilidades que permitan entonces realizar una reconfiguración de la máquina virtual.
- *Dominio público:* El paquete PVM se distribuye como un software de uso público y se puede bajar gratuito de internet [74].

Por otro lado, en la PVM podemos encontrarnos con disfunciones si el diseño de la red no es suficientemente avanzado debido a posibles colisiones en la demanda de recursos ya muchas aplicaciones en paralelo pueden inducir un acoplamiento fuerte. Además la abstracción de la máquina virtual, la independencia del hardware y la independencia de la codificación tienen un coste. La PVM no va a ser tan rápida como los sockets. Sin embargo, si el grado de acoplamiento se mantiene lo suficientemente bajo, no es observable esta diferencia.

2.1.7.2. Arquitectura de la PVM

La arquitectura de PVM se basa en considerar que una aplicación es una colección de tareas que se comunican y sincronizan mediante el paradigma de paso de mensajes. Las características más importantes que proporciona este modelo de paso de mensajes de PVM son:

- *Cada tarea puede enviar un mensaje a cualquier otra tarea:* No hay límites en el número y tamaño de los mensajes, condicionados sólo por la cantidad de memoria disponible en cada uno de los computadores que forman la máquina virtual para construir los buffers de envío y recepción.
- *Envío asíncrono con bloqueo:* Este tipo devuelve el control a la tarea tan pronto como el buffer de envío esté libre para ser utilizado de nuevo, con independencia de que el destinatario haya realizado una llamada a una función de recepción.
- *Recepción asíncrona con/sin bloqueo:* Una recepción no bloqueante devuelve el control a la tarea con independencia de la llegada de los datos, existiendo una bandera de control que permite determinar qué ha ocurrido en la recepción. Por el contrario, la recepción bloqueante no devuelve el control a la tarea en tanto no se produzca la llegada de los datos al buffer de recepción.
- *Se garantiza el orden de los mensajes:* El modelo asegura que los mensajes enviados por una tarea X a una Y llegarán siempre en el mismo orden. También proporciona funciones de filtrado que permiten extraer un mensaje de la cola de llegadas.

- *Envío de mensajes en modo difusión (broadcast):* Un mensaje puede ser enviado a un grupo de tareas mediante una sola función de envío.
- *Grupos dinámicos de tareas:* Permite agrupar un conjunto de tareas como un todo, permitiendo que una tarea pueda formar parte de diferentes grupos y salir y entrar a ellos dinámicamente.
- *Envío de señales y eventos asíncronos:* PVM oferta la posibilidad del envío de señales entre tareas así como la recepción de eventos asíncronos.

2.1.7.3. Modelos de computación en PVM

PVM considera que una aplicación se descompone en un conjunto de tareas que se ejecutan en los distintos computadores que forman la máquina virtual y este modelo posibilita la paralelización explícita de la aplicación desde varios puntos de vista.

PVM reconoce el paralelismo a nivel funcional o paralelismo a nivel de datos y soporta ambos paradigmas de programación o una mezcla de ellos La primera implica que cada tarea realiza una función diferente, esto se ve con un enfoque MPMD (Múltiples Programas, Múltiples Datos). La segunda, bajo un enfoque SPMD (Único Programa, Múltiples Datos), consiste en que cada tarea ejecuta el mismo código pero sobre un subconjunto diferente de datos. Dependiendo de las funciones, cada tarea puede ejecutarse en paralelo y puede necesitar sincronizarse o intercambiar datos con otra tarea.

Por otro lado, PVM permite la utilización de diferentes paradigmas algorítmicos de paralelización permitiendo varias formas de estructurar los algoritmos para su ejecución. Según la estructuración de las tareas, se tienen estos modelos:

- *Fases paralelas:* Se basa en la sucesión de un número indeterminado de fases para alcanzar la solución, estando cada fase compuesta de dos etapas: cálculo e interacción. En la etapa de cálculo, cada tarea realiza un conjunto de operaciones de forma independiente. Concluida ésta se pasa a la etapa de interacción, en la que se realiza una comunicación síncrona entre todas las tareas. Luego es que se aborda la siguiente fase. Este modelo se caracteriza por facilitar la depuración y análisis de la aplicación; pero presenta el inconveniente de que las etapas de cálculo e interacción no se solapan.
- *Segmentación:* Las tareas de la aplicación se comportan como etapas de una segmentación virtual a través de la cual los datos van circulando de forma continua y se ejecutarán de forma solapada sobre diferentes subconjuntos de la carga de trabajo.
- *Maestro - esclavo:* Una tarea maestra realiza la parte secuencial del algoritmo y lanza un conjunto de tareas esclavas para ejecutar de forma paralela la carga de trabajo. La tarea maestra es la encargada de crear e inicializar las tareas esclavas, de recoger los resultados y de realizar las funciones de sincronización. Este modelo consta de tres fases. La primera consiste en la inicialización de los grupos de procesos y la distribución dinámica o estática de la carga de trabajo. La segunda es la etapa de cálculo y la tercera fase es la recopilación y visualización de los resultados. El principal inconveniente es que el maestro puede convertirse en un verdadero cuello de botella.
- *Híbrido:* Es una mezcla de los anteriores. Se basa en una generación arbitraria de la estructura de las tareas en función de la carga de trabajo a ser procesada, obteniéndose un grafo

dinámico que va evolucionando de acuerdo a la carga de trabajo.

- *Arborescente:* Un proceso padre divide la carga de trabajo entre varios procesos hijos, que a su vez realizan el mismo proceso de división sobre el subconjunto de la carga de trabajo que les ha correspondido. Al final, el proceso padre original recogerá los resultados parciales de los hijos y los combinará para obtener un resultado final. Este modelo es adecuado para problemas en los que la carga de trabajo se desconoce a priori y su dificultad radica en el correcto balanceo de la carga.

De entre los modelos anteriores hemos usado en este trabajo el modelo maestro - esclavo, donde los esclavos son los diferentes hilos de búsqueda y el maestro es el coordinador de los mismos. Más adelante, en el capítulo 3, daremos detalles de esta implementación.

2.2. Metaheurísticas Paralelas

Desde el punto de vista del diseño de algoritmos, las estrategias de computación paralela "puras" explotan el orden parcial de las instrucciones de los algoritmos, o sea, los conjuntos de operaciones que puedan ser ejecutados concurrentemente en el tiempo, sin modificar el método de solución y la solución final obtenida, y así se corresponden al paralelismo natural presente en los algoritmos.

Como se explicó anteriormente el orden parcial de los algoritmos brinda dos fuentes principales de paralelismo: paralelismo de datos y paralelismo funcional.

La computación paralela basada en el paralelismo de datos o funcional es particularmente eficiente cuando los algoritmos manipulan estructuras de datos que sean fuertemente regulares, como por ejemplo las matrices en una operación de multiplicación. Cuando los algoritmos manipulan estructuras de datos irregulares, como los grafos, o manejan datos con dependencias fuertes entre las diferentes operaciones se dificulta obtener una paralelización eficiente usando solo paralelismo funcional o de datos.

Las metaheurísticas pertenecen a esta categoría de algoritmos que son difíciles de paralelizar [75], o sea tienen limitado el paralelismo de datos o funcional; pero como métodos de solución de problemas ofrecen otras oportunidades para la computación paralela.

2.2.1. Estrategias de paralelización de metaheurísticas

Una metaheurística puede comenzar desde diferentes soluciones iniciales para así explorar diferentes regiones del espacio de búsqueda y retornar diferentes soluciones. Estas regiones diferentes del espacio de búsqueda se pueden convertir en una fuente de paralelismo, sin embargo el análisis de sus implementaciones resultan complejos.

Así, de acuerdo a la fuente de paralelismo usada, se reconocen 3 estrategias para paralelizar las metaheurísticas [75]:

- *Paralelismo del Tipo 1 o de Bajo Nivel*: Esta fuente de paralelismo se encuentra normalmente dentro de una iteración del método de búsqueda, y se puede obtener a través de ejecuciones concurrentes de las operaciones o de evaluaciones concurrentes de varios movimientos que

llevan a cabo una iteración del método de búsqueda. Está orientado directamente a reducir el tiempo de ejecución del método y no a lograr mayor exploración o soluciones de mayor calidad; y conviene señalar que, dado el mismo número de iteraciones, tanto la implementación secuencial como en paralelo produce el mismo resultado. Algunas implementaciones modifican el método secuencial para tomar ventajas de la potencia de cálculo extra disponible; pero sin alterar el método básico de búsqueda. Por ejemplo evaluar el vecindario de la solución actual con varios movimientos en paralelo en vez de usar uno solo.

- *Paralelismo del Tipo 2 o Descomposición del Dominio*: Se obtiene el paralelismo descomponiendo las variables de decisión en conjuntos disjuntos. Esta descomposición reduce el tamaño del espacio de búsqueda, pero se necesita ser repetida para permitir la exploración completa del espacio de solución. Una heurística particular es aplicada a cada variable en el subconjunto que se trate, los otros se consideran fijos. Esta estrategia se implementa generalmente en un modelo maestro-esclavo, donde el proceso maestro divide las variables de decisión. Durante la búsqueda, el maestro puede modificar esta división. Estas modificaciones que se hacen a intervalos pueden ser fijadas antes o durante la ejecución, o ser ajustadas en el momento de reiniciar el método. Los esclavos exploran de forma concurrente e independiente solo su espacio asignado. Las otras variables se mantienen fijas y no son afectadas por los movimientos realizados, aunque también los esclavos pudieran tener acceso a todo el conjunto global de variables. Cuando los esclavos tienen acceso a todo el vecindario, el maestro debe realizar operaciones complejas de combinación de las soluciones parciales obtenidas de cada subconjunto para formar una solución completa del problema.

- *Paralelismo del Tipo 3 o Búsqueda Múltiple*: Se lleva a cabo con varias búsquedas concurrentes en el espacio de soluciones. Cada hilo concurrente puede o no ejecutar el mismo método, puede empezar por las mismas o diferentes soluciones iniciales, etc. Los hilos pueden comunicarse durante la búsqueda o sólo al final para identificar la mejor de todas las soluciones. La primera estrategia se denomina método cooperativo de búsqueda múltiple y la segunda es conocida como método de búsqueda independiente. La comunicación puede ser realizada de manera sincrónica o asincrónica y puede ser manejada por eventos o ejecutada en momentos decididos dinámicamente o predeterminados de antemano. Esta estrategia se usa frecuentemente para llevar a cabo una mayor exploración del espacio de búsqueda. Varios estudios [55, 76] han mostrado que las técnicas multihilos proporcionan mejores soluciones que su correspondiente contraparte secuencial, incluso cuando el tiempo disponible de ejecución para cada hilo es menor que el de la computación secuencial. Dichos estudios han mostrado también que la combinación de distintos hilos, implementando cada una de ellos diferentes estrategias, incrementa la robustez de la búsqueda global en relación con las variaciones de las características de las instancias del problema. También indican que mientras la estrategia de búsquedas independientes es fácil de implementar y obtiene buenos resultados, éstos se pueden mejorar usando la estrategia cooperativa de búsqueda múltiple.

Habría que señalar que el uso de las medidas clásicas de desempeño para estas metaheurísticas en paralelo es en cierto modo problemático [55]. El desempeño de los algoritmos paralelos está fuertemente vinculado al ambiente de programación en el que nos encontremos, así como de factores cuya homogeneidad en el campo secuencial se da por entendida: diseño de la red de comunicaciones, aspectos de sincronización, latencia de los mensajes, etc. Además los hilos con frecuencia interactúan asincrónicamente, de forma que las distintas ejecuciones pueden producir distintas salidas para la misma entrada. De esta manera, el tema de comparación de estas estrategias en paralelo con sus correspondientes secuenciales es un campo abierto y activo como puede

leerse en la cita anterior.

Para calificar la medida clásica del factor de aceleración se usa la noción de calidad de la solución, o sea comparar cuál método encuentra la mejor solución, aunque esto tampoco permite una comparativa clara debido a lo complejo de lograr recursos computacionales similares.

2.2.2. Otras taxonomías

Muchas de las clasificaciones acerca de las metaheurísticas paralelas que se proponen en la literatura se relacionan a un tipo específico de metaheurística, así:

Greening [77] divide las estrategias paralelas del Recocido Simulado de acuerdo al grado de seguridad en la evaluación de la función de costo asociada con un movimiento. Los algoritmos que brindan una evaluación libre de error las identifica como sincrónicos y las otros como asincrónicos. A su vez los sincrónicos se dividen en los que mantienen las propiedades de convergencia del método secuencial (parecido al serie) y aquellos que tienen una evaluación segura pero difieren del secuencial en la generación del patrón de búsqueda (generación alterada).

Cantú-Paz [78] da una clasificación de los algoritmos genéticos paralelos. La primera categoría llamada paralelización global es idéntica a la paralelización del tipo 1. Las otras dos categorías clasifican los algoritmos genéticos de acuerdo al tamaño de su población, así están los de grano grueso y los de grano fino. Hay también una clase de algoritmos genéticos paralelos híbridos, por ejemplo para enmarcar aquellos de paralelización global que tienen subpoblaciones de algoritmos de grano fino.

Por otro lado, de una forma más general en [79, 80, 61] presentan una clasificación de acuerdo al número de trayectorias investigadas en el vecindario de búsqueda, en la cual los métodos se pueden clasificar en: métodos de un único camino y métodos de múltiples caminos.

En el caso de la paralelización de un único camino el objetivo es aumentar la velocidad del recorrido secuencial del vecindario de búsqueda. La tarea cuya ejecución se paraleliza entre los procesadores puede ser la evaluación de la función de coste de cada vecino de la solución actual o la construcción del vecindario. En el primer caso, el aumento de la velocidad puede obtenerse sin modificación en la trayectoria seguida por la implementación secuencial. En el segundo caso, se descompone el vecindario y se distribuye entre varios procesadores permitiendo un examen profundo a una porción mayor del vecindario que el examinado por una implementación secuencial, la cual usa frecuentemente técnicas de reducción del vecindario. Así, la trayectoria seguida en el vecindario puede ser guiada mejor en modo paralelo que en el modo secuencial y posiblemente conduciendo a soluciones mejores. La idea de vecindario distribuido sirve para entender y formalizar la paralelización basada en descomposición del dominio. Cada iteración de la búsqueda local comienza enviando la solución actual a todos los procesadores, junto con una definición de vecindarios locales, o sea, porciones del vecindario global las cuales serán investigadas por cada procesador. Esta descomposición es temporal y puede cambiar de un vecindario a otro. Luego, cada procesador encuentra y propone un movimiento dentro de su vecindario local. Estos movimientos se combinan o se selecciona el mejor movimiento encontrado dentro del vecindario local, y así una solución posible es generada. Esto se repite hasta que la solución actual no puede ser mejorada mucho más. Este tipo de paralelización de un único camino tiene granularidad media o pequeña y necesita sicronizaciones frecuentes, es dependiente de la aplicación en el sentido de la estructura del vecindario y la definición de la función de costo. Sus primeras aplicaciones aparecieron bajo el Templado Simulado y los Algoritmos Genéticos [61].

La paralelización de múltiples caminos se caracteriza por la investigación en paralelo de múltiples trayectorias, cada una en un procesador diferente. Las hebras o hilos de búsqueda pueden ser independientes o cooperativos. En los hilos independientes hay dos métodos. Uno, cuando cada camino arranca desde una solución o con una población diferente. Aquí las hilos pueden usar el mismo algoritmo o uno diferente, con los mismos parámetros o no y no intercambian información entre ellos durante la búsqueda. Y el otro método es cuando se descompone el dominio en varios vecindarios y cada uno se explora en paralelo sin intersección entre las diferentes trayectorias. Esta estrategia se implementa con facilidad y puede lograrse que sea muy robusta ajustando los parámetros de cada procesador de manera diferente.

Los hilos cooperativos intercambian y comparten información recogida a lo largo de las trayectorias que investigan y esta es usada para mejorar las otras trayectorias. Esta información compartida puede ser implementada lo mismo como variables globales en una memoria compartida, o como una cola en una memoria local de un procesador dedicado que pueda ser accedido por los otros procesadores. Con este método no solo se espera acelerar la convergencia a las mejores soluciones sino también encontrar soluciones de mas calidad que las obtenidas por las estrategias independientes en el mismo tiempo de cómputo.

Es por ello que esta es la estrategia elegida en nuestro trabajo con el fin de cubrir los objetivos de robustez e independencia del problema de optimización que se aborde.

2.2.3. Estado del arte

Varias propuestas de paralelizar metaheurísticas aparecen en la literatura, muchas de ellas con resultados exitosos para ciertos tipos de problemas. Una referencia a varios de estos trabajos aparecen en [61, 81].

Los métodos multihilos o de búsqueda múltiple ofrecen las perspectivas muy interesantes en este campo, y algunos trabajos que muestran esto aparecen referenciados en [82].

Por ejemplo, las implementaciones sincrónicas de estos métodos, donde la información se intercambia a intervalos regulares y que de manera general obtienen mejores resultados en la calidad de la solución que los métodos secuenciales. Crainic [83] en la Búsqueda Tabú y Grafiggne [84] en el Templado Simulado a su vez muestran que los métodos de búsqueda independiente mejoran a los métodos cooperativos sincrónicos. Pero Lee y Lee [85] usando un método de sincronización del intervalo ajustado dinámicamente obtiene mejores resultados que los métodos independientes. Cohoon et al.[86] muestran que la búsqueda paralela con operadores de migración aplicados de forma sincrónica mejora los resultados respecto al mismo método sin la migración.

Últimamente se destaca el diseño de implementaciones asincrónicas cooperativas multihilos, o también llamadas búsquedas cooperativas multinivel [76] que según los informes que aparecen en [75] ofrecen mejores resultados que las implementaciones sincrónicas y la búsqueda independiente. También han observado que la cooperación debida al intercambio de información no solo mejora el factor de aceleración sino que modifica los patrones de búsqueda de los programas. Otros trabajos que muestran un rendimiento superior de los métodos cooperativos aparecen referenciados en [82].

Estos métodos no se basan en una descomposición explícita del dominio del problema. El paralelismo se logra lanzando concurrentemente varios algoritmos de búsqueda independientes, donde se usan diferentes soluciones iniciales y diferentes valores de los parámetros de búsqueda para cada algoritmo independiente, lo cual conlleva una descomposición implícita del espacio de

solución.

Los sistemas de colonias de hormigas y los métodos basados en enjambres aparecen como uno de los primeros mecanismos cooperativos inspirados en la naturaleza; pero el principio de cooperación que hasta ahora presentan es demasiado rígido para ofrecer un método de propósito general [75]. La búsqueda dispersa por otro lado ofrece un mecanismo de combinación relativa al contexto y memorias que manejan la evolución de la población. En los sistemas cooperativos multihilos el intercambio de información va más allá del intercambio según principios genéticos, aunque los operadores genéticos pueden usarse para controlar a éste.

Un único mecanismo de cooperación no cubre todas las posibilidades y se recomienda el uso de mecanismos híbridos donde se combine diferentes heurísticas. Buenos ejemplos de esto son el trabajo de Krasnogor en los algoritmos meméticos autogenerados [87], los de Crainic [82] y Le Boutillier [88] que demuestran que con esta combinación se logran resultados superiores. Se abre así todo un campo a investigar en el que surgen preguntas tales como: ¿Cuáles metaheurísticas combinar? ¿Qué rol jugaría cada tipo? ¿Qué mecanismos de cooperación usar?.

Se han hecho algunas investigaciones [89] para intentar identificar algún tipo de comportamiento debido a las interacciones entre dos o más procesos de búsqueda de manera que permita diseñar métodos de cooperación más eficientes. Los métodos cooperativos de búsqueda no se basan en una descomposición explícita del dominio del problema. Estos lanzan de manera concurrente varios procesos o hilos de búsqueda de forma independiente que al ejecutarlos con soluciones iniciales y/o parámetros de búsqueda diferentes ya hacen una descomposición implícita del espacio de soluciones. Se define entonces un método que permita que estos procesos se comuniquen y colaboren entre sí.

Uno de los métodos de cooperación más usado es compartir la información recopilada en la búsqueda. Este mecanismo se basa casi siempre en tener en cuenta al menos cuatro parámetros:

- la información recopilada disponible para compartirla,
- identificar quienes comparten la información,
- la frecuencia en que se comparte la información y
- el número de procesos concurrentes.

Estos parámetros se identifican en lo que la literatura llama “parámetros del sistema de control” [90] y que sus valores impactan en la organización de la cooperación como una entidad global, a diferencia de parámetros de cada proceso individual que solo afecte su patrón de búsqueda.

Los métodos cooperativos logran un factor de aceleración similar a los independientes de búsqueda, solo que éstos desafortunadamente pueden resultar inestables y aunque muchas veces mejoran el factor en algunas instancias del problema, la degradan en otras. Los cooperativos obtienen soluciones de mejor calidad que las logradas por métodos independientes.

La información a compartir por los procesos o hilos de búsqueda debe ser significativa, en el sentido de que debe ser útil para el proceso de decisión de la búsqueda en cada uno de ellos. Debe ser información que brinde indicaciones correctas acerca del estado actual de la búsqueda global.

¿Qué información debe intercambiarse? ¿Cuándo hacer el intercambio de información? ¿Qué hacer con la información? Estas preguntas aún quedan sin una respuesta clara, ya que todavía no se han definido procedimientos para definir estos parámetros teniendo en cuenta que

intervienen muchos otros factores. Este trabajo pretende aportar información y criterios para responder a algunas de estas preguntas en el contexto de la optimización combinatoria. Concretamente en el capítulo 5 diseñamos experimentos con el fin de dirimir qué sentido tiene el uso de la memoria en la búsqueda multihilos.

Estudios previos [82, 89] demuestran que estos métodos cooperativos con acceso no restringido a la información compartida pueden experimentar serios problemas de convergencia. Esto parece ser debido a la influencia de los contenidos de la memoria compartida por intensos intercambios de las mejores soluciones encontradas por los procesos de búsquedas y la subsiguiente estabilización de los contenidos de la memoria compartida y la información compartida.

Estos estudios también resaltan el hecho de que el simple intercambio de información, por ejemplo el mejor valor, puede dirigir la búsqueda a regiones no esperadas sin tener en cuenta una lógica en el proceso de optimización.

De todo esto se concluyen algunos principios:

- Las metaheurísticas a menudo poseen fuertes dependencias de datos. Por tanto, intentar aplicarles directamente técnicas de paralelización funcional o de datos solo podrían identificar un paralelismo limitado.
- No obstante, aplicarle la paralelización es muy beneficioso, solo que hay buscar los componentes en los cuales se obtienen ganancias computacionales significativas, tales como la evaluación del vecindario de soluciones.
- Las estrategias de paralelización de las metaheurísticas serán mas ventajosas si se incorporan esquemas jerárquicos donde el más alto nivel lo mismo explore divisiones del dominio del problema, que implemente técnicas cooperativas multihilos de búsqueda.
- Usar métodos híbridos, donde se incorporen principios y estrategias de una clase de metaheurística a otra, puede lograr mejores rendimientos tanto en los algoritmos secuenciales como en los paralelos.
- Los métodos paralelos cooperativos multihilos se presentan como los más promisorios no solo porque mejoran la velocidad y el factor de aceleración, sino también la calidad de las soluciones, la eficiencia computacional y la robustez de la búsqueda.
- Hay un campo abierto a la experimentación de estas técnicas y de sus mecanismos de cooperación.

2.3. Conclusiones

En este capítulo se dieron conceptos básicos acerca de la Computación Paralela. Uno de los objetivos de este trabajo abarca el campo de la paralelización de metaheurísticas en la resolución de problemas de optimización. Por este motivo hemos profundizado en el análisis de la programación paralela, los algoritmos paralelos, los métodos de la programación paralela y fundamentalmente los ambientes de programación. Hemos elegido como ambiente de programación paralela para el desarrollo de este trabajo la PVM. Los motivos por lo que se ha tomado esta decisión se exponen en este capítulo. Hemos también profundizado en los conceptos de las metaheurísticas paralelas, estrategias de paralelización, taxonomías y hemos hecho una revisión del estado del arte donde se muestran referencias a varias propuestas con resultados exitosos.

Capítulo 3

Estrategia Cooperativa Paralela basada en Soft-Computing

A pesar de los resultados prometedores de las diferentes metaheurísticas analizadas, es difícil encontrar en la bibliografía estrategias para resolver problemas de optimización que posean características generales, que sean independientes del problema, robustas, fáciles de implementar y que produzcan resultados aceptables en tiempos razonables. Por otro lado, resultados teóricos relativamente recientes son desalentadores en este sentido [**?**]. Esto nos lleva a pensar en la posibilidad de reunir diferentes estrategias y métodos de búsqueda bajo un mismo esquema coordinado de forma que podemos soslayar el conocido resultado citado anteriormente.

En este capítulo se presentará nuestra propuesta para resolver problemas de optimización combinatoria, una estrategia cooperativa basada en reglas difusas que ejecuta en forma paralela un conjunto de algoritmos, y controla y modifica su comportamiento de manera coordinada. Para ello, se mostrarán los detalles de su implementación mostrando el pseudocódigo de sus componentes y explicando la base de reglas que definen sus diferentes mecanismos de coordinación.

Se explica en detalles la metaheurística FANS, por su uso como parte fundamental de la estrategia propuesta en este trabajo. Se describen los elementos básicos del algoritmo y su esquema general, destacando la utilidad de su principal componente: la valoración difusa, y se muestra como esta puede ser utilizada para variar el comportamiento del algoritmo.

Como introducción al capítulo se describirán conceptos básicos de la Soft-Computing y su utilidad al campo de las metaheurísticas.

3.1. Soft-Computing, conceptos básicos

La asignación de recursos limitados o bajo restricción y el control de acciones cambiantes con el tiempo son situaciones a las que constantemente se enfrentan los directivos en toda institución, ya sea pequeña o grande, pública o privada. Son problemas en los cuales los modelos y sus respectivos métodos de solución existentes, conocidos muchas veces, no representan a la realidad, o sus métodos de solución presentan limitaciones muy serias dejando un campo abierto por modelar y donde plantear los correspondientes métodos de solución.

De esta amplitud de problemas, se destacan estas situaciones:

- cuando los métodos de solución existentes que sirven para casos simples presentan limitaciones o no se pueden utilizar en casos complejos.
- cuando los datos no se conocen exactamente, es decir, cuando los datos son imprecisos y se está en un ambiente semi-estructurado.

En el primer caso están incluidos aquellos problemas en los cuales el uso de métodos exactos de solución de los modelos existentes tiene limitaciones en el tiempo de ejecución, cuando se utilizan dichos métodos en situaciones con muchas variables de decisión y muchas restricciones. En estos casos hace que se enfoque su solución entendiendo que es mejor satisfacer que optimizar. O sea, aunque en muchos casos no se encuentren las soluciones exactas, si se encontrarán buenas soluciones. Se hace notar la vaguedad que hay en definir esa satisfacción; por ejemplo, podría ser que el decisor deseara una solución "aceptable".

A partir de que se publicara en 1965 el primer artículo sobre Conjuntos Difusos, y luego se hiciera la definición de *Soft-Computing* se comienza a abordar con cierto éxito tanto la primera como la segunda situación.

Ahora, ¿qué se entiende por *Soft-Computing*?

Según Zadeh [91]: "*Básicamente, Soft Computing no es un cuerpo homogéneo de conceptos y técnicas. Mas bien es una mezcla de distintos métodos que de una forma u otra cooperan desde sus fundamentos. En este sentido, el principal objetivo de la Soft Computing es aprovechar la tolerancia que conllevan la imprecisión y la incertidumbre, para conseguir manejabilidad, robustez y soluciones de bajo costo. Los principales ingredientes de la Soft Computing son la Lógica Fuzzy, la Neuro-computación y el Razonamiento Probabilístico, incluyendo este último a los Algoritmos Genéticos, las Redes de Creencia, los Sistemas Caóticos y algunas partes de la Teoría de Aprendizaje. En esa asociación de Lógica Fuzzy, Neurocomputación y Razonamiento Probabilístico, la Lógica Fuzzy se ocupa principalmente de la imprecisión y el Razonamiento Aproximado; la Neurocumputación del aprendizaje, y el Razonamiento Probabilístico de la incertidumbre y la propagación de las creencias.*"

Hay algunos intentos de ajustar mas esta definición; así, por ejemplo en [92], se define como: "*Cualquier proceso de computación que expresamente incluya imprecisión en los cálculos en uno o mas niveles, y que permita cambiar la granularidad del problema o suavizar los objetivos de optimización en cualquier etapa, se define como perteneciente al campo de la Soft Computing.*"

Existe otro punto de vista interesante que da Verdegay [93] como otra forma de definir Soft Computing: "*Se trata de considerarla como antítesis de lo que se denomina Hard Computing, de manera que podría verse la Soft Computing como un conjunto de técnicas y métodos que permitan tratar las situaciones prácticas reales de la misma forma que suelen hacerlo los seres humanos, es decir, en base a inteligencia, sentido común, consideración de analogías, aproximaciones, etc. En este sentido Soft Computing es una familia de métodos de resolución de problemas cuyos primeros miembros serían el Razonamiento Aproximado y los Métodos de Aproximación Funcional y de Optimización, incluyendo los de búsqueda*" (ver en la Figura 3.1).

En este sentido, la Soft Computing queda situada como la base teórica del área de los Sistemas Inteligentes, y se hace patente que la diferencia entre el área de la Inteligencia Artificial clásica, y la de los Sistemas Inteligentes, es que la primera se apoya en la denominada Hard Computing, mientras que la segunda lo hace en la Soft Computing.

Moviéndonos a un mayor nivel de profundidad del esquema, tal como se muestra en la Figura

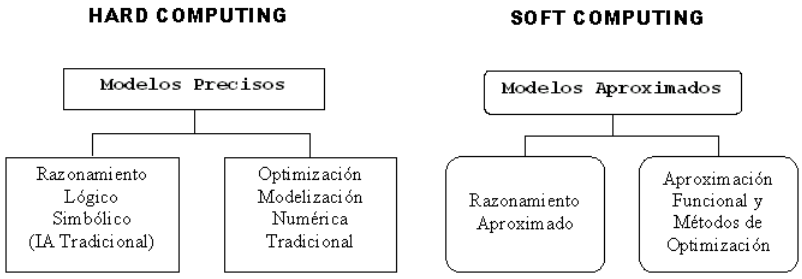


Figura 3.1: Hard Computing - Soft Computing

3.2 el área que abarca la Aproximación Funcional/Métodos de Optimización tiene un primer componente que lo forman las Redes Neuronales, y otro compuesto por las Metaheurísticas.

