FACULTAD DE CIENCIAS
UNIVERSIDAD DE CANTABRIA

Proyecto Fin De Carrera

Metaheurísticas basadas en Scatter Search y Path Relinking para resolver el Problema del Fuzzy Job Shop Scheduling

(Scatter Search and Path Relinking based metaheuristics
to solve the Fuzzy Job Shop Scheduling Problem)

Para acceder al Título de
INGENIERO EN INFORMÁTICA

Autor: Alberto Polidura Pérez
Marzo 2013

FACULTAD DE CIENCIAS

INGENIERÍA EN INFORMÁTICA

CALIFICACIÓN DEL PROYECTO DE FIN DE CARRERA

Realizado por:	Alberto Polidura Pérez
Directora del PFC:	Inés González Rodríguez
Título:	Metaheurísticas basadas en Scatter Search y Path Relinking para resolver el problema del Fuzzy Job Shop Scheduling
Title:	Scatter Search and Path Relinking based metaheuristics to solve the Fuzzy Job Shop Scheduling Problem

Presentado a examen el día:

para acceder al Título de
INGENIERO EN INFORMÁTICA

Composición del Tribunal:

Presidente (Apellidos, Nombre): González Harbour, Michael
Secretaria (Apellidos, Nombre): Martínez Fernández, María del Carmen
Vocal (Apellidos, Nombre): Menéndez de Llano Rozas, Rafael
Vocal (Apellidos, Nombre): Sánchez Barreiro, Pablo
Vocal (Apellidos, Nombre): Sanz Gil, Roberto

Este Tribunal ha resuelto otorgar la calificación de: .....

Fdo: El Presidente

Fdo: El Secretario

Fdo: Vocal

Fdo: Vocal

Fdo: Vocal

Fdo: El Director del PFC

Agradecimientos

A mi directora de PFC, Inés González Rodríguez, por darme la oportunidad de trabajar con ella y resolverme todas las dudas que fueron surgiendo, así como motivarme a seguir adelante durante los momentos más difíciles de su realización.

A mi familia, por el apoyo constante durante este periodo en el que siempre se interesaron por su estado y me aconsejaron todo lo que pudieron en diversos momentos.

A mis amigos y todas aquellas personas que en algún momento mostraron interés por los progresos en el proyecto y aguantaron mis explicaciones por muy aburridas que fuesen.

Con este proyecto da por finalizada una de las etapas de mi vida más enriquecedoras en todos los aspectos y que seguro guardaré en mi memoria para siempre. Por ello, a todos aquellos que han formado parte de ella, muchas gracias.

Resumen

El presente Proyecto de Fin de Carrera tiene como objetivo aplicar una metaheurísitca conocida como Scatter Search con Path Relinking al problema de planificación de tareas conocido como Job Shop Scheduling Problem (JSP) con duraciones inciertas. Esto es, al problema de planificación clásico del Job Shop, que ha sido tratado desde diversos enfoques tradicionalmente, le añadiremos un grado de incertidumbre mediante el uso de intervalos difusos para modelar las duraciones inciertas de las tareas.

Para la búsqueda de soluciones utilizaremos el mencionado esquema algorítmico de búsqueda evolutiva Scatter Search (o Búsqueda Dispersa) que orienta la exploración del espacio de búsqueda tomando como referencia un conjunto de "buenas" soluciones. Al mismo, le incorporaremos la estrategia de Path Relinking (o reenlazado de caminos) para explorar en más detalle las trayectorias que conectan estas "buenas"soluciones. Para ello, estudiaremos cómo adaptar los operadores genéricos de este esquema de búsqueda al problema bajo consideración.

Como resultado del proyecto se pretende obtener una implementación de los algoritmos de búsqueda mencionados para la resolución del problema de planificación JSP. El método resultante se evaluará sobre una batería de problemas de referencia (benchmarks), con el correspondiente análisis de los resultados obtenidos. El lenguaje de programación utilizado será C++ y para la implementación se utilizará un desarrollo basado en componentes, adoptando una metodología iterativa e incremental.

Palabras clave: Planificación, Job Shop, Duraciones Inciertas, Metaheurística, Scatter Search, Path Relinking

Preface

The main goal of this Final Year Project is to apply a metaheuristic known as Scatter Search with Path Relinking to the Job Shop Scheduling Problem (JSP) with fuzzy durations. This means that, to the classic Job Shop Scheduling Problem commonly studied from different points of view, we will add a degree of uncertainty using fuzzy intervals in order to model the uncertain duration of the tasks.

In order to explore the solution space, we will use the so called evolutive Scatter Search algorithm scheme, which directs the exploration using a set of "good"solutions. This will be combined with the Path Relinking strategy to explore in more detail the paths that connect these "good"solutions. To do this, we will study how to adapt the generic operators from this search algorithm to the problem under consideration.

As a result of the project we get an implementation of the search algorithms previously discussed for the resolution of the Job Shop Scheduling Problem. The resulting method will be evaluated with a benchmark of reference problems with the corresponding analysis of the obtained results. The programming language used will be C++ and we will use a component based development adopting an incremental and iterative methodology.

Keywords: Scheduling, Job Shop, Uncertain Durations, Metaheuristic, Scatter Search, Path Relinking

Índice general

Índice de figuras

Índice de cuadros

Capítulo 1

Introducción

En este capítulo se hace una pequeña introducción al problema que pretende solventar este Proyecto Fin de Carrera. Tras la descripción se expone la estructura que seguirá el documento.

Índice

1.1. Introducción

Los problemas de planificación de tareas han constituido un notable campo de investigación desde los años 50 con importantes aplicaciones en diversos campos de la industria, finanzas y ciencia [4]. Parte de ésta investigación ha sido focalizada en aquellos problemas de planificación de tareas en los que existe un cierto grado de incertidumbre o imprecisión en los datos de que disponemos tratando de modelar las situaciones típicas de la vida real [12]. La incorporación de incertidumbre a estos problemas requiere un enfoque distinto en los métodos de resolución y representación con los que habitualmente son tratados.

Uno de los problemas de planificación de tareas más abordado ha sido el conocido como Job Shop Problem[1] que trata sobre la asignación de tareas limitadas a recursos a lo largo del tiempo y tiene como finalidad la optimización de uno o más objetivos. Estos recursos pueden ser máquinas de un taller, pista de aterrizaje en un aeropuerto, ladrillos en una construcción, unidades de procesamiento en un ambiente computacional, etc. Las tareas pueden ser operaciones del proceso de producción, despegues y aterrizajes en el aeropuerto, etapas de un proyecto de ingeniería, ejecuciones de un programa computacional, etc.

[1] A lo largo de esta memoria, hemos preferido que algunos de los términos técnicos figuren en inglés, en lugar de sustituirlos por su traducción al español. Este ha sido el criterio a seguir cuando no existe una traducción directa al castellano o cuando el uso del término original en inglés está extendido en la comunidad científica hispanoparlante.

El Job Shop Problem puede ser un problema difícil (entra en la categoría de los considerados problemas NP-completos) tanto desde un punto de vista técnico como de implementación. Las dificultades encontradas son similares a las de otras ramas de optimización combinatoria y modelado estocástico. Es precisamente la complejidad computacional del Job Shop Problem lo que motiva el uso de técnicas metaheurísticas para su resolución.

Entre todas las variantes posibles del problema, algunos intentos han sido llevados a cabo para extender los métodos heurísticos clásicos de problemas de planificación de tareas a aquellos casos con duraciones de tareas inciertas. Una posibilidad es modelar la incertidumbre en la duración de las tareas mediante intervalos difusos en los que conocemos tres valores: el menor valor posible de duración, el más probable y el mayor. El problema resultante se denomina Fuzzy Job Shop Scheduling Problem. Este Fuzzy Jop Shop Problem ha sido tratado, entre otros, mediante redes neuronales [14], algoritmos genéticos [15], [16], [17], simulated annealing [13] y algoritmos genéticos unidos a búsquedas locales [18].

Uno de estos métodos heurísticos es el llamado algoritmo de Scatter Search (o Búsqueda Dispersa). Dicho algoritmo genera soluciones teniendo en cuenta las características de varias partes del espacio de búsqueda y orienta la exploración tomando como referencia un conjunto de soluciones de alta calidad. Una variante del mismo puede encontrarse con la estrategia de Path Relinking (o reenlazado de caminos) para explorar en más detalle las trayectorias que conectan estas soluciones de alta calidad.

En nuestro trabajo, proponemos un nuevo enfoque de resolución para el Job Shop Scheduling Problem con duraciones de tareas inciertas, basado en la combinación de Scatter Search con Path Relinking. Para ello desarrollaremos una sencilla aplicación que implemente tanto dicho problema de planificación como nuestros algoritmos de búsqueda. Ésta será sometida a una batería de pruebas usando una serie de instancias del problema conocidas por su elevada complejidad. Anotaremos los resultados obtenidos para su posterior análisis en busca de explicaciones para su comportamiento.

1.2. Objetivos del Proyecto

El objetivo principal del proyecto es aplicar el algoritmo Scatter Search con Path Relinking al Job Shop Scheduling Problem con duraciones de tareas inciertas. Pero más concretamente, podemos distinguir una serie de objetivos en particular:

1. Estudiar a partir de la literatura científica existente el problema del Fuzzy Job Shop y las metaheurísticas Scatter Search y Path Relinking.
2. Proponer cómo se puede adaptar el esquema general de Scatter Search y Path Relinkin al problema Fuzzy Job Shop.
3. Implementar la adaptación propuesta.
4. Evaluar experimentalmente las distintas componentes de la técnica Scatter Search con Path Relinking, así como el método final resultante.

1.3. Estructura del Documento

A continuación se hace un breve resumen de los contenidos a tratar en cada uno de los capítulos posteriores del documento.

Capítulo 2: El problema de Job Shop Scheduling
Se describe en detalle el problema de planificación de tareas al que nos vamos a enfrentar en nuesta versión particular con duraciones inciertas. Para ello hablaremos de sus requisitos, objetivos, formas de representación y propiedades de las soluciones.

Capítulo 3: Solución Metaheurística
Se explica a fondo los algoritmos de Scatter Search y Path Relinking que serán utilizados en nuestro problema de planificación de tareas.

Capítulo 4: Diseño y Desarrollo
Se explican mediante diagramas UML la arquitectura y el diseño propuestos para que la aplicación a desarrollar sea correcta y segura en el cumplimiento de los requisitos, además de para tener una visión global de su funcionamiento.

Capítulo 5: Resultados
Se presenta el entorno de pruebas construido para la aplicación y se muestran los resultados obtenidos en cada una de las ejecuciones que se llevaron a cabo. Además, Se analizan en detalle estos resultados dando razones para explicar su comportamiento.

Capítulo 6: Conclusiones y Trabajo Futuro
Se exponen las conclusiones obtenidas de la elaboración del proyecto y una serie de posibles líneas de desarrollo futuras.

Capítulo 2

El Problema de Job Shop Scheduling

En este capítulo se tratará detalladamente el problema de planificación sobre el que vamos a trabajar, el Job Shop Scheduling Problem con duraciones inciertas.