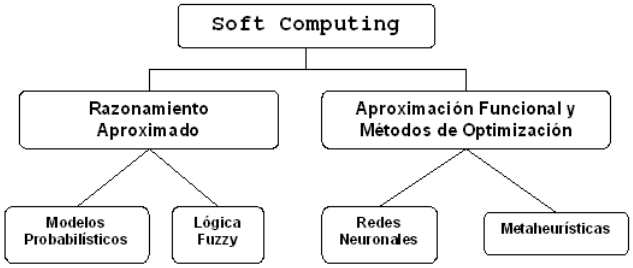


Figura 3.2: Componentes de la Soft Computing

En este aspecto y "*...en contraste con la computación tradicional (hard), la Soft Computing se beneficia de la tolerancia asociada a la imprecisión, la incertidumbre, y las verdades parciales para conseguir tratabilidad, robustez, soluciones de bajo costo y mejores representaciones de la realidad*" [93] es que las metaheurísticas tienen un fácil acomodo con las demás componentes de la Soft Computing, y sus sinergias pueden facilitar la aparición de nuevas metodologías, esquemas y marcos teóricos y prácticos que ayuden a entender y tratar mejor la generalizada imprecisión del mundo real.

Algunos ejemplos de cómo la Soft-Computing ha sido incluida dentro del campo de las metaheurísticas son los siguientes:

- *Criterios de parada difusos.* Los criterios de parada forman una parte importante de cualquier metaheurística. La inclusión de la Soft Computing en este elemento es algo relativamente reciente y viene de la mano de Vergara y Verdegay [94, 95]. Concretamente se ha aplicado con éxito a los algortimos del Simplex, de Karmakar y de puntos interiores, así como a algunos algortimos para resolver los Problemas de la Mochila y del Viajante de Comercio [96, 97].

- *Memes difusos en algoritmos multimeméticos.* Krasnogor y Pelta [98] han propuesto y probado una estrategia híbrida entre un algoritmo de búsqueda local basado en Soft Computing (FANS) [99, 100] que se explicará en detalles más adelante, y un Algoritmo Memético

(MMA). Ha sido utilizado en problemas de predicción de estructura de proteínas cuya dificultad es conocida. El uso de técnicas de Soft Computing facilita la comprensión de términos como "aceptabilidad" de las soluciones, "calidad" de las soluciones, soluciones "cercanas" a otras, etc.

- *Algoritmo de División Adaptable Difuso para Optimización Global.* Bazak [101] propone un algoritmo de división adaptable difuso (FAPA) para localizar el óptimo global de funciones multi-modales sin uso de conjeturas ni aproximaciones. El criterio de estimación difusa usado en la evaluación de las regiones y en la transformación de los datos brinda medios más flexibles y libres de suposiciones.

Para una revisión exhaustiva de estos temas se recomienda ver [102].

3.2. FANS, método de búsqueda por entornos, adaptable y difuso

Introducimos este apartado con el fin de presentar un elemento importante de nuestra propuesta que está directamente relacionado con las técnicas y conceptos de la Soft Computing. Aunque desde un punto de vista jerárquico correspondería su presentación más adelante se incluye aquí al ser una técnica específica de búsqueda que hace un uso continuado de las funciones de pertenencia de los conjuntos difusos. En concreto FANS evalúa las soluciones del espacio de búsqueda de manera cualitativa, de forma que el decisor pueda escoger o rechazar una solución *buena, aceptable, mala,* etc.

FANS (Fuzzy Adaptive Neighborhood Search) [103], se define como un método de búsqueda por entornos, adaptable y difuso. Es de búsqueda por entornos porque el algoritmo realiza transiciones desde una solución de referencia a otra de su entorno, produciendo "trayectorias" o caminos. Es adaptativo porque su comportamiento varía en función del estado de la búsqueda. Y es difuso porque las soluciones son evaluadas, además de la función objetivo, mediante una valoración difusa que representa algún concepto subjetivo o abstracto, y gracias a esto el método es capaz de capturar el comportamiento cualitativo de otras heurísticas basadas en búsqueda por entornos, con lo cual, será posible obtener un "framework" de heurísticas. Dado que este método se usa en nuestra investigación se hace una descripción detallada del mismo y para ello se utilizan las siguientes convenciones:

$s_i \in \mathcal{S}$ es una solución del espacio de búsqueda, $\mathcal{O}_i \in \mathcal{M}$ es un operador de Modificación o de Movimiento del espacio de operadores; $\mathcal{P}$ representa el espacio de parámetros (tuplas de valores) y $\mathcal{F}$ representa el espacio de los conjuntos difusos cuyos elementos se denotan como $\mu_i()$.

FANS está basado en cuatro componentes principales [100, 99]:

- un Operador de Modificación, para construir soluciones.
- una Valoración Difusa, para cualificarlas.
- un Administrador de Operación, para adaptar el comportamiento o características del operador.
- un Administrador de Vecindario, para generar y seleccionar una nueva solución.

Con estas definiciones, *FANS* queda especificado con la 7-tupla siguiente:

$$Fans(\mathcal{NS}, \mathcal{O}, \mathcal{OS}, \mu(), Pars, (cond, accion))$$

donde $\mathcal{NS}$ es el *Administrador de Vecindario*, $\mathcal{O}$ es el operador utilizado para construir soluciones, $\mathcal{OS}$ es el *Administrador de Operación* y $\mu()$ es una valoración difusa o subjetiva.

Además, $Pars$ representa un conjunto de parámetros y el par $(cond, accion)$ es una regla de tipo $IF\ cond\ THEN\ accion$ que se utiliza para detectar y actuar en consecuencia cuando la búsqueda se ha estancado.

A continuación se describen con más detalles estos elementos:

1) El Operador de Modificación Dada una solución de referencia $s \in \mathcal{S}$, el operador de modificación $\mathcal{O}_i \in \mathcal{M}$, con $\mathcal{O} : \mathcal{S} \times \mathcal{P} \rightarrow \mathcal{S}$, construye nuevas soluciones s_i a partir de s.

Cada aplicación del operador sobre la misma solución s debe devolver una solución diferente. Es decir, debe existir algún elemento de aleatoriedad en su definición. Un requerimiento adicional es que el operador provea algún elemento o parámetro adaptable $t \in \mathcal{P}$ para controlar su comportamiento. Por lo tanto, utilizaremos $\mathcal{O}^t$ para referirnos al operador con ciertos parámetros t.

2) La Valoración Difusa En el contexto de *FANS* las soluciones son evaluadas (además de la función objetivo) en términos de la *valoración difusa*. La motivación de este aspecto parte de la idea que en ocasiones se suele hablar de soluciones *Diferentes*, *Similares*, *Razonables*, etc, siendo estas características de naturaleza subjetiva o vaga. Una forma de modelizar adecuadamente este tipo de conceptos vagos se logra representando la valoración difusa mediante un conjunto difuso $\mu() \in \mathcal{F}$, con $\mu : \mathcal{R} \rightarrow [0, 1]$. Como sinónimo de valoración difusa, son aceptables términos como *concepto difuso* o *propiedad difusa*, lo que permite hablar de soluciones que verifican dicha propiedad en cierto grado.

Por lo tanto, la valoración difusa nos permite obtener el grado de pertenencia de la solución al conjunto difuso representado por $\mu()$. Por ejemplo, teniendo el conjunto difuso de las soluciones "buenas", consideraremos el grado de bondad de la solución de interés. Así, dadas dos soluciones $a, b \in \mathcal{S}$ podemos pensar en cuan *Similar* es a con b, o cuan *Cerca* están, o también cuan *Diferente* es b de a. *Similar, Cerca, Diferente* serán conjuntos difusos representados por funciones de pertenencia $\mu()$ apropiadas y naturales en el área de la optimización combinatoria.

3) El Administrador de Operación *FANS* utiliza varios operadores de modificación en el proceso de búsqueda y este Administrador de Operación $\mathcal{OS}$ es el responsable de definir el esquema de adaptación u orden de aplicación de los diferentes operadores utilizados.

El cambio de operador, o lo que es lo mismo, la ejecución del administrador, puede realizarse en cada iteración, o cuando el estado de la búsqueda así lo requiera. Esta adaptación puede estar basada en estadísticas del algoritmo tales como cantidad de evaluaciones de la función de costo realizadas, o teniendo en cuenta medidas particulares del proceso de búsqueda, como por ejemplo el número de iteraciones sin cambios en la mejor solución hallada, etc.

Básicamente se sugieren dos posibilidades para el Administrador. En la primera, el adminis-

trador $\mathcal{OS}$ ajustará los parámetros del operador y en consecuencia, devolverá un operador cuyo comportamiento será diferente: $\mathcal{OS}$ $(\mathcal{O}^{ti}) \Rightarrow \mathcal{O}^{tj}$.

La segunda alternativa, consiste en disponer de una familia de operadores de modificación $\{\mathcal{O}_1, \mathcal{O}_2, \ldots, \mathcal{O}_k\}$. Bajo esta situación, $\mathcal{OS}$ podría definir el orden de aplicación de los mismos. Dado que cada operador implica un vecindario diferente, obtendremos una estructura que recuerda a la de VNS (*Variable neighborhood search*) [27]. Sin embargo, este esquema es diferente al de VNS ya que *FANS* no utiliza un método explícito de búsqueda local ni requiere una métrica de distancia entre soluciones como la necesaria en VNS.

Claramente el operador y el administrador de operación estarán fuertemente acoplados y es factible pensar en una situación donde operadores específicos requieran administradores o gestores específicos.

4) El Administrador de Vecindario Este componente es el responsable de generar y seleccionar una nueva solución del vecindario. Podemos verlo como una función cuyo tipo es:

$$\mathcal{NS} : \mathcal{S} \times \mathcal{F} \times \mathcal{M} \times \mathcal{P} \Rightarrow \mathcal{S}$$

En el contexto de *FANS* se utilizan dos tipos de vecindarios: el *operativo* y el *semántico*, ambos definidos respecto a cierta solución de referencia s.

Dados el operador $\mathcal{O}$ y la solución actual s, se define el vecindario operativo como:

$$\mathcal{N}(s) = \{\hat{s}_i \mid \hat{s}_i = \mathcal{O}_i(s)\} \tag{3.1}$$

donde $\mathcal{O}_i(s)$ indica la i-ésima aplicación de $\mathcal{O}$ sobre s.

Para la definición del *vecindario semántico* de s se utiliza la valoración difusa $\mu()$, dando lugar a la siguiente definición:

$$\hat{\mathcal{N}}(s) = \{\hat{s}_i \in \mathcal{N}(s) \mid \mu(\hat{s}_i) \geq \lambda\} \tag{3.2}$$

Es decir, $\hat{\mathcal{N}}(s)$ representa el λ-corte del conjunto difuso de soluciones representado por $\mu()$. En otras palabras, las soluciones de interés serán aquellas que satisfagan nuestra valoración con, al menos, cierto grado λ.

La operación del administrador es simple: primero se ejecuta un *generador* para obtener soluciones del vecindario semántico a partir de varias aplicaciones del operador de modificación $\mathcal{O}$. Posteriormente, el procedimiento *selector* debe decidir cuál de estas soluciones retornar teniendo en cuenta: los grados de pertenencia de dichas soluciones, su costo o una combinación de ambos valores.

Por ejemplo, si se utiliza una valoración difusa de "Similaridad" con respecto a la solución actual, el selector podría utilizar reglas de selección como las siguientes:

- *Mejor:* Devolver la solución más similar disponible,
- *Peor:* Devolver la menos similar,
- *Primera:* Devolver la primer solución suficientemente similar.

Naturalmente, también podría utilizarse el costo de esas soluciones similares para obtener reglas de selección como:

- *MaxMax:* De las soluciones similares, retornar la de mayor costo,
- *MaxMin:* De las soluciones similares, retornar la de menor costo.

5)Parámetros Globales *FANS* mantiene un conjunto de parámetros globales $Pars \in \mathcal{P}$ para llevar el registro del estado de la búsqueda y este conjunto es utilizado por varios componentes para decidir las acciones a tomar.

Es sabido que todos los métodos de búsqueda local presentan como principal inconveniente, el quedar atrapados en óptimos locales. *FANS* maneja esta situación a través de dos mecanismos que se utilizan para escapar de dichos óptimos.

El primer mecanismo está contenido en la variación del operador en las etapas de búsqueda. El otro mecanismo de escape esta basado en el par $(cond, accion)$ donde *cond*, llamada $HayEstancamiento?() : \mathcal{P} \rightarrow [True, False]$, es utilizada para determinar cuando hay suficiente evidencia de que la búsqueda esta definitivamente estancada. Cuando *cond* se verifique, entonces se ejecutará la acción $accion = Escape()$. Por ejemplo, se podría reiniciar el algoritmo desde una nueva solución inicial, reiniciar desde una solución obtenida a partir de una modificación especial de la actual, o cualquier otra opción que se considere adecuada.

```
Procedimiento FANS:
Begin
  /* O : el operador de modificacion */
  /* μ() : la valoracion difusa */
  /* S_cur, S_new : soluciones */
  /* NS: el administrador de vecindario */
  /* OS: el administrador de operacion */
  Inicializar Variables();
  While ( not-fin ) Do
    /* ejecutar el administrador de vecindario NS */
    S_new = NS(O, μ(), S_cur);
    If (S_new es ''buena'' en terminos de μ()) Then
      S_cur := S_new;
      adaptar-Valoracion-Difusa(μ(), S_cur);
      Else
        /* No fue posible encontrar una solucion buena con  */
        /* el operador actual */
        /* Sera modificado */
        O := OS-ModificarOperador(O);
      Fi
      If (HayEstancamiento?()) Then
        Escape();
      Fi
   Od
  End.
```

Figura 3.3: Esquema de *FANS*

3.2.1. Algoritmo de FANS

En esta sección se describe el esquema de *FANS*, el cual se muestra en la Fig. 3.3. Se puede observar que cada iteración comienza con una llamada al administrador de vecindario $\mathcal{NS}$ con los siguientes parámetros: la solución actual S_{cur}, la valoración difusa $\mu()$ y el operador de modificación $\mathcal{O}$. Como resultado de la ejecución de $\mathcal{NS}$ pueden ocurrir 2 cosas: se pudo encontrar una solución vecina aceptable S_{new} (en términos de $\mu()$), o no se pudo.

En el primer caso, S_{new} pasa a ser la solución actual y los parámetros de $\mu()$ son adaptados. Por ejemplo, si $\mu()$ representa la similaridad respecto a la solución actual, entonces la valoración difusa debe ser adaptada para reflejar este cambio en la solución de referencia. Si $\mathcal{NS}$ no pudo retornar una solución aceptable, es decir, en el vecindario inducido por el operador no se pudo encontrar una solución suficientemente buena, entonces se aplica el nuevo mecanismo de escape: se ejecuta el administrador de operación OS el cual retornara una versión modificada del operador $\mathcal{O}$. La próxima vez que $\mathcal{NS}$ se ejecute, dispondrá de un operador modificado para buscar soluciones, y por lo tanto es posible que el resultado sea diferente.

La condición $HayEstancamiento?()$ será verdadera cuando (por ejemplo) se hayan realizado $Tope$ llamadas a OS sin haber obtenido mejoras en la solución actual. Es decir, se probaron varios operadores y no se obtuvieron mejoras. En este caso, se ejecutará el procedimiento $Escape()$, se evaluará el costo de la nueva solución y se adaptará la valoración difusa $\mu()$. Posteriormente, se reinicia la búsqueda.

Debe quedar claro que lo que varía en cada iteración son los parámetros utilizados en las llamadas a $\mathcal{NS}$. El algoritmo comienza con $\mathcal{NS}\ (s_0, \mathcal{O}^{t_0}, \mu_0)$. Si $\mathcal{NS}$ puede retornar una solución aceptable, entonces en la siguiente iteración la llamada será $\mathcal{NS}\ (s_1, \mathcal{O}^{t_0}, \mu_1)$; es decir cambia la solución actual y por lo tanto se modifica la valoración difusa. Si en cierta iteración l, $\mathcal{NS}$ no puede retornar una solución aceptable, entonces se ejecutará el administrador de operación el cual devolverá una versión modificada del operador. La iteración siguiente, $\mathcal{NS}$ será llamado con $\mathcal{NS}\ (s_l, \mathcal{O}^{t_1}, \mu_l)$.

Durante la ejecución del algoritmo pueden ocurrir dos situaciones problemáticas: primero, se realizaron varias llamadas $\mathcal{NS}(s_j, \mathcal{O}^{t_i}, \mu_j)$ con $i \in [1, 2, \ldots, k]$, lo cual significa que se probaron k formas diferentes de obtener una solución aceptable y ninguna tuvo éxito; o segundo , la búsqueda se está moviendo entre soluciones aceptables pero en las ultimas m llamadas no se consiguió mejorar la mejor solución de todas las visitadas hasta el momento.

Cuando cualquiera de las dos situaciones ocurra, se ejecutará el procedimiento $Escape()$ lo cual llevará a alguna de las dos posibilidades siguientes: $\mathcal{NS}\ (\hat{s}_0, \mathcal{O}^{t_j}, \hat{\mu}_0)$ o $\mathcal{NS}\ (\hat{s}_0, \mathcal{O}^{t_0}, \hat{\mu}_0)$, donde $\hat{s}_0$ es una nueva solucion inicial (generada aleatoriamente, por ejemplo), y $\hat{\mu}_0$ es la valoración difusa obtenida en función de $\hat{s}_0$. Respecto al operador, se puede mantener con los mismos parámetros $(\mathcal{O}^{t_j})$ o también se lo puede "reinicializar" $(\mathcal{O}^{t_0})$.

El criterio de parada del algoritmo se basa en la verificación de una condición externa. Por ejemplo, el algoritmo finaliza cuando el número de evaluaciones de la función de costo alcanza cierto límite o cuando se realizaron un número predeterminado de evaluaciones.

3.2.2. FANS como heurística de propósito general

El comportamiento global de *FANS* depende de la definición de sus componentes y de la interacción entre ellos. A través de la utilización de diferentes definiciones y parámetros para la valoración difusa se consigue que el algoritmo se comporte de diferente manera, lo que da lugar, en consecuencia, a variaciones en los resultados que se puedan obtener.

Por ejemplo puede tener un Comportamiento tipo Caminos Aleatorios. Para obtenerlo, todo lo que se necesita es considerar cualquier solución del vecindario como aceptable. Por lo tanto, utilizando una valoración difusa con $\mu(f(s), f(\hat{s})) = 1,\ \forall\, \hat{s} \in \mathcal{N}(s)$, cualquier solución del vecindario operacional tendrá la opción de ser seleccionada. De esta manera, *FANS* se comportará como un

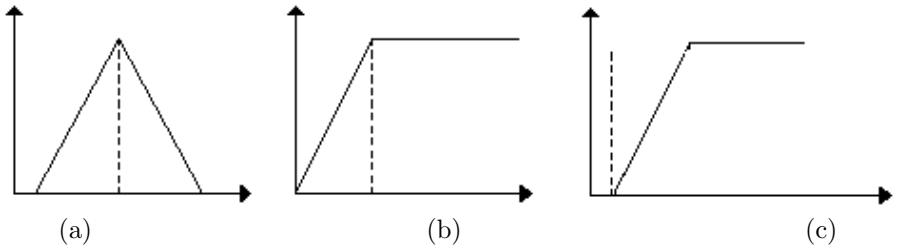


Figura 3.4: Ejemplos de Valoraciones Difusas.

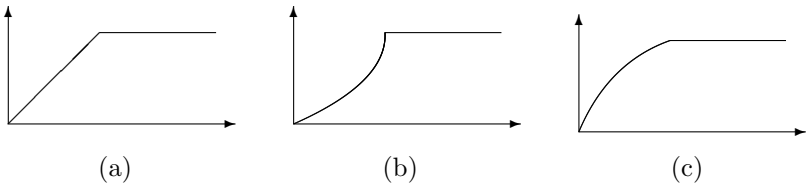


Figura 3.5: Aplicación de los modificadores lingüisticos *Algo*, en (b), y *Muy*, en (c), sobre la definición de Aceptabilidad (a).

método que produce caminos aleatorios.

Para lograr un Comportamiento tipo Hill Climbing es necesario utilizar una valoración difusa tal que $\mu(s, \hat{s}) = 1$ if $f(\hat{s}) < f(s)$ y asignar $\lambda = 1$. De esta manera, unicamente serán consideradas como aceptables aquellas soluciones que mejoren el costo actual. El método de Hill Climbing termina cuando se alcanza un óptimo local. En el contexto de *FANS*, la obtención de un método multiarranque es directo.

FANS también puede plantearse como una heurística en sí misma capaz de mostrar una variedad de comportamientos muy amplia. Este potencial queda claro si consideramos que la valoración difusa representa el criterio de un decisor buscando soluciones para un problema.

Por ejemplo, en la Figura 3.4 se representan varios criterios posibles para aceptabilidad. Las definiciones están basadas en el costo de cierta solución actual, representado en los gráficos por una línea de puntos.

La definición triangular (a), representa un decisor buscando soluciones similares en costo pero diferentes en estructura. Este tipo de comportamiento puede aparecer cuando las soluciones presentan características dificiles de cuantificar (por ejemplo, aspectos estéticos) con lo cual el decisor desea obtener un conjunto de opciones de costo similar para luego hacer su elección en términos de otros criterios diferentes al costo. La definicion central (b) representa un decisor "insatisfecho" con la solución actual. Cualquier opción que la mejore, tendrá un valor máximo de aceptabilidad. Finalmente la tercera opción (c), representa un decisor "conservador" en el sentido que para cambiar la solución actual, la mejora obtenida debe ser considerable.

Otra posibilidad de conseguir comportamientos diferentes, se obtiene mediante la aplicación de modificadores lingüisticos sobre cierta definición dada de una valoración difusa. Un ejemplo aparece en la Figura 3.5 donde se muestra en primer lugar la definición de Aceptabilidad y a continuación, las definiciones respectivas para *Algo Aceptable* y *Muy Aceptable*.

En este trabajo, la valoración difusa se basa en el costo de la solución actual o de referencia y representa la noción de aceptabilidad, o sea, se tendrá un conjunto difuso de soluciones aceptables. De esta manera, aquellas soluciones vecinas que mejoren el costo de referencia son consideradas

como aceptables con grado 1. Aquellas soluciones con un costo peor que cierto umbral tendrán grado 0 de aceptabilidad. Por tanto, las soluciones aceptables tendrán grados de pertenencia que varían entre [0,1].

La figura 3.6 representa un esquema de Aceptabilidad, donde α es el costo de referencia y β es el límite de aceptabilidad.

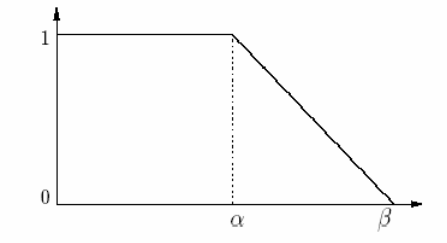


Figura 3.6: Noción de Aceptabilidad para controlar transiciones a soluciones vecinas.

FANS se mueve entre soluciones que satisfacen la idea de aceptabilidad con al menos cierto grado λ, y variaciones en este parámetro posibilitan al algoritmo lograr el comportamiento de diferentes esquemas clásicos de búsqueda local. Así, por ejemplo, para un ajuste de $\lambda = 1$, FANS se comportará como un esquema ávido o "hill climber", donde las soluciones que mejoren el costo serán consideradas como aceptables con un alto grado. Por otro lado, para un ajuste de $\lambda = 0$, FANS producirá caminos aleatorios. Diferentes valores de ajuste para FANS permitirán entonces tener agentes con diferentes comportamientos, con los cuales se pueden construir diferentes esquemas de búsqueda y manejar en la estrategia el equilibrio entre la intensificación y la diversificación fácilmente.

En síntesis, *FANS* brinda la posibilidad de variar su comportamiento y en última instancia las soluciones que se pueden obtener, de forma simple y comprensible. Si consideramos que cada comportamiento permite obtener resultados cuantitativamente diferentes, y que para cada problema resulta mejor un algoritmo que otro, la potencialidad de *FANS* como herramienta genérica de optimización resulta clara. Por ello, se puede considerar a *FANS* como un framework de métodos de búsqueda local.

FANS se ha aplicado con éxito a diferentes problemas de optimización, entre los cuales se encuentran el problema de la mochila simple y la mochila con múltiples restricciones, la minimización de funciones reales de complicada definición estándar para optimización y el problema de predicción de estructura de proteínas [104, 105, 106, 107].

3.3. Estrategia cooperativa paralela basada en Soft-Computing

La idea principal de nuestra propuesta puede resumirse imaginando un comité de expertos encargado de resolver algún problema concreto. El comité estará dirigido por un Coordinador, que conoce los aspectos generales relativos al problema en cuestión y las capacidades y características de cada uno de los miembros del equipo de expertos. Además, el Coordinador sabe cuándo puede considerarse aceptable una solución obtenida, o sea, sabe cuándo parar.

Cada miembro del comité es un experto en resolver el problema mediante un tratamiento

específico, que puede ser compartido por más de uno. En este contexto cada uno de los expertos del comité trabaja de forma independiente; pero todos al mismo tiempo.

El Coordinador tendrá dos tareas principales:

- recolectar información del rendimiento de los expertos.
- darles directivas que ajusten su manera de solucionar el problema.

La idea del comité se diseña como una estrategia cooperativa multiagentes, organizada jerárquicamente en dos niveles, bajo un esquema de coordinación de planificación centralizada (Ver Figura 3.7) en la cual en la jerarquía superior está el Coordinador y a él se subordinan todos los agentes que resuelven el problema.

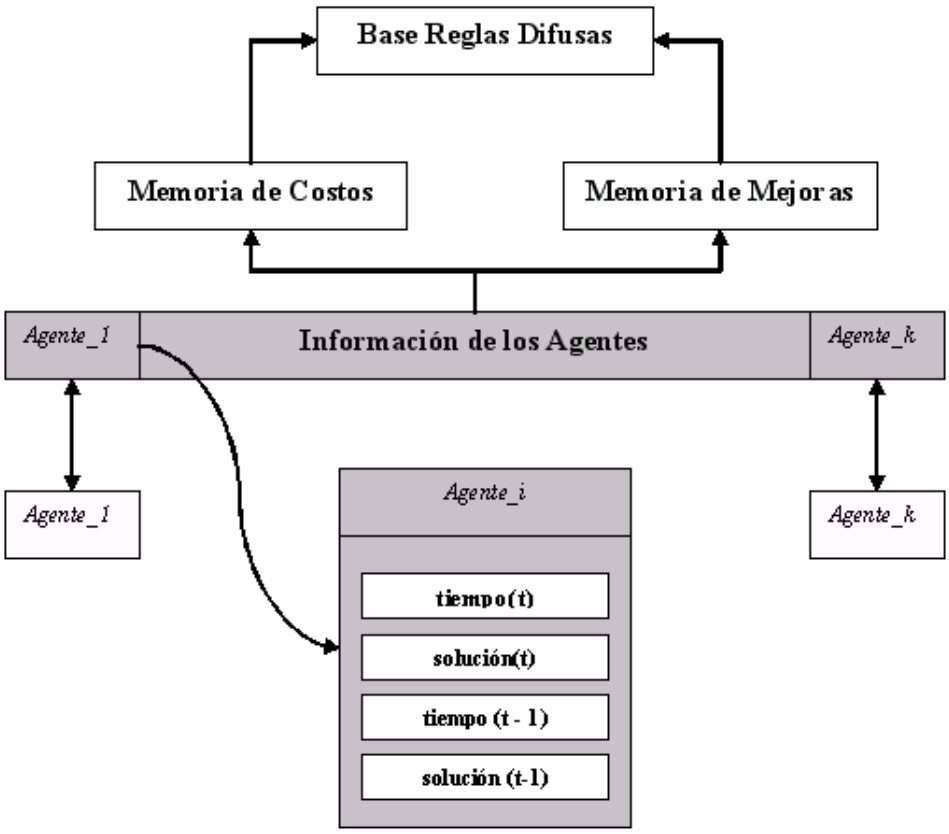


Figura 3.7: Esquema de la Estrategia

El Coordinador se modela a través de un sistema de reglas difusas, donde se propone representar el conocimiento a través de tres grupos de reglas:

- *Reglas Generales:* que representan el conocimiento general que el Coordinador tiene acerca del problema (tipo, tamaño, características, etc.), y posiblemente cuales son los mejores expertos para resolverlo.
- *Reglas acerca de los Agentes:* que representan el conocimiento específico del Coordinador sobre las características de los agentes resolvedores, tales como el conocimiento sobre su comportamiento a lo largo del tiempo, preferencias, ajuste, etc.

- *Reglas de Parada:* que representan el criterio de parada que usa el Coordinador. Por ejemplo, pueden basarse en el costo de la solución actual, en el lapso de tiempo de computación desde el inicio de la ejecución, etc.

La obtención de tal sistema de reglas no es trivial. En una primera aproximación, el conjunto de reglas difusas es generado manualmente, intentando capturar los siguientes principios esenciales:

- Si algo funciona bien, mantenerlo. Pero, si algo no marcha bien, provocar algún cambio.
- Aprovechar el conocimiento existente.
- Es mejor satisfacer que optimizar.

El primer principio gobierna una gran cantidad de algoritmos y su significado es claro, debe mantenerse el modo de funcionamiento del agente resolvedor si éste funciona bien, sino enviarle directivas que cambien su estado. El segundo sigue la idea de compartir buenas soluciones con el fin de concentrar la búsqueda alrededor de ellas. El último ítem se relaciona con una situación donde se dispone de una solución razonablemente buena, y es completamente innecesario comenzar una búsqueda por la mejor solución.

En el nivel inferior de la estrategia, se encuentran los agentes resolvedores, que representan a los expertos y que ejecutan algoritmos de optimización. O sea, son entidades que poseen el conocimiento necesario para resolver un problema de optimización específico y a los cuales se les envía desde el agente Coordinador los parámetros de ajuste de su comportamiento.

En el diseño presentado cada agente resolvedor se modela a través de un algoritmo de búsqueda por entornos, adaptable y difuso, FANS, ya explicado anteriormente. La elección de FANS para estos agentes se debe a su facilidad para obtener comportamientos de diferentes heurísticas al variar sus parámetros de ajuste, además debido a que es una técnica de búsqueda local, fácil de comprender, implementar, y que no necesita muchos recursos computacionales.

3.3.1. Implementación de la estrategia

A continuación, con la ayuda de los pseudocódigos, se describe con más detalles esta propuesta. En la Figura 3.8 se muestra el pseudocódigo para el Coordinador.

En un primer paso, el Coordinador determina el comportamiento inicial (conjunto de parámetros) para cada agente resolvedor. El Coordinador pasa dichos comportamientos a cada agente y entonces estos se ejecutan.

La comunicación entre el Coordinador y los agentes resolvedores se realiza a través de mensajes de intercambio donde los agentes envían la información sobre su desempeño y reciben órdenes de adaptación; el Coordinador recibe y almacena comportamientos y recupera medidas de desempeño que usa para adaptar su base de reglas interna. Como ya se dijo antes, los agentes se encuentran en ejecución resolviendo el problema, enviando los resultados y esperando las directivas del Coordinador.

El Coordinador almacena los costos $f(s^t)$ producidos por cada algoritmo y un historial de las tasas de mejoras definido por la siguiente ecuación:

$$\Delta_f = \frac{(f(s^t) - f(s^{t-1}))}{(time^t - time^{t-1})} \tag{3.3}$$

```
Procedimiento Coordinador:
Begin
  /* Agente_i, agente resolvedor */
  /* Par_i, conjunto de parámetros */
  /* Coordinador */
  /* MC_i, memoria para alamcenar la información de cada Agente_i */
  /* FRB, La Base de reglas difusas del Coordinador */
  For i := 1 To k Do
    obtener Par_i para Agente_i usando la FRB;
    ajustar Agente_i con Par_i;
    Agente_i comienza a ejecutar;
  Od
  Repeat Until ( no se alcance el criterio de parada ) Do
    MC_i es actualizada por Agente_i (asincrónicamente);
    For (cada Agente_i que haya informado nuevos resultados) Do
      Coordinador decide si Agente_i necesita ser cambiado;
      If (SI) Then
        obtener Par_i para Agente_i usando la FRB;
        Coordinador almacena Par_i en MC_i;
      Fi
    Od
  Od
  For i := 1 To k Do
    detener Agente_i;
  Od
End.
```

Figura 3.8: Pseudo código del Coordinador

```
Procedimiento Agente_i:
Begin
  Repeat Until ( no haya orden de parar ) Do
    verificar si hay información del Coordinador en MC_i;
    If (hay nueva información) Then
      adaptar los parámetros internos;
    Fi
    ;
    /* aquí se define la estrategia de Agente_i */
    ;
    If (se verifican las condiciones para comunicación) Then
      enviar información de su desempeño al Coordinador;
    Fi
   End.
```

Figura 3.9: Pseudo código del Agente Resolvedor.

de forma tal que, ante un nuevo informe, calcula la "calidad" de la solución reportada, en términos del historial almacenado y donde $time^t - time^{t-1}$ representa el tiempo transcurrido entre dos informes consecutivos.

Entonces, los valores Δ_f y $f(s^t)$ y son guardados en arreglos ordenados separados a longitud fija o "memorias" M_{Δ_f}, M_f.

El Coordinador comprueba qué algoritmo proporciona nueva información y decide si su comportamiento necesita ser adaptado. En este caso, usa su base de reglas para obtener un nuevo comportamiento que será enviado al agente. Cuando se cumple el criterio de parada, el Coordinador detiene la ejecución de los algoritmos.

La operación de los algoritmos es sencilla (Ver Figura 3.9). Cada uno se ejecuta de forma asíncrona, alternando el envío del comportamiento de su desempeño (solución actual, su coste y tiempo) y la recepción de la información de adaptación del Coordinador.

Esto se implementa de dos formas. Primero, se decidió simular el paralelismo usando un diseño de tiempo compartido en el que se hace una ejecución aleatoria de lecturas y escrituras en una memoria compartida (que es implementada como una estructura de datos). Luego, se desarrolla una implementación paralela real, usando el paquete de software PVM.

Como FANS es una heurística basada en búsqueda local, se decidió que la información enviada por el Coordinador a cada algoritmo sea una nueva solución, o sea, un nuevo punto en el espacio de búsqueda o cambiarle su comportamiento mediante un ajuste de algún parámetro. Cuando el algoritmo lo recibe, vuelve a empezar la búsqueda a partir de ese nuevo punto o en el otro caso bajo un esquema de búsqueda modificado.

3.3.2. Base de Reglas Difusas

Para la implementación de la Base de Reglas Difusas, se proponen las siguiente reglas:

1. **Regla SM, Sin Memoria:**

   IF `la solución del` $agente_i$ `es` **peor**
   `que la mejor vista hasta el momento`
   THEN `enviarle al` $agente_i$ `la mejor solución.`

   Bajo esta regla si el $agente_i$ reporta una solución *peor* respecto a la mejor de todas las que ya se han visto entonces se le indica que se "mueva" en el espacio de búsqueda asignándole como solución de reinicio esa mejor solución.

   Entiéndase como *peor* que el costo de la solución obtenida por el $agente_i$ sea mayor o menor a la mejor de todas las ya vistas, según se maximice o minimice en el problema. Por tanto, en este caso no se tiene en cuenta la memoria , al no tener en cuenta el historial del comportamiento de los agentes.

2. **Regla MR1, con Memoria e independiente de los agentes:**

   IF `la calidad de la solución del` $agente_i$ `es` **mala**
   AND `razón de mejora del` $agente_i$ `es` **mala**
   THEN `enviarle al` $agente_i$ `la mejor solución.`

   donde la etiqueta difusa *mala* se define por la siguiente función de pertenencia:

$$\mu(x,\alpha,\beta) = \begin{cases} 0 & \text{si } x > \beta \\ (\beta - x)/(\beta - \alpha) & \text{if } \alpha \leq x \leq \beta \\ 1 & \text{si } x < \alpha \end{cases}$$

   Donde x es el rango percentil de la variable que se trate, ya sea el costo de la solución, $f(s^t)$, o la tasa de mejora, Δ_f, de entre los valores almacenados en la memoria. Los otros parámetros fueron fijados empíricamente a los siguientes valores: $\alpha = 10$ y $\beta = 20$.

   En otros términos, si el $agente_i$ reporta soluciones malas respecto a las que ya se han visto durante la optimización, y si además, su tasa de mejora entre dos reportes consecutivos es de las peores, eso indica que el $agente_i$ está estancado. Por lo tanto, lo que se hace será "moverlo" en el espacio de búsqueda asignándole una nueva solución que puede ser

generada aleatoriamente o puede ser la mejor vista por el Coordinador hasta ese momento. Esta última es la usada en este trabajo y puede ser vista como un esquema *greedy* de coordinación.

De esta forma y usando los criterios estándares en el cálculo de las reglas difusas, si denominamos por μ_{costo} y μ_{tasa} a los valores obtenidos en la función de pertenencia descrita anteriormente para $x = f(s^t)$ y $x = \Delta_f$, respectivamente, el antecedente de las reglas se calcula como:

$$\min\ (\mu_{costo},\ \mu_{tasa}\ )$$

y el consecuente se produce si este valor anterior es mayor que un cierto umbral fijado empíricamente en 0.7. Como se ve esta regla es independiente del tipo de agentes resolvedores que use la estrategia.

3. **Regla MR2, con Memoria y dependiente de los agentes:**

IF `la calidad de la solución informada por` $agente_i$ `es` **mala**
AND `razón de mejora del` $agente_i$ `es` **mala**
THEN `enviarle al` $agente_i$ `la mejor solución`
AND `ajustar el valor de` λ `del` $agente_i$`.`

En esta, igual que en la Regla **MR1**, se lleva un historial de las tasas de mejoras y los costos producidos por los algoritmos, de forma tal que, ante un nuevo informe, se calcula la "calidad" de la solución reportada. Se usa la misma función difusa de pertenencia para definir la etiqueta *mala*.

La diferencia en esta Regla está en además de lo que hace la regla MR1 se le añade *ajustar* el valor de λ del $Agente_i$.

O sea, se le enviará un nuevo valor de λ, denominado $\lambda_i^{'}$, que acerca el valor de λ_i al del agente que haya obtenido la mejor solución vista hasta el momento, λ^*. Para ello, se define una etiqueta difusa Δ_λ de esta forma:

$$\Delta_\lambda = \mu_\delta^{-1}(min(\mu_{\cos to}, \mu_{tasa}))$$

donde, si $\left|\lambda^* - \lambda^i\right| \neq 0$:

$$\mu_\delta(x) = \begin{cases} \frac{x}{|\lambda^* - \lambda_i|} & \text{si} \quad 0 \leq x \leq 1 \\ 1 & \text{si} \quad x \geq 0 \end{cases}$$

si $\left|\lambda^* - \lambda^i\right| = 0$: entonces $\mu_\delta(x) = 1, \forall x$, en este caso la expresión de Δ_λ carece de sentido, pero se hace innecesario su cálculo ya que $\lambda^* = \lambda^i$ y por lo tanto $\Delta_\lambda = 0$.

y de esta forma $\lambda_i^{'}$ queda definido como:

$$\lambda_i^{'} = \begin{cases} min(1, \lambda_i + \Delta_\lambda) & \text{si} \quad \lambda_i \leq \lambda^* \\ max(1, \lambda_i - \Delta_\lambda) & \text{si} \quad \lambda_i > \lambda^* \end{cases}$$

Como se ve esta regla depende de los parámetros que configuran el comportamiento de búsqueda de los agentes. Está implementada en este trabajo específicamente para el algoritmo FANS y su parámetro de umbral, la valoración difusa; pero puede usarse con cualquier otro agente resolvedor que use algoritmos cuyo ajuste de alguno o varios de sus parámetros puedan ser manipulados sin esfuerzo.

4. **Regla RR1, retrasa el uso de la regla MR1:**

   IF `tiempo de ejecución del` $agente_i \leq \frac{1}{3}$ `tiempo total de ejecución`
   THEN `usar la Regla SM.`
   ELSE IF `tiempo de ejecución del` $agente_i > \frac{1}{3}$ `tiempo total de ejecución`
   THEN `usar la Regla MR1.`

   Esta regla se define para permitir a los agentes resolvedores que se estabilicen en su búsqueda evitando de este modo convergencias prematuras hacia las zonas de los mejores valores encontrados al inicio y por tanto permitiendo una mayor diversificación en la estrategia durante esta fase.

   Por tanto, con esta regla lo que se hace es demorar el uso de la Regla **MR1** un tiempo determinado, en este caso un tercio del tiempo total de ejecución asignado, y mientras se usará la Regla (**SM**).

5. **Regla RR2, retrasa el uso de la regla MR2:**

   IF `tiempo de ejecución del` $agente_i \leq \frac{1}{3}$ `tiempo total de ejecución`
   THEN `usar la Regla SM.`
   ELSE IF `tiempo de ejecución del` $agente_i > \frac{1}{3}$ `tiempo total de ejecución`
   THEN `usar la Regla MR2.`

   Esta regla también se construye para permitir a los agentes resolvedores que se estabilicen en su búsqueda evitando de este modo convergencias prematuras hacia las zonas de los mejores valores encontrados al inicio y por tanto permitiendo una mayor diversificación en la estrategia; pero lo que se hace es demorar el uso de la Regla **MR2** un tiempo determinado, un tercio del tiempo total de ejecución asignado, y mientras tanto se usará la Regla **SM**.

3.3.3. Verificación Experimental

Para comprobar la validez de la estrategia paralela cooperativa propuesta se evaluará su comportamiento en forma empírica a partir de experimentos computacionales sobre un conjunto de instancias de problemas de prueba.

Para nuestros experimentos se tendrán en cuenta los siguientes aspectos: cantidad razonable de instancias de prueba en cada problema, se restringirá el análisis a los valores alcanzados de la función de costo, y sobre todo un conocimiento nulo o escaso del dominio del problema en cuestión lo que implica la utilización y ajuste de operadores muy simples.

Como banco de pruebas se utilizarán los siguientes problemas:

1. El problema de la mochila “clásico”.

a) Se realiza primero un juego de experimentos preliminares para configurar y comprobar el funcionamiento de la estrategia.

b) Se hace un segundo juego de experimentos con los siguientes objetivos: comprobar los beneficios de la coordinación, hacer un análisis de la influencia del uso de agentes con diferentes comportamientos, y estudiar la influencia del número de agentes participantes en la estrategia. Los experimentos del ítem anterior y estos se realizan simulando el paralelismo a través de una implementación de tiempo compartido.

c) Se prueba la estrategia ante un conjunto de instancias duras de la mochila que constituyen todo un reto para la mayoría de los algoritmos exactos y métodos heurísticos de la actualidad. Estos experimentos se realizan en un cluster de computadores bajo una implementación paralela real.

2. El problema de la p-mediana.

a) Se realiza primero un juego de experimentos preliminares ante instancias pequeñas del problema para configurar, ajustar y comprobar el funcionamiento de la estrategia en este problema.

b) Se pone a prueba la estrategia ante instancias medias y grandes del problema.

3.4. Conclusiones

Como introducción al capítulo se mostraron conceptos básicos de la Soft-Computing y su utilidad al campo de las metaheurísticas. Basándonos en estos conceptos intentamos alcanzar los objetivos de este trabajo.

Concretamente en este capítulo se ha presentado una propuesta para resolver problemas de optimización combinatoria a través de una estrategia cooperativa basada en reglas difusas en la que se controla y modifica el comportamiento de un conjunto de algoritmos y los ejecuta en forma paralela y coordinada. Esto se construyó como un sistema multiagentes en una estructura jerárquica de dos niveles, con mecanismos de coordinación bajo planificación centralizada. En el nivel superior hay un Coordinador al que se subordinan los agentes que resuelven el problema y que están situados en el nivel inferior. El Coordinador cumple las tareas básicas de recolectar información del rendimiento de los agentes resolvedores y darles directivas para que ajusten su manera de solucionar el problema. Los agentes resolvedores poseen el conocimiento necesario para resolver un problema de optimización específico.

El Coordinador posee como componente clave una base de reglas difusas que definen los diferentes mecanismos de coordinación de la estrategia. Con estas reglas comprueba qué algoritmo proporciona nueva información y decide si su comportamiento necesita ser adaptado. Se propusieron varias reglas en las que se combina el uso o no de la memoria en la que se guarda el historial de las tasas de los agentes, hay reglas dependientes e independientes de los agentes, y otras en las que se combinan algunas reglas entre sí tanto para aprovechar lo positivo de cada una como para retrasar el uso de la memoria.

Para la implementación de los agentes resolvedores se usa la metaheurística FANS, un método de búsqueda por entornos, adaptable y difuso debido a su con ella es fácil obtener comportamientos de diferentes heurísticas al variar sus parámetros de ajuste, además debido a que es una técnica de búsqueda local, fácil de comprender, implementar, y que no necesita muchos recursos

computacionales. Por ello, se describen los elementos básicos del algoritmo y su esquema general, destacando la utilidad de su principal componente: la valoración difusa, y se muestra como esta puede ser utilizada para variar el comportamiento del algoritmo.

Capítulo 4

Problema de la Mochila

En este capítulo queremos verificar la estrategia presentada en nuestro trabajo. Para ello hemos analizado una batería de problemas clásicos de optimización combinatoria. De entre ellos hemos escogido algunos problemas bien conocidos, de los cuales se tengan instancias estándares y cuya dificultad esté bien graduada. El Problema de la Mochila (Knapsack Problem, KP) cumple con estas carácterísticas. Esto nos ha permitido analizar la influencia de dicha dificultad en el ajuste de los distintos factores de nuestra estrategia. Se explica brevemente el problema y se muestra su formulación matemática. Para comprobar el comportamiento de la estrategia se hacen varios juegos de experimentos que se agrupan en dos categorías, en dependencia de la implementación del paralelismo y los recursos empleados. Se hacen experimentos en un único computador bajo una implementación donde se simula el paralelismo a través de un esquema de tiempo compartido. Teniendo en cuenta que la estrategia necesita probarse ante instancias más duras se hace una versión implementada bajo el paquete de programas del PVM y se prueba en un cluster de 8 computadores. Finalmente, se presentan las conclusiones del capítulo.

4.1. Formulación del problema

El problema de la mochila es uno de los más estudiados tanto en la Investigación de Operaciones como en Ciencias de la Computación. Se dice que es NP en un sentido débil ya que puede ser resuelto en tiempo pseudo-polinomial [108]. Además, tantos años de mejoras algorítmicas han hecho posible resolver casi todas las instancias estándares de la literatura. Algunos ejemplos de estos métodos, así como las últimas técnicas aplicadas para resolver diferentes instancias de la Mochila aparecen en [108, 109].

Este problema puede ser formalmente definido así: dado un conjunto N, compuesto de n items i con un beneficio o valor p_i y peso w_i, y un valor C de capacidad de la mochila. El objetivo es seleccionar un subconjunto de N tal que el total de los beneficios de los items seleccionados sea máximo y el total de los pesos no exceda a C. Su formulación matemática es:

$$Max \quad \sum_{i=1}^{n} p_i \times x_i$$

$$s.t. \quad \sum_{i=1}^{n} w_i \times x_i \leq C, \quad x_i \in \{0,1\}, i = 1, \ldots, n$$

donde

n es el número de items,
x_i indica si el item i está incluido o no en la mochila,
p_i es el beneficio asociado al i,
$w_i \in [0, .., r]$ es el peso del item i,
C es la capacidad de la mochila.

Se asume que $w_i < C$, $\forall i$; y $\sum_{i=1}^{n} w_i > C$

4.1.1. Instancias del problema

En el Problema de la Mochila se pueden generar instancias diferentes teniendo en cuenta algunas de sus características, tales como: el tamaño de las instancias, el tipo de correlación entre los pesos y los beneficios y el rango de valores disponibles para los pesos.

Así, para este trabajo se ha generado primero para probar la estrategia propuesta un grupo de instancias que denominamos "clásicas" que se usarán en el primer grupo de experimentos, bajo una implementación con el paralelismo simulado en un único computador. Las instancias que fueron generadas para este problema en el seno del grupo de investigación del autor de la memoria se crean dado $w_i = U(1, max)$ y donde $U(1, max)$ es un número aleatorio con distribución uniforme en el intervalo $(1, max)$. Tenemos entonces los siguientes tipos de instancias definidos por medio de la correlación entre pesos y beneficios:

- No Correlacionadas, NC: $p_i = U(1, max)$
- Débilmente Correlacionadas, WC: $p_i = U(w_i - t, w_i + t)$
- Fuertemente Correlacionadas, SC: $p_i = w_i + k$

donde $t = 100$, $max = 1000$ y $k = 10$ son constantes arbitrarias.

La Capacidad de la Mochila se calcula así:

$$C = \alpha * \sum_{i=1}^{n} w_i \text{ ,con } \alpha \in [0, 1]$$

Se definen 5 tamaños del problema:

$$n = (150, 300, 450, 600, 750)$$

y para cada uno se generan aleatoriamente varias instancias de cada tipo. Cada ejecución terminará cuando se hayan hecho $k * n$ evaluaciones de la función objetivo. Donde $k = 100$ para $n = (150, 300, 450)$ y $k = 300$ para $n = (600, 750)$.

Debido a que no se conoce el valor óptimo de la solución para estas instancias, se usa como referencia la cota de Dantzig para medir la eficacia del algoritmo [110]. La cota de Dantzig se obtiene "relajando" el problema de forma que puedan obtenerse soluciones no binarias, es decir, los x_i no son valores binarios sino pertenecientes al intervalo unidad, y con esto en cada ejecución se calcula el *Error*:

$$Error = 100 * \frac{CotadeDantzig - Costo}{CotadeDantzig}$$

Se tiene también el número de evaluaciones de la función de costo usadas para alcanzar dicho error $E2B$. Este valor es transformado para mostrar el porcentaje de evaluaciones usadas sobre el total de evaluaciones disponibles, nombrado $\%\ Evals$

También se define la variable Desempeño, como medida de comparación de algoritmos independiente del problema en el sentido de que sólo depende del error y del porcentaje de las evaluaciones que se han realizado. Esta tiene la siguiente formulación:

$$Desempeño = \frac{100}{w_1 \times error^2 + w_2 \times (\frac{\%Evals}{100})^2 + 1} \tag{4.1}$$

La idea de la construcción de la variable Desempeño es que dos algoritmos A, B tendrán el mismo desempeño si: ${Error_A}^2 + \%\ {Evals_A}^2 = {Error_B}^2 + \%\ {Evals_B}^2$. Como se puede ver en la figura 4.1 el recorrido del Desempeño está en el intervalo [0,100]. Los pesos w_1, w_2 permiten equilibrar la importancia relativa de cada termino. En este caso, se asigna la misma importancia al *error* y a $\%Evals$, por ello $w_1 = w_2 = 1$.

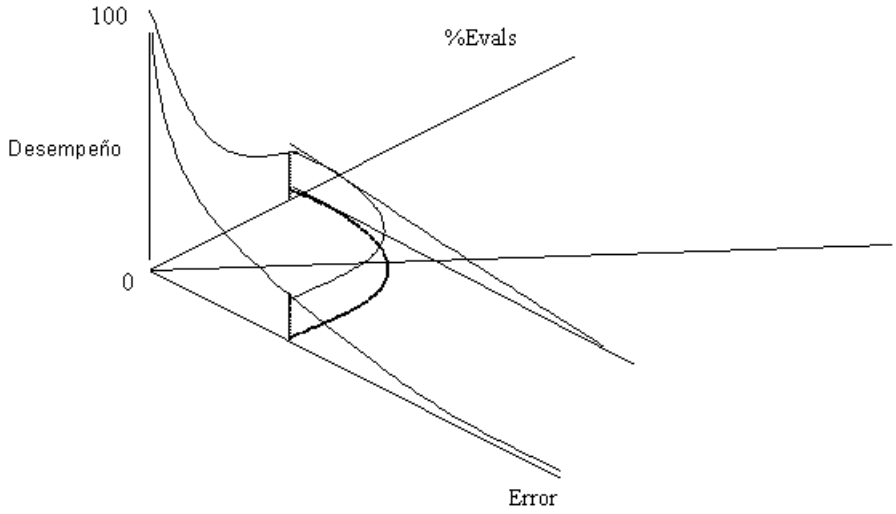


Figura 4.1: Variable Desempeño

En el Problema de la Mochila también se pueden construir instancias muy duras de resolver y que constituyen todo un reto para la mayoría de los algoritmos exactos de la actualidad. En las referencias [108, 109] se presenta un conjunto de estas instancias siguiendo dos direcciones, una en la que construye las instancias tradicionales pero con coeficientes muy grandes, y otra en la cual los coeficientes son de tamaño medio, pero todas sus cotas superiores tienen un rendimiento malo. De estas últimas se tomaron las siguientes instancias, brindadas amablemente por su autor, el Dr. Pisinger [111], para probarlas ante una implementación de la estrategia bajo paralelismo real en un cluster de ordenadores:

- spanner, span(v, m): Estas instancias son construidas de manera que todos sus items son múltiplos de un pequeño conjunto de items llamado conjunto llave y se caracterizan por los siguientes parámetros: v es el tamaño del conjunto llave, m es el limite multiplicador y en el conjunto llave se puede tener cualquier distribución de los items.

  El conjunto v de items se genera con los pesos en el intervalo [1, R] y los beneficios acordes a la distribución. Los items (p_k, w_k) en el conjunto llave se normalizan dividiendo los pesos

y los beneficios entre $m+1$. Los items se construyen tomando repetidamente un ítem (p_k, w_k) del conjunto llave y un multiplicador generado aleatoriamente en el intervalo [1, m]. El ítem construido queda así ($a \times p_k$, $a \times w_k$).

Para este experimento se consideraron tres distribuciones en las cuales el limite multiplicador se tomó $m = 10$ y tal como los experimentos computacionales lo demuestran [108] las instancias mas duras son para valores pequeños de v.

- Instancias No Correlacionadas span
  NC: $w_i = U(1,R), p_i = U(1,R), span(2,10)$.
- Débilmente Correlacionadas span
  WC: $w_i = U(1,R), p_i = U(w_i - \frac{R}{10}, w_i + \frac{R}{10})$ con $p_i \geq 1$ y $span(2,10)$
- Fuertemente Correlacionadas span
  SC: $w_i = U(1,R), p_i = w_i + \frac{R}{10}, span(2,10)$.

- profit ceiling, pceil(d): Estas instancias tienen la propiedad de que todos los beneficios son múltiplos de un parámetro dado d. Los pesos de los n items se distribuyen aleatoriamente en el intervalo [1, R], y los beneficios se obtienen calculando $p_i = d \times \lceil w_i/d \rceil$. Se tomó $d = 3$.
  $pceil(d=3)$: $w_i = U(1,R), p_i = d\lceil (w_i/d) \rceil$

- circle, circle(d): estas instancias se generan de manera que los beneficios son una función de los pesos en forma de la curva de un círculo (generalmente una elipse). Los pesos se distribuyen aleatoriamente en el intervalo [1, R]. Se tomó $d = 2/3$;
  $circle(d=\frac{2}{3})$: $w_i = U(1,R), p = d\sqrt{(4R)^2 - (w-2R)^2}$

La Capacidad de la Mochila se calcula de esta manera:

$$C = \frac{h}{1+H} \sum_{i=1}^{n} w_i,$$

donde $H = 100$ y h es el número de instancia.

Se conoce el óptimo de cada instancia, suministrados también por el Dr. Pisinger [111] y que fueron calculados usando el mejor algoritmo exacto desarrollado por este autor. Usando este valor, se calcula el *Error*:

$$Error = 100 * \frac{Optimo - Costo\,Obtenido}{Optimo}$$

Resumiendo, para comprobar el comportamiento de la estrategia ante este problema se hacen varios juegos de experimentos que se pueden agrupar en dos categorías en dependencia de la implementación del paralelismo y los recursos empleados:

- Bajo paralelismo simulado en un único computador.
- Bajo paralelismo real con PVM en un cluster de computadores.

El primer grupo de experimentos se realiza en un único ordenador y bajo una implementación en la que se simula el paralelismo. En este grupo se hacen dos juegos experimentos: uno preliminar, para probar la implementación realizada y para ello se toma un solo tipo de instancia; y otro avanzado, para confirmar la validez de la estrategia, ampliándose para ello el conjunto de instancias y tamaños del problema. En este grupo de experimentos se usan las primeras instancias explicadas anteriormente, aquellas llamadas clásicas.

Con los resultados del primer grupo de experimentos y viendo la validez de la propuesta se pasa a realizar entonces una implementación en un cluster de computadores bajo el paquete de programas PVM. Se harán dos juegos de experimentos, uno preliminar para probar la aplicación realizada; y otro ante un conjunto mayor de instancias y tamaños del problema. Para estos casos se toman las instancias consideradas muy difíciles de resolver [108, 109] para la mayoría de los algoritmos del estado del arte.

4.2. Experimentos bajo Paralelismo Simulado

Este primer grupo de experimentos se hace bajo un paralelismo simulado en una implementación secuencial de tiempo compartido, donde no hay una concurrencia real, y en la cual se arma un vector de agentes resolvedores a los que se va asignando cierto tiempo de ejecución en forma de *round-robin*. En esta implementación puede haber cierto sincronismo en la ejecución de las tareas debido al propio sistema operativo; por ello, para obtener algún tipo de asincronismo se fija una probabilidad de comunicación tanto al agente coordinador como a cada agente resolvedor. Se realizó un análisis empírico de la influencia de este parámetro en el desempeño del algoritmo. Las diferencias encontradas en los desempeños del algoritmo para los diferentes valores de analizados desde 0.2 a 0.8 sugirieron este último valor como el que producía mejores resultados.

Cada ejecución del algoritmo se hace en un sólo computador. Sin embargo, por razones de organización se llevaron a cabo en distintos momentos y en ordenadores diferentes; pero esto no influye en los resultados obtenidos ya que en este caso se toma como criterio de parada el número de evaluaciones de la función objetivo y este parámetro hace independiente al experimento del procesador usado.

Se realizan dos experimentos:

- Experimentos preliminares.
- Experimentos avanzados.

Los experimentos preliminares se hacen para probar la implementación realizada, por ello se toma un solo tipo de instancia y 3 tamaños del problema. Luego, en un segundo juego de experimentos, que se hacen para confirmar la validez de la estrategia, se amplía el conjunto de instancias y tamaños del problema y se comprueban los beneficios de la coordinación, se analiza de la influencia del uso de los agentes resolvedores con diferentes comportamientos y ante la variación del número de estos. En ambos grupos de experimentos las instancias del problema usadas fueron las que denominamos instancias “clásicas”.

4.2.1. Experimentos Preliminares

En este experimento se usan seis agentes, todos con igual comportamiento de búsqueda, con $\lambda = 0{,}95$. Se definen 3 tamaños de las instancias del problema, $n = \{150, 300, 450\}$, y para cada n se generan aleatoriamente 5 instancias en las que hay una correlación débil entre pesos y beneficios (WC), donde $p_i = U(w_i - t, w_i + t)$, para $t = 100$ (constante arbitraria). Se toma este tipo de instancia debido a que las evidencias experimentales muestran que son las más cercanas a problemas reales [110].

En el Coordinador sólo se usa la regla **MR1** de su Base de Reglas Difusas (FRB), es decir:

IF `la calidad de la solución del` $agente_i$ `es` **mala**
AND `razón de mejora del` $agente_i$ `es` **mala**
THEN `enviarle al` $agente_i$ `la mejor solución.`

En los experimentos sucesivos y hasta que no se especifique lo contrario esta será la regla usada.