Índice

2.1. Descripción del Problema

El problema clásico de planificación de tareas conocido como Job shop Problem consiste en planificar un conjunto de trabajos $J_1, ..., J_n$ en un conjunto de máquinas $M_1, ..., M_m$ sujetas a ciertas restricciones. Una de estas restricciones se refiere al orden secuencial en el que las tareas $\theta_{11}, ..., \theta_{1m}$ de cada trabajo $J_i, i = 1, ..., n$ deben ejecutarse. Además, hay restricciones de capacidad en las que una cierta tarea θ_{ij} requiere el uso exclusivo e ininterrumpido de una máquina durante su tiempo de procesamiento, denotado p_{ij}.

Una planificación factible será aquella que defina los instantes de inicio de cada una de las tareas de tal manera que todas las restricciones sean respetadas.

El objetivo es encontrar una planificación factible que sea óptima de acuerdo con un cierto criterio, que en nuestro caso particular será minimizar el tiempo

total de su procesamiento (también conocido como makespan).

2.2. Duraciones Inciertas

Es común en la vida real encontrarnos con situaciones en las que existe una incertidumbre a la hora de dictaminar el tiempo que se tarda en procesar una tarea de nuestro sistema. Sin embargo, un experto del tema tendrá, gracias a su experiencia previa, cierto conocimiento sobre su duración, lo le que permitirá restringirla dentro de unos márgenes temporales. En nuestro caso, supondremos que disponemos de un intervalo difuso A definido por tres valores (a_1, a_2, a_3) que representan los extremos del intervalo de valores posibles $[a_1, a_3]$ y el valor más habitual o posible a_2. Este intervalo modela una distribución de posibilidad sobre los valores de la duración. También puede verse como un número difuso [19] (un conjunto difuso sobre los reales) cuya función de pertenencia viene dada por la siguiente ecuación:

$$\mu_A(x) = \begin{cases} \frac{x-a_1}{a_2-a_1} & : a_1 \leq x \leq a_2 \\ \frac{x-a_3}{a_2-a_3} & : a_2 < x \leq a_3 \\ 0 & : x < a_1 \text{ o } a_3 < x \end{cases}$$

Para nuestro problema, esencialmente solo necesitaremos dos operaciones sobre estos intervalos inciertos, la suma y el máximo. Éstas pueden ser obtenidas mediante el Principio de Extensión desde números reales a intervalos difusos [20]. Sin embargo, computar estas operaciones en intervalos difusos puede llegar a ser intratable, por lo que las simplificaremos realizando una aproximación que consistirá en ejecutar estas operaciones en los tres puntos que definen nuestro intervalo incierto, tal y como se propone en [13]. Por lo tanto, para dos números difusos M y N cualesquiera, la operación aproximada de la suma será de la siguiente manera:

$$M + N = (m_1 + n_1, m_2 + n_2, m_3 + n_3)$$

Y la aproximación del máximo será:

$$\text{máx}(M, N) = (\text{máx}(m_1, n_1), \text{máx}(m_2, n_2), \text{máx}(m_3, n_3))$$

Los intervalos difusos pueden entenderse como distribuciones de posibilidad sobre los números reales. Esto permite obtener el valor esperado de un intervalo difuso cualquiera A utilizando la siguiente expresión [21]:

$$E[A] = \tfrac{1}{4}(a^1 + 2a^2 + a^3)$$

El valor esperado coincide con el substituto escalar neutro del intervalo incierto, que a su vez se trata del centro de gravedad del triangulo formado por dicho intervalo [22]. Dicho valor puede utilizarse para establecer un orden total $\leq_E$ en el conjunto de intervalos difusos [13], de manera que para cualesquiera dos intervalos M y N, $M \leq_E N$ si y sólo si $E[M] \leq E[N]$. Este orden puede extenderse trivialmente para hallar el elemento maximal de un conjunto de varios intervalos.

2.3. Representacion Gráfica del Problema

El Job Shop Problem se puede representar a través de un grafo conocido como grafo disyuntivo $G = (V, A \cup D)$:

- Cada nodo del conjunto V representa una tarea del problema. Además, se incluyen dos nodos 0 y ∞ que corresponden a tareas ficticias con duración 0 y tales que la tarea 0 ha de preceder a todas las demás tareas y la tarea ∞ ha de ser posterior al resto de tareas.
- Los arcos A son llamados arcos conjuntivos y representan las restricciones de precedencia entre tareas.
- Los arcos D corresponden a los arcos disyuntivos y representan las restricciones de capacidad de las máquinas.

Cada uno de los arcos posee un peso equivalente al tiempo que tarda en procesarse la tarea de la que parte el arco en nuestro problema.

Un grafo solución determina unívocamente un orden de procesamiento de tareas y viceversa, un orden de procesamiento de tareas puede representarse como un grafo. Así, en lo que sigue hablaremos indistintamente de orden de procesamiento o solución factible. Encontrar un orden de procesamiento factible será equivalente a hallar un subgrafo acíclico de $G = (V, A \cup R)$ donde R es una selección hamiltoniana de D.

El makespan es el tiempo de finalización de la última tarea o equivalentemente, en el grafo solución, la longitud del camino más largo entre el nodo inicial 0 y el nodo final ∞. Por tanto, dado un grafo solución, para obtener el makespan sólo es necesario usar propagación hacia delante de los pesos de cada arco desde el nodo 0 hasta el nodo final.

Puesto que en nuestra versión del problema los tiempos son intervalos difusos, deberemos utilizar las operaciones de suma y máximo explicadas anteriormente para propagar las restricciones. De esta manera, la planificación resultante será una planificación incierta. Esto siginifica que los tiempos de inicio y fin de cada tarea y, por tanto, del makespan, ya no son precisos, sino que vienen dados por intervalos difusos.

Cuando las duraciones son números reales, un camino crítico del grafo es un camino de duración máxima desde 0 hasta ∞; su longitud coincide con el makespan. Cuando las duraciones son intervalos difusos, es necesario redefinir el concepto de camino crítico.

Todos estos conceptos se ilustran con un ejemplo en la Sección 2.9.

2.4. Caminos Críticos

Otro concepto que debe ser reformulado para intervalos difusos es el de caminos críticos.

Para el caso exacto definimos camino crítico como el camino más largo de nuestro grafo solución desde un nodo de inicio hasta un nodo de fin. A su vez, arco crítico será cualquier arco dentro del camino crítico de nuestro grafo. Sin embargo, para duraciones inciertas, no es trivial extender éste concepto. Por ejemplo, podría darse el caso de que el makespan (un intervalo incierto) no coincida con el tiempo de finalización de alguno de los trabajos.

Para ello, en [24] se propone una nueva estructura de vecindades basada en la idea presentada previamente, en la cual hablábamos de que las operaciones aritméticas para intervalos difusos pueden simplificarse a su realización en los tres puntos críticos de los mismos. Esto nos permite descomponer el grafo de soluciones de duraciones inciertas en tres grafos paralelos con duraciones fijas. Por lo tanto, en cada uno de éstos podremos calcular el camino crítico tal y como definimos en el párrafo anterior.

Usando la representación de los grafos paralelos, un camino es crítico para un Job Shop Problem de duraciones inciertas si es crítico en alguno de sus grafos paralelos.

Una vez más, todos estos conceptos se ilustran con un ejemplo en la Sección 2.9.

2.5. Modelo de Makespan Esperado

Como hemos dicho, el objetivo del Job Shop Problem es encontrar una planificación óptima de las tareas de tal manera que el makespan de ésta sea mínimo. Sin embargo en nuestro caso el makespan es un intervalo difuso y la operación máximo no define un orden total en nuestro conjunto de intervalos difusos. Es por esto que utilizaremos el valor esperado y el orden total que define en el conjunto de intervalos difusos. Así, entenderemos que una solución óptima es aquella que minimiza el valor esperado del makespan.

2.6. Representación Binaria y Cálculo de Distancias

Una representación binaria del Job Shop Problem es propuesta por Nakano y Yamada (1991) en [25]. Su vector binario contiene elementos que denotan las relacciones de precedencia entre las operaciones que son procesadas por la misma máquina. Un elemento en particular determina si la operación v está planificada antes de la operación w ($v < w = 1$) o no ($w < v = 0$). Puesto que cada trabajo debe procesarse en cada máquina, necesitamos un vector del tamaño $m \cdot n$ (donde m es el número de trabajos y n el número de máquinas). Pero ya que el gen $v < w$ está especificado, el gen $w < v$ es redundante y por tanto omitido. Así, terminaremos con un vector de tamaño $mn(n-1)/2$ bits.

El algoritmo que describe el procedimiento para calcular la representación binaria de una solución puede encontrarse en Algoritmo 1.

Algoritmo 1 Cálculo de la representación binaria de una solución

```
RepBinaria=∅ y n=0
para i=0 hasta i < m hacer
  para j=0 hasta j < n-1 hacer
    para k=j+1 hasta k < n-1 hacer
      Obtener el identificador de la tarea del trabajo j y máquina i, id1
      Obtener el identificador de la tarea del trabajo k y máquina i, id2
      si id1 planificada antes que id2 entonces
        RepBinaria[n]=1
      si no
        RepBinaria[n]=0
      fin si
      Incrementa n
    fin para
  fin para
fin para
devolver RepBinaria
```

En términos de la representación gráfica propuesta previamente, un elemento de nuestro vector determina la dirección de un arco entre tareas.

Dado que lo que diferencia a una solución de otra en nuestro problema del Job Shop son las direcciones de los arcos, podemos utilizar esta representación binaria para calcular distancias entre ambas. Esto es, ir aplicando bit a bit un operación XOR que nos dé como resultado el número de bits (o arcos) que tienen diferentes y por tanto su distancia. El Algoritmo 2 desarrolla el procedimiento explicado.

Algoritmo 2 Cálculo de la distancia entre dos soluciones s' y $s"$

```
distancia=0
repBinaria1 es el resultado de aplicar 1 sobre s'
repBinaria2 es el resultado de aplicar 1 sobre s"
para cada bit b' de repBinaria1 y b" de repBinaria2 hacer
  distancia=distancia + b' ⊕ b"
fin para
devolver distancia
```

2.7. Representación en Forma de Vector de Permutaciones

Un orden de procesamiento cualquiera de nuestro Job Shop Problem puede codificarse como una permutación con repetición [1]. Esto es, una permutación del conjunto de tareas en el que cada tarea viene representada simplemente por el número del trabajo al que pertenece. Por ejemplo, si tenemos un problema con 3 trabajos de 3 tareas cada uno, su representación en forma de permutación con repetición puede ser la siguiente (2 3 1 1 3 2). En esta codificación, el primer

2 representa la primera tarea del segundo trabajo, el segundo 2 la segunda y así sucesivamente. La principal ventaja de este esquema es que siempre representa soluciones factibles, lo que facilita la definición de los distintos operadores de algoritmos de búsqueda metaheurística, como es el caso de la Búsqueda Dispersa.

Este orden de procesamiento puede decodificarse y representarse, como vimos antes, mediante un grafo en el que mediante la propagación de restricciones podemos obtener el tiempo de inicio de cada tarea. Este tipo de decodificación da lugar a nuestro primero tipo de planificaciones atendiendo a los tiempos de holgura entre tareas, las semis-activas:

Planificaciones semi-activas: Una planificación factible es llamada semi-activa si ninguna operación puede ser completada antes sin alterar el orden de procesamiento de ninguna de las máquinas.

Un subconjunto dentro de éstas son las planificaciones activas:

Planificaciones activas: Una planificación factible es llamada activa si no es posible construir otra planificación, mediante cambios en el orden de procesamiento de las maquinas, con al menos una tarea terminando antes y ninguna terminando más tarde.