Para cada una de las 15 instancias, se ejecuta 30 veces el esquema con y sin la base de reglas activada. Cuando la base de reglas está desactivada, no hay flujo de de información entre el Coordinador y los agentes resolvedores.

Cada ejecución termina cuando se han hecho $100 \times n$ evaluaciones de la función objetivo. En cada ejecución, se calcula el *Error* (respecto a la cota de Dantzig [110]) de la mejor solución encontrada, y el número de evaluaciones de la función de costo usadas para alcanzar dicho error que se transforma para mostrar el porcentaje de evaluaciones usadas sobre el total de evaluaciones disponibles ($\%\, Evals$).

Se analizan los resultados desde dos puntos de vista. Primero, considerando el conjunto de las instancias globalmente; y segundo, analizando los resultados en términos del tamaño del problema.

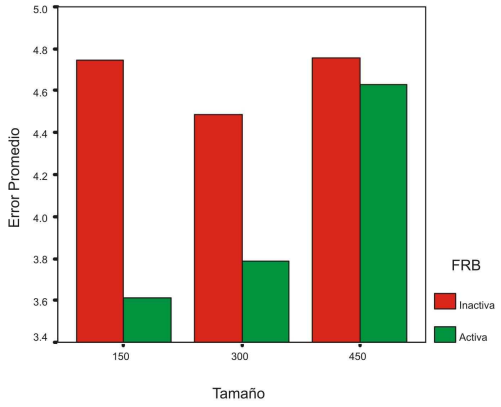
En la Tabla 4.1 se presentan los resultados sobre el conjunto total de instancias para las dos variables de interés: *Error* y $\%\, Evals$. En relación con el *Error*, es claro que cuando la base de reglas, FRB, está activa, la estrategia consigue valores medios significativamente menores que cuando está inactiva. La desviación estándar es ligeramente mayor para el caso en el que FRB está disponible.

Tabla 4.1: Instancia WC, Media y Desviación estándar del *Error* y $\%\, Evals$

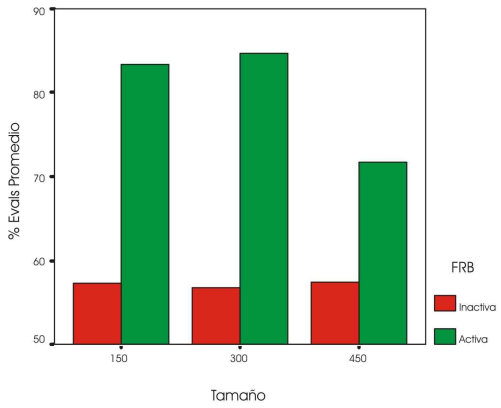
	Error		$\%\, Evals$	
FRB	*Media*	*DT*	*Media*	*DT*
inactiva	4.66	(0.37)	57.19	(27.24)
activa	4.01	(0.59)	79.95	(18.53)

En términos de $\%\, Evals$, la media es mayor cuando FRB está disponible. Si consideramos el caso en el que FRB estaba inactiva, las mejores soluciones se alcanzaron utilizando alrededor del 57 % de las evaluaciones disponibles, esto significa, que el 43 % de las evaluaciones disponibles fueron inútiles para mejorar.

Los resultados en términos del tamaño del problema se muestran en la Figura 4.2 El gráfico (a) muestra la media del *Error* en función del tamaño. Puede verse fácilmente que para cada



(a)



(b)

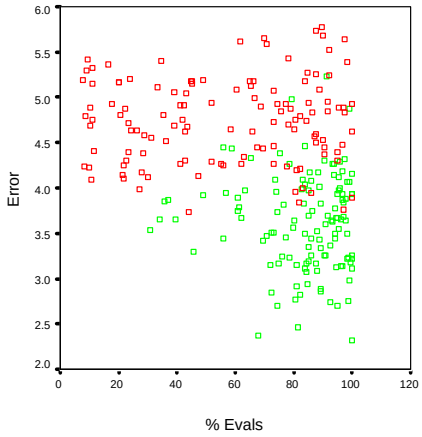
Figura 4.2: Gráfico de Barras, (a) media de Error y (b) media del $\%\,Evals$ en función del tamaño para FRB activa / inactiva.

tamaño considerado, el uso de la FRB permite obtener valores menores del *Error*. También que la media de este crece junto con el tamaño del problema. Cuando la FRB está inactiva no hay una tendencia clara para el *Error*. El gráfico (b), muestra la media de $\%\,Evals$ para cada tamaño considerado. Cuando FRB está activa, no puede detectarse un comportamiento claro; cuando FRB está desactivada los valores son prácticamente idénticos para cada tamaño. Además, el primer caso presenta mayores valores que el segundo.

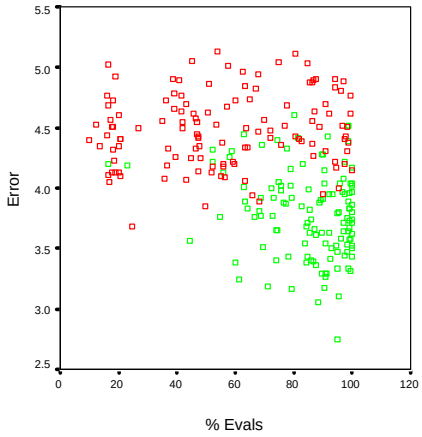
Para finalizar el análisis, en la Fig. 4.3 se muestran los diagramas de dispersión del *Error* vs. $\%Evals$ para cada tamaño de instancia considerado $n = \{150, 300, 450\}$. Los puntos más oscuros (color rojo) representan los resultados sin la utilización de la FRB, o sea, sin coordinación; y los puntos claros (color verde) los resultados bajo coordinación.

Para los tamaños $n = \{150, 300\}$, los gráficos muestran una diferencia clara de comportamiento entre ambos esquemas. La utilización de coordinación permite obtener errores de menor valor, y en general, haciendo uso de más evaluaciones de la función de costo. Esto significa que la estrategia cooperativa es capaz de evitar los óptimos locales de mala calidad que obtiene el esquema sin coordinación, permitiendo una exploración más adecuada de zonas mejores del espacio de búsqueda.

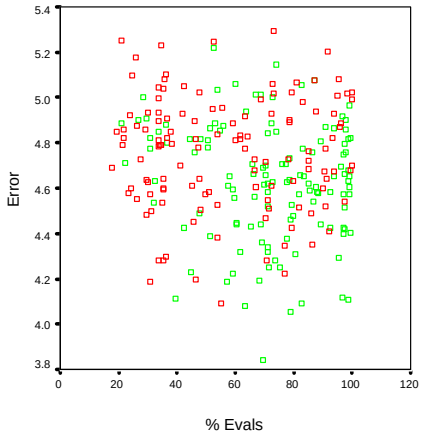
Cuando $n = 450$, la diferencia de comportamiento no parece tan clara a simple vista. Sin



$n = 150$



$n = 300$



$n = 450$

Figura 4.3: Gráficos de dispersión del *Error* vs. *%Evals*

embargo, si consideramos la zona del gráfico por debajo de $Error < 4{,}5\,\%$, podemos observar que allí aparece una mayoría de soluciones provistas por el esquema con coordinación.

Como conclusión a estos resultados preliminares se obtiene que los resultados obtenidos sobre esta instancia del problema muestran la validez de la estrategia propuesta y su mecanismo de coordinación bajo una base de reglas difusas.

4.2.2. Experimentos Avanzados

Viendo en los experimentos anteriores lo positivo que resulta utilizar la estrategia propuesta se probará entonces a continuación ante un conjunto mayor de instancias y tamaños del problema con los que se comprobará los beneficios de la coordinación, se analizará la influencia del uso de los agentes resolvedores con diferentes comportamientos y cuánto se afectan los resultados al variar el número de agentes usados. En estos experimentos se usarán también las primeras instancias del problema explicadas anteriormente, las llamadas clásicas. Se hacen tres conjuntos de experimentos, con los siguientes objetivos:

1. **Comprobar los beneficios de la coordinación:** El objetivo de este experimento es evaluar si el uso del esquema de coordinación propuesto mejora los resultados con respecto a los obtenidos sin el uso de la coordinación ante un conjunto de diferentes instancias del problema.
2. **Análisis de la influencia del uso de agentes resolvedores con diferentes comportamientos:** El objetivo de este experimento es analizar el comportamiento de la estrategia al variar los parámetros de ajustes de los agentes.
3. **Análisis de la influencia del número de agentes:** El objetivo de este experimento es evaluar el comportamiento de la estrategia de acuerdo al número de agentes y teniendo éstos los mismos parámetros de ajuste.

En estos experimentos se usan 3 tipos de instancias, dado $w_i = U(1, max)$:

- No Correlacionadas, NC: $p_i = U(1, max)$
- Débilmente Correlacionadas, WC: $p_i = U(w_i - t, w_i + t)$
- Fuertemente Correlacionadas, SC: $p_i = w_i + k$

donde $t = 100$, $max = 1000$ and $k = 10$ son constantes arbitrarias.

La Capacidad de la Mochila se calcula así:

$$C = \alpha * \sum_{i=1}^{n} w_i \text{ ,con } \alpha = 0{,}5$$

Se definen 5 tamaños del problema: $n = \{150, 300, 450, 600, 750\}$, y para cada uno se generan aleatoriamente 20 instancias de cada tipo. Cada ejecución termina cuando se han hecho $k * n$ evaluaciones de la función objetivo. Donde $k = 100$ para $n = \{150, 300, 450\}$ y $k = 300$ para $n = \{600, 750\}$.

Se calcula nuevamente el *Error* en base a la cota de Dantzig por no conocerse los valores óptimos de las soluciones. También se obtiene el porcentaje de evaluaciones usadas sobre el total de evaluaciones disponibles, $\%\ Evals$ y con estas dos variables se calcula el *Desempeño*, ya explicado anteriormente.

4.2.2.1. Comprobación de los beneficios de la coordinación

En este experimento se usan seis agentes resolvedores, todos con $\lambda = 0{,}95$, evitando con esto que las diferencias de comportamiento entre ellos no dejara ver con claridad el objetivo fundamental a revisar aquí que es analizar el efecto de usar o no la coordinación. Para cada una de las 15×20 instancias, se ejecuta 25 veces el esquema con y sin la base de reglas (FRB) activada. Cuando se activa la base de reglas se usa la regla **MR1**.

En este caso se analizan los resultados obtenidos desde dos puntos de vista. Primero, considerando cada tipo de instancia; y segundo, analizando los resultados en términos del tamaño del problema, tal como se muestra en la Tabla 4.2 para las variables Error, $\%\,Evals$ y Desempeño.

Tabla 4.2: Promedio y Desviación Típica del *Error*, $\%\,Evals$ y Desempeño para cada tipo de instancia. *Si*, *FRB* habilitada; *No*, *FRB* no habilitada.

	Error		$\%\,Evals$		*Desempeño*	
Inst	*No*	*Si*	*No*	*Si*	*No*	*Si*
NC	6,65 (1,20)	5,65 (0,98)	68,47 (22,27)	85,61 (14,55)	2,39 (0,85)	3,22 (1,13)
WC	4,90 (0,58)	4,51 (0,91)	50,08 (28,02)	70,94 (25,53)	4,11 (0,95)	5,07 (2,08)
SC	0,65 (0,06)	0,63 (0,06)	54,62 (28,64)	64,67 (25,74)	56,99 (9,71)	54,41 (9,14)

Tabla 4.3: Promedio y Desviación Típica del *Error*, $\%\,Evals$ y *Desempeño* para cada tamaño. *Si*, *FRB* habilitada; *No*, *FRB* no habilitada.

	Error		$\%\,Evals$		*Desempeño*	
Tam	*No*	*Si*	*No*	*Si*	*No*	*Si*
150	4,28 (2,99)	3,33 (2,32)	54,37 (27,04)	75,53 (21,12)	22,18 (27,47)	22,16 (24,55)
300	4,07 (2,79)	3,30 (2,13)	56,54 (24,54)	78,41 (22,29)	21,58 (26,21)	21,85 (24,61)
450	4,00 (2,55)	3,87 (2,44)	66,37 (28,80)	74,18 (24,04)	21,03 (25,70)	20,50 (24,66)
600	3,73 (2,19)	3,48 (2,03)	55,49 (28,15)	70,92 (26,30)	21,10 (25,48)	20,34 (23,69)
750	4,24 (2,55)	4,00 (2,38)	55,87 (27,54)	69,64 (25,83)	19,91 (24,90)	19,66 (24,05)

En relación con el Error, es claro que cuando la FRB está activa, la estrategia consigue valores medios significativamente menores que cuando está inactiva. La desviación estándar es ligeramente mayor para el caso en el que FRB está disponible en las instancias WC y SC, no así en la instancia NC, en la cual es menor. El test no parámetrico U de Mann - Whitney para $\alpha \leq 0{,}01$ muestra que hay diferencias significativas entre los valores de Error cuando está activa la FRB que cuando está inactiva para los diferentes tipos de instancias.

En términos de $\%\,Evals$, los valores promedios son mayores cuando la FRB está disponible. Esto implica que cuando no se usa coordinación los mejores resultados son obtenidos más rápido. Como se vio anteriormente este hecho podría ser considerado como no muy bueno a primera vista.

Acerca de la variable Desempeño, los valores son mejores cuando FRB está activa en las instancias NC y WC. En la instancia SC, que es la más fácil de resolver, los valores son ligeramente mayores cuando no hay coordinación, debido a similares valores en su Error, y a que sus valores en el $\%\,Evals$ son más bajos que con la FRB activa.

Los resultados en términos del tamaño del problema se muestran en la Tabla 4.3. Se muestra que los valores mas bajos de Error en cada tamaño se obtienen con el uso de la coordinación. Los

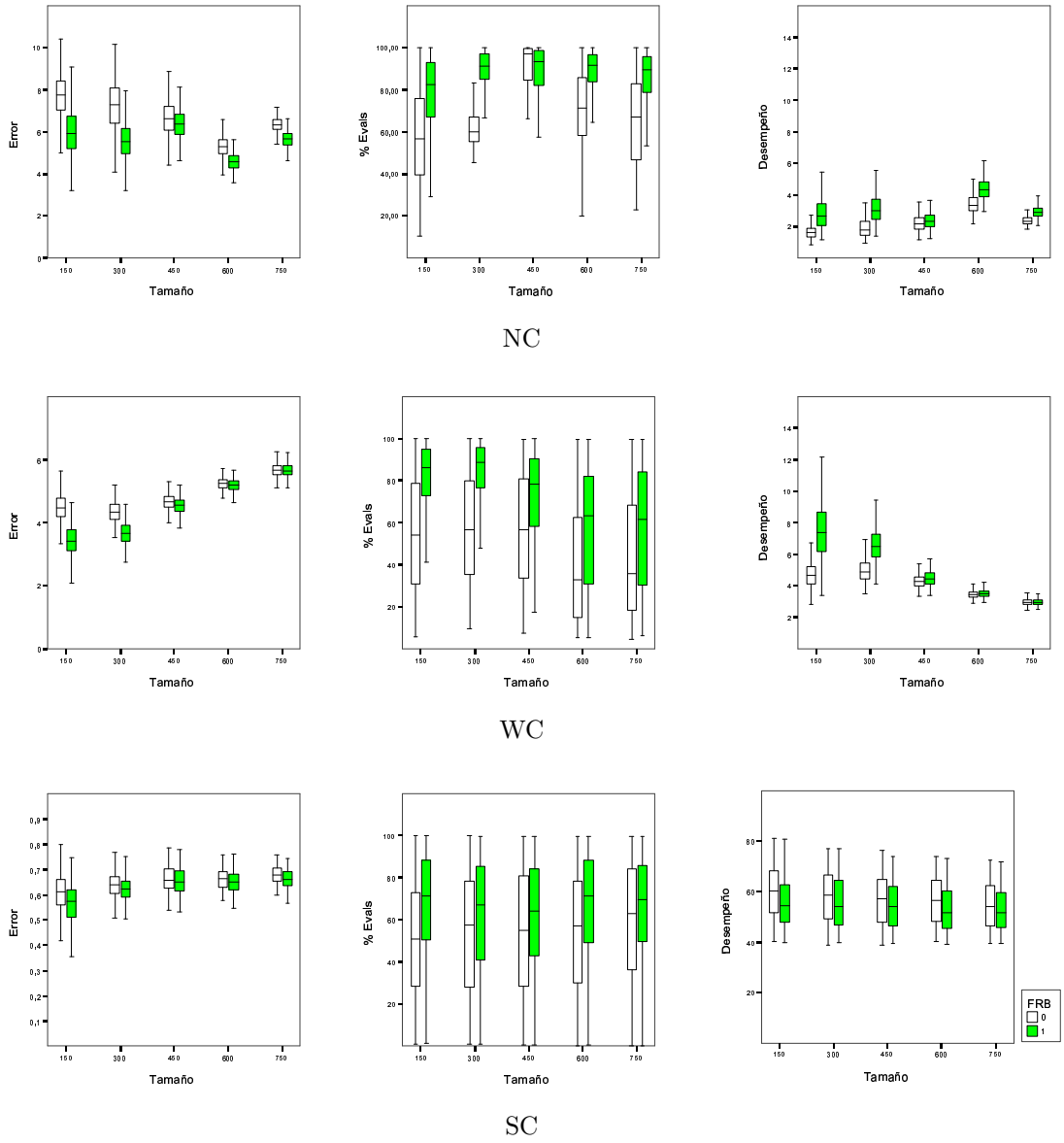


Figura 4.4: Promedio del *Error*, *%Evals* y *Desempeño* para cada tipo de instancia en función del tamaño con FRB on/off.

valores mas altos de % *Evals* son obtenidos con la coordinación activada en cada tamaño. Esto significa que con el uso de la coordinación se hace un mejor uso de los recursos disponibles. Sin embargo, su incremento en el % *Evals* produce una reducción del Desempeño, donde los valores más altos son obtenidos cuando no se usa la coordinación. Se hace notar que las pruebas no paramétricas U de Mann - Whitney para $\alpha \leq 0{,}05$ mostraron que hay diferencias estadísticamente significativas en la medias del Error entre la FRB activada y no activada para todos los tamaños.

Para ampliar el análisis se muestran los gráficos de cajas para el promedio del Error, % *Evals* y Desempeño (ver Figura 4.4), como una función del tipo de instancia y el tamaño, tanto para la FRB activada y no activada. Las escalas se han ajustado en cada caso para facilitar su visualización.

Cuando el tipo de instancia se tiene en cuenta, las diferencias entre los valores de FRB son claros en las instancias NC y WC: los valores de Error son más bajos cuando la FRB está activa y las diferencias disminuyen cuando el tamaño se incrementa. No hay un patrón claro en el comportamiento del % *Evals*, aunque los valores más altos se obtienen con la FRB activada.

Para ambos tipos de instancias el valor del Desempeño es mejor cuando la FRB está activada y los análisis estadísticos muestran que estas diferencias son significativas.

En la instancia SC, las diferencias sobre el Error y el % *Evals* no son suficientes para lograr un mejor valor del Desempeño. Esto sólo pasa con SC, que son las instancias más fáciles, debido a que lo ganado con la coordinación en una mayor exploración del espacio de búsqueda se pierde en recursos computacionales. Todo esto indica que nuestra estrategia sólo tiene sentido ante instancias difíciles del problema.

4.2.2.2. Influencia del uso de agentes con diferentes comportamientos

Como se explicó en el capítulo 3, en el epígrafe 3.2.2, el comportamiento del algortimo FANS, usado en los agentes resolvedores, depende de la definición de sus componentes y de la interacción entre ellos. A través de la utilización de diferentes definiciones y parámetros para la valoración difusa se consigue que el algoritmo se comporte de diferente manera, lo que da lugar, en consecuencia, a variaciones en los resultados que se puedan obtener. Así, si se consideran los siguientes valores de la valoración difusa:

$$\lambda = \{0{,}45, 0{,}6, 0{,}75, 0{,}8, 0{,}9, 1\}$$

se pueden construir tres esquemas diferentes con 6 agentes en cada uno, tal como se ve en la Tabla 4.4.

Tabla 4.4: Esquemas de Búsqueda

Esquemas	*Agente*1	*Agente*2	*Agente*3	*Agente*4	*Agente*5	*Agente*6
Greedy	0.80	0.90	1.00	0.80	0.90	1.00
Permisivo	0.45	0.60	0.75	0.45	0.60	0.75
Mixto	0.45	0.90	0.75	0.80	0.60	1.00

En el primer esquema los valores de λ están cercanos a 1; o sea, que la aceptabilidad de aquellas soluciones que sean peores que la referencia es baja, por tanto solo se aceptarán soluciones que mejoren la solución actual, por ello se denomina *greedy*. El segundo esquema emplea agentes que usan valores bajos de aceptabilidad, en los cuales las transiciones a soluciones peores son permitidas frecuentemente. O sea, es *permisivo*, y aceptará también soluciones que no mejoran la solución actual. El tercer esquema es *mixto* al mezclar agentes con valores de aceptabilidad del primer y segundo esquema. Nótese que se usan todos los valores sugeridos. Cada una de las 300 instancias del problema y cada esquema se ejecuta 80 veces para un total de 72000 ejecuciones de la estrategia.

Los resultados de este experimento se analizan en términos de los tipos de instancias en cada esquema para las variables de interés: Error, % *Evals* y Desempeño en las Tablas 4.5, 4.6 y 4.7.

La Tabla 4.5 muestra que el esquema Mixto obtiene los valores más bajos del Error. Este esquema permite que estén presentes simultáneamente diferentes niveles de explotación y explo-

Tabla 4.5: Promedio y Desviación Típica del *Error* para cada tipo de instancia agrupada por esquema

Instancia	*Greedy*	*Permisivo*	*Mixto*
NC	5,55 (0,78)	21,69 (5,36)	5,40 (0,80)
WC	3,02 (0,33)	6,65 (0,72)	2,94 (0,32)
SC	0,47 (0,05)	0,66 (0,06)	0,47 (0,05)

Tabla 4.6: Promedio y Desviación Típica del % *Evals* para cada tipo de instancia agrupada por esquema

	Esquemas		
Instancias	*Greedy*	*Permisivo*	*Mixto*
NC	88,64 (11,60)	65,93 (25,06)	86,57 (12,89)
WC	89,22 (11,23)	61,85 (26,31)	87,95 (12,27)
SC	88,01 (12,84)	56,28 (27,82)	86,03 (14,75)

Tabla 4.7: Promedio y Desviación Típica del Desempeño para cada tipo de instancia agrupada por esquema

	Esquemas		
Instancias	*Greedy*	*Permisivo*	*Mixto*
NC	3,25 (1,03)	0,28 (0,21)	3,43 (1,11)
WC	9,38 (1,90)	2,27 (0,57)	9,80 (1,79)
SC	50,15 (5,67)	56,36 (9,53)	51,02 (6,31)

ración, obteniéndose así un equilibrio entre la intensificación y la diversificación lo que permite obtener soluciones de muy alta calidad. Es destacable la relación inversa entre el Error y el tipo de instancia, en que la más alta correlación implica el error más bajo respecto a la cota de Dantzig.

En la Tabla 4.6, se muestra una clara reducción del % *Evals* en el esquema Permisivo según la correlación de las instancias se incrementa. Esto debe verse junto a los correspondientes altos niveles de Error en la Tabla 4.5. Este comportamiento se explica debido a que los agentes en este esquema permiten transiciones a casi cualquier otra solución del vecindario, poniendo una baja presión en la búsqueda de mejores soluciones. Así la estrategia alcanza rápidamente una meseta del espacio de búsqueda con soluciones de costo muy similar y se mantiene en un ciclo entre ellas.

Los valores del Desempeño en la Tabla 4.7 son mejores bajo el esquema Mixto en las instancias NC y WC. En la instancia SC los mejores resultados son obtenidos con el esquema Permisivo. También aquí puede verse que a medida que aumenta la correlación, y por lo tanto disminuye la dificultad de las instancias, el Desempeño aumenta.

Para concluir, se muestran los gráficos de cajas (ver Figura 4.5) para las variables Error, % *Evals* y Desempeño como una función del tamaño de la instancia para cada tipo de esquema y para cada tipo de instancia considerado. Las escalas se han ajustado en cada caso para facilitar su visualización.

Los valores de Error pueden ser vistos en relación con el tipo de instancia; así, según aumenta la correlación entre los pesos y los beneficios, el valor del Error disminuye. El tamaño de las instancias solo afecta al esquema Permisivo, mientras que no se detecta ninguna relación entre

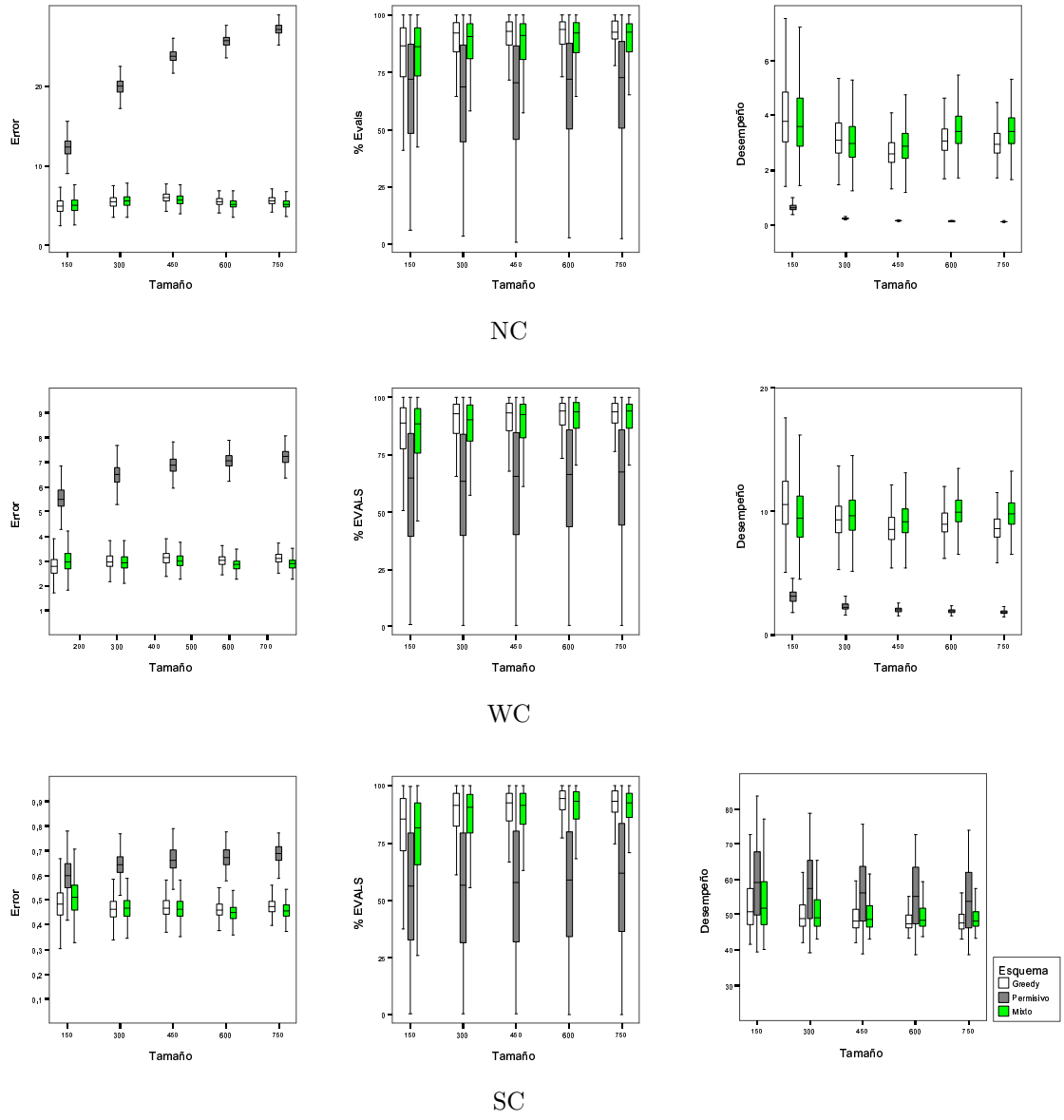


Figura 4.5: Gráficos de Caja de promedios de *Error*, *%Evals* y *Desempeño* vs. Tamaño

tamaño y Error. Los valores del % *Evals* parecen no ser afectados por el tamaño y el tipo de instancia. El patrón de comportamiento de los diferentes esquemas es también claro: el esquema Permisivo siempre tiene los valores mas bajos de % *Evals*, indicando que éste no hace un uso rentable de los recursos. En cambio, los esquemas Ávido y Mixto siempre usan más de un 80 % del % *Evals* disponibles. Esto sugiere que el esquema Permisivo es sensible al tamaño del espacio de búsqueda por su falta de eficacia en intensificación, y que los otros esquemas están actuando aún (en términos de tamaño) de manera eficiente. Sospechamos que aumentando el tamaño o la dificultad de las instancias estos esquemas mostrarán sensibilidad a este cambio.

Los valores del Desempeño están relacionados con el Error. Se puede ver que en las instancias NC y WC, los esquemas Ávido y Mixto son mejores que el Permisivo, independientemente del tamaño considerado. Para las instancias SC, los mejores valores del Desempeño son obtenidos

por el esquema Permisivo. Esto no es extraño y está en relación directa con la construcción de esta variable; y es que todos los esquemas obtuvieron valores muy bajos de Error y el esquema Permisivo tiene el valor más bajo del $\%\,Evals$. Respecto al tamaño del problema, se muestra que en todos los esquemas el Desempeño disminuye según el tamaño del problema se incrementa.