Y a su vez, dentro de las activas podemos encontrarnos las planificaciones non-delay (o densas):

Planificaciones non-delay: Una planificación factible es llamada non-delay si ninguna maquina está parada mientras una operación esta esperando a ser procesada.

En lo que respecta a nuestro problema, sabemos que en el caso de duraciones exactas (no difusas), es seguro que el conjunto de soluciones activas contiene al menos una solución óptima (mínimo makespan) [4]. Por ello, sería interesante reducir la búsqueda al sub-espacio de soluciones activas, ya que es más pequeño y a la vez es seguro que contiene un óptimo.

La búsqueda de planificaciones activas es lo que nos lleva a utilizar un algoritmo de decodificación diferente al de propagación de restricciones. Éste es, precisamente, el tema sobre el que discutimos en la Sección siguiente.

2.8. Algoritmo de Planificación G&T

El Algoritmo de Giffler & Thompson [23] (G&T a partir de ahora) nos permite obtener las deseadas soluciones activas (en el Job Shop Problem determinista) a partir de un orden de procesamiento. Se trata de un algoritmo voraz que intenta asignar tiempos de inicio a cada tarea evitando que queden huecos en la planificación final. Dicho algoritmo utiliza como referencia el orden de procesamiento dado, pero puede ocurrir que no lo respete (lo modifique) si

así consigue mejorar la solución final.

El esquema básico del Algoritmo G&T, tal y como es descrito en [2], se muestra en el Algortimo 3.

Algoritmo 3 Algoritmo G&T

A es el vector de solución
Determinar la operación o' de A con el tiempo de procesamiento más temprano.
Determinar la maquina M' de o' y construir el conjunto B de todas las operaciones en A que se procesan en M'.
Eliminar las operaciones en B que no comienzan antes del tiempo de procesamiento de o'.
Planificar la operación de B más a la izquierda en la permutación y eliminarla de A.

Para nuestro caso de duraciones inciertas, podemos extender el G&T aunque ya no podemos asegurar que la planificación resultante vaya a ser activa, dada la incertidumbre existente en la duración de las tareas. Sólo podremos decir que la planificación incierta obtenida es posiblemente activa en el sentido de que existe una configuración de posibles duraciones para la cual dicha planificación es activa.

2.9. Ejemplo Ilustrativo

Para facilitar el entendimiento de todos estos conceptos se presenta un ejemplo del Job Shop Problem para duraciones inciertas. El problema del ejemplo tendrá un tamaño 3x2 (tres trabajos con dos tareas cada uno). Para definirlo tendremos que conocer las duraciones difusas de las tareas y las tareas que comparten máquina. Lo primero vendrá definido por una matriz de duraciones que contiene las duraciones de las tareas, organizadas por filas las de un mismo trabajo. En nuestro caso se muestra a continuación:

$$\begin{pmatrix} (3,4,7) & (1,2,3) \\ (4,5,6) & (2,3,4) \\ (1,2,6) & (1,2,4) \end{pmatrix}$$

Y para definir qué tareas comparten máquina se utiliza una matriz de máquinas que contiene organizadas por filas las tareas de un trabajo, denotando por un mismo entero la máquina que utiliza cada una. En nuestro caso será la siguiente matriz:

$$\begin{pmatrix} 0 & 1 \\ 1 & 0 \\ 1 & 0 \end{pmatrix}$$

Tal y como explicamos previamente, nuestro problema se puede representar mediante un grafo como podemos ver en la Figura 2.1.

Como también explicamos anteriormente, existen unas restricciones de precedencia entre las tareas indicadas de antemano, lo que será representado por las

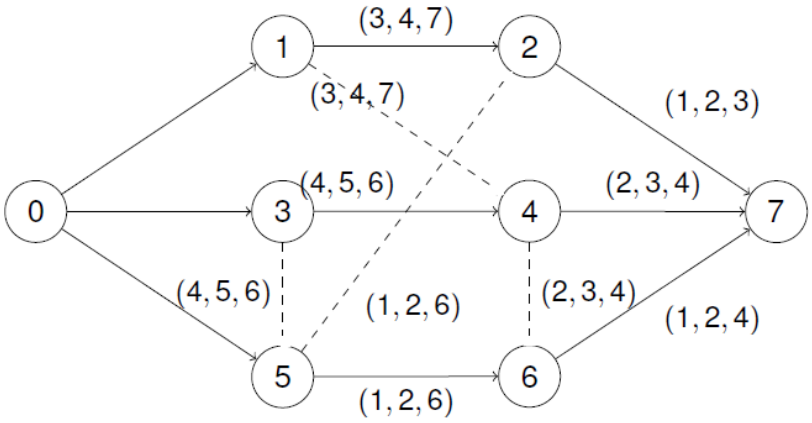

Figura 2.1: Grafo

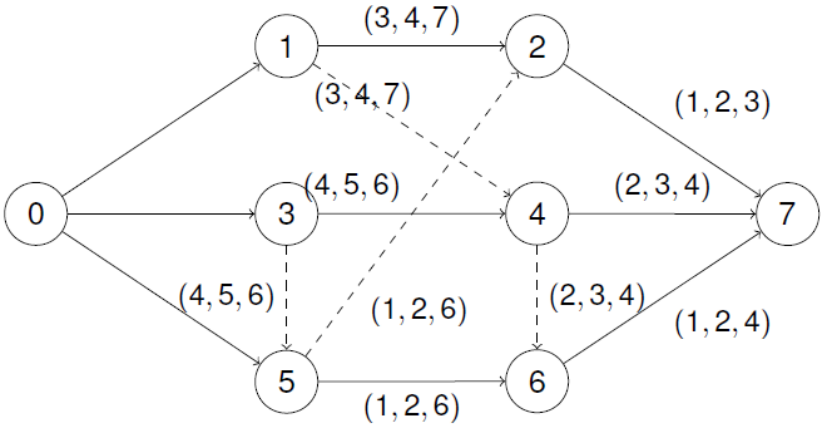

Figura 2.2: Grafo solución

flechas con línea continua. Éstas tienen unos pesos en los arcos que indican el tiempo que toma cada una de ellas en finalizar como mínimo, en valor medio y como máximo respectivamente. Las tareas que comparten un recurso están unidas por líneas discontinuas. Una solución al problema consistirá en elegir el orden de precedencia entre tareas que comparten recurso, asignando un sentido a las líneas discontinuas y eliminando los arcos disyuntivos que sobran. Una solución para el problema anterior se muestra en la Figura 2.2.

Ésta corresponde al orden de procesamiento de tareas (1 3 5 4 2 6) tal y como se explicó en la Sección 2.7. aplicando el Algoritmo 3. A su vez, la representación binaria de la misma es 010010 aplicando el Algoritmo 1.

Para calcular el camino crítico de ésta solución, procedemos a descomponer el grafo en tres grafos paralelos con cada una de las duraciones de los intervalos difusos. Ésto lo podemos observar en las Figuras 2.3, 2.4 y 2.5.

Así, hemos podido calcular los caminos críticos de cada uno de los subgrafos

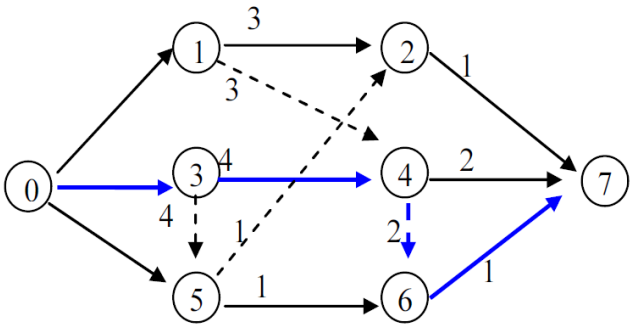

Figura 2.3: Camino Crítico (3 4 6) Makespan 7

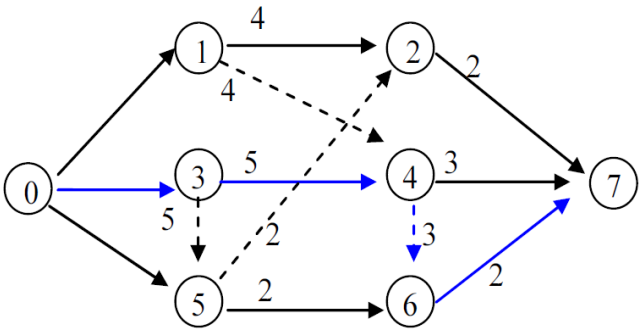

Figura 2.4: Camino Crítico (3 4 6) Makespan 10

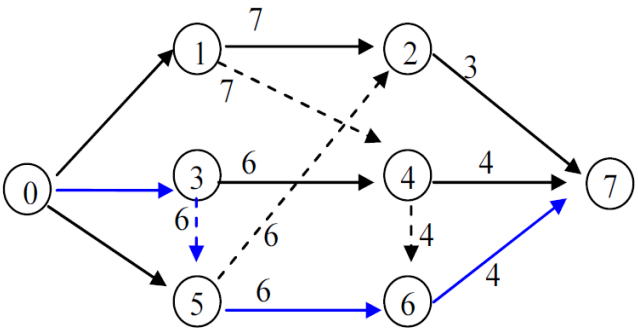

Figura 2.5: Camino Crítico (3 5 6) Makespan 16

y sumando los costes de cada uno de éstos podemos hallar los makespans respectivos.

Capítulo 3

Solución Metaheurística

En este capítulo daremos a conocer las principales ideas y conceptos del Scatter Search y Path Relinking, el esquema de búsqueda que vamos a utilizar en nuestro Fuzzy Job Shop.

Índice

3.1. Scatter Search

Scatter Search (SS) [1] es un procedimiento metaheurístico basado en formulaciones y estrategias introducidas originalmente por Glover en la década de los setenta [8]. SS opera sobre un conjunto de soluciones, llamado conjunto de referencia, combinando éstas para crear nuevas soluciones de modo que mejoren a las que las originaron. En este sentido decimos que es un método evolutivo. Sin embargo, a diferencia de otros métodos evolutivos, como los algoritmos genéticos, SS no está fundamentado en la aleatorización sobre un conjunto relativamente grande de soluciones, sino en elecciones sistemáticas y estratégicas sobre un conjunto pequeño. Como ilustración basta decir que los algoritmos genéticos suelen considerar una población de 100 soluciones, mientras que en la búsqueda dispersa es habitual trabajar con un conjunto equivalente de tan sólo 10 soluciones. SS se basa en el principio de que la información sobre la calidad o el atractivo de un conjunto de reglas, restricciones o soluciones puede ser utilizado mediante la combinación de éstas en lugar de aisladamente.

[1] Scatter Search podría traducirse como Búsqueda Dispersa, sin embargo, hemos optado por el término original en inglés por ser el más habitual en la literatura, incluso en la escrita en castellano, como puede verse en [26]

Una de las características más notables de SS es que se basa en integrar la combinación de soluciones con la búsqueda local. Aunque en diseños avanzados esta búsqueda local puede contener una estructura de memoria (búsqueda tabú), no es necesario que así sea, limitándose, en la mayoria de los casos a una búsqueda local convencional.