4.2.2.3. Influencia del número de agentes

El objetivo de este experimento es ver cuánto pueden afectarse los resultados al variar la cantidad de agentes resolvedores que se usan en la estrategia. Para ello se toman 10 instancias del problema de tamaño 450, generadas aleatoriamente y sin correlación entre los pesos y los beneficios (NC). El algoritmo se ejecuta 20 veces para cada instancia con y sin coordinación y variando el número de agentes de 1 a 6. Todos los agentes resolvedores tienen el mismo comportamiento de búsqueda "greedy", con $\lambda = 0,95$. Se configuran todos iguales con el objetivo de evitar que los resultados se vean influenciados por otros factores como el tener cada agente diferentes parámetros de ajuste. Se realizan 100000 evaluaciones de la función objetivo en la estrategia y cada 20000 se hace una toma de datos. Así, el número de datos obtenido es: 20 ejecuciones $\times$ 10 instancias del problema $\times$ 6 agentes $\times$ 5 capturas de información $\times$ 2 esquemas de coordinación (excepto para el caso de 1 agente, donde no tiene sentido la coordinación) = 11000 datos.

En las Tablas 4.8 y 4.9 se presenta el promedio y la desviación estándar del Error, el porcentaje de las evaluaciones para el mejor ($\%\,Evals$) y el Desempeño para los conjuntos de 1 a 6 agentes con y sin coordinación (coordinado: Si, no coordinado ó independiente: No). Se recuerda que para el caso de un agente solo tiene sentido el esquema independiente. Se observa que todas las variables en el esquema coordinado son mejores que en el esquema independiente, o sea, los Errores son menores, mientras que el Desempeño y $\%\,Evals$ son mayores. La variable $\%\,Evals$ muestra que casi todas las evaluaciones dadas se usaron para mejorar la búsqueda.

	Error		$\%\,Evals$		*Desempeño*	
Clon	*No*	*Si*	*No*	*Si*	*No*	*Si*
1	9,07	-	50,27	-	1,21	-
2	9,15	4,90	55,08	95,40	1,18	4,00
3	9,14	5,55	58,88	94,06	1,19	3,14
4	9,21	5,67	61,60	93,81	1,17	3,02
5	9,19	5,84	64,63	93,45	1,17	2,87
6	9,27	5,60	67,63	93,86	1,15	3,10

Tabla 4.8: Promedio del *Error*, $\%\,Evals$ y *Desempeño*. *Si*, *FRB* habilitada; *No*, *FRB* no habilitada.

	Error		$\%\,Evals$		*Desempeño*	
Clon	*No*	*Si*	*No*	*Si*	*No*	*Si*
1	0,55	-	27,46	-	0,15	-
2	0,45	0,58	25,40	4,63	0,12	0,91
3	0,50	0,54	23,34	5,50	0,13	0,59
4	0,47	0,58	21,82	5,84	0,12	0,59
5	0,51	0,63	21,64	5,78	0,14	0,62
6	0,49	0,57	19,12	5,84	0,13	0,64

Tabla 4.9: Desviación estándar del *Error*, $\%\,Evals$ y *Desempeño*. *Si*, *FRB* habilitada; *No*, *FRB* no habilitada.

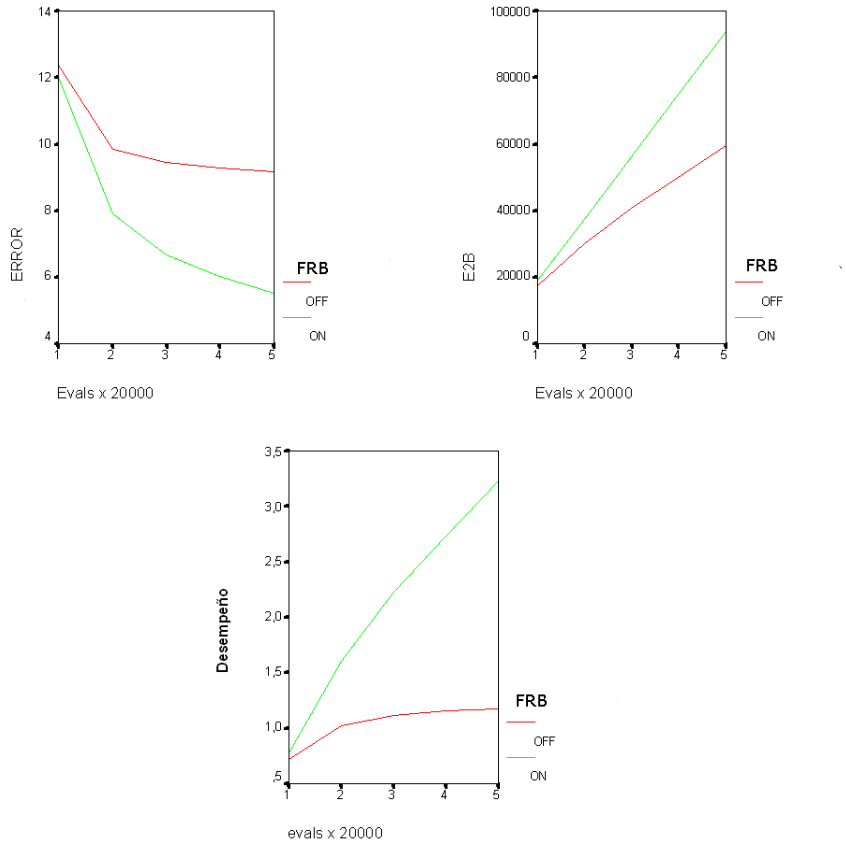


Figura 4.6: Promedio del Error, E2B y Desempeño vs. Evaluaciones

De acuerdo al número de agentes, los mejores valores se obtuvieron con 2 agentes, bajo el esquema coordinado, en todas las variables del estudio. Las diferencias de los promedios de error de los grupos de agentes son significativas para este caso de 2 agentes respecto a los demás. Esto mismo sucede para el Desempeño.

Es lógico pensar que la estrategia tendría una mayor capacidad de exploración según se incremente el número de agentes; sin embargo, la cantidad de evaluaciones disponibles se fija de manera global y por tanto, ésta se divide proporcionalmente entre los agentes. O sea, a medida que aumenta el número de agentes, la cantidad de evaluaciones disponibles a cada agente disminuye y entonces la posibilidad de una mayor intensidad de la explotación en la búsqueda decrece.

Esto explica los resultados logrados en el esquema independiente donde el Error obtiene resultados peores que el esquema coordinado y en el cual tampoco se ve una clara tendencia de su crecimiento asociada al número de agentes. Esto es debido a que no hay interrelación entre ellos. En el esquema coordinado las diferencias del Error son menores debido a que hay una comunicación entre los agentes a través del Coordinador. Así se obtienen los mejores resultados con 2 agentes, donde al parecer este número de agentes, con el número global de evaluaciones de la función objetivo divido entre ellos, y para este tamaño del problema, logra el mejor equilibrio entre explotación y exploración. Suponemos que para instancias del problema de mayor tamaño este número de agentes podría crecer.

En la Figura 4.6 se muestra la evolución del promedio del Error, E2B y el Desempeño según las evaluaciones usadas. El eje horizontal (evals x 20000) muestra las llamadas a la función objetivo

en muestreos de información cada 20000 llamadas.

En ella se muestra como el error prácticamente se mantiene constante a partir de las 40000 evaluaciones en el esquema independiente, mientras que bajo coordinación muestra una mejora continua.

Según se hacen ejecuciones se van obteniendo valores cada vez más altos de E2B, significando esto que se están encontrando mejores soluciones, y por tanto se hace mejor uso de los recursos disponibles.

En cuanto al comportamiento del Desempeño se nota que el esquema coordinado mantiene un incremento lineal hasta el fin de las ejecuciones. Esto no es así para el esquema independiente que solo se incrementa hasta aproximadamente la mitad de las evaluaciones y luego se mantiene casi constante.

4.3. Experimentos bajo Paralelismo Real

Los buenos resultados en paralelismo simulado nos inducen a pensar que con las ventajas que brindan los ambientes de programación paralela, como se vio en el capítulo 2, podamos aún mejorar los resultados y pasar a un análisis en términos de tiempo de ejecución y no de número de evaluaciones de la función objetivo. Por este motivo hemos decidido pasar la implementación de la estrategia propuesta de un paralelismo simulado a un paralelismo real.

Estos experimentos se realizarán en un cluster de ordenadores bajo una implementación paralela usando el paquete de software PVM (Parallel Virtual Machine), donde hay concurrencia real y un total asincronismo.

La configuración del cluster de computadores usado es la siguiente: 8 ordenadores AMD Athlon duales, con 2 GHz, 256 k en la memoria caché, y 2 GBytes de RAM, con el Sistema Operativo Linux Fedora Core release 3.

En esta implementación cada agente solucionador es creado como un clon de FANS en una tarea del PVM.

En este se realizan dos juegos experimentos:

- Experimentos preliminares.
- Experimentos avanzados.

Los experimentos preliminares se hacen para probar la implementación realizada, por ello se toma un solo tipo de instancia y tamaño, una de las que resultó difícil de resolver en la implementación anterior: NC 450. Se realizan algunas pruebas para tener una visión clara del funcionamiento de la estrategia bajo paralelismo real. Estos experimentos preliminares nos han servido también para ajustar diferentes detalles de implementación que dependen fuertemente del ambiente y lenguaje escogido y cuya descripción amén de tediosa nos parece innecesaria ya que no aportaría nada al análisis de nuestra propuesta.

Luego, en un segundo juego de experimentos, para confirmar la validez de la estrategia, ante un conjunto de instancias consideradas duras de resolver, se comprueban los beneficios de la coordinación y se analiza de la influencia del uso de agentes resolvedores con diferentes comportamientos. Se prueban todas las reglas de la Base de Reglas del Coordinador

En ambos experimentos el criterio de parada que se toma es el tiempo de ejecución, a diferencia del epígrafe anterior que fue el número de evaluaciones. Esto se hace igual a lo que muestra la literatura en la mayoría de los casos. Por tanto ya no se toma aquí el número de evaluaciones para alcanzar el mejor valor, ni la variable Desempeño usada anteriormente.

4.3.1. Experimentos Preliminares

En este juego de experimentos se usan 10 instancias sin correlación entre los pesos y los beneficios (NC), de tamaño 450, y generadas aleatoriamente. Cada instancia se ejecuta 20 veces durante 30 segundos.

Se comprobará cuánto pueden afectarse los resultados debido al uso o no de la coordinación y ante la variación del número de agentes resolvedores que se usan en la estrategia. Todos los agentes tienen el mismo comportamiento de búsqueda "greedy", con $\lambda = 0,95$. Se configuran de esta forma, todos iguales, con el objetivo de evitar que los resultados se vean influenciados por otros factores como el tener cada agente diferentes parámetros de ajuste. Luego se usará un esquema mixto, en el cual los agentes tendrán los siguientes valores de configuración: $\lambda = \{0{,}45, 0{,}6, 0{,}75, 0{,}8, 0{,}9, 1\}$. En todos los casos bajo coordinación ya sea con el esquema greedy o el mixto se usa la regla **MR1** de la Base de Reglas Difusas.

En la Tabla 4.10 se muestran en una columna los resultados del promedio del Error y la Desviación Típica obtenidos bajo un esquema greedy coordinado (*Si*), y en otra columna los valores logrados bajo un esquema independiente, sin coordinación (*No*). Se aclara que no aparecen los valores con un solo agente para el caso bajo coordinación porque se necesitan al menos dos agentes para que cooperen entre sí. Para el esquema mixto solo tiene sentido tener 6 agentes, cada uno con una configuración diferente. Los mejores resultados se obtienen con 6 agentes, ya que logra el menor valor de Error promedio, tanto en el esquema coordinado como en el independiente. Comparándolos entre ellos, se ve que siempre el esquema coordinado logra mejores valores que el esquema independiente para los diferentes números de agentes usados.

Clon	*Si*	*No*	*Mixto*
1	-	8,79 (0,45)	-
2	3,59 (1,02)	3,96 (0,36)	-
3	3,40 (1,13)	3,94 (0,41)	-
4	3,66 (1,51)	3,91 (0,39)	-
5	3,42 (1,30)	3,79 (0,44)	-
6	3,31 (1,19)	3,71 (0,39)	1,73 (0,25)

Tabla 4.10: Promedio Error y Desviación Típica, *Si*, esquema greedy; *No*, sin coordinación

Los resultados del esquema mixto son mejores a los anteriores, ya sea usando un esquema independiente o un esquema greedy coordinado. Aunque creemos que no es factible comparar estos resultados con los de la implementación bajo paralelismo simulado porque las condiciones de la verificación experimental son otras, aquí también se logran resultados de mas calidad que los obtenidos en aquellos experimentos.

Como conclusiones a estos experimentos preliminares se tienen las siguientes: se comprobó que al aumentar el número de agentes se mejoran los resultados, que en este caso se hizo hasta la cantidad de 6 agentes. Creemos que esto será así hasta cierto valor en que lógicamente puedan aparecer cuellos de botella en la comunicación con el Coordinador que afecten el trabajo de la

estrategia. No se usaron más agentes porque el cluster tenia 8 ordenadores, de los cuales se usa uno para el Coordinador y se dejó uno libre por si ocurría algún fallo, además al definir el esquema mixto se tomaron 6 valores de λ que en experimentos anteriores con el algoritmo FANS resultaron adecuados lograr determinados comportamientos de búsqueda. Se comprobó también la validez de usar una estrategia coordinada comparada ante una estrategia sin coordinación. Se verifica que bajo coordinación los mejores resultados se logran con un esquema mixto, donde se mezclen agentes resolvedores con características diferentes para realizar la búsqueda, ante un esquema greedy en el que todos los agentes se comportan iguales. Por ello, a continuación se probará la estrategia ante un conjunto de instancias más difíciles de resolver.

4.3.2. Experimentos Avanzados

Viendo que los resultados preliminares obtenidos son positivos bajo esta implementación se pasa ahora a probar la estrategia ante instancias con mayor nivel de dificultad en su solución. En estos experimentos se usarán aquellas instancias del problema de la mochila denominadas duras [108, 109], brindadas amablemente por el Dr. Pisinger y que fueron explicadas anteriormente.

Se tomaron las siguientes instancias:

- *Instancias No Correlacionadas span*
- *Débilmente Correlacionadas span*
- *Fuertemente Correlacionadas span*
- *Profit Ceiling, pceil*
- *Circle*

Se conoce el óptimo de cada instancia, suministrado también por Pisinger [111] y que fue calculado usando el mejor algoritmo exacto desarrollado por éste. Usando este valor, se calcula el *Error*:

$$Error = 100 * \frac{Optimo - Costo\,Obtenido}{Optimo}$$

Ante este tipo de instancias se hacen 3 juegos de experimentos para cumplir los objetivos siguientes:

1. **Comprobar los beneficios de la coordinación**
2. **Analizar la influencia del uso de agentes resolvedores con diferentes comportamientos**
3. **Analizar la influencia del uso de memoria**

4.3.2.1. Comprobación de los beneficios de la coordinación

El objetivo de este experimento es analizar si es beneficioso el uso de la coordinación ante estas instancias, para ello se compararán los esquemas de búsqueda siguientes:

- esquema secuencial
- esquema independiente.
- esquema cooperativo.

El algoritmo secuencial, FANS, ajustado con una valoración difusa $\lambda = 0{,}95$ se ejecutará durante $t = 180$ segundos, para así darle la misma oportunidad de tiempo de cálculo que los métodos paralelos.

La búsqueda independiente difiere de la cooperativa en que no hay flujo de información entre los agentes resolvedores; o sea, sólo se guarda la mejor solución global obtenida al finalizar la ejecución. Por el contrario, en el esquema cooperativo se activa la base de reglas difusas, FRB, con la regla **MR1**, permitiendo que los agentes resolvedores cooperen entre sí al compartir información a través del coordinador.

Tanto en la búsqueda independiente como en la cooperativa se usan 6 agentes resolvedores y se ejecuta cada instancia durante 30 segundos. Todos estos agentes son iguales, con la valoración difusa $\lambda = 0{,}95$, en ambos métodos; esto se hace para lograr una consistencia en la comparación entre ellos, de manera que la comparativa no se vea afectada por ninguno de los parámetros de configuración de los agentes. Aunque los experimentos preliminares mostraron que el esquema mixto obtiene los mejores resultados se quiso probar primero la estrategia con un esquema greedy ante estas instancias.

Se consideran 4 tamaños del problema $n = \{500, 1000, 1500, 2000\}$ y para cada tipo de instancia y rango de datos se genera aleatoriamente una serie de $H = 100$ instancias para un total de 4000. Las instancias usadas tienen R = 1000 y cada algoritmo se ejecuta una vez en cada instancia, ya que se usan 100 instancias diferentes del mismo tipo.

En la tabla 4.11 se presentan los promedios de *Error* y *Desviación Estándar* para cada tipo de instancia y tamaño. La estrategia de la búsqueda independiente obtiene los valores más altos de estas variables. Estos valores son muy cercanos a los que obtiene el algoritmo ejecutado de forma secuencial. El análisis del test no paramétrico de Kolmogorov-Smirnov y U Mann-Whitney confirman que las diferencias en el promedio del *Error* sobre el conjunto total de las instancias entre ambos no son significativas. Se puede concluir, que bajo las mismas condiciones respecto al tiempo, es casi lo mismo realizar una ejecución larga de un algoritmo específico que realizar varias ejecuciones cortas en paralelo sin coordinación.

Los mejores valores de *Error* y *Desviación Típica* se logran con el esquema cooperativo tanto para cada tipo de instancia como tamaño. El test no paramétrico mostró que las diferencias de este esquema respecto a los dos anteriores sobre el conjunto total de las instancias son significativas.

Analizando la influencia del tamaño de las instancias, se ve que a medida que este crece se incrementa el promedio de *Error* en los tres esquemas: secuencial, independiente y cooperativo (usando la regla **MR1** de la Base de Reglas).

En las Tabla 4.12 se muestran cuantas instancias fueron solucionadas con $Error \leq 1$ para cada tamaño y tipo de instancia por el esquema secuencial, el independiente y el greedy coordinado. Se destacan los resultados obtenidos en las instancias PCeil, donde casi todos los casos fueron resueltos con $error \leq 1$ por todos los esquemas. En las instancias Circle, que resultan las más difíciles de resolver ocurre lo contrario, los valores nunca alcanzan el 10 % de las instancias disponibles. En las otras instancias se observa la superioridad del esquema coordinado respecto a los otros dos esquemas y también se ve que no hay diferencias entre el esquema secuencial a los

NC

Tamaño	*Seq*	*Indep*	*Coop*
500	3,22 (2,14)	3,49 (2,41)	0,56 (0,61)
1000	8,70 (6,09)	8,78 (6,28)	4,44 (3,49)
1500	12,01 (9,07)	12,10 (9,07)	8,17 (6,51)
2000	14,15 (11,14)	14,25 (11,14)	10,86 (8,98)

WC

Tamaño	*Seq*	*Indep*	*Coop*
500	2,03(1,84)	2,08(1,91)	0,60(0,76)
1000	3,08(2,78)	3,07(2,80)	1,59(1,64)
1500	3,64(3,41)	3,64(3,46)	2,38(2,32)
2000	3,99(4,06)	4,02(4,12)	2,91(3,07)

SC

Tamaño	*Seq*	*Indep*	*Coop*
500	2,20(2,18)	2,28(2,37)	0,71(0,71)
1000	3,16(3,56)	3,20(3,62)	1,95(2,23)
1500	3,53(4,04)	3,75(4,21)	2,74(3,22)
2000	3,81(4,49)	4,02(4,69)	3,22(3,87)

PCeil

Tamaño	*Seq*	*Indep*	*Coop*
500	0,16(0,13)	0,19(0,24)	0,16(0,25)
1000	0,19(0,16)	0,21(0,27)	0,18(0,27)
1500	0,20(0,20)	0,22(0,28)	0,19(0,29)
2000	0,21(0,22)	0,22(0,29)	0,20(0,30)

Circle

Tamaño	*Seq*	*Indep*	*Coop*
500	5,10(2,78)	5,48(2,78)	2,42(1,19)
1000	10,42(5,28)	10,65(5,31)	6,17(2,77)
1500	12,31(6,34)	12,86(6,47)	8,48(3,71)
2000	13,41(6,93)	13,87(7,18)	10,28(4,54)

Tabla 4.11: Promedio de Error sobre 100 instancias de cada tipo y tamaño.

180 segundos y el esquema independiente a los 30 segundos. Respecto al tamaño en cada tipo de instancia se tiene que a medida que este aumenta se logra un menor número de casos.

4.3.2.2. Influencia del uso de agentes con diferentes comportamientos

El objetivo de este experimento es analizar el comportamiento de la estrategia ante estas instancias al usar agentes con diferentes comportamientos de búsqueda.

Como se ha visto anteriormente, las trayectorias de búsqueda de FANS pueden variar en base a una función del nivel mínimo de aceptabilidad de las soluciones. Teniendo esto en cuenta se usan dos tipos de esquemas , Greedy y Mixto, en los que se usan 6 agentes como se muestra en la Tabla 4.13.

Se usa un conjunto total de 4000 instancias y los pesos w_i se distribuyen uniformemente con $R = \{1000, 10000\}$. Se mantienen los 4 tamaños anteriores del problema $n = \{500, 1000, 1500, 2000\}$. Cada instancia se ejecuta un tiempo de 30 segundos. Se hace una ejecución para cada una de las 100 instancias de cada tipo. Para poder hacer un estudio del comportamiento

		NC	
Tamaño	*Seq*	*Indep*	*Coop*
500	23	22	77
1000	15	13	26
1500	10	11	19
2000	10	9	15
Total	58	55	137

		WC	
Tamaño	*Seq*	*Indep*	*Coop*
500	39	40	83
1000	29	30	52
1500	27	28	39
2000	26	27	36
Total	121	125	210

		SC	
Tamaño	*Seq*	*Indep*	*Coop*
500	45	44	71
1000	37	35	48
1500	33	33	41
2000	32	32	38
Total	147	144	198

		PCeil	
Tamaño	*Seq*	*Indep*	*Coop*
500	100	97	98
1000	99	98	97
1500	99	98	97
2000	99	97	97
Total	397	390	389

		Circle	
Tamaño	*Seq*	*Indep*	*Coop*
500	4	11	9
1000	3	2	3
1500	3	3	3
2000	4	3	3
Total	14	19	18

Tabla 4.12: Instancias con Error ≤ 1.

Esquemas	*Agente*1	*Agente*2	*Agente*3	*Agente*4	*Agente*5	*Agente*6
Greedy	0.95	0.95	0.95	0.95	0.95	0.95
Mixto	0.45	0.90	0.75	0.80	0.60	1.00

Tabla 4.13: Esquemas de búsqueda según valores de λ

dinámico de la estrategia se hacen toma de datos cada 5 segundos.

Los resultados de este experimento obtenidos en base al tamaño, tipo y rango de las instancias se muestran en la Tabla 4.14. Son los datos tomados al final de la ejecución, o sea a los 30 segundos.

Se observa en esta Tabla que respecto al tamaño del problema, el esquema Greedy obtiene los mejores resultados del *Error* en las instancias más pequeñas. Al incrementarse el tamaño, el promedio del *Error* se incrementa en el esquema Greedy, y es el esquema Mixto el que logra los

NC

	$R = 10^3$		$R = 10^4$	
Tamaño	*Greedy*	*Mixto*	*Greedy*	*Mixto*
500	0,56(0,61)	3,08 (3,43)	0,43(0,55)	2,68 (3,66)
1000	4,44(3,49)	2,19 (2,99)	3,59(3,02)	2,32 (2,97)
1500	8,17(6,51)	1,72 (2,37)	7,05(5,61)	1,74 (2,35)
2000	10,86(8,98)	1,81 (2,15)	9,66(7,77)	2,04 (2,45)

WC

	$R = 10^3$		$R = 10^4$	
Tamaño	*Greedy*	*Mixto*	*Greedy*	*Mixto*
500	0,60 (0,76)	1,01 (1,43)	0,46 (0,59)	1,16 (1,40)
1000	1,59 (1,64)	1,22 (1,61)	1,35 (1,44)	1,38 (1,65)
1500	2,38 (2,32)	1,50 (1,77)	2,38 (2,59)	1,56 (1,64)
2000	2,91 (3,07)	1,77 (1,88)	3,08 (3,38)	2,01 (1,93)

SC

	$R = 10^3$		$R = 10^4$	
Tamaño	*Greedy*	*Mixto*	*Greedy*	*Mixto*
500	0,71 (0,71)	1,07 (1,20)	0,51 (0,59)	0,72 (0,83)
1000	1,95 (2,23)	1,17 (1,28)	1,24 (1,42)	0,94 (0,92)
1500	2,74 (3,22)	1,35 (1,42)	1,97 (2,34)	1,60 (1,67)
2000	3,22 (3,87)	1,59 (1,91)	2,50 (3,10)	1,69 (2,00)

PCeil

	$R = 10^3$		$R = 10^4$	
Tamaño	*Greedy*	*Mixto*	*Greedy*	*Mixto*
500	0,16 (0,25)	0,13 (0,17)	0,02 (0,03)	0,02 (0,02)
1000	0,18 (0,27)	0,14 (0,18)	0,02 (0,03)	0,02 (0,03)
1500	0,19 (0,29)	0,16 (0,19)	0,02 (0,03)	0,02 (0,02)
2000	0,20 (0,30)	0,17 (0,20)	0,02 (0,03)	0,02 (0,03)

Circle

	$R = 10^3$		$R = 10^4$	
Tamaño	*Greedy*	*Mixto*	*Greedy*	*Mixto*
500	2,42 (1,19)	4,85 (2,45)	2,66 (1,29)	5,08 (2,62)
1000	6,17 (2,77)	6,65 (2,90)	6,99 (3,21)	6,73 (2,97)
1500	8,48 (3,71)	7,60 (2,92)	9,30 (4,12)	7,40 (2,85)
2000	10,28(4,54)	8,44 (3,39)	10,97(4,98)	8,10 (3,13)

Tabla 4.14: Promedio del Error sobre 100 instancias para cada tipo y tamaño.

mejores resultados. Se destaca el hecho que en las instancias No Correlacionadas, NC, el promedio del *Error* disminuye al incrementarse el tamaño. Esto puede ser debido a que cada agente resolvedor puede jugar bien su rol cuando la convergencia a un mínimo local no es muy rápida y esto ocurre para tamaños grandes del problema. En los otros tipos de instancias independientemente del rango el promedio del error se incrementa con una variación muy lenta.

Respecto al tipo de instancia los mejores resultados se obtienen en PCeil y los peores en Circle, tanto para el esquema Mixto como para el Greedy; esto era esperado teniendo en cuenta los resultados obtenidos por el algoritmo secuencial, FANS, en el experimento anterior.

En la Tabla 4.15 se muestra la cantidad de instancias que fueron solucionadas con $Error \leq 1$ para las instancias con $R = 10^4$ y tamaños $n = \{500, 1000, 1500, 2000\}$ a los 30 segundos. Aquí se destaca aún más los resultados logrados en las instancias PCeil, donde todas las instancias fueron resueltas con $error \leq 1$ en ambos esquemas, no así en las instancias Circle, donde ocurre lo

	NC		*WC*		*SC*		*PCeil*		*Circle*	
Tamaño	G	M	G	M	G	M	G	M	G	M
500	88	51	83	62	84	71	100	100	8	6
1000	28	52	53	56	56	61	100	100	4	5
1500	20	53	40	48	46	47	100	100	3	4
2000	15	46	34	38	42	46	100	100	3	4
Total	151	202	210	204	228	225	400	400	18	19

Tabla 4.15: Instancias con Error ≤ 1 a los 30 segundos.

contrario, en la que los valores nunca alcanzaron el 10 % de las instancias disponibles.

Observando los valores obtenidos se nota como se acentúa la diferencia en los éxitos alcanzados por ambos esquemas, donde el esquema Mixto (M) logra un mayor número, sobre todo en la medida que crece el tamaño, esto independientemente del tipo de instancia. El esquema Greedy (G) logra los mejores valores para tamaño $n = 500$ en todas las instancias. Ténganse esto en cuenta al valorar el total de cada columna, sobre todo en aquellos casos donde el esquema Greedy supera al Mixto y se vea como una consecuencia de las diferencias logradas en las instancias mas pequeñas.

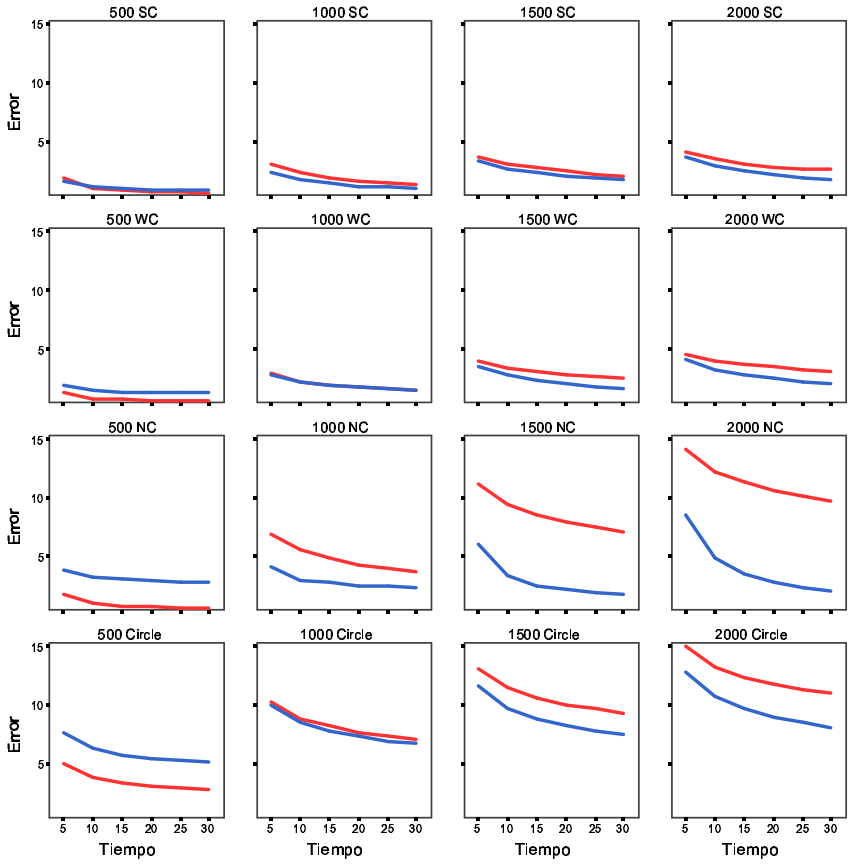


Figura 4.7: Promedio del *Error* vs. Tiempo para cada esquema (greedy o mixto), en cada tipo de instancia y tamaño.