3.1.1. Diseño Básico de Scatter Search

Una primera plantilla de este esquema la podemos encontrar en [9], que ha servido como referencia para la mayoría de implementaciones a fecha de hoy. Esta metodología es muy flexible pues cada uno de sus elementos puede ser implementado de varias maneras y con distintos grados de sofisticación. A continuación describimos un diseño básico para implementar Scatter Search basado en los conocidos como "cinco métodos":

1. Método de generación diversificada: para generar una serie de soluciones de prueba usando una solución arbitraria o una semilla como input.
2. Método de mejora: para transformar una solución de prueba en una o más soluciones mejoradas.
3. Método de actualización del conjunto de referencia: para construir y mantener un conjunto de referencia que albergue las mejores soluciones encontradas, organizadas para proporcionar un acceso eficiente desde otros métodos. Las soluciones obtienen pertenencia de acuerdo con su calidad o diversidad.
4. Método de generación de subconjuntos: para producir un subconjunto del conjunto de referencia como base para crear soluciones combinadas.
5. Método de combinación de soluciones: para transformar un subconjunto dado por el método anterior en una o más soluciones nuevas utilizando combinación.

El procedimiento que describe la interacción de estos cinco métodos y el esquema general de Scatter Search es el planteado en [3] y reproducido en el Algoritmo 4.

Un par de apuntes que se podrían hacer sobre este esquema de scatter search son los siguientes:

- El conjunto de referencia, $RefSet$, es una colección tanto de soluciones de alta calidad como de soluciones diversas, que son usadas para generar nuevas soluciones mediante su combinación. Si el tamaño del conjunto de referencia es denotado por $b = b_1 + b_2 = |RefSet|$, la construcción del mismo consiste en seleccionar las b_1 mejores soluciones de la población generada. Estas soluciones son añadidas a $RefSet$ y borradas de la población generada. Para cada solución restante, la mínima de las distancias a las soluciones de $RefSet$ es calculada. Después, la solución con la máxima de estas distancias mínimas es seleccionada. Esta solución es añadida a $RefSet$, borrada de la población generada y las distancias mínimas son

Algoritmo 4 Algoritmo de Scatter Search

Empezar con P vacío.
mientras Tamaño de P no sea el deseado **hacer**
  Usar el $Metodo\ de\ generacion\ diversificada$ para construir una solución x y aplicar en ella el $Metodo\ de\ Mejora$.
  si x no pertenece a P **entonces**
    Añadir x a P
  fin si
fin mientras
Usar el *Método de actualizacion del conjunto de referencia* para construir el conjunto de referencia, *RefSet*, con las mejores soluciones de P.
Ordenar de mayor a menor las soluciones de *RefSet* de acuerdo con el valor de su función objetivo.
$NuevasSoluciones = TRUE$
mientras $NuevasSoluciones$ **hacer**
  Generar $NuevoSubconjunto$ con *Método de generación de subconjuntos.*
  $NuevasSoluciones = FALSE$
  mientras $NuevoSubconjunto$ no esté vacío **hacer**
    Seleccionar el siguiente subconjunto s de $NuevoSubconjunto$
    Aplicar el *Método de combinación de soluciones* a s para obtener una o más soluciones de prueba x.
    Aplicar el *Método de Mejora* a las soluciones de prueba.
    Ejecutar el *Método de actualización del conjunto de referencia*
    si $RefSet$ ha cambiado **entonces**
      $NuevasSoluciones = TRUE$
    fin si
    Eliminar s de $NuevoSubconjunto$
  fin mientras
fin mientras

recalculadas. El proceso es repetido b_2 veces. El conjunto de referencia resultante tiene b_1 soluciones de alta calidad y b_2 soluciones diversas.

- De los cinco métodos de nuestra metodología de Scatter Search, solo cuatro son estrictamente necesarios. El *Método de Mejora* normalmente es necesario en aquellos casos en los que buscamos resultados con una alta calidad pero el procedimiento puede ser implementado sin él.

3.1.2. Reconstrucción del Conjunto de Referencia

Cuando se consideran estrategias avanzadas en un entorno metaheurístico, el objetivo de mejorar el rendimiento entra en conflicto con el de diseñar un procedimiento sencillo de implementar y afinar. Los diseños avanzados generalmente, pero no siempre, se traducen en una mayor complejidad y en parámetros adicionales de búsqueda. Uno de los mismos le describimos a continuación.

El Método de Reconstrucción del conjunto de referencia descrito en [3], consiste en un procedimiento en el que el $RefSet$ es parcialmente reconstruido mediante una actualización diversificada. Esto es, si el tamaño del conjunto de referencia es $b = b_1 + b_2$, las soluciones $x^{b_1+1}, ..., x^b$ son eliminadas de $RefSet$. El metodo de generación diversificada es reinicializado considerando que el objetivo es generar soluciones que son diversas respecto a las de referencia $x^1, ..., x^{b_1}$. Para ello, el método de generación diversificada es usado para construir un conjunto de nuevas soluciones. Las b_2 soluciones $x^{b_1+1}, ..., x^b$ de $RefSet$ son seleccionadas secuencialmente con el criterio de maximizar la diversidad explicado anteriormente.

Esta reconstrucción tiene lugar cuando ninguna solución de prueba es admitida en el $RefSet$. Esta mejora añade un mecanismo para reconstruir parcialmente el conjunto de referencia cuando los métodos de combinación y mejora no proveen soluciones de suficiente calidad para desplazar a las actuales del $RefSet$.

3.1.3. Adaptación del Scatter Search al FJSP

Visto lo explicado en la sección anterior, se antoja necesario realizar una adaptación de todos estos conceptos a nuestro caso particular del Fuzzy Job Shop Problem.

En primer lugar, para representar las soluciones de nuestro problema utilizaremos la representación en forma de vector de permutaciones explicada en el capítulo anterior. Esto nos facilitará el trabajo a la hora de generar el conjunto inicial de soluciones en nuestro algoritmo de Scatter Search, puesto que para ello podremos generar un vector de permutaciones con repetición de números consecutivos (teniendo en cuenta el número de trabajos y tareas de nuestro problema) y posteriormente desordenarlo aleatoriamente.

Antes de actualizar el $RefSet$ con el conjunto de soluciones inicial tendremos que evaluar las mismas. Para ello utilizaremos el Algoritmo 3 explicado también

en el capitulo anterior.

A la hora de actualizar el $RefSet$ con las soluciones diversas necesitamos medir distancias entre las mismas. Para ello utilizaremos el Algoritmo 1 para el cálculo de sus representaciones binarias y con las mismas podremos efectuar cálculo de distancias a partir de lo descrito en el Algoritmo 2 del capítulo anterior.

Nuestro $Metodo\ de\ Mejora$ consistirá en una búsqueda local con escalada simple basada en lo expuesto en [5]. En esencia consiste en calcular una vecindad a partir de una solución dada y evaluar esta vecindad en busca de una solución mejor. Con *hill-climbing*, el primer vecino encontrado que sea mejor, reemplaza a la solución original y la búsqueda local se relanza sobre la nueva solución. El procedimiento termina cuando ningún vecino mejora lo encontrado. Claramente, la clave de la búsqueda local pasa por definir la vecindad. En [5] se propuso una nueva vecindad, como resultado de invertir el orden relativo de más de dos tareas consecutivas dentro de un bloque crítico.

Para la combinación de soluciones se aplicará un cruce JOX. El cruce JOX calcula un número aleatorio hasta el cual todos los elementos del vector de permutaciones de la primera solución se mantendrán y el resto los intercambia con los de la otra solución.

3.2. Path Relinking

Path Relinking (PR de ahora en adelante) fue originalmente sugerido como un enfoque que integraba estrategias de intensificación y diversificación en el contexto de búsqueda tabú [10], [11]. Este enfoque genera nuevas soluciones explorando trayectorias que conectan soluciones de alta calidad, empezando desde una de estas soluciones y generando un camino en el espacio de vecindades que conduce a otras soluciones. Esto es conseguido seleccionando movimientos que introducen atributos de las soluciones de destino.

Para generar los caminos deseados, sólo es necesario realizar aquellos movimientos que cumplan lo siguiente: empezando desde una solución inicial, los movimientos deben introducir progresivamente atributos de la solución final (o reducir la distancia entre los atributos de ambas soluciones). Los roles de solución inicial y final son intercambiables; cada solución puede ser inducida a moverse simultáneamente hacia la otra con el fin de generar combinaciones. Primero consideremos la creación de caminos que unen dos soluciones seleccionadas x' y x", centrando nuestra atención en la parte del camino que se encuentra entre las soluciones, produciendo una secuencia $x' = x(1), x(2), ..., x(r) = x$". Para reducir el numero de opciones a considerar, la solución $x(i+1)$ podría crearse a partir de $x(i)$ en cada paso eligiendo un movimiento que minimice el número de movimientos restantes para alcanzar x". El camino construido podría encontrar soluciones peores que la solución inicial o final, pero que den lugar a puntos de acceso para alcanzar otras (algunas mejores) soluciones. Por este motivo, merece la pena examinar las soluciones vecinas a lo largo del camino construido y tomar nota de aquellas que puedan dar lugar a puntos de inicio para lanzar

búsquedas adicionales. En [3] se plantea un esquema básico del procedimiento de Path Relinking, que nosotros reproducimos en el Algoritmo 5.

Algoritmo 5 Algoritmo de Path Relinking

Obtener un $RefSet$ de b soluciones élite.
Evaluar las soluciones de $RefSet$ y ordenarlas de acuerdo con su función de fitness de tal manera que x^1 sea la mejor y x^b la peor.
$NuevasSoluciones = TRUE$.
mientras $NuevasSoluciones$ **hacer**
  Generar $NuevoSubconjunto$ con el *Método de generación de subconjuntos.*
  Hacer $NuevasSoluciones = FALSE$ y $Pool=\emptyset$.
  mientras $NuevoSubconjunto \neq \emptyset$ **hacer**
    Seleccionar el siguiente par $(x', x")$ de $NuevoSubconjunto$.
    Aplicar el método de enlazado para construir la secuencia $x' = x'(1), x'(2), ..., x'(r) = x"$ y añadir las soluciones a $Pool$.
    para i=1 hasta $i < r/NumImp$ **hacer**
      Aplicar el *Método de Mejora* a $x'(i * NumImp)$ y añadir soluciones a $Pool$
    fin para
    Aplicar el método de enlazado para construir la secuencia $x" = x"(1), x"(2), ..., x"(s) = x'$ y añadir las soluciones a $Pool$.
    para i=1 to $i < s/NumImp$ **hacer**
      Aplicar el *Método de Mejora* a $x"(i * NumImp)$ y añadir soluciones a $Pool$
    fin para
    para cada solución $x \in Pool$ **hacer**
      si $x \notin RefSet$ y $f(x) < f(x^b)$ **entonces**
        Hacer $x^b = x$ y reordenar $RefSet$.
        $NuevasSoluciones = TRUE$.
      fin si
    fin para
    Eliminar $(x', x")$ de $NuevoSubconjunto$.
  fin mientras
fin mientras

En el caso de nuestro Fuzzy Job Shop Problem estos movimientos serán inversiones de arcos en el camino crítico de la solución inicial x' tomando como referencia el sentido de los mismos en la solución final $x"$. Esto se debe a que la definición que dimos de distancia entre dos soluciones de nuestro problema (Algoritmo 2) está conforme a lo explicado en el párrafo anterior. Además sabemos por [24] que al invertir un arco crítico (perteneciente a un camino crítico) no se introducen ciclos, es decir, la solución resultante sigue siendo factible. Por tanto, realizaremos las inversiones de arcos del camino crítico de x' para que coincidan con el sentido de los mismos en $x"$ usando como guía sus representaciones binarias. Esto puede dar lugar a cambios en el camino crítico de x', por lo que después de cada inversión éste deberá ser recalculado.