En la Figura 4.7 se presenta el comportamiento dinámico de ambos esquemas mostrando la

evolución del promedio de error durante el tiempo de ejecución en cada tipo de instancia y tamaño para $R = 10^4$. La línea de color azul representa al esquema Mixto y la de color rojo al esquema Greedy. La instancia PCeil fue omitida debido a los valores tan bajos de error alcanzados. El gráfico muestra claramente como el esquema Mixto supera al esquema Greedy, excepto en aquellas instancias de menor tamaño y se nota como las diferencias se van haciendo mayores de izquierda a derecha según aumenta el tamaño y de arriba hacia abajo según aumenta la dificultad de las instancias, destacándose aún más en la instancia NC para tamaño = 2000.

4.3.2.3. Influencia del uso de la memoria

El objetivo de este experimento es analizar el impacto del uso de la memoria en la búsqueda de mejores soluciones. En los experimentos anteriores al usar el esquema cooperativo sólo se usó la regla **MR1**, por ello en éste se probarán cada una de las reglas definidas en la FRB y se tomará la regla Sin Memoria, **SM**, como base de la comparación. Se toman 3 tamaños del problema, $n = (1000, 1500, 2000)$, y los pesos w_i se distribuyen uniformemente con $R = \{10000\}$ que fueron los casos más difíciles de resolver en los experimentos anteriores. No se usa aquí el tipo de instancia PCeil debido a que ha sido relativamente fácil de resolver.

Se configuran los agentes resolvedores bajo el esquema Mixto y se ejecuta 10 veces la estrategia durante 15 segundos para cada instancia y regla. Para poder hacer un estudio del comportamiento dinámico de la estrategia se toman datos cada 5 segundos.

El primer análisis se enfoca en las tres primeras reglas y como se muestra en la Tabla 4.16 el uso de la memoria, con las reglas **MR1** y **MR2**, muy pocas veces alcanzan o mejoran la calidad de las soluciones obtenidas con la regla **SM**. Con estas tres reglas no se alcanza a nivel global de la estrategia un equilibrio adecuado entre la exploración y la explotación del espacio de búsqueda.

El hecho básico de tener varios agentes diferentes trabajando de forma paralela, que se iniciaron a partir de soluciones diferentes y sólo almacenando la mejor solución global encontrada por ellos en el tiempo de ejecución, como hace la regla **SM**, parece ser suficiente para alcanzar soluciones de calidad. Consideramos que esto puede mejorarse, ya que se nota que hay cierta convergencia de los agentes en aquella zona donde se alcanzan los primeros mejores valores; y por ello, se intentará resolver permitiendo que los agentes se "establezcan" y progresen por sí mismos antes de comenzar a tener en cuenta su historial. Esto se hace con las reglas **RR1** y **RR2**, en las que se deshabilita el uso de la memoria durante un tercio del tiempo total de la computación de la estrategia (en este caso los primeros 5 segundos), y luego ésta se activa. Precisando el funcionamiento de estas reglas, lo que se hace es activar la regla **SM** en los inicios de la búsqueda dejando que los agentes resolvedores se reorganicen en aquellas zonas del espacio de búsqueda potencialmente promisorias, y luego del tiempo fijado activar la memoria a través de las reglas **MR1** o **MR2** obligando a los agentes a ir moviéndose a las regiones donde se van obteniendo los mejores resultados. Se intenta lograr con esto que la exploración alcance un peso mayor respecto a la explotación durante los primeros momentos de la búsqueda.

En la Tabla 4.16 se ve que las reglas que aplican cierto retraso en el uso de la memoria siempre mejoran los promedios de Error respecto a las demás, en particular la regla **RR2** que obtiene los mejores resultados promedios.

El test no paramétrico de Kruskal-Wallis muestra que esas diferencias son estadísticamente significativos ($\alpha < 9{,}7E^{-8}$)cuando se considera el conjunto total de datos.

Haciendo un mismo análisis pero teniendo en cuenta el tipo y tamaño de las instancias algunas

NC

Regla	1000	1500	2000
SM	3,73 (3,66)	2,87 (3,18)	3,61 (3,37)
MR1	3,92 (4,06)	3,10 (3,39)	3,68 (3,57)
MR2	3,00 (3,68)	2,72 (3,41)	3,57 (3,61)
RR1	3,85 (3,78)	3,09 (3,64)	3,61 (3,40)
RR2	3,33 (3,74)	2,65 (3,22)	3,25 (3,36)

WC

Regla	1000	1500	2000
SM	2,19 (1,59)	2,45 (1,59)	3,04 (1,95)
MR1	2,37 (1,72)	2,67 (1,77)	3,20 (2,09)
MR2	2,18 (1,61)	2,69 (2,10)	3,24 (2,32)
RR1	2,18 (1,52)	2,48 (1,59)	3,00 (1,91)
RR2	2,07 (1,56)	2,30 (1,57)	2,88 (1,86)

SC

Regla	1000	1500	2000
SM	1,96 (1,43)	1,90 (1,32)	2,32 (1,79)
MR1	2,10 (1,55)	2,15 (1,58)	2,65 (2,23)
MR2	2,43 (2,56)	2,41 (2,48)	2,96 (3,00)
RR1	1,92 (1,37)	1,92 (1,31)	2,35 (1,85)
RR2	1,86 (1,31)	1,80 (1,20)	2,23 (1,69)

Circle

Regla	1000	1500	2000
SM	11,86 (1,81)	10,22 (1,60)	9,79 (1,01)
MR1	12,06 (1,81)	10,41 (1,55)	10,00 (0,97)
MR2	11,46 (2,04)	9,99 (2,23)	10,16 (2,69)
RR1	11,87 (1,85)	10,17 (1,51)	9,84 (1,08)
RR2	11,45 (1,98)	9,60 (1,64)	9,20 (1,09)

Tabla 4.16: Promedio de Error y Desviación Típica para cada regla y tamaño.

diferencias significativas desaparecen. En la Tabla 4.17 se muestran aquellas que se mantienen a un nivel de significancia $\alpha < 0{,}05$. NS significa que no hay diferencias significativas entre esas reglas y el símbolo "$>$" aquí se usa en el sentido de "es peor que". La tabla muestra que las diferencias entre la regla sin memoria, **SM**, y las demás reglas, que usan la memoria, es mucho más clara según la complejidad teórica de las instancias y el tamaño aumenta.

Los resultados también muestran que la información almacenada en la memoria es mucho más útil al combinar varias reglas reglas. Si se mira en la Tabla 4.17 se ve que siempre la reglas **MR2** y **RR2** se comportan mejor que la reglas **MR1** y **RR1** respectivamente. La regla **MR2** combina aspectos que son independientes de los agentes, como lo hace la regla **MR1**, con aspectos que son intrínsecos al agente como es actuar sobre algún parámetro que pueda cambiar dinámicamente su comportamiento de búsqueda, y mejor aún, como en la regla **RR2**, si a esto se le añade un retraso en la activación del mecanismo de la memoria.

Esto indica que las estrategias cooperativas junto al uso de memoria y reglas difusas tienen sentido ante instancias difíciles, sobre todo cuando estas reglas se aplican dejando libre al agente resolvedor durante un cierto tiempo para que realice la búsqueda sin coordinación.

Tamaño = 1000

NC	WC	SC	$Circle$
$MR1 > MR2$	NS	$MR2 > SM$	$MR1 > MR2$
$SM > MR2$		$MR2 > RR1$	$MR1 > RR2$
$RR1 > MR2$		$MR2 > RR2$	$SM > MR2$
			$RR1 > MR2$
			$RR1 > RR2$

Tamaño = 1500

NC	WC	SC	$Circle$
NS	$MR1 > RR2$	$MR1 > RR2$	$MR1 > MR2$
	$MR2 > RR2$	$MR2 > SM$	$MR1 > RR2$
		$MR2 > RR1$	$SM > MR2$
		$MR2 > RR2$	$MR2 > RR2$
			$RR1 > RR2$

Tamaño = 2000

NC	WC	SC	$Circle$
NS	NS	$MR1 > RR2$	$MR1 > SM$
		$MR2 > SM$	$MR1 > RR2$
		$MR2 > RR1$	$SM > RR2$
		$MR2 > RR2$	$RR1 > RR2$
			$MR2 > RR2$

Tabla 4.17: Comparativa de Reglas según test no paramétrico U de Mann - Whitney.

4.4. Conclusiones

En este capítulo hemos puesto a prueba la estrategia presentada en nuestro trabajo. Para ello hemos usado el problema de la mochila (Knapsack Problem, KP). Es un problema NP muy bien conocido, del cual es posible graduar bien la dificultad de las instancias. Esto nos ha permitido analizar la influencia de dicha dificultad en el ajuste de los distintos factores de nuestra estrategia.

Se muestran los resultados que se obtienen en una implementación donde se simula el paralelismo a través de un esquema de tiempo compartido. En estos se analizó los beneficios de la coordinación, la influencia del uso de agentes resolvedores con diferentes comportamientos y la afectación en los resultados al variar el número de estos agentes. Los resultados obtenidos en los experimentos muestran los beneficios de la estrategia de coordinación propuesta, con la que se obtienen menores valores promedio de *Error* para todas las instancias y tamaños del problema.

Se ve que los mejores resultados se obtienen siempre bajo el esquema mixto donde se combinan algoritmos con diferentes comportamientos de búsqueda demostrando que una adecuada definición de los agentes donde tengan roles equilibrados entre intensificación y diversificación puede mejorar aún mas los resultados.

Si se desea resolver instancias muy duras entonces se hacen necesarios mayores recursos computacionales y se justifica asumir el costo de diseño y programación de una implementación paralela real que se ejecute sobre un cluster de computadores.

También se muestra cuanto sentido tiene el uso de la memoria combinado con reglas difusas dentro de las estrategias cooperativas para obtener soluciones de alta calidad ante instancias muy difíciles sobre todo si se combinan reglas independientes y dependientes de los agentes y se

aplican con cierto retraso de tiempo, ya que se obtiene un mejor equilibrio entre la exploración que se logra en mayor medida al inicio de la ejecución y una mayor explotación que se alcanza luego al encaminar el comportamiento del proceso de optimización de manera más directa hacia el refinamiento de las buenas soluciones que se van obteniendo.

Partiendo de todos estos resultados se concluye que mecanismos de coordinación modelados gracias a las técnicas de la lógica difusa y basados en el retraso del uso de la memoria mejoran la potencia de búsqueda de la estrategia cooperativa.

Capítulo 5

Problema de la P-Mediana

Como dijimos en el capítulo anterior, hemos escogido problemas bien conocidos, con instancias con dificultad conocida y medible. El Problema de la p-mediana responde también a estas características. El objetivo principal es verificar la utilidad de la estrategia y en especial sus diferentes mecanismos de coordinación modelados en su base de reglas difusas en la solución de problemas de optimización manteniendo un nivel de generalidad en su implementación. Se explica brevemente la formulación matemática del problema y las instancias a usarse. Para comprobar el comportamiento de la estrategia se harán dos grupos de experimentos: uno preliminar ante instancias pequeñas del problema para probar y ajustar la implementación y otro avanzado ante instancias lo suficientemente grandes que permiten los recursos empleados como comprobación de la validez de la estrategia. Al final se presentarán las conclusiones del capítulo.

5.1. Formulación del problema

Un problema de localización se puede describir de la siguiente manera. Considere un número de clientes distribuidos espacialmente en un área geográfica que demandan un cierto servicio o producto, y que esta demanda debe ser cubierta por una o varias instalaciones. Las instalaciones pueden operar en un ambiente de cooperación o competencia en dependencia del tipo de servicio requerido por los clientes. El proceso de decisión es intentar resolver dónde ubicar las instalaciones en el área teniendo en cuenta los requerimientos de los clientes y las restricciones geográficas.

Aunque se acredita a Euclides y Pitágoras como los primeros que construyeron los modelos geométricos de la distancia, el primero que planteó formalmente un problema de localización y sugirió una solución fue el Emperador Constantino en base a las posiciones discretas disponibles para la localización de las legiones romanas.

En este tipo de problema se identifican 3 elementos:

- las instalaciones, son un conjunto de objetos que serán localizados para brindar un servicio o producto.
- las localizaciones, forman un conjunto de posibles puntos para situar las instalaciones.
- los clientes, son los usuarios de las instalaciones que demandan ciertos servicios o productos.

Un problema clásico de localización es el problema de la p-mediana, *PM*, donde dado un número total de instalaciones o medianas que van a ser abiertas, hay que decidir en qué puntos éstas deben ser abiertas y asignar cada punto de demanda a una mediana abierta, de forma tal que la distancia total que se atraviesa para cubrir toda la demanda sea mínima.

Para ello, considere un conjunto L de m localizaciones potenciales para p instalaciones o medianas y un conjunto U de localizaciones para n usuarios. Considere una matriz D, de tamaño $n \times m$, que contiene las distancias o costos para satisfacer la demanda desde la medianas j hasta el usuario localizado en i, para todos los $j \in L$ e $i \in U$.

El problema de la p-mediana intenta localizar simultáneamente las p medianas a fin de minimizar el costo total para satisfacer las demandas de los usuarios, estando cada uno de ellos suministrado por la mediana abierta mas cercana. Matemáticamente el objetivo es seleccionar p columnas de D, tal que la suma de los coeficientes mínimos en cada línea dentro de esas columnas sea la menor posible.

$$\text{mín} \sum_{i \in U} \underset{j \in J}{\text{mín}}\, d_{ij}$$

donde,
$J \subseteq L$
$|J| = p$,

En Investigación de Operaciones se definen dos conjuntos de variables de decisión

y_j : variables de decisión para las instalaciones, donde

$$y_j = \begin{cases} 1 & \text{si se ubica la mediana en la localización } j \in L \\ 0 & \text{en otro caso} \end{cases} \tag{5.1}$$

x_{ij} : variables de asignación, donde

$$x_{ij} = \begin{cases} 1 & \text{si el cliente } i \text{ es servido desde una mediana} \\ & \quad \text{localizada en } j \in L \\ 0 & \text{en otro caso} \end{cases} \tag{5.2}$$

asegurando con ello que todo punto de demanda va a quedar asignado a una mediana.

La formulación matemática del problema para la programación entera es la siguiente:

$$
\begin{aligned}
Min \quad & \sum_{i=1}^{m} \sum_{j=1}^{n} d_{ij} x_{ij} & \\
s.a. \quad & \sum_{i=1}^{n} x_{ij} = 1 & i = 1, \ldots, m \\
& x_{ij} \leq y_j & i = 1, \ldots, m \\
& & j = 1, \ldots, n \\
& \sum_{j=1}^{n} y_j = P & \\
& y_j \in \{0, 1\} & j = 1, \ldots, n \\
& x_{ij} \in \{0, 1\} & i = 1, \ldots, m \\
& & j = 1, \ldots, n
\end{aligned}
$$

Desde los trabajos de Hakimi [112, 113, 114] sobre este problema, quién demostró la posibilidad de localizar óptimamente por lo menos p facilidades en los nodos de una red, se han desarrollado varios métodos para resolverlo.

Entre los algoritmos exactos bien conocidos están los de Beasley [115] y los de Hanjoul y Peeters [116]. Una amplia referencia sobre estos algoritmos y problemas relacionados puede encontrarse en Cornuejols et al. [117], Brandeau y Chiu [118], Mirchandani y Francis [119], Drezner [120], y Labbé y Louveaux [121].

Entre las heurísticas clásicas desarrolladas están *Greedy* de Kuehn y Hamburger [122], *Alternate* de Maranzana [123], con su procedimiento de mejora y modificación de cluster, e *Interchange* de Teitz y Bart [124], quienes crearon un procedimiento de sustitución de vértice. Luego aparecen algunos híbridos de estas heurísticas tales como, *GreedyG* [125], donde se mezcla un procedimiento *Alternate* dentro de *Greedy*. Pizzolato presentó una combinación de *Alternate* e *Interchange* [126]. Moreno et al. [127] comparan una variante de *Greedy*, frente a *Greedy+Alternate* y *Multi-arranque Alternate*. Otros procedimientos híbridos han presentado también Voß [128] y Hansen y Mladenovic [129] comparándolos con diferentes métodos.

Otro tipo de heurísticas basadas en la relajación dual de la formulación de programación entera de PM fueron propuestas por Galvao [130] y Captivo [125]. Beasley [131] usa métodos basados en la relajación Lagrangiana. En esta línea Avella et al. [132] presentan algoritmos exactos que manejan instancias de hasta 3795 nodos.

Algunas variantes de la Busqueda Tabú se han propuesto para resolver el problema, tales como: Mladenovic et al. [133], Voß [128] y Rolland et al. [134]. Taillard [135] aplica principios de descomposición dentro de un marco de Tabú y obtiene buenos resultados con instancias de 3038 nodos y 500 medianas. Rosing y Hodgson [136] con la heurística Concentration también logran resultados satisfactorios en instancias de tamaño moderado. Desde los trabajos de Hansen y Mladenovic [129] también se aplica la VNS (Variable Neighborhood Search)y sus variantes a la PM con buenos resultados. Así Hansen, Mladenovic y Pérez-Brito [137] con la VNDS (Variable Neighborhood Descent Search) logran excelentes resultados sobre 1400, 3038 y 5934 nodos. García-López et al. [138] usan varias estrategias paralelas de la VNS en instancias de 1400 nodos. Crainic et al. [76] proponen un método cooperativo multi-búsqueda para la VNS basado en un mecanismo de memoria central en instancias que van de 1400 to 11948 nodos y 10 a 1000 medianas.

Una propuesta interesante es la de Canós et al. [139, 140] en la cual aplicando técnicas de la Soft-Computing presentan una versión difusa del problema de la p-mediana. Basado en estos trabajos Verdegay y Kutangila [141, 142] formulan un modelo del problema borroso de la p-mediana para los grafos borrosos y lo extienden a los problemas de la p-mediana capacitada y no capacitada.

5.1.1. Instancias del problema

En el presente trabajo para comprobar el desempeño de la estrategia ante el problema de la p-mediana se usarán las instancias de 100 nodos tomadas del conjunto de Galvao [143, 144] e instancias de 1400, 3039 y 5934 nodos y diferentes valores de sus medianas tomadas de la biblioteca TSPLIB [145]. Estas últimas instancias usadas originalmente en el contexto del problema del viajante de comercio, y tal como se ha hecho en otros trabajos [76, 138, 137] al usarlas en el problema de la p-mediana, consideran a cada nodo tanto un cliente a ser servido como una potencial facilidad y las distancias son euclidianas.

Los experimentos se realizarán en un cluster de ordenadores bajo una implementación paralela usando el paquete de software PVM (Parallel Virtual Machine), cuya configuración ya fue expuesta en el capítulo anterior y en la cual cada agente resolvedor es creado como un clon de FANS en una tarea del PVM bajo una concurrencia real. Desde el punto de vista de la implementación los cambios que se hacen aquí a la estrategia respecto a la implementación bajo paralelismo real anterior son sólo aquellos referidos a la resolución del problema y a la configuración de los datos que usan la solución, que por supuesto son diferentes en este caso. Lo demás, incluyendo el tamaño de los vectores donde se almacena el historial de los agentes, se mantiene igual.

A diferencia del capítulo anterior donde se hicieron varios juegos de experimentos para estudiar el funcionamiento de la estrategia desde varios puntos de vista, aquí sólo se analizará el rendimiento de la estrategia acorde a los diferentes mecanismos de coordinación que brinda la Base de Reglas Difusas con cada una de sus reglas.

Los experimentos se agrupan en dos categorías:

- Experimentos preliminares.
- Experimentos avanzados.

Los experimentos preliminares se hacen para probar la implementación realizada y la validez de la estrategia ante este problema, por ello se toma una instancia pequeña en la que debe resultar fácil obtener las soluciones en poco tiempo.

Luego, en un segundo juego de experimentos, se amplia el conjunto de instancias de prueba con tamaños medios y grandes del problema.

En todos los experimentos la estrategia usará seis agentes resolvedores, clones del algoritmo FANS, en un esquema de trabajo independiente (*indep*) por un lado, y por otro en un esquema de trabajo cooperativo usando cada una de las diferentes reglas de la Base de Reglas Difusas (FRB). Los agentes resolvedores se configuran todos iguales y con comportamiento Greedy para el esquema independiente, y con una configuración Mixta donde cada agente tiene comportamientos de búsqueda diferentes para el esquema cooperativo, según se muestra en la Tabla 5.1

En cada caso se harán 10 ejecuciones de la estrategia. El rendimiento en términos de la calidad

Tabla 5.1: Esquemas de Trabajo

Esquemas	*Agente*1	*Agente*2	*Agente*3	*Agente*4	*Agente*5	*Agente*6
Indep	0.95	0.95	0.95	0.95	0.95	0.95
Coop	0.45	0.90	0.75	0.80	0.60	1.00

de la solución se mide en base a un % de *Error* que se calcula teniendo en cuenta el mejor valor aparecido en la literatura [76] para cada tipo de instancia y número de mediana:

$$Error = 100 * \frac{Optimo - Costo\,Obtenido}{Optimo}$$

Se fija un tiempo máximo de CPU para la búsqueda de las soluciones en dependencia del tamaño de la instancia de 5 segundos para 100 nodos, 60 segundos para 1400 nodos, 120 segundos para 3038 nodos y 180 segundos para 5934 nodos. Luego se prueba la estrategia aumentando el tiempo de ejecución a 250 segundos para el caso de la mediana $p = 40$ en la instancia de 1400 nodos.

5.2. Experimentos preliminares

El objetivo de este experimento es comprobar la validez de la estrategia ante el problema de la p-mediana; para ello, se usa una instancia de 100 nodos escogida del conjunto de Galvao [143, 144] con medianas que están en un rango de 5 a 40, con intervalos de 5 entre ellas y los óptimos reportados. Se comparan los resultados obtenidos en el experimento con estos óptimos calculando el Error.

Tabla 5.2: Media del Error y Desviación Típica, $m = 100$ nodos,

p	**Indep**	**SM**	**MR1**	**MR2**	**RR1**	**RR2**
5	0,00 (0,00)	0,00 (0,00)	0,00 (0,00)	0,00 (0,00)	0,00 (0,00)	0,00 (0,00)
10	0,73 (0,41)	0,39 (0,54)	0,28 (0,25)	0,50 (0,47)	0,46 (0,44)	0,56 (0,46)
15	0,23 (0,25)	0,39 (0,39)	0,29 (0,22)	0,47 (0,37)	0,30 (0,24)	0,50 (0,35)
20	0,04 (0,07)	0,08 (0,08)	0,14 (0,10)	0,16 (0,13)	0,09 (0,11)	0,19 (0,13)
25	0,13 (0,09)	0,05 (0,07)	0,13 (0,07)	0,09 (0,06)	0,04 (0,07)	0,09 (0,07)
30	0,21 (0,08)	0,09 (0,11)	0,17 (0,08)	0,16 (0,09)	0,04 (0,08)	0,05 (0,09)
35	0,24 (0,11)	0,09 (0,14)	0,11 (0,10)	0,15 (0,10)	0,12 (0,14)	0,10 (0,12)
40	0,21 (0,11)	0,09 (0,11)	0,15 (0,10)	0,20 (0,08)	0,03 (0,07)	0,05 (0,09)

En la Tabla 5.2 se muestran los valores medios del Error y la Desviación Típica para cada valor de mediana obtenidos por la estrategia bajo un esquema independiente (**Indep**) y bajo un esquema cooperativo usando las 5 reglas de la Base de Reglas Difusas, FRB (el resto de las columnas), todos para un tiempo $t_{max} = 5$ segundos.

Como se observa para una instancia tan pequeña del problema se obtiene prácticamente el valor óptimo para los diferentes valores de mediana. También los valores de la desviación estándar son bajos. Todo esto demuestra la robustez de la implementación de la estrategia ante este tipo de problema.

El caso mas difícil de resolver resultó con la mediana $p = 10$, obteniéndose los valores de Error más altos, tanto en el esquema independiente como en la mayoría de las reglas del esquema cooperativo. De todas, es la regla **MR1** la que mejor se comporta aquí; o sea, aquella donde se calcula la "calidad" de la solución reportada, en términos del historial almacenado y se envían directivas a los agentes resolvedores que funcionan "mal" para "moverlos" en el espacio de búsqueda, asignándole una nueva solución que es la mejor vista hasta ese momento por el Coordinador.

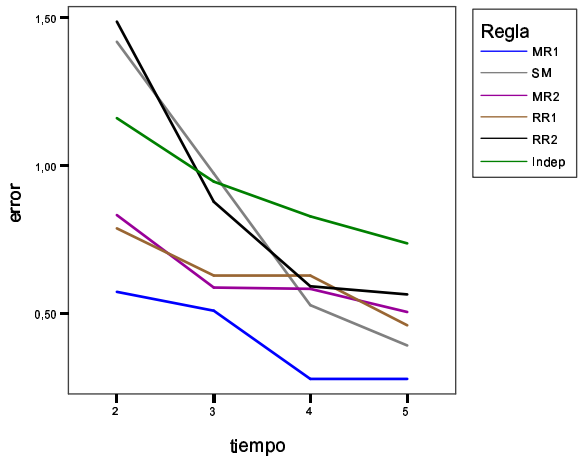


Figura 5.1: Media de Error vs. Tiempo, $m = 100$ y $p = 10$

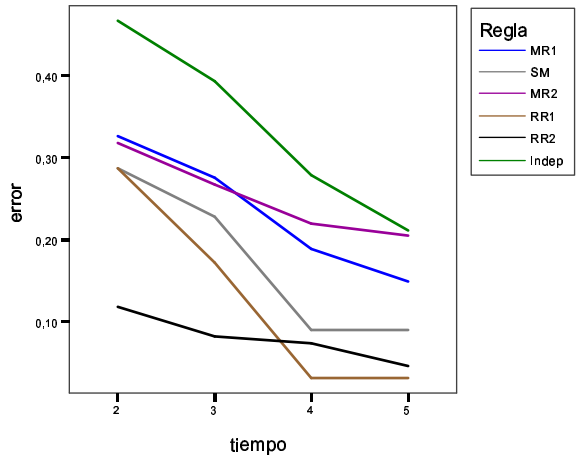


Figura 5.2: Media de Error vs. Tiempo, $m = 100$ y $p = 40$

A continuación se muestran los gráficos 5.1 y 5.2 donde se ve el comportamiento del Error vs. Tiempo para las medianas $p = 10$ y $p = 40$. A estos se les quitó la muestra para $t = 0$ con el objetivo de reajustar la escala y poder mostrar con claridad las diferencias entre las reglas, ya que para este valor de tiempo los valores de Error son altos comparados con los demás.

Para $p = 10$ se ve cuan positivo resulta el uso de un esquema cooperativo y el funcionamiento de la regla **MR1**, tal como se explicó con los datos de la tabla anterior. Para $p = 40$ se corrobora

con mas claridad lo anterior y lo interesante que resulta ver como las reglas que retrasan el uso de la memoria, **RR1** y **RR2**, son las que obtienen los mejores resultados. También se ve como la regla **SM**, en la que no se usa memoria, funciona mejor que las reglas **MR1** y **MR2** que si hacen uso de esta desde el inicio.

Todo esto puede interpretarse que es mejor hacer uso de una mayor exploración al inicio del proceso de búsqueda y luego de cierto tiempo priorizar la explotación. Al no hacerse uso de la memoria, la estrategia de manera global tendrá mayor diversificación que la intensificación en la búsqueda y también hará un uso más activo del tiempo exclusivo del CPU dedicado a la búsqueda. Luego, al usar el historial almacenado para controlar la búsqueda se obtiene una mayor intensificación.

Tabla 5.3: Número casos con $Error = 0$, $m = 100$ nodos

p	**Indep**	**SM**	**MR1**	**MR2**	**RR1**	**RR2**
5	10	10	10	10	10	10
10	.	5	3	2	1	2
15	4	1	.	.	1	.
20	6	4	2	3	5	2
25	2	6	.	1	6	3
30	.	5	.	1	7	7
35	.	6	3	.	4	5
40	.	5	2	.	8	7

En la Tabla 5.3 se muestran los veces en que se obtiene un Error igual a cero para cada valor de mediana en las ejecuciones del experimento. Como se ve esto ocurre en mayor grado para el esquema cooperativo respecto al independiente. Respecto a las reglas los mejores resultados se obtienen con las reglas **SM** (sin memoria), **RR1** (regla **MR1** con retraso), y **RR2** (regla **MR2** con retraso). Revisando la mediana $p = 40$, la mas difícil de resolver, se nota que son las reglas con retraso **RR1** y **RR2** superan a las demás. Resumiendo, se alcanza el óptimo con mas frecuencia usando un esquema cooperativo y en las instancias más duras se logra retrasando el uso de la memoria durante cierto tiempo, ya sea a través de la regla **MR1** o de la **MR2**.

5.3. Experimentos avanzados

Luego de comprobar en el experimento anterior que la estrategia propuesta es capaz de resolver este tipo de problema se pasa entonces a probarla ante instancias de más envergadura. Para ello se usarán algunas instancias tomadas del TSPLIB [145].

Se harán experimentos ante las siguientes instancias, con algunas medianas seleccionadas al azar y 10 ejecuciones para cada mediana, el tiempo de ejecución de cada iteración aparece al lado:

- fl1400, con $m = 1400$, $p = 20, 40, 60, 80$ y $t_{max} = 60$ segundos.
- pcb3038, con $m = 3038$, $p = 100, 200, 300, 400$ y $t_{max} = 120$ segundos.
- rl5934, con $m = 5934$, $p = 100, 200, 300, 400$ y $t_{max} = 180$ segundos.

Luego, en un último experimento se toma la instancia fl1400, con la mediana $p = 40$ y se le

da un tiempo mucho mayor de ejecución, $t = 250$ segundos, de forma que pueda verse mejor la actuación de las diferentes reglas de coordinación.