A su vez, de entre todos estos elementos del camino, elegiremos un número de ellos (vendrá dado por un parámetro de entrada) uniformemente distribuidos,

a partir de los cuales se lanzarán búsquedas locales y actualizaremos el conjunto de referencia $RefSet$ con los mismos.

Dadas las características de nuestro problema y requisitos temporales, habrá una serie de puntos a tener en cuenta en la adaptación del Algoritmo 5 para nuestro Fuzzy Job Shop Problem:

- Tendremos que hallar la manera de convertir un arco del camino crítico a la posición equivalente en el vector de su representación binaria.
- Puesto que solo podemos invertir arcos de máquina de una solución, tendremos que obviar aquellos arcos del camino crítico que son del mismo trabajo.
- Para evitar que se demore demasiado el procedimiento, tendremos que establecer una cota al número de inversiones de arcos (o elementos del camino enlazado) a calcular. Para ello, existen varias posibilidades:
  - Terminar una vez que se ha recorrido el camino crítico y que todas las posibles inversiones de arcos han sido realizadas.
  - O bien, establecer un límite de inversiones de arcos basado en la distancia entre ambos extremos del camino.

Teniendo en cuenta lo expuesto previamente y basándonos en el Algoritmo 5, podemos adaptar este algoritmo para nuestro problema en el Algoritmo 6.

Algoritmo 6 Algoritmo de Path Relinking para FJSS

Input: $RefSet$ de b soluciones élite, $TopeInv$ tope de inversiones a realizar.
Evaluar las soluciones de $RefSet$ y ordenarlas de acuerdo con su makespan de tal manera que x^1 sea la mejor y x^b la peor.
$NuevasSoluciones = TRUE$.
mientras $NuevasSoluciones$ **hacer**
  Generar $NuevoSubconjunto$ con el *Método de generación de subconjuntos.*
  Hacer $NuevasSoluciones = FALSE$
  mientras $NuevoSubconjunto \neq \emptyset$ **hacer**
    Seleccionar el siguiente par $(x', x")$ de $NuevoSubconjunto$.
    mientras No se alcanza $TopeInv$ y No se recorre todo el camino crítico **hacer**
      Obtener siguientes componentes (u, v) de Camino crítico de x'
      si (u, v) corresponde a arco de trabajo **entonces**
        Saltar al siguiente par de componentes
      fin si
      si si v se procesa antes que u en x **entonces**
        Inviertir el sentido del arco dado por (u, v) en x'
        Recalcular el camino crítico de x'
        Actualizar la representación binaria de x' con el valor de x"
        Añadir x' a $Pool$
      fin si
    fin mientras
    para i=1 hasta $i < |Pool|/NumImp$ **hacer**
      Aplicar *Método de Mejora* a $Pool(i*NumImp)$
      Ejecutar el *Método de actualización del* $RefSet$ con $Pool(i*NumImp)$

      si $RefSet$ ha cambiado **entonces**
        $NuevasSoluciones = TRUE$
      fin si
    fin para
    Eliminar $(x', x")$ de $NuevoSubconjunto$.
  fin mientras
fin mientras

Capítulo 4

Diseño y Desarrollo

En este capitulo del proyecto se pasarán a describir todos los componentes de la aplicación desarrollada conforme a lo explicado en el capitulo anterior y los objetivos iniciales.

Índice

4.1. Decisiones Tecnológicas

Para la construcción del sistema se ha optado por utilizar el lenguaje de programación C++. Esto se debe principalmente a los requisitos temporales de nuestro problema y al rendimiento que ofrece C++ frente a otros lenguajes como Java y su "Java Virtual Machine". Otro motivo para elegir C++ es facilitar la posterior integración de nuestro código en la librería de algoritmos para scheduling desarrollada por el grupo de investigación GTC (Grupo de Tecnologías de la Computación) de la Universidad de Oviedo.

El entorno de desarrollo utilizado ha sido Eclipse 3.7.2 con el plugin de CDT que provee un entorno de desarrollo completamente funcional en C/C++. Los motivos de esta elección en este caso son la posibilidad de utilizar plugins (como CDT), su carácter gratuito y que goza de un gran soporte en la red.

Sobre el sistema operativo utilizamos Ubuntu 12.04 de 32 bits y la versión del kernel 3.2.0-27. La elección de Linux frente a otras posibilidades se basó en la versatilidad que ofrece a la hora de sacar provecho a la computación masiva

en supercomputadores y clústers. Ésto nos permitiría lanzar nuestra aplicación en una máquina de estas características en caso de desearlo en un futuro.

Se ha aprovechado parte de código existente para resolver el FJSP con un algoritmo memético (genético combinado con búsqueda local). Es por esto que alguna de las clases y/o métodos que aparecen a continuación tienen nombres propios de algoritmos genéticos (cromosoma, gen...). El hecho de encontrarse escrito en lenguaje C++ también representó otra razón de peso para escoger este lenguaje frente a otras posibilidades.

Puesto que se ha seguido una metodología incremental iterativa, se fueron añadiendo diversas características para construir una propuesta de SS+PR para el FJSP de manera iterativa, a partir de una propuesta básica y funcional de SS, evaluando así en cada versión la verdadera aportación de los cambios o elementos adicionales que se incluyen. Se creyó interesante distinguir unas versiones de otras a la hora de analizar los resultados, por lo que se explicarán los diferentes refinamientos que contiene cada una de ellas a continuación.

4.2. Primera Versión: PFCv1

4.2.1. Modelado del Problema y Soluciones

En primer lugar, necesitamos una manera de representar nuestro Fuzzy Job Shop Problem sobre el que vamos a trabajar. Para ello se optamos por crear la clase **proJSS** cuya misión principal es leer los datos del problema de un fichero de texto y almacenarlos en su estructura interna. Ésta consiste básicamente en una matriz de **Trabajos** cada uno de ellos con su duración, recurso que utiliza, tiempo mínimo de inicio y tiempo máximo de finalización.

Por otro lado, habrá que modelar las diferentes soluciones de nuestro problema. Basándonos en lo explicado en la sección 2.4, crearemos una clase **crJSS** que, entre otras cosas, albergará:

- Representación de la solución en la forma vector de permutaciones con repetición.
- Planificación actual de la solución. Se trata a su vez de una instancia de la clase **plan2** que contiene:
  - las tareas previa y siguiente en máquina para todas ellas.
  - las cabezas y colas de cada tarea.
- Planificación temporal de las tareas correspondientes al vector de permutaciones con repetición dado.
- Valor de la función de evaluación (o makespan).

Para poder evaluar los **crJSS** mediante el Algoritmo 3, creamos la clase **opFitGYT**. Esta clase contiene los métodos necesarios para planificar una solución, calcular su makespan y evaluar si es solucion al problema dado.

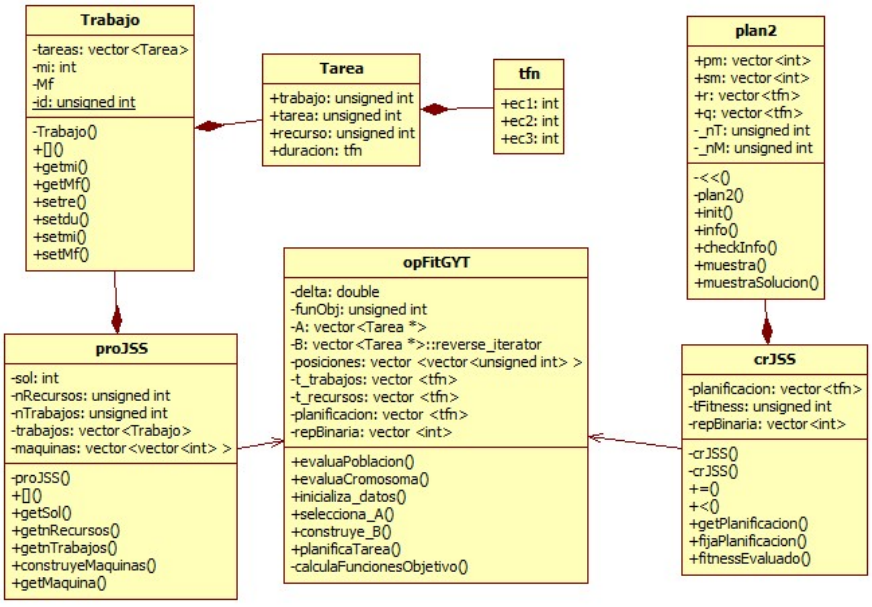

Figura 4.1: Diagrama UML del Job Shop Problem

Además, dada la naturaleza de duraciones inciertas en nuestro problema, fue necesario crear una clase **tfn** (*triangular fuzzy number*) que implementase la funcionalidad de los intervalos inciertos (o difusos). Fundamentalmente, es una estructura que almacena los tres valores del número triangular (a_1, a_2, a_3) y las operaciones típicas redefinidas, según se explicó en la Sección 2.2.

Un diagrama UML de lo explicado hasta ahora lo podemos encontrar en la Figura 4.1. Todas estas clases están basadas en código ya existente de un Algoritmo genético para el FJSP.

4.2.2. Modelado del Scatter Search

Dada la naturaleza del Scatter Search y el uso de conjuntos de soluciones en su funcionamiento, optamos por crear una clase **poblacion**. Ésta es basicamente una clase contenedor de un vector de cromosomas con diversas operaciones para modificar tamaño y obtener el mejor y peor individuo de la misma.

Siguiendo las pautas de lo planteado en el Capítulo 3 sobre Scatter Search, lo que se antoja más sencillo es implementar primero los "cinco métodos" y más tarde ponerlos en funcionamiento mediante el Algoritmo 4.

A continuación pasamos a explicar la implementación realizada de cada uno de los "cinco métodos":

- Método de generación diversificada

Implementado a través de la clase **opGenJSS**, se encarga de generar soluciones aleatorias a partir de un **proJSS** dado. Para ello, rellena aleatoriamente el vector de representación de cada uno de los **crJSS** de los que se compone la **poblacion**. de acuerdo con la definición que se dió para esa representación.

- Método de actualización del conjunto de referencia

  Creamos la clase **actRefset**. Básicamente se encarga de ordenar de menor a mayor *makespan* las soluciones de una **poblacion** y tomar las b primeras (b es el tamaño del conjunto de referencia) para guardarlas en el conjunto de referencia. También cabe la posibilidad de actualizar el conjunto de referencia a partir de un único **crJSS**, para lo cual tiene que ir comparando uno por uno y reemplazar con desplazamiento en caso de ser necesario.

- Método de generación de subconjuntos

  Implementado mediante la clase **subsetGen**. Para ello guarda todas las posibles combinaciones de parejas de elementos del conjunto de referencia como pares de índices en un vector.

- Método de combinación de soluciones

  La clase **opCruJOX** aplica un cruce JOX a dos soluciones **crJSS** para dar lugar a dos soluciones hijas. La implementación se basa en lo explicado en el capítulo anterior acerca del cruce JOX.

Puesto que el *Método de mejora* no era estrictamente necesario para el funcionamiento del Scatter Search, éste se incorporó en versiones posteriores.

Con todo esto, no queda más que construir el Algoritmo 4 utilizando las clases descritas anteriormente. Esto lo llevamos a cabo en la clase **prueba**.

Todo esto conforma nuestra primera versión del código **PFCv1**. Podemos encontrar un diagrama UML del mismo en la Figura 4.2.

4.3. Segunda Versión: PFCv2

Nuestra segunda iteración consiste en considerar la diversidad de soluciones en nuestro conjunto de referencia aparte de su calidad. Es decir, el conjunto de referencia contendrá soluciones de alta calidad y muy diversas a partes iguales. Para ello tendremos que modificar el *Método de actualización del conjunto de referencia* y por lo tanto los métodos pertinentes de la clase **actRefset**.

Se sustituye la generación del $RefSet$ inicial por la correspondiente al método de actualización del $RefSet$ explicado en la Sección 3.1.1, de manera que se introduzcan b_1 soluciones de calidad y b_2 soluciones diversas. La aparición del concepto de distancia entre soluciones en lo recientiemente sustituido conlleva alguna modificación adicional. Tal y como vimos en la Sección 2.6, para calcular la distancia entre dos soluciones es necesario conocer la representación binaria de las mismas. Es por esto que en la clase **opFitGYT** tendremos que incluir unas lineas de código que calculen la representación binaria de cada **crJSS** (según el

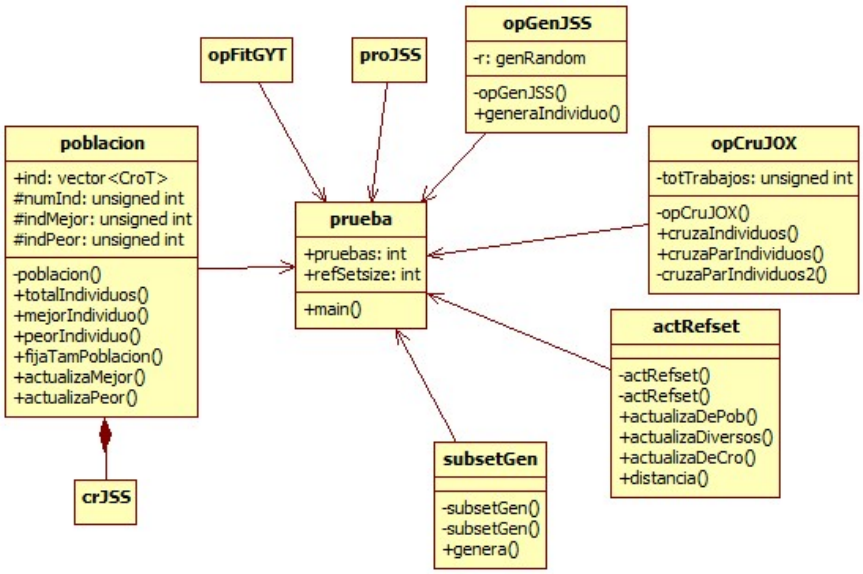

Figura 4.2: Diagrama UML del Job Shop Problem con Scatter Search

Algoritmo 1) y lo almacenen en un campo del propio cromosoma para su futuro uso.

Todo esto junto con lo explicado en la sección anterior, conforma nuestra segunda versión del código **PFCv2**.

4.4. Tercera Versión: PFCv3

En la tercera iteración añadiremos la reconstrucción del conjunto de referencia (ver Sección 3.1.2) a nuestro Algoritmo de Scatter Search. Para ello tendremos que modificar nuestra clase principal **prueba** y la clase **actRefset** para que incluya un método que actualice nada más los individuos diversos de nuestro conjunto de referencia. El funcionamiento de ambas modificaciones será conforme a lo que se explicó en la Sección 3.1.2.

Todo esto junto con lo explicado en la sección anterior, conforma nuestra tercera versión del código **PFCv3**.

4.5. Cuarta Versión: PFCv4

En esta iteración incluiremos el *Método de mejora* del Scatter Search ya implementado. Éste se basa en lo planteado durante el capitulo anterior cuando hablamos de la adaptación del SS a nuestro problema particular.

Así, nuestra clase **opFitGYT** será reemplazada por **opFitGYTEsc2**, que implementa lo mismo que la anterior pero añade la búsqueda local a cada **crJSS**

evaluado. Esta nueva clase está basada en el código del trabajo previo con algoritmos genéticos para FJSP y no es trabajo original de este PFC.

Todo esto junto con lo explicado en la sección anterior, conforma nuestra cuarta versión del código **PFCv4**.

4.6. Quinta Versión: PFCv5

Para esta iteración añadimos toda la funcionalidad del Path Relinking. Ésta concuerda con lo planteado previamente en el Algoritmo 6.

En primer lugar, necesitamos una manera de obtener el camino crítico de cada **crJSS**. Para ello, podemos modificar la parte añadida en la versión anterior (**PFCv4**) para el cálculo de bloques críticos y extenderlo a un camino completo. Éste se almacenará en una variable que guardaremos con cada **crJSS** de una manera similar a lo que se hizo para la representación binaria en la segunda versión del programa.

El siguiente paso será sustituir la parte actual de cruce JOX por una implementación del Algoritmo 6. La clase **pathRelinking** contiene los métodos utilizados por dicho algoritmo tales como:

- Función que calcula la posición equivalente en la representación binaria de un arco a partir de sus dos componentes. Ésto se hizo posible estudiando el orden que sigue el Algoritmo 1 para rellenarla y así poder acceder a la posición adecuada.
- Función para invertir el sentido de un arco de máquina de un **crJSS** dadas sus dos componentes. Para ello recalcula la planificación actual del cromosoma reutilizando lo ya implementado para el cálculo de vecindades en la búsqueda local de la versión anterior.
- Función que recalcula camino crítico y makespan de un **crJSS** una vez se ha realizado una inversión de arco.
- Función que lanza una búsqueda local a un **crJSS** como *Método de mejora* del Path Relinking.

Dejaremos como parámetros modificables a elección del usuario la variable $NumImp$ que dictaba cada cuantos elementos del camino se aplicaba el *Método de mejora* y la cota máxima de elementos recorridos que será un divisor de la distancia entre ambas soluciones.

Todo esto junto con lo explicado en la sección anterior, conforma nuestra quinta versión del código **PFCv5**. Podemos encontrar un diagrama UML del mismo en la Figura 4.3.

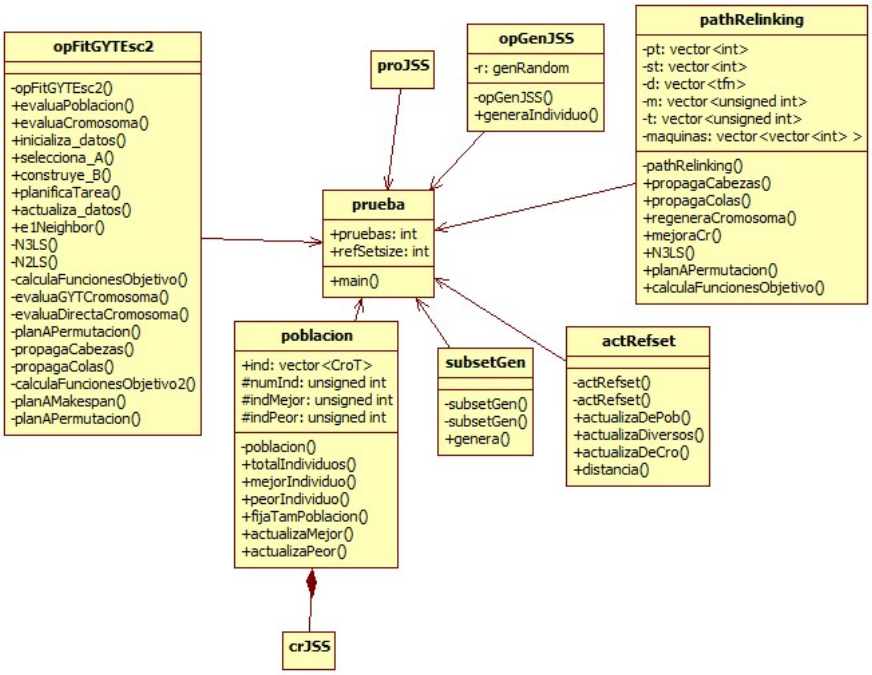

Figura 4.3: Diagrama UML del Job Shop Problem con Path Relinking

Capítulo 5

Resultados

Índice

5.1. Diseño de Pruebas

Dado el carácter estocástico (no determinista) de nuestro algoritmo metaheuristico, lanzaremos varias ejecuciones del mismo para obtener resultados valorables. En nuestro caso serán 30 ejecuciones.

Para el *input* de las pruebas, usaremos un conjunto de instancias de problemas considerados como díficiles de resolver a modo de *benchmark*: FT10 (tamaño 10 x 10) y FT20 (20 x 5), La11, La12, La21, La24, La25 (15 x 10), La27, La29 (20 x 10), La38, La40 (15 x 15), y ABZ7, ABZ8, ABZ9 (20 x 15). Cada una de estas instancias será ejecutada sobre las diferentes versiones de nuestro código (*PFCv1*,*PFCv2*,*PFCv3*,*PFCv4*,*PFCv5*). De cada uno de estos problemas anotaremos varios resultados de cada ejecución:

- Valor esperado del makespan de la mejor solución entre todas las instancias lanzadas.
- Valor esperado del makespan de la peor solución entre todas las instancias lanzadas.
- Media aritmética de los mejores makespan de cada instancia lanzada.
- Varianza de los mejores makespan de cada instancia lanzada.
- Tiempo medio de una ejecución entre todas las instancias lanzadas.

Como parámetros, hemos decidido fijar el tamaño del *Refset* a 10 individuos y el de la población inicial a diez veces éste último valor. Se han elegido estos valores en línea con lo que suele ser más habitual en la literatura consultada.

5.2. Resultados Experimentales

El equipo en el que se han ejecutado las siguientes pruebas cuenta con un procesador Intel Core i5 M460 @ 2.53GHz y 4GB de memoria RAM.