5.3.1. Instancia fl1400

En la Tabla 5.4 se muestran los valores medios del Error y la Desviación Típica, y en la Tabla 5.5 los valores mínimos del Error para cada mediana obtenidos por la estrategia bajo un esquema independiente (**Indep**) y bajo el esquema cooperativo usando cada una de las 5 reglas de la FRB (el resto de las columnas, **SM, MR1, MR2, RR1, RR2**).

Tabla 5.4: Media del Error y Desviación Típica, $m = 1400$ nodos,

p	Indep	SM	MR1	MR2	RR1	RR2
20	3,86 (1,43)	2,43 (2,02)	2,22 (1,19)	2,26 (1,34)	3,09 (2,64)	2,17 (2,20)
40	9,34 (0,30)	3,78 (0,81)	3,36 (1,40)	3,57 (1,47)	3,27 (1,00)	2,82 (0,82)
60	9,29 (1,05)	7,22 (2,37)	5,06 (2,08)	5,86 (2,39)	7,72 (2,19)	7,20 (1,81)
80	7,28 (1,13)	11,11 (2,76)	7,75 (1,93)	7,33 (1,38)	9,64 (1,72)	10,79 (3,23)

Como se muestra en la Tabla 5.4 los mejores valores medios del Error se obtienen en casi todos los casos bajo esquemas cooperativos con aquellas reglas que hacen un uso de la historia precedente (**MR1** y **MR2**). Para el caso de la mediana en esta instancia que resulta mas difícil de resolver, $p = 40$, es la regla **MR2** la que obtiene los mejores resultados (esta regla va ajustando los valores de la valoración difusa λ de los agentes resolvedores acorde a un historial de su comportamiento).

Tabla 5.5: Valores Mínimos de Error, $m = 1400$ nodos

p	Indep	SM	MR1	MR2	RR1	RR2
20	1,43	0,54	0,35	0,50	0,26	0,33
40	9,01	2,59	1,49	1,73	1,73	2,02
60	8,15	3,86	1,97	2,84	5,07	4,97
80	4,63	7,38	3,16	5,34	6,84	5,40

Los mejores valores mínimos, tal como se ve en la Tabla 5.5 se obtienen con el esquema cooperativo usando las reglas **RR1** para $p = \{20, 40\}$ y **MR1** para $p = \{60, 80\}$.

De manera general en este caso, tanto para los valores medios como mínimos, siempre son las reglas que hacen uso de memoria, al tener en cuenta el historial precedente del comportamiento de los agentes resolvedores, las que obtienen los mejores resultados.

En la Figura 5.3 se presenta el comportamiento dinámico de los diferentes esquemas de la estrategia para una de las medianas, en este caso $p = 40$, mostrando la evolución de los valores mínimos del Error durante el tiempo total de ejecución.

El gráfico muestra claramente como el esquema independiente (**Indep**) se estanca en un mínimo local y no mejora casi nada a partir de $t = 20$ segundos hasta el final de la ejecución. También se destaca como las reglas que hacen uso de la memoria, ya sea con retraso o no en su aplicación, obtienen mejores resultados que la regla **SM**, que no hace uso de la memoria.

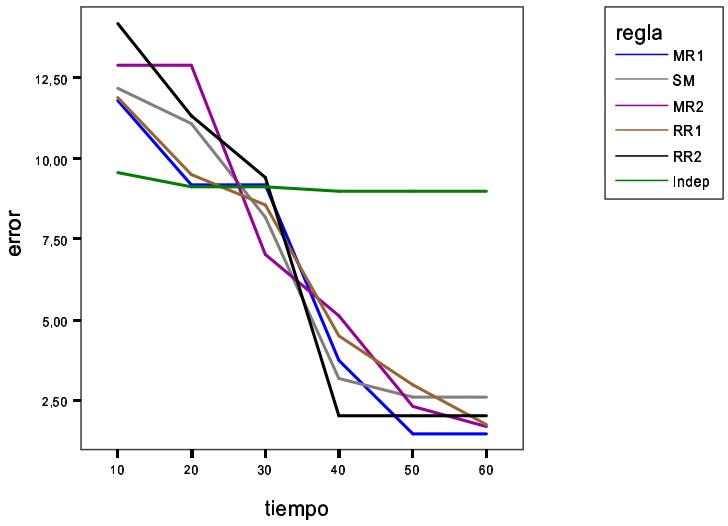


Figura 5.3: Valor mínimo Error vs. Tiempo, $m = 1400$ y $p = 40$

5.3.2. Instancia pcb3038

En la Tabla 5.6 se muestran los valores medios del Error y la Desviación Típica, y en la Tabla 5.7 los valores mínimos del Error para cada mediana obtenidos por la estrategia bajo el esquema independiente (**Indep**) y bajo el esquema cooperativo usando las 5 reglas de la FRB (**SM, MR1, MR2, RR1, RR2**).

Tabla 5.6: Media del Error y Desviación Típica, $m = 3038$ nodos

p	Indep	SM	MR1	MR2	RR1	RR2
100	7,33 (0,45)	6,55 (0,69)	5,47 (0,66)	5,20 (0,60)	6,60 (0,60)	6,28 (0,63)
200	10,10 (0,55)	8,42 (0,92)	6,74 (0,45)	6,68 (0,64)	8,35 (0,98)	8,17 (1,32)
300	12,65 (0,66)	9,22 (1,50)	7,17 (0,65)	7,13 (0,73)	9,92 (2,08)	9,60 (1,41)
400	11,58 (2,88)	9,46 (2,56)	7,22 (0,50)	7,31 (0,76)	9,78 (2,67)	9,10 (2,28)

Tabla 5.7: Valores Mínimos de Error, $m = 3038$ nodos,

medianas	Indep	SM	MR1	MR2	RR1	RR2
100	6,56	5,60	4,45	4,32	5,62	5,24
200	8,82	7,39	5,90	5,77	6,58	6,90
300	11,52	6,44	6,42	6,58	6,85	7,08
400	7,87	7,33	6,59	6,47	6,65	6,79

En la Tabla 5.6 se observa que los mejores valores medios del Error se obtienen en todos los casos bajo el esquema cooperativo con aquellas reglas que usan la memoria en su mecanismo de cooperación. En esta instancia es la regla **MR2** la de mejores resultados, excepto para $p = 400$ en que la regla **MR1** se comporta mejor. Se destaca que en algunos casos la regla sin memoria, **SM**, obtiene mejores valores de media que las reglas que usan memoria, sobre todo mejor que las

reglas de retraso. Consideramos que esto sucede debido a que el tiempo de ejecución establecido no permite a estas últimas reglas desempeñarse en todas sus posibilidades.

Los mejores valores mínimos, tal como se ve en la Tabla 5.7 se obtienen con las reglas **MR2** para $p = \{100, 200, 400\}$ y **MR1** para $p = 300$. Aquí se ve una tendencia de las reglas con memoria a superar en casi todos los casos a la regla sin memoria, indicando que las primeras son más capaces de encontrar mínimos locales de mejor "calidad".

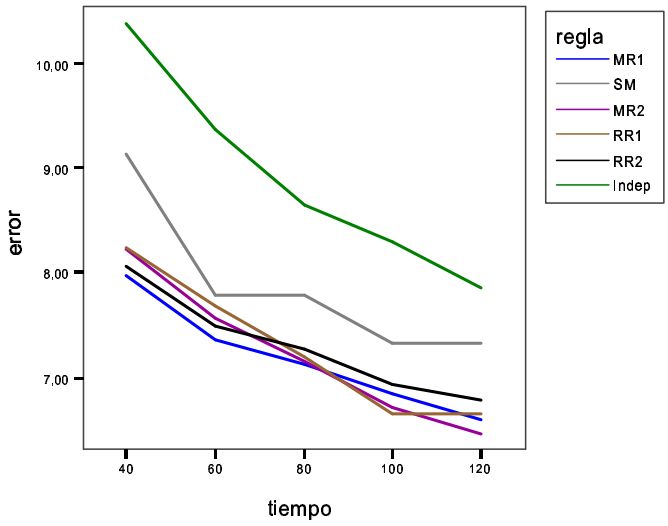


Figura 5.4: Valor mínimo Error vs. Tiempo, $m = 3038$ y $p = 400$

En la Figura 5.4 se presenta el comportamiento dinámico de los diferentes esquemas de búsqueda para la mediana $p = 400$, mostrando la evolución de los valores mínimos de Error durante el tiempo total de ejecución. Se excluyen los valores para $t = 0$ segundos para que el esquema se vea con mayor precisión.

El gráfico muestra las diferencias entre el esquema independiente (**Indep**) y todas las reglas del esquema cooperativo. También se ve como las reglas que hacen uso de la memoria van encontrando mejores valores que los obtenidos por la regla sin memoria, la cual se va estancando durante ciertos intervalos de la ejecución.

5.3.3. Instancia rl5934

Las Tablas 5.8 y 5.9 muestran los valores medios del Error y Desviación Típica y los valores mínimos del Error respectivamente para cada mediana obtenidos por la estrategia bajo el esquema independiente (**Indep**) y usando cada una de las reglas de la FRB (**SM, MR1, MR2, RR1, RR2**) del esquema cooperativo.

Como se muestra en la Tabla 5.8 los mejores valores medios se obtienen en todos los casos bajo el esquema cooperativo con las reglas que hacen un uso de la memoria durante todo el tiempo de ejecución, en este caso las reglas **MR1** y **MR2**. Los valores de la Desviación Típica son bajos, sobre todo en estas dos reglas, confirmando la robustez de su funcionamiento.

Tabla 5.8: Media del Error y Desviación Típica, $m = 5934$ nodos

p	Indep	SM	MR1	MR2	RR1	RR2
100	6,82 (0,31)	6,25 (0,87)	5,42 (0,31)	5,23 (0,48)	5,89 (0,63)	5,93 (0,60)
200	8,41 (0,78)	8,02 (0,74)	6,48 (0,33)	6,59 (0,27)	8,20 (0,66)	7,67 (0,96)
300	10,04 (1,73)	9,47 (0,99)	7,39 (0,24)	7,21 (0,20)	9,53 (1,09)	9,67 (1,16)
400	11,05 (1,82)	10,58 (2,17)	7,59 (0,33)	7,59 (0,19)	9,55 (2,17)	11,08 (2,06)

Tabla 5.9: Valores Mínimos de Error, $m = 5934$ nodos

p	Indep	SM	MR1	MR2	RR1	RR2
100	6,34	6,20	4,77	4,68	5,07	4,64
200	6,55	7,61	6,04	6,17	6,60	5,88
300	6,68	7,60	6,96	6,91	7,50	7,49
400	7,61	7,33	7,07	7,30	7,31	7,51

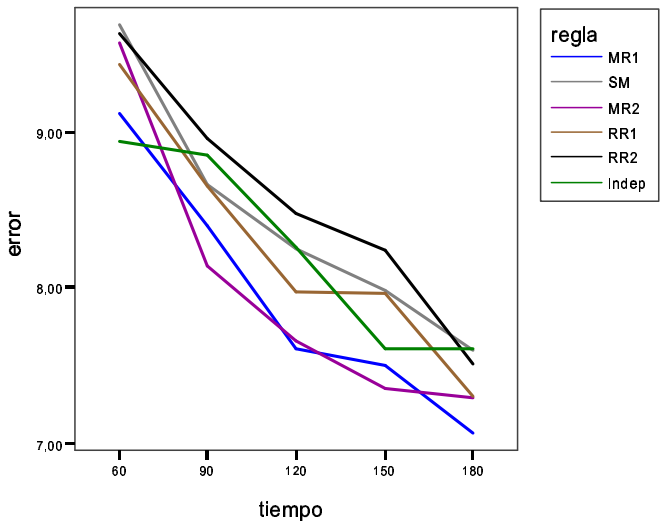


Figura 5.5: Valor mínimo Error vs. Tiempo, $m = 5934$ y $p = 400$

Para el caso de mediana que aquí resulta mas difícil de resolver, $p = 400$, siguen siendo las reglas **MR1**y **MR2** las que obtienen los mejores resultados.

Analizando la Tabla 5.9, en la cual aparecen los valores mínimos del Error, se ve que en casi todos los casos los mejores valores se obtienen con el esquema cooperativo con reglas que hacen uso de la memoria, como **RR2** para $p = \{100, 200\}$ y **MR1** para $p = 400$. Nótese que solamente para $p = 300$ es que el esquema independiente (**Indep**) logra el valor mas bajo del Error; sin embargo en la Tabla anterior de los valores medios este esquema tiene los peores valores.

En la Figura 5.5 se presenta el comportamiento dinámico de los diferentes esquemas de la estrategia para $p = 400$, el caso que aquí resulta más difícil de resolver, mostrando la evolución de los valores mínimos de Error durante el tiempo total de ejecución.

En el gráfico se destaca el buen rendimiento que se logra con casi todas las reglas que hacen uso de la memoria, ya sea con retraso o no en su aplicación, ya que son las que durante la ejecución van obteniendo mejores resultados que la regla del esquema independiente y la regla sin memoria. Se ve como la regla del esquema independiente se estanca a los 150 segundos y como a partir de este mismo instante de tiempo las reglas que usan la memoria mejoran los resultados de las que no usan la memoria.

5.3.4. Análisis estadístico de los datos

Al realizar las pruebas no paramétricas de Kruskal-Wallis con $\alpha \leq 0{,}008$ considerando el conjunto total de datos se obtiene que las diferencias son estadísticamente significativas en todos los casos.

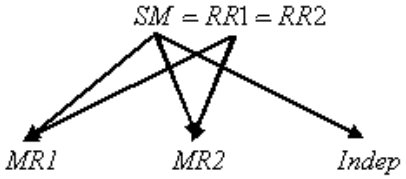


Figura 5.6: Significaciones U de Mann-Whitney, $m = 100$ nodos

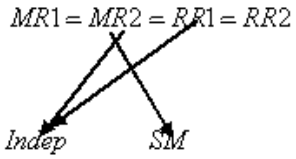


Figura 5.7: Significaciones U de Mann-Whitney, $m = 1400$ nodos

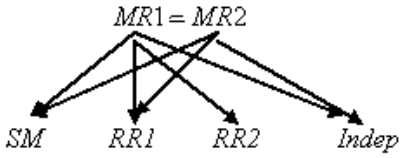


Figura 5.8: Significaciones U de Mann-Whitney, $m = 3039$ nodos

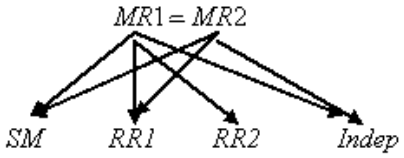


Figura 5.9: Significaciones U de Mann-Whitney, $m = 5934$ nodos

Haciendo un mismo análisis teniendo en cuenta el tipo y tamaño de las instancias para el tiempo máximo de ejecución en las Figuras 5.6, 5.7, 5.8 y 5.9 se resumen los resultados de las

pruebas no paramétricas U de Mann-Whitney para $\alpha \leq 0{,}05$ mostrando las reglas en las cuales no hay diferencias significativas entre ellas y aquellas reglas en que esto si ocurre. No hay diferencias entre las reglas que aparecen arriba y si hay diferencias entre las reglas de arriba respecto a las que aparecen debajo. Las flechas salen de aquellas reglas en las que alcanzan los mejores resultados. Como se observa en todas las instancias las reglas que usan la memoria tienen mejor desempeño que aquellas que no lo hacen. También se observa que las reglas **MR1** y **MR2** se comportan de manera general mejor que las reglas **RR1** y **RR2** que retrasan el uso de la memoria, algo que creemos es debido a que estas instancias necesitan más tiempo de ejecución para que estas reglas tengan mejores posibilidades de actuar.

5.3.5. Instancia fl1400, con mayor tiempo de ejecución

En este experimento se deja un mayor tiempo de ejecución a la estrategia de manera que pueda verse mejor la actuación de las diferentes reglas de coordinación, en especial aquellas que retrasan el uso de la memoria. Para ello, se toma la instancia de 1400 nodos, con la mediana $p = 40$, y se le da un tiempo $t_{max} = 250$ segundos. Se hacen 10 ejecuciones a cada una de las reglas. El tiempo t_{max} asignado es comparable al usado por Crainic et al. [76] en uno de los últimos trabajos publicados acerca del problema de la p-mediana y en el que también usan una estrategia cooperativa para resolverlo.

Tabla 5.10: Media del Error y Desviación Típica, $m = 1400$ nodos, $t = 250$ seg,

t	Indep	SM	MR1	MR2	RR1	RR2
0	32,30 (5,12)	33,72 (4,61)	37,48 (4,05)	35,49 (5,36)	39,97 (7,06)	34,85 (4,63)
50	9,06 (0,93)	3,78 (0,81)	3,20 (1,07)	3,08 (1,24)	3,15 (1,01)	2,95 (0,78)
100	8,31 (0,66)	2,15 (0,82)	2,09 (0,72)	1,68 (0,64)	1,84 (0,57)	1,67 (0,87)
150	8,23 (0,69)	1,76 (0,78)	1,23 (0,62)	1,51 (0,67)	1,51 (0,47)	1,03 (0,62)
200	7,65 (0,37)	1,42 (0,45)	1,23 (0,62)	1,08 (0,49)	1,25 (0,58)	0,82 (0,44)
250	7,60 (0,34)	1,33 (0,44)	1,05 (0,54)	1,08 (0,49)	1,01 (0,57)	0,82 (0,44)

En la Tabla 5.10 se muestran los valores medios del Error y la Desviación Típica para esta mediana obtenidos por la estrategia tanto bajo el esquema independiente (**Indep**), como bajo un esquema cooperativo usando cada una de las reglas de la FRB (**SM, MR1, MR2, RR1, RR2**).

Como se observa hay una gran diferencia entre los resultados logrados por el esquema cooperativo y los obtenidos por el esquema independiente, siendo peores estos últimos. Nótese que el independiente tiene los valores mas bajos de Desviación Típica, esto debido a la poca diversificación que se logra bajo este esquema al usar sólo agentes con características *greedy*. Los mejores valores se logran con una regla que hace uso de la memoria con retraso, en este caso la **RR2**. Todo esto puede verse mejor en el gráfico 5.10 donde se muestra el comportamiento en el tiempo del Error medio de todas las reglas.

En el gráfico 5.11 para lograr mas claridad se muestran solamente las reglas del esquema cooperativo y se ve que las reglas que usan la memoria se comportan mejor que la regla sin memoria **SM**.

En la Tabla 5.11 aparecen los valores mínimos del Error, y se observa como todas las reglas del esquema cooperativo a partir de los 100 segundos de ejecución prácticamente alcanzan el valor reportado en la literatura como el mejor para esta instancia y mediana.

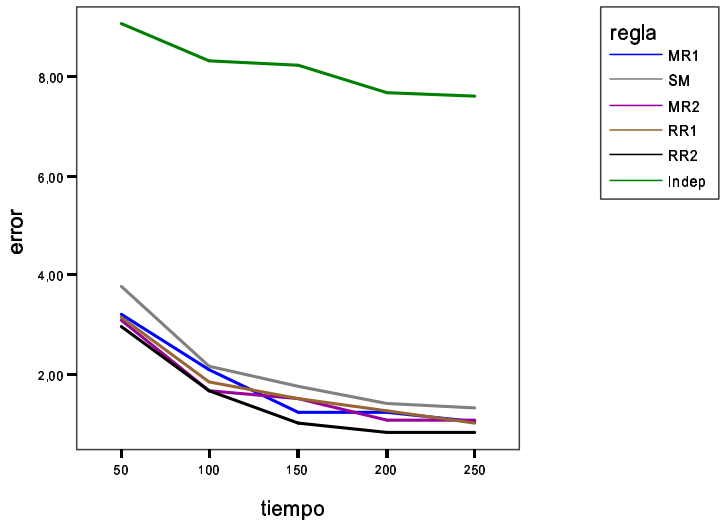


Figura 5.10: Media de Error vs. Tiempo, $m = 1400$, $p = 40$ y $t_{max} = 250$ seg

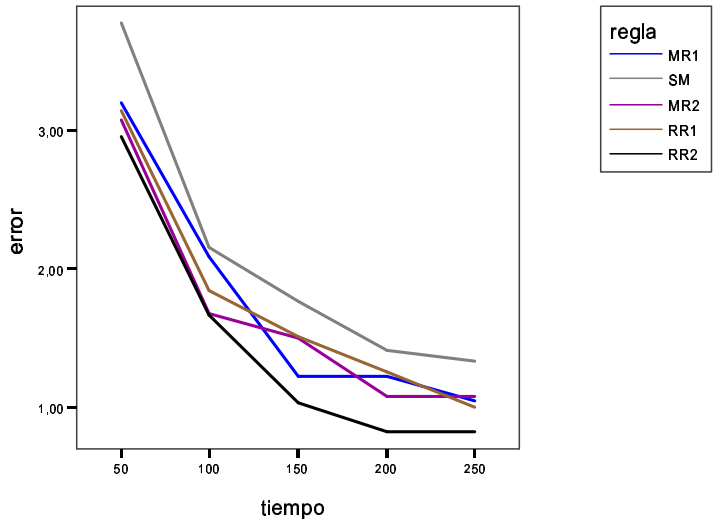


Figura 5.11: Media Error vs. Tiempo, $m = 1400$, $p = 40$ y $t_{max} = 250$ seg

El comportamiento dinámico en el tiempo de la estrategia en la obtención de sus valores mínimos se muestra en la figura 5.12. Para ver con mas precisión las diferencias entre las reglas del esquema cooperativo se tiene la figura 5.13, donde se observa que las reglas que hacen un uso de la memoria se comportan mejor que la regla sin memoria, la cual se estanca y no mejora más el resto de la ejecución. Además se ve que aquellas reglas que retrasan el uso de la memoria, **RR1** y **RR2**, obtienen los mejores valores de todas.

En la Tabla 5.12 se muestran los veces en que se obtiene un Error <= 1 en las 10 ejecuciones

Tabla 5.11: Valores Mínimos de Error, $m = 1400$, $p = 40$ y $t_{max} = 250$ seg

t	Indep	SM	MR1	MR2	RR1	RR2
0	24,60	26,22	32,33	28,45	28,99	27,52
50	7,88	2,59	1,49	1,47	1,73	2,02
100	7,44	0,78	1,10	0,67	1,00	0,43
150	7,44	0,78	0,42	0,62	0,83	0,31
200	6,87	0,78	0,42	0,32	0,32	0,31
250	6,87	0,78	0,42	0,32	0,10	0,31

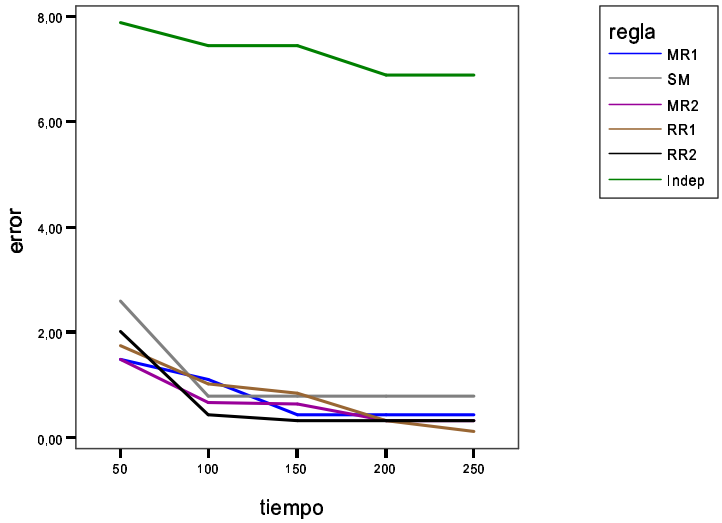


Figura 5.12: Mínimo de Error vs. Tiempo, $m = 1400$, $p = 40$ y $t_{max} = 250$ seg

Tabla 5.12: Casos con $Error <= 1$, $m = 1400$ nodos, $p = 40$, 10 ejecuciones

tiempo	SM	MR1	MR2	RR1	RR2
250	3	5	4	5	8

de este experimento. Como se ve esto no ocurre para el esquema independiente. Respecto a las reglas del esquema cooperativo se ve que aquellas reglas que usan memoria en su esquema de búsqueda se comportan mejor que la regla sin memoria. Los mejores resultados se obtienen con la regla **RR2**, al obtener 8 casos de 10 posibles. Esta es una regla que retrasa del uso de la memoria y que va ajustando el funcionamiento de búsqueda de los agentes según su comportamiento.

Es de destacar también que se obtienen resultados comparables a los logrados en el trabajo de Crainic et al., que se tomó como referencia para asignar el tiempo máximo de ejecución, y donde hacen sus experimentos en un servidor SUN Enterprise 10000 con 64-procesadores, 400 MHz re reloj y 64 gigabytes of RAM.

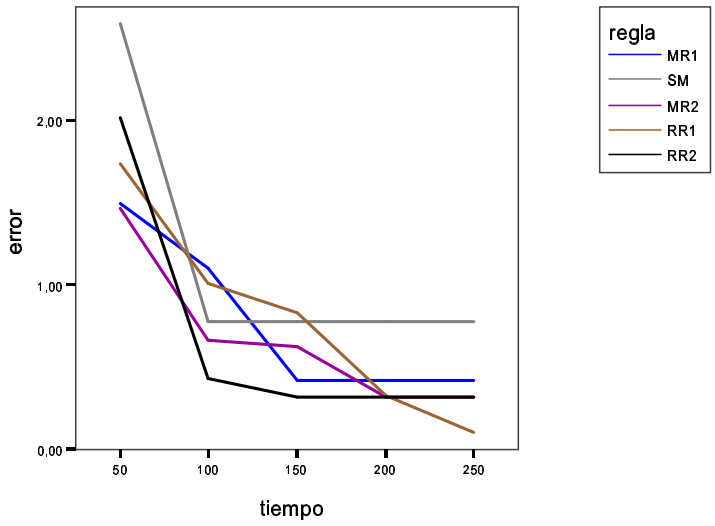


Figura 5.13: Mínimo Error vs. Tiempo, $m = 1400$, $p = 40$ y $t_{max} = 250$ seg

5.4. Conclusiones

En este capítulo se comprobó la validez y generalidad de la estrategia al probarla esta vez ante un nuevo problema, el de la P-Mediana y manteniendo prácticamente igual su implementación. Los cambios que se hacen respecto a la implementación bajo paralelismo real del capítulo anterior son sólo los referidos a la resolución del problema y a la configuración de los datos que usan la solución, que necesariamente son diferentes al problema de la mochila.

Se realizaron experimentos ante instancias de diferentes tamaños y varias medianas donde se analizó el rendimiento de la estrategia acorde a los diferentes mecanismos de coordinación que brinda la Base de Reglas Difusas del Coordinador en cada una de sus reglas, probando el uso o no de la memoria y retrasando esta cierto tiempo para su aplicación, considerando que estos aspectos son los más importantes del diseño de la estrategia.

Se confirmó el uso positivo de estas reglas difusas que junto a llevar un registro del historial del comportamiento de la búsqueda de cada agente establecen mecanismos de coordinación dentro de las estrategias cooperativas que permiten alcanzar soluciones de alta calidad ante instancias grandes y difíciles; y en especial al combinar un esquema básicamente explorador al inicio de la ejecución con una mayor explotación que se aplique con cierto retraso de tiempo. Los excelentes resultados del uso de estas reglas con retraso se vio con claridad en el último experimento realizado, al dejarle mas tiempo de ejecución a la estrategia lo que les permitió a estas reglas que pudieran funcionar con toda la potencialidad de su diseño.

Los análisis de los resultados obtenidos permiten concluir que los mecanismos de coordinación modelados gracias a técnicas de la lógica difusa y basados en especial en el uso de la memoria ya sea con retraso o no mejoran la búsqueda potencial de la estrategia cooperativa.

Capítulo 6

Conclusiones y Trabajo Futuro

Se presentó en este trabajo una estrategia cooperativa paralela basada en técnicas de la soft-computing para resolver problemas de optimización combinatoria.

La idea básica fue disponer de un conjunto de agentes resolvedores, cuya función básica es actuar como algoritmos metaheurísticos de solución de problemas de optimización combinatoria, y ejecutarlos de forma paralela y cooperativa desde un agente coordinador. El diseño de la estrategia tuvo como premisa fundamental la generalidad basada en un mínimo de conocimiento del problema a resolver, o sea que tuviera la mayor independencia posible de éste para así poder realizar con el menor coste de tiempo y esfuerzo los posibles ajustes a su implementación para poder asumir la resolución de un nuevo problema.

Cada agente resolvedor actúa de forma autónoma y se comunica solamente con el agente coordinador para enviarle las soluciones que va encontrando y para recibir de este las directivas que le indiquen cómo debe seguir actuando. El agente coordinador recibe las soluciones al problema encontradas por cada agente y siguiendo la base de reglas difusas que modelan su comportamiento crea las directivas que envía a éstos, llevando de esta manera todo el control de la estrategia.

Como algoritmo de solución para los problemas de optimización combinatoria del agente calculador se usó la metaheurística FANS, un método método de búsqueda por entornos, adaptable y difuso que basa en la evaluación cualitativa de las soluciones del espacio de búsqueda. Se eligió FANS debido a que puede considerarse una herramienta genérica de métodos de búsqueda local al brindar la posibilidad de variar su comportamiento de forma sencilla permitiendo diferentes comportamientos, con los cuales se pueden construir algoritmos con diferentes esquemas de búsqueda.

La estrategia se implementó en el lenguaje C++ y como marco de computación paralela se usó la biblioteca de dominio público PVM, que brinda un ambiente de programación concurrente, distribuido y de propósito general. Todo esto soportado en un cluster de ocho computadores.

Para dar a conocer esto se elaboró esta memoria, estructurada de la siguiente manera:

Un primer capítulo en el cual se dieron definiciones básicas relativas a la optimización combinatoria y las metaheurísticas. Se presentaron algunas de las metaheurísticas más exitosas agrupadas en métodos de trayectorias y de poblaciones para una mejor comprensión de su funcionamiento.