Cuadro 5.1: Resultados experimentales

Problema	PFCv	Ec				Tiempo
		Mín	Med	Máx	Var	(ms)
ft10	V1	1176.75	1293.29	1485	4738.93	881182
	V2	1127.75	1266.53	1362	3452.2	2349729
	V3	1134	1254.22	1404.25	4760.31	3704521
	V4	937.75	965.53	998.25	210.23	3607293
	V5	936.75	970.65	1001	255.22	8300709
ft20	V1	1496	1730	1581.11	3092.35	1970568
	V2	1447	1579.72	1663.25	2288.8	5579393
	V3	1473	1599.53	1748.25	4015.55	9057008
	V4	1180.5	1218.53	1276	648.40	8374601
	V5	1182	1239.06	1328.5	859.34	9519767
la11	V1	1264	1443	1328.03	3350.03	826298
	V2	1245	1312.1	1424	1630.86	4601981
	V3	1251	1316.17	1414	1492.73	8360610
	V4	1222	1222	1222	0	7022237
	V5	1222	1222	1222	0	7969839
la12	V1	1062.5	1275	1184.98	5003.16	629163
	V2	1054.75	1160.68	1308	3454.89	4396427
	V3	1059.25	1146.48	1270.75	2512.08	8380514
	V4	1040.75	1040.81	1042.5	0.09	7066897
	V5	1040.75	1040.75	1040.75	0	8122609
la21	V1	1338.75	1544.25	1425.77	3005.21	1408454
	V2	1259.25	1404.97	1505	3069.89	5104777
	V3	1308.5	1412.78	1521.75	3201.57	8856308
	V4	1053.25	1081.73	1124.25	241.92	9354213
	V5	1060	1103.38	1149.75	403.99	10870813
la24	V1	1148.5	1406.75	1297.06	4836.6	1100105
	V2	1121	1250.78	1381	4456.85	5042189
	V3	1165	1276.88	1363.25	3018.26	8775972
	V4	948.25	966.61	1005	140.92	9169386
	V5	966.25	989.76	1026	210.79	10307408
la25	V1	1217.75	1475	1336.31	4384.73	994485
	V2	1203.5	1326.27	1456	4231.2	5019369
	V3	1142	1307.68	1399	2795.9	8775223
	V4	985	1009.31	1048	198.62	9220391
	V5	1006.5	1034.19	1068.25	278.52	10573426
la27	V1	1537.25	1742	1647.35	2877.36	2190194
	V2	1516.5	1661.41	1852.25	5979.67	12990759
	V3	1542	1666.85	1753	2839.15	24382716
	V4	1266.25	1302.28	1328	288.28	24970211
	V5	1294.5	1339.21	1382.75	417.57	28154294
la29	V1	1457.5	1832.5	1577.47	7452.5	1881375
	V2	1439.5	1567.01	1690	2870.11	12985684
	V3	1382	1563.58	1756	5523.3	24128145
	V4	1231.71	1265	1297.75	254.68	25265843
	V5	1206	1263.55	1336.75	818.76	28523583

la38	V1	1538.25	1712.5	1633.57	2724.06	2198760
	V2	1516.25	1605.47	1708.5	2868.86	258886752
	V3	1484.5	1617.93	1813	3919.97	15127855
	V4	1246	1271.19	1309	204.11	16944234
	V5	1286.5	1328.8	1369	325.17	18760466
la40	V1	1481.25	1731.75	1606.85	4554.31	1586079
	V2	1513.25	1639.51	1756.25	4326.48	8560020
	V3	1476.25	1605.23	1740.25	3475.49	15358093
	V4	1246.25	1261.86	1287.5	126.51	18264093
	V5	1263	1289.98	1314.75	277.66	19078573
abz7	V1	837	938.5	880.33	578.98	3105349
	V2	809.75	867.50	972.25	1036.94	27823092
	V3	808.75	863.85	972.25	1136	52643787
	V4	679	694.95	713.75	64.64	55425824
	V5	697	711.05	742.25	106.96	53500956
abz8	V1	845.25	958.25	905.99	712.19	3598275
	V2	863	905.47	972.25	738.40	28217291
	V3	837	909.6	991	907.96	52310381
	V4	694.75	710.65	727.5	64.67	56758737
	V5	701.25	725.02	758	137.21	63080879
abz9	V1	882	1031.25	943.3	1096.27	3762839
	V2	867	939.32	1007	1078.95	27795765
	V3	879.5	948.54	1007	1149.55	52464332
	V4	707.75	733.74	759.75	142.47	56163926
	V5	729.25	759.05	784.5	206.46	62908595

5.3. Análisis de Resultados

A la luz de lo presentado en la sección anterior, podemos efectuar un análisis en busca de patrones en los resultados obtenidos e intentaremos encontrar razones que expliquen los mismos.

5.3.1. Análisis en función del Makespan

En primer lugar y con motivo de ofrecer una visión más clara de los resultados obtenidos en el Cuadro 5.1, llevamos a cabo unos cálculos intermedios para cuantificar la mejora del *makespan* en cada caso. Estos consistieron en calcular el porcentaje de mejora del makespan medio entre versiones de la aplicación para todos los problemas de prueba. Dependiendo de las versiones que se tomasen como referencia para evaluar la mejora, se obtuvieron dos tablas de porcentajes:

- Cuadro 5.2: la mejora de cada versión con respecto a la versión de partida (**PFCv1**). Puede verse como la calidad del método implementado en cada versión respecto a una cota inferior.
- Cuadro 5.3: la mejora de cada versión respecto de la anterior. Permite cuantificar la aportación de la componente que se ha incluido en cada versión que no había en la anterior (mide la mejora que genera el único

cambio introducido) por lo que nos permite evaluar la bondad de cada una de las componentes.

Además, para cada versión de la aplicación, se calculó la media de los porcentajes de mejora y se marcó en gris la casilla en la que se había registrado un mayor porcentaje de mejora.

La mejora introducida entre nuestras versiones **PFCv1** y **PFCv2** consiste en considerar también la diversidad entre las soluciones pertenecientes al *Refset*. Como se puede ver en el Cuadro 5.1, para toda la batería de problemas observamos una mejora en el *makespan* medio entre ambas soluciones y para más de la mitad de problemas la varianza se reduce igualmente. Más concretamente, según el Cuadro 5.2, observamos que el *makespan* mejora en media un 0.9 % llegando hasta 3.57 % en algún problema. Una explicación para esto puede encontrarse en la ampliación del espacio de búsqueda a regiones alternativas gracias a las soluciones "diversas" del *Refset* que pueden dar lugar a soluciones "buenas" en sitios inesperados. En cambio, si nos limitamos a soluciones de "buena" calidad, nuestro *Refset* puede ser rápidamente poblado por soluciones muy parecidas que nos restringen considerablemente nuestro espacio de búsqueda, pudiendo dar lugar a una convergencia prematura de la búsqueda, obteniendo así óptimos locales como solución final.

Entre las versiones **PFCv2** y **PFCv3** introducimos la reconstrucción del *Refset* en caso de no encontrarse soluciones nuevas. Sin embargo, como podemos ver en el Cuadro 5.1, en casi ningún problema de prueba notamos una mejora del *makespan* obtenido. Esto se puede ver de una manera más clara en el Cuadro 5.3 donde podemos observar que el porcentaje de mejora medio es negativo con un valor de -0.03 %. Una posible explicación para esto puede encontrarse en el hecho de que la diversidad que se introduce mediante la reconstrucción del *Refset* desaparece inmediatamente en la siguiente iteración pues cualquier solución generada con una mínima "calidad" va a sustituir a estas diversas.

Para la versión **PFCv4** introducimos la mejora con búsqueda local basada en invertir el orden de tareas de un bloque crítico. Aquí sí que observamos una mejora notable en el *makespan* para todos los problemas de prueba, mucho mayor que en casos anteriores. Según el Cuadro 5.2, obtenemos como media una mejora del 20.94 % con valores máximos de 25.48 %. Esto se debe a que la búsqueda local proporciona una intensificación de la búsqueda en zonas prometedoras (aquellas donde se encuentran las soluciones del $RefSet$). Para ello, se explora la vecindad de las soluciones iniciales en busca de elementos de mayor calidad.

Finalmente, con la versión **PFCv5** se incluyó un Path Relinking como cruce entre soluciones del *Refset*. Los resultados obtenidos en esta versión, siendo notablemente mejores que los obtenidos en las primeras versiones (Cuadro 5.2 donde la media de mejora es 18.26 % frente a 0.90 % y 0.88 %), no superan los alcanzados por **PFCv4** y su búsqueda local en ninguno de los casos. Como prueba de esto último tenemos el Cuadro 5.3, donde podemos ver que el porcentaje de mejora media tiene un valor de -3.46 %. Entendemos que una posible expli-

Problema	Porcentaje			
	V2	V3	V4	V5
ft10	2.07	3.02	25.34	25.05
ft20	0.09	-1.17	22.93	19.12
la11	1.20	0.89	7.98	7.98
la12	2.05	3.25	12.17	12.17
la21	1.46	0.91	24.13	21.50
la24	3.57	1.56	25.48	22.45
la25	0.75	2.14	24.47	21.12
la27	-0.85	-1.18	22.35	18.71
la29	0.66	0.88	19.81	18.62
la38	1.72	0.96	22.18	18.66
la40	-2.03	0.10	21.47	18.02
abz7	1.46	1.87	21.06	16.38
abz8	0.06	-0.40	21.56	17.68
abz9	0.42	-0.56	22.22	18.15
MEDIA	**0.90**	**0.88**	**20.94**	**18.26**

Cuadro 5.2: Tabla de porcentajes de mejora de *makespan* medio respecto a V1

cación para esta falta de mejora es la falta de equilibrio entre intensificación y diversificación existente en la búsqueda de **PFCv5**. En efecto, en las versiones anteriores, el operador de combinación JOX tiene un efecto implícito de mutación que proporciona diversidad a la búsqueda y en concreto para **PFCv4** esta diversidad se complementa con la intensificación de la búsqueda local. Entendemos que el efecto diversificador queda eliminado en **PFCv5** al considerar la combinación mediante path-relinking, tal y como se ha definido para esta versión. Una posible solución para un futuro sería modificar el operador de combinación para reforzar la diversificación, bien mediante una ligera mutación de los individuos del camino generado por el algoritmo de path-relinking, bien mediante técnicas de path-relinking más sofisticadas, tal y como se describe en [3].

Para una visión más clara de los porcentajes de mejora medios del *makespan* en cada uno de los problemas y versiones de la aplicación, se elaboraron dos gráficos de barras con los datos de los Cuadros 5.2 y 5.3. Éstos pueden verse representados en las Figuras 5.1 y 5.2 respectivamente.

5.3.2. Análisis Temporal

Cuando se usan técnicas metaheurísticas para resolver problemas de optimización combinatoria, como es el caso del Fuzzy Job Shop, un factor a tener en cuenta, además de la bondad de la solución, es el tiempo de computación necesario para obtener dicha solución. Por ello, no sólo debemos analizar las distintas versiones aquí propuestas en términos de valores de maskespan, sino también en términos de carga computacional.

Problema	Porcentaje			
	V2	V3	V4	V5
ft10	2.07	0.97	23.02	-0.40
ft20	0.09	-1.25	23.82	-4.95
la11	1.20	-0.31	7.15	0.00
la12	2.05	1.22	9.22	0.01
la21	1.46	-0.56	23.43	-3.46
la24	3.57	-2.09	24.30	-4.07
la25	0.75	1.40	22.82	-4.44
la27	-0.85	-0.33	23.26	-4.70
la29	0.66	0.22	19.10	-1.48
la38	1.72	-0.78	21.43	-4.53
la40	-2.03	2.09	21.39	-4.39
abz7	1.46	0.42	19.55	-5.93
abz8	0.06	-0.46	21.87	-4.95
abz9	0.42	-0.98	22.65	-5.22
MEDIA	**0.90**	**-0.03**	**20.21**	**-3.46**

Cuadro 5.3: Tabla de porcentajes de mejora de *makespan* medio respecto a versión previa

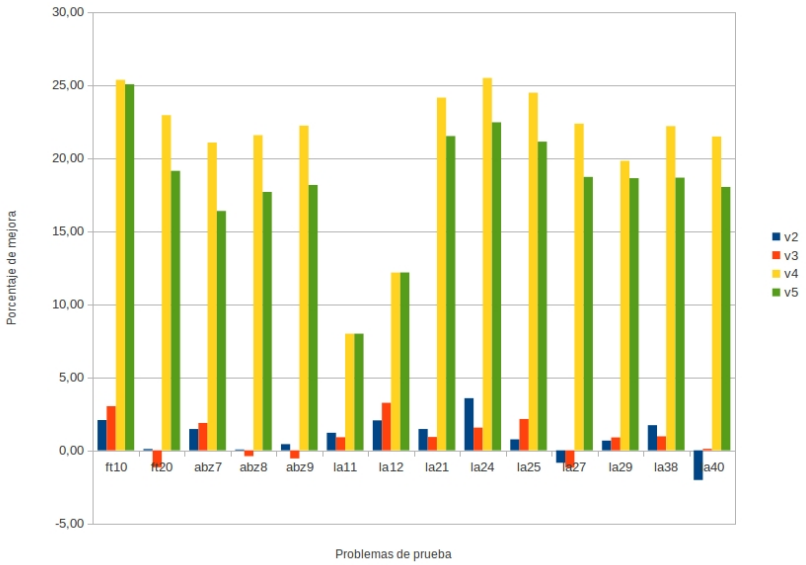

Figura 5.1: Gráfico de barras del Cuadro 5.2.