En el segundo capítulo se explicaron conceptos básicos acerca de la Computación Paralela presentando las arquitecturas de máquinas computadoras más usadas, los modelos de la pro-

gramación paralela, conceptos sobre algoritmos paralelos y se mostraron diferentes ambientes de programación que facilitan el trabajo de los programadores, haciendo énfasis en PVM, por ser la herramienta de software usada en la implementación de la propuesta. Se abordaron las metaheurísticas paralelas explicando conceptos, estrategias, taxonomías y se hace una revisión del estado del arte donde se muestran referencias a varias propuestas con resultados exitosos.

En el tercer capítulo se describieron los conceptos básicos de la Soft-Computing y su utilidad al campo de las metaheurísticas a través de ejemplos de su utilidad en solucionar la vaguedad implícita en la manera de solucionar muchos de los problemas de optimización donde su complejidad, dimensión y/o requerimientos computacionales para resolverlos hace que se enfoque entendiendo que es mejor satisfacer que optimizar. Se explica a FANS, describiendo los elementos básicos del algoritmo y su esquema general, destacando la utilidad de su principal componente: la valoración difusa. Luego, se aborda nuestra propuesta de estrategia mostrando los detalles de su implementación a través del pseudocódigo de sus componentes y explicando la base de reglas que definen sus diferentes esquemas de trabajo.

En los capítulos cuatro y cinco se hace la verificación experimental de la estrategia diseñada y propuesta en este trabajo, al comprobar su validez ante el problema de la Mochila y el problema de la P-Mediana. Para ello se realizaron varios juegos de experimentos ante diferentes instancias de estos problemas.

Desde los primeros experimentos ya se mostró una manera sencilla de mejorar el comportamiento de un algoritmo particular $\mathcal{A}$. Simplemente "clonando" k veces $\mathcal{A}$ e introduciendo un algún esquema de cooperación como el propuesto aquí. Incluso si solo se desea mejorar la calidad de los resultados y no la velocidad de computación no se necesita un paralelismo real, basta definir un conjunto de clones de $\mathcal{A}$, ejecutarlos en *round robin* un cierto número de iteraciones, capturar la información necesaria desde los clones, actualizar con ella algunas variables del sistema y decidir si algún algoritmo del conjunto de clones debe ser "movido" en el espacio de búsqueda.

La información que se intercambia dentro de la estrategia global juega un papel crucial, ya en la literatura [90, 146] se muestran que las metaheurísticas cooperativas paralelas pueden ser vistas como sistemas dinámicos donde su evolución puede estar determinada más por el esquema de cooperación que por el proceso de optimización. En el presente trabajo se demuestra también que una adecuada definición de los clones de $\mathcal{A}$ puede mejorar aún mas los resultados, sobretodo teniendo éstos unos roles equilibrados entre intensificación y diversificación. Se ve que los mejores resultados se obtienen siempre bajo el esquema mixto donde se combinan algoritmos con diferentes comportamientos de búsqueda.

También se justifica asumir el costo de diseño y programación de una implementación paralela real que se ejecute sobre un cluster de computadores si se desea resolver instancias grandes y duras donde se hacen necesarios mayores recursos computacionales.

Se ve cuánto sentido tiene el uso de una memoria que registre un historial de la calidad de las soluciones encontradas por cada agente, combinándola con reglas difusas dentro del mecanismo de coordinación de la estrategia para obtener soluciones de alta calidad ante instancias muy difíciles sobretodo si se aplican con cierto retraso de tiempo, ya se que obtiene un mejor equilibrio entre la exploración que se logra en mayor medida al inicio de la búsqueda y una mayor explotación que se alcanza luego al encaminar el comportamiento del proceso de optimización de manera más directa hacia el refinamiento de las buenas soluciones que se van obteniendo.

Partiendo de todos estos resultados se concluye que una estrategia paralela cooperativa for-

mada por agentes con diferentes comportamientos formando un esquema mixto de trabajo y que base sus mecanismos de coordinación en reglas difusas que hacen uso de memoria, sobretodo retrasando su uso de manera que los algoritmos actúen por sí solos, mejora la búsqueda potencial de soluciones de muy alta calidad del problema en cuestión al lograr un equilibrio global entre la exploración y la explotación.

En resumen, las principales aportaciones del trabajo han sido:

- Proponer una estrategia cooperativa paralela basada en técnicas de Soft-Computing de carácter general para resolver problemas de optimización.
- Mostrar que esta estrategia es una herramienta útil dentro de la optimización combinatoria y relevante de cara a su incorporación en un sistema de ayuda a la decisión por su capacidad para resolver problemas de diferentes tipos y tamaños sin incidir dentro de un conocimiento estricto del problema.
- Mostrar la generalidad de esta herramienta para ajustarla a diferentes problemas sin hacer grandes cambios.
- Verificar experimentalmente la validez de la estrategia propuesta ante instancias duras y grandes del problema de la mochila y el problema de la p-mediana.
- Comprobar la utilidad de usar un esquema mixto de algoritmos cooperando entre sí para enfrentar problemas duros de resolver.
- Mostrar la utilidad del uso de reglas basadas en la lógica difusa para construir diferentes estrategias de coordinación.
- Mostrar la utilidad del uso de memoria con retraso para enfrentar instancias muy duras por lograr un adecuado equilibrio entre la explotación y la exploración del espacio de búsqueda.

Aún quedan muchas cosas por investigar y experimentar y que constituyen pautas del trabajo a seguir, tales como:

- Mejorar y ampliar la base de reglas difusas, por ejemplo se pueden incorporar nuevas reglas para modelar el criterio de parada o para dividir el espacio de búsqueda.
- Usar funciones de pertenencia que adapten su definición a lo largo la ejecución mejorando el desempeño.
- Probar diferentes vías para ajustar el comportamiento de los algoritmos.
- Ampliar el banco de experimentos ante otros problemas de optimización.
- Incorporar mecanismos dinámicos que permitan la elección del tipo de heurística a usar para resolver el problema en cuestión.
- Profundizar acerca del tipo de información a ser guardada en la memoria del coordinador y su uso en el control del esquema global de búsqueda.
- Probar la estrategia con otros algoritmos diferentes a FANS como agentes resolvedores que realizan la optimización.

Bibliografía

[1] F. Glover. Future paths for integer programming and links to artificial intelligence. *Computers and Operations Researchl*, 5:533–549, 1986.

[2] I.H. Osman and G. Laporte. Metaheuristics: a bibliography. *Annals of Operations Research*, (63):513–623, 1996.

[3] S. Voss, S. Martello, I.H. Osman, and C. Roucariol, editors. *Meta-Heuristics. Advances and Trends in Local Search Paradigms for Optimization.* Kluwer Academic Publishers, 1999.

[4] B. Melián, J.A. Moreno, and J.M. Moreno. Metaheurísticas: un visión global. *Revista Iberoamericana de Inteligencia Artificial*, 19:7–28, 2003.

[5] E. Aarts and J.K. Lenstra, editors. *Local Search in Combinatorial Optimization.* John Wiley & Sons, 1997.

[6] R. Martí. *Handbook of Metaheuristics*, chapter Multistart Methods. Kluwer Academic, 2003.

[7] R. Marti and J.M. Moreno. Métodos multiarranque. *Revista Iberoamericana de Inteligencia Artificial*, 19:49–60, 2003.

[8] C. Blum and A. Roli. Metaheuristics in combinatorial optimization: Overview and conceptual comparison. *ACM Computing Surveys*, 35(3):268–308, 2003.

[9] D. Corne, M. Dorigo, and F. Glover, editors. *New Ideas in Optimization.* McGraw-Hill, 1999.

[10] B.A. Díaz and K.A. Dowsland. Diseño de heurísticas y fundamentos del recocido simulado. *Revista Iberoamericana de Inteligencia Artificial*, 19:93–102, 2003.

[11] D. Henderson, S.H. Jacobson, and A.W Johnson. *Handbook of Metaheuristics*, chapter The theory and practice of simulated annealing. Kluwer Academic, 2003.

[12] O. Martin and S.W. Otto. Combining simulated annealing with local search heuristics. *Annals of Operations Research*, 63:57–75, 1996.

[13] G. Laporte and I.H. Osman. Metaheuristics: A bibliography. *Annals of Operations Research*, 63:513 – 623, 1996.

[14] M.G.C. Resende and C.C. Ribeiro. *Handbook of Metaheuristics*, chapter Greedy Randomized Adpative Search Procedures. Kluwer Academic, 2003.

[15] M.G.C. Resende and J.L. González. Grasp: Procedimientos de búsqueda miopes aleatorizados y adaptativos. *Revista Iberoamericana de Inteligencia Artificial*, 19:61–762, 2003.

[16] T.A. Feo and M.G.C. Resende. Greedy randomized adaptive search procedures. *Journal of Global Optimization*, 6:109–133, 1995.

[17] P. Moscato. *Handbook of Applied Optimization*, chapter Memetic Algorithms. Oxford University Press, 2002.

[18] F. Glover and R. Martí. *Handbook of Metaheuristics*, chapter Scatter Search and Path Relinking: Advances and applications. Kluwer Academic, 2003.

[19] P. Festa and M.G.C. Resende. Grasp: An annotated bibliography. Technical report, AT and T Labs Research Technical Report, 2001.

[20] F. Glover and B. Melián. Búsqueda tabú. *Revista Iberoamericana de Inteligencia Artificial*, 19:29–48, 2003.

[21] F. Glover and M. Laguna, editors. *Tabu Search*. Kluwer Academic, 1997.

[22] Tabu search. `http://www.tabusearch.net`.

[23] P. Hansen and N. Mladenovic. *Handbook of Applied Optimization*, chapter Variable Neighborhood Search. Oxford University Press, 2002.

[24] P. Hansen and N. Mladenovic. *Handbook of Metaheuristics*, chapter Variable Neighbourhood Search. Kluwer Academic, 2003.

[25] P. Hansen, Mladenovic N., and J.A. Moreno. Búsqueda de entorno variable. *Revista Iberoamericana de Inteligencia Artificial*, 19:77–92, 2003.

[26] P. Hansen and N. Mladenovic. Variable neighborhood search: Principles and applications. *European Journal of Operational Research*, 130:449–467, 2001.

[27] P. Hansen and N. Mladenovic. An introduction to variable neighborhood search. In S. Voss, S. Martello, I. Osman, and C. Roucairol, editors, *Metaheuristics: Advances and Trends in Local Search Procedures for Optimization*, pages 433–458. Kluwer, 1999.

[28] O. Martin, H. Ramalhino, and T. Stützle. *Handbook of Metaheuristics*, chapter Iterated Local Search. Kluwer Academic, 2003.

[29] O. Martin, H. Ramalhino, and T. Stützle. A beginner's introduction to iterated local search. In *Metaheuristics International Conference, MIC 2001*, 2001.

[30] T. Stützle. Iterated local search for the quadratic assignment problem. Technical report, Darmstadt University of Technology, Computer Science Department, Intellectics Groupt, 1999.

[31] H.R. Lourenco, O. Martin, and T. Stützle. Iterated local search. In Fred Glover and Gary Kochenberger, editors, *Handbook on Metaheuristics*. Kluwer Academic Publishers, 2002. To appear. Preliminary version at http://www.intellektik.informatik.tu-darmstad.de/tom/pub.html.

[32] E.P.K. Tsang and C. Voudoris. *Handbook of Metaheuristics*, chapter Guided Local Search. Kluwer Academic, 2003.

[33] Guided local search. http://cswww.essex.ac.uk/CSP/gls.html.

[34] C. Voudouris. *Guided Local Search for Combinatorial Optimisation Problemss*. PhD thesis, Department of Computer Science, University of Essex, 1997.

[35] Z. Michalewicz. *Genetic Algorithms + Data Structures = Evolution Programs*. Springer Verlag, 1999.

[36] T. Bäck, D. Fogel, and Z. Michalewicz, editors. *Handbook of Evolutionary Computation*. Institute of Physics Publishing and Oxford University Press, 1997.

[37] A. Diaz, F. Glover, H. Ghaziri, J. Gonzalez, M. Laguna, P. Moscato, and F. Tseng. *Optimización Heurística y Redes Neuronales*. Ed. Paraninfo, 1996.

[38] J.H. Holland. Genetic algorithms and the optimal allocation of trials. *SIAM J. Comput.*, 2:88–105, 1973.

[39] C. Reeves. *Handbook of Metaheuristics*, chapter Guided Local Search. Kluwer Academic, 2003.

[40] F. Glover and M. Laguna, editors. *Genetic Algorithms in Search, Optimization and Machine Learning*. Addison-Wesley, 1989.

[41] C. Cotta and P. Moscato. *Handbook of Metaheuristics*, chapter A Gentle Introduction to Memetic Algorithms. Kluwer Academic, 2003.

[42] C. Cotta and P. Moscato. Una introducción a los algoritmos meméticos. *Revista Iberoamericana de Inteligencia Artificial*, 19:131–148, 2003.

[43] P. Larrañaga and J. Lozano, editors. *Estimation of Distribution Algorithms*. Kluwer Academic Publisher, 2002.

[44] P. Larrañaga, J.A. Lozano, and H. Mühlenbein. Estimation of distribution algorithms applied to combinatorial optimization problems. *Revista Iberoamericana de Inteligencia Artificial*, 19:149–168, 2003.

[45] H. Mühlenbein and G. Paaß. *Parallel Problem Solving from Nature-PPSN IV*, volume 1411 of *Lecture Notes in Computer Science*, chapter From recombination of genes to the estimation of distributions I. Binary parameters, pages 178–187. Springer Verlag, 1996.

[46] M. Laguna and R. Martí. Scatter search: Diseño básico y estrategias avanzadass. *Revista Iberoamericana de Inteligencia Artificial*, 19:123–130, 2003.

[47] M. Laguna and R. Martí, editors. *Scatter Search. Methodology and Implementations in C*. Kluwer Academic Publisher, 2003.

[48] M. Dorigo and T. Stützle. *Handbook of Metaheuristics*, chapter Ant Colony Optimization Metaheuristic. Kluwer Academic, 2003.

[49] G. Di Caro, L.M. Gambardella, and M. Dorigo. Ant algorithms for discrete optimization. *Artificial Life*, 5:137–172, 1999.

[50] B. Bullnheimer, Hartl R.F., and C. Strauss. A new rank based version of the ant system — a computational study. Technical report, University of Viena, Institute of Management Science, 1997.

[51] Ant colony optimization. http://iridia.ulb.ac.be/ mdorigo/ACO/ACO.html.

[52] R.C. Eberhart and J. Kennedy, editors. *Swarm Intelligence*. Academic Press, 2001.

[53] Particle swarm optimization. http://web.ics.purdue.edu/ hux/PSO.shtml.

[54] Particle swarm optimization. http://www.engr.iupui.edu/ shi/Coference/psopap4.html.

[55] T.G. Crainic and M. Toulouse. *Fleet Management and Logistics*, chapter Parallel Metaheuristics. Kluwer Academic, 1998.

[56] Seyed H. Roosta, editor. *Parallel Processing and Parallel Algorithms. Theory and Computation*. Springer-Verlag, 1999.

[57] S. Gill. Parallel programming. *The Computer Journal*, 1:2–10, 1958.

[58] J. Holland. A universal computer capable of executing an arbitrary number of subprograms simultaneously. In *Proceedings of East Joint Computer Conference*, pages 08–113, 1967.

[59] M.E. Conway. A multiprocessor system design. In *Proceedings of AFIPS Fall Joint Computer Conference*, pages 139–146, 1963.

[60] M. Allen and B. Wilkinson, editors. *Parallel Programming, Tecniques and Applications Using Networked Workstations and Parallel Computers*. Prentice Hallg, 1999.

[61] V. Cung, S.L. Martins, C. Ribeiro, and C. Roucairol. *Essays and Surveys in Metaheuristics*, chapter Strategies for the Parallel Implementation of Metaheuristics, pages 263–308. Kluwer Academic Publisher, 2001.

[62] M.J. Flynn. Very high speed computing systems. In *Proceeding of the IEEE, 54(12)*, pages 1901–1909, 1966.

[63] M.J. Flynn. Some computer organizations and their effectivenes. *IEEE Transactions on Computing*, 21(9):948–948, 1972.

[64] M.J. Flynn and K.W. Rudd. Parallel architectures. *ACM Computing Surveys*, 28(1):67–70, 1996.

[65] S. Dormido Canto, editor. *Procesamiento Paralelo. Teoria y Programación*. Ed. Sanz y Torres, S.L., 2003.

[66] G.M. Amdahl. Validity of single-processor approach to achieving large-scale computing capability. In *Proceedings of AFIPS Conference, Reston, VA*, pages 1901–1909, 1967.

[67] R. Butler and E. Lusk. Monitors, messages and clusters: The p4 parallel programming system. Technical report, Argonne National Laboratory, 1993.

[68] p4 system. http://www-fp.mcs.anl.gov/ lusk/p4/.

[69] J. Flower, A. Kolawa, and S. Bharadwaj. The express way to distributed processing. *Supercomputing review*, pages 54–55, 1991.

[70] N. Carriero and D. Gelernter. Linda in context. *Communications of the ACM*, 32:444–458, 1989.

[71] D. Gelernter and D. Kaminsky. Supercomputing out of recycled garbage: Preliminary experience with piranha. In *International Conference on Supercomputing, ACM Press*, pages 417–427, 1992.

[72] Message passing interface. http://www.mpi-forum.org/.

[73] A. Geist, A. Beguelin, J. Dongarra, W. Jiang, R. Manchek, and V. Sunderam. Pvm user's guide and reference manual. Technical report, Oak Ridge National Laboratory, 1994.

[74] Parallel virtual machine. http://www.netlib.org/pvm3/.

[75] T.G. Crainic and M. Toulouse. *Parallel Strategies for Metaheuristics*, volume Handbook of Metaheuristics, pages 475 – 513. Kluwer Academic Publisher, 2003.

[76] T.G. Crainic, M. Gendreau, P. Hansen, and N. Mladenovic. Cooperative parallel variable neighborhood search for the p-median. *Journal of Heuristics*, 10:293–314, 2004.

[77] D.R. Greening. Parallel simulated annealing tecniques. *Physica D*, 42:293–306, 1990.

[78] E. Cantú-Paz. A summary of parallel genetic algorithms. *Calculateurs Parallèls, Réseaux et Systèmes répartis*, 10:141–170, 1998.

[79] M. Verhoeven and E. Aarts. Parallel local search. *Journal of Heuristics*, 1:43–65, 1995.

[80] M. Verhoeven and M. Severen. Parallel local search for steiner trees in graphs. *Annals of Operations Research*, 90:185–202, 1999.

[81] P.M. Pardalos S. Duni Ekisoglu and M.G.C. Resende. *Models for Parallel and Distributed Computation - Theory, Algorithmic Techniques and Applications*, chapter Parallel metaheuristics for combinatorial optimization. Kluwer Academic, 2002.

[82] T.G. Crainic and M. Gendreau. Cooperative parallel tabu search for capacitated network design. *Journal of Heuristics*, 8:601–627, 2002.

[83] T.G. Crainic, M. Toulouse, and M. Gendreau. Towards a taxonomy of parallel tabu search algorithms. *INFORMS Journal on Computing*, 9:61–72, 1997.

[84] T.G. Crainic and M. Toulouse. *Simulated Annealing Parallelization Techniques*, chapter Parallel annealing by periodically interacting multiple searches: an experimental study. Wiley & Sons, 1992.

[85] K. Lee and S. Lee. Efficient parallelization of sumulated annealing using multiple markov chains: an application to graph partitioning. In *Proceedings of the International Conference on Parallel Processing*, pages 177–180, 1992.

[86] J. Cohoon, W. Martin, and D. Richards. *Parallel Problem solving from Nature*, volume 496 of *Lecture Notes in Computer Science*, chapter Genetic algortihm and punctuated equilibria in VLSI, pages 134–144. Springer Verlag, 1991.

[87] N. Krasnogor. Self-generating metaheuristics in bioinformatics: The proteins structure comparison case. *Genetic Programming and Evolvable Machines*, 5(2), 2004.

[88] A. Le Bouthillier and T.G. Crainic. A cooperative parallel meta-heuristic for the vehicle routing problem with time windows. *Computers and Operations Research*, 32(7):1685–1708, 2003.

[89] T.G. Crainic, M. Toulouse, and B. Sansó. Systemic behavior of cooperative search algorithms. *Parallel Computing*, 30:57–79, 2004.

[90] M. Toulouse, T.G. Crainic, and K. Thulasiraman. Global optimization properties of parallel cooperative search algorithms: a simulation study. *Parallel Computing*, 26(1):91–112, 2000.

[91] L.A. Zadeh. Fuzzy logic, neural networks, and soft computing. *Commun. ACM*, 37(3):77–84, 1994.

[92] X. Li, D. Ruan, and A.J. van der Wal. Discussion on soft computing at flins´96. *International Journal of Intelligent Systems*, 13(2-3):287–300, 1998.

[93] J.L. Verdegay. Una revisión de las metodologías que integran la soft computing. IV Congreso Español sobre Metaheurísticas, Algoritmos Evolutivos y Bioinspirados, 2005.

[94] E. Vergara. *Programación lineal difusa y criterios de parada difusas en la programación lineal*. PhD thesis, Dpto. Ciencias de la Computación e Inteligencia Artificial, Universidad de Granada, 1999.

[95] J.L. Verdegay and E. Vergara. Fuzzy sets-based control rules for terminating algorithms. *Computer Science Journal*, 10:9–27, 2002.

[96] A. Sancho-Royo, J.L. Verdegay, and E. Vergara-Moreno. Some practical problems in fuzzy sets-based decision support systems. *MathWare and Soft Computing*, 6(2-3):173–187, 1999.

[97] J.L. Verdegay and E. Vergara. Fuzzy termination criteria in knapsack problem algorithms. *MathWare and Soft Computing*, 7(2-3):89–97, 2000.

[98] D. Pelta and N. Krasnogor. *Recent Advances in Memetic Algorithms*, chapter Multimeme Algorithms using fuzzy logic based memes for protein structure prediction. Studies in Fuzziness and Soft Computing. Physica-Verlag, 2004.

[99] D. Pelta, A. Blanco, and J.L. Verdegay. A fuzzy adaptive neighborhood search for function optimization. In *Fourth International Conference on Knowledge-Based Intelligent Engineering Systems & Allied Technologies, KES 2000*, volume 2, pages 594–597, 2000.

[100] D. Pelta, A. Blanco, and J.L. Verdegay. Introducing fans: a fuzzy adaptive neighborhood search. In *Eigth International Conference on Information Processing and Management of Uncertainty in Knowledge-based Systems, IPMU 2000*, volume 3, pages 1349–1355, 2000.

[101] M. Basak and L.A Özdamar. *Fuzzy Sets based Heuristics for Optimization*, chapter A Fuzzy Adaptive Partitioning Algorithm (FAPA) for Global Optimization. Studies in Fuzziness and Soft Computing. Physica-Verlag, 2003.

[102] J.L. Verdegay, editor. *Fuzzy Sets based Heuristics for Optimization*. Studies in Fuzziness and Soft Computing. Physica-Verlag, 2003.

[103] D. Pelta, A. Blanco, and J.L. Verdegay. *Fuzzy Sets based Heuristics for Optimization*, chapter Fuzzy Adaptive Neighborhood Search: Examples of Application. Studies in Fuzziness and Soft Computing. Physica-Verlag, 2003.

[104] D. Pelta, A. Blanco, and J.L. Verdegay. A fuzzy valuation-based local search framework for combinatorial problems. *Journal of Fuzzy Optimization and Decision Making*, 2(1):177–193, 2002.

[105] D. Pelta and N. Krasnogor. Measuring protein structure similarities by means of the universal similarity metric. *Bioinformatics*, 7(20):1015 – 1021, 2004.

[106] D. Pelta, N. Krasnogor, C. Bousono-Calzon, J.H. Edmund Burke, and J.L. Verdegay. Applying a fuzzy sets-based heuristic for the protein structure prediction problem. *Fuzzy Sets and Systems*, 152(1):103–123, 2005.

[107] D. Pelta, A. Blanco, and J.L. Verdegay. Applying a fuzzy sets-based heuristic for the protein structure prediction problem. *International Journal of Intelligent Systems*, 17(7):629–643, 2002.

[108] D. Pisinger. Where are the hard knapsack problems? Technical report, University of Copenhagen, DIKU, Denmark, 2003/08.

[109] H. Kellerer, U. Pferschy, and D. Pisinger. *Knapsack Problems.* Springer Verlag, 2004.

[110] S. Martello and P. Toth. *Knapsack Problems: Algorithms and Computer Implementations.* John Wiley & Sons, 1990.

[111] D. Pisinger. Personal communication.

[112] S.L. Hakimi. Optimum locations of swiching centers and the absolute centers and medians of a graph. *Operations Research*, 12:450–459, 1964.

[113] S.L. Hakimi. Optimum distributions of swiching centers in a communications network and some related graph theoretic problems. *Operations Research*, 13:462–475, 1965.

[114] S.L. Hakimi. An algorithmic approach to network location problems; part 2. the p-medians. *SIAM Journal on Applied Mathematics*, 37:539–560, 1969.

[115] J.E. Beasley. A note on solving large p-median problems. *European Journal of Operational Research*, 21:270–273, 1985.

[116] P. Hanjoul and D. Peeters. A comparison of two dual-based procedures for solving the p-median problem. *European Journal of Operational Research*, 20(3):387–273, 1985.

[117] G. Cornuejols, M.L. Fisher, and G.L. Nemhauser. Location of bank accounts to optimize float: An analytic study of exact and approximate algorithms. *Management Science*, 23:789–810, 1977.

[118] M.L. Brandeau and S.S. Chiu. An overview of representative problems in location research. *Management Science*, 35(6):645–674, 1989.

[119] P.S. Mirchandani and R.L. Francis, editors. *Discrete Location Theory.* John Wiley and Sons, 1990.

[120] Z. Drezner, editor. *Facility Location.A Survey of Applications and Methodsy.* Springer-Verlag, 1995.

[121] M. Labbé and F.V. Louveaux. *Annotated Bibliographies in Combinatorial Optimization*, chapter Location Problem. John Wiley and Sons, 1997.

[122] A.A. Kuehn and M.J. Hamburger. A heuristic program for locating warehouses. *Management Science*, 9:643–666, 1990.

[123] F.E. Maranzana. A heuristic program for locating warehouses. *Operations Research Quaterly*, 12:138–139, 1964.

[124] M.B. Teitz and P. Bart. Heuristic methods for estimating the generalized vertex median of a weighted graph. *Operations Research*, 16(5):955–961, 1968.

[125] E.M. Captivo. Fast primal and dual heuristics for the p-median location problem. *European Journal of Operational Research*, 52:65–74, 1991.

[126] N.D. Pizzolato. A heuristic for large-size p-median location problems with application to school location. *Annals of Operations Research*, 50:473–485, 1994.

[127] J.A. Moreno, C. Rodríguez, and N. Jiménez. Heuristic cluster algorithm for multiple facility location allocation problem. *RAIRO: Recherche Operationnelle Operations Research*, 25:97–107, 1991.

[128] S. Voß. A reverse elimination approach for the p-median problem. *Studies in Locational Analysis*, 8:49–58, 1996.

[129] P. Hansen and N. Mladenovic. Variable neighborhood search for the p-median. *Location Science*, 5(4):207–226, 1997.

[130] R.D. Galvão. A dual bounded algorithm for the p-median problem. *Operations Research*, 28:270–273, 1980.

[131] J.E. Beasley. Lagrangean heuristics for location problems. *European Journal of Operational Research*, 65:383–399, 1993.

[132] P. Avella, A. Sassano, and I. Vasiliev. Computational study on large-scale p-median problems. Technical report, Dipartimento di Informatica e Sistemistica,Universitá di Roma La Sapienza, Italy, 2003.

[133] N. Mladenovic, J.A. Moreno, and J.M. Moreno Vega. Tabu search in solving p-facility location-allocation problems. Technical report, Les Cahiers du GERAD, GERAD, Montréal, Canada, 1995.

[134] E. Rolland, D.A. Schilling, and J.R. Current. An efficient tabu search procedure for the p-median problem. *European Journal of Operational Research*, 96:329–342, 1996.

[135] E. Taillard. Heuristic methods for large centroid clustering problems. *Journal of Heuristics*, 9 (1):51–73, 2003.

[136] K.E. Rosing and M.J. Hodgson. Heuristic concentration for the p-median: An example demonstrating how and why it works. *Computers and Operations Research*, 29(10):1317–1330, 2002.

[137] P. Hansen, N. Mladenovic, and D. Perez-Brito. Variable neighborhood decomposition search. *Journal of Heuristics*, 7:335–350, 2001.

[138] F. Garcia, B. Melian, J.A. Moreno, and J.M. Moreno. The parallel variable neighborhood search for the p-median problem. *Journal of Heuristics*, 8:375–388, 2002.

[139] M.J. Canós, C. Ivorra, and V. Liern. An exact algorithm for the fuzzy p-median problem. *European Journal of Operational Research*, 116:80–86, 1999.

[140] M.J. Canós, C. Ivorra, and V. Liern. The fuzzy p-median problem: A global analysis of the solutions. *European Journal of Operational Research*, 130:430–436, 2001.

[141] D. Kutangila. *Modelos basados en Soft Computing para resolver problemas de localización.* PhD thesis, Departamento de Ciencias de la Computación e Inteligencia Artificial, Universidad de Granada, 2005.

[142] D. Kutangila and J.L. Verdegay. Fuzzy graphs as basis for solving some public transportation problems in kinshasa, emergent solutions for information and knowledge economy. In *Proceedings of the 10th SIGEF Congress*, 2003.

[143] p-median test problems (galvao sets). http://www.bus.ualberta.ca/eerkut/testproblems/.

[144] R.D. Galvão and C. ReVelle. A lagrangean heuristic for the maximal covering location problem. *European Journal of Operational Research*, 88:114–123, 1996.

[145] G. Reinelt. Tsplib: A traveling salesman problem library. *ORSA Journal on Computing*, 3:376–384, 1991.

[146] M. Toulouse, T.G. Crainic, and B. Sanso. An experimental study of systemic behaviour of cooperative search algorithms. In S. Voss, S. Martello, C. Roucariol, and I. Osman, editors, *Meta Heuristics 98: Theory and Applications*, pages 373–392. Kluwer Academic Publishers, 1999.

Lista de Publicaciones