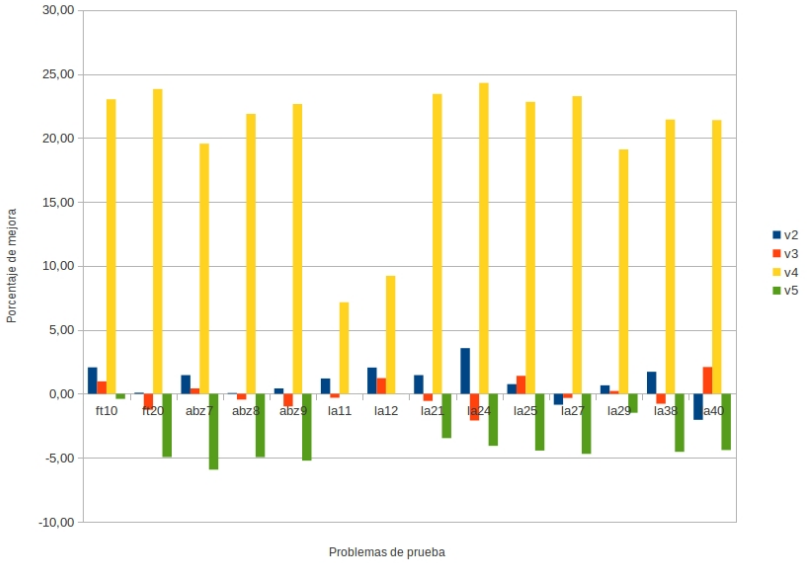

Figura 5.2: Gráfico de barras del Cuadro 5.3.

Claramente, para una misma versión del algoritmo, habrá grandes diferencias en los tiempos invertidos para resolver los distintos problemas de nuestro banco de pruebas, dada la variedad de tamaños y niveles de dificultad. Ésto se debe, en primer lugar, a que al aumentar el tamaño aumentan las posibles combinaciones de órdenes de procesamiento, es decir, aumenta el tamaño del espacio de búsqueda. Además, dentro de un mismo tamaño, no todos los problemas presentan el mismo nivel de dificultad y los costes de operaciones tales como evaluación de soluciones, cálculo de representaciones binarias, cálculo de distancias, etc son mayores cuanto mayor es el tamaño del problema.

La tendencia que se observa analizando los tiempos de ejecución de las diferentes versiones para un mismo problema es que éstos aumentan a medida que incrementamos el número de versión ejecutada. Esto se debe principalmente a que los cambios introducidos en las sucesivas versiones buscan incrementar la proporción del espacio de búsqueda explorado y por lo tanto manejan un mayor número de individuos. Sin embargo, como podemos ver en los resultados obtenidos, un mayor tiempo de ejecución no significa estrictamente un mejor resultado final, puesto que puede explorar regiones del espacio donde no haya soluciones de buena calidad o puede converger a óptimos locales.

En la Figura 5.3 se muestran los tiempos medios de una ejecución de entre todas las instancias de problemas para cada versión de nuestro código.

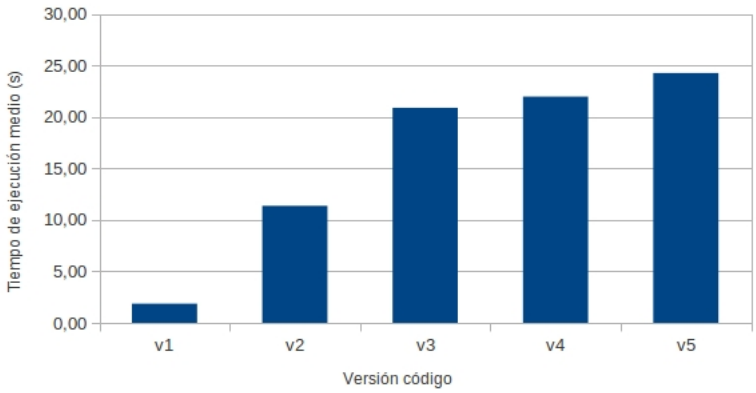

Figura 5.3: Gráfico de barras de tiempos de ejecución medios

Capítulo 6

Conclusiones y Trabajo Futuro

Índice

6.1. Conclusiones

En este proyecto de fin de carrera se ha diseñado e implementado una aplicación que permite resolver el Fuzzy Job Shop Problem mediante un algoritmo de Scatter Search con Path Relinking.

Dada la falta de conocimientos previos en algunos de los temas a estudiar, se tuvo que utilizar como fuente no un libro de texto, sino la literatura científica, por tratarse de un tema de investigación actual. Así, primero se estudió el problema del Job Shop con duraciones inciertas para tener conocimientos suficientes del mismo y poder tomar así decisiones razonadas más adelante. A continuación, se estudió y trató la técnica de búsqueda metaheurística que se iba a utilizar para buscar soluciones, Scatter Search, y se adaptó a nuestro problema en particular. Igualmente, se realizó el mismo proceso para la extensión Path Relinking, que se incluyó junto a lo ya presentado del Scatter Search. Todo ésto se plasmó en un diseño elaborado siguiendo una metodología incremental iterativa, lo que dió lugar a distintas versiones con diferentes refinamientos. Finalmente, estas versiones fueron sometidas a pruebas utilizando una batería de problemas conocidos por su dificultad y sus resultados fueron analizados en busca de patrones.

Por tanto, se ha llevado a cabo un trabajo de investigación en el que se han estudiado y adaptado técnicas de la literatura científica para resolver un problema de cierta dificultad. Como consecuencia, se ha elaborado una aplicación funcional que lee un problema de Job Shop con duraciones inciertas desde un fichero de texto y aplica una metodología metaheurística basada en Scatter

Search con la extensión de Path Relinking. De cada ejecución se recogen no sólo las soluciones proporcionadas (órdenes de procesamiento y tiempos de inicio para cada tarea del problema), sino también diversas medidas del makespan obtenido y de su tiempo de ejecución total.

Sobre los resultados obtenidos, las mejoras que se fueron añadiendo sucesivamente a nuestro algoritmo de Scatter Search efectivamente reflejaron una mejoría sobre los resultados experimentales. Sin embargo, este comportamiento no se repitió a la hora de incluir el Path Relinking en nuestro algoritmo de búsqueda.

6.2. Trabajo Futuro

Como continuación de este proyecto, podrían desarrollarse las siguientes lineas de investigación:

- Estudiar mediante un análisis paramétrico la influencia de los parámetros del algoritmo (tamaño *Refset*, tamaño población inicial, porcentaje de Path Relinking, etc.) en los resultados finales. Debido a los requisitos computacionales de una experimentación de este tipo, sería deseable poder lanzar la experimentación en un equipo de mejores característica técnicas.
- Analizar más a fondo la parte del Path Relinking para encontrar soluciones a su relativo mal rendimiento.
- Explorar nuevas estrategias para el Scatter Search y el Path Relinking que puedan llevar a mejores resultados, como puede ser la hibridación de la Scatter Search con búsqueda tabú.

Bibliografía

[1] Mattfeld, D. (1995). Evolutionary Search and the Job Shop. *Springer-Verlag*, Berlin. p.68.

[2] Bierwirth, C., Mattfeld, D. (1999). Production Scheduling and Rescheduling with Genetic Algorithms. *Evolutionary Computation*, 7, 1-17.

[3] Marti, R., Laguna, M., Glover, F. (2006). Principles of scatter search. *European Journal of Operational Research*, 169, 359-372,

[4] Pinedo, M. (2008). Scheduling: Theory, Algorithms, and Systems. *Springer*.

[5] Puente, J., Vela, C. R., González Rodríguez, I. (2010). Fast Local Search for Fuzzy Job Shop Scheduling. *Proceedings of the 2010 conference on ECAI 2010: 19th European Conference on Artificial Intelligence*, 739-744, IOS Press Amsterdam, The Netherlands.

[6] Palacios, J., González-Rodríguez, I., Vela, C. R., Puente, J. (2011). Particle Swarm Optimisation for Open Shop Problems with Fuzzy Durations. *Proceeding IWINAC'11 Proceedings of the 4th international conference on Interplay between natural and artificial computation* - Volume Part I, 362-371, Springer-Verlag Berlin.

[7] Glover, F., Laguna, M. (2000). Fundamentals of Scatter Search and Path Relinking. *Control and Cybernetics*, 39, 653-684.

[8] Glover, F. (1977). Heuristics for Integer Programming Using Surrogate Constraints. *Decision Sciences*, 8, 156-166.

[9] Glover, F. (1998). A Template for Scatter Search and Path Relinking, in *Artificial Evolution*, Lecture Notes in Computer Science 1363, J.-K. Hao, E. Lutton, E. Ronald , M. Schoenauer and D. Snyers (Eds.)

[10] Glover, F., Laguna, M. (1997). Tabu Search, *Kluwer Academic Publishers*, Boston.

[11] Glover, F., Laguna, M. (1993). Integrating Target Analysis and Tabu Search for Improved Scheduling Systems. *Expert Systems with Applications*, 6, 287-297.

[12] Herroelen, W., Leus, R. (2005). Project scheduling under uncertainty: Survey and research potentials, *European Journal of Operational Research*, 165, 289-306.

[13] Fortemps, P. (1997). Jobshop scheduling with imprecise durations: a fuzzy approach, *IEEE Transactions of Fuzzy Systems*, 7, 557–569.

[14] Tavakkoli-Moghaddam, R., Safei, N., Kah. M.M. O. (2008). Accessing feasible space in a generalized job shop scheduling problem with the fuzzy processing times: a fuzzy-neural approach, *Journal of the Operational Research Society*, 59, 431–442.

[15] Sakawa, M., Kubota, R. (2000). Fuzzy programming for multiobjective job shop scheduling with fuzzy processing time and fuzzy duedate through genetic algorithms, *European Journal of Operational Research*, 120, 393–407.

[16] Petrovic, S., Fayad, C., Petrovic, D., Burke, E., Kendall, G. (2008). Fuzzy job shop scheduling with lot-sizing, *Annals of Operations Research*, 159, 275-292.

[17] González-Rodríguez, I., Puente, J., Vela, C. R., Varela, R. (2008). Semantics of schedules for the fuzzy job shop problem, *IEEE Transactions on Systems, Man and Cybernetics*, Part A, 38(3), 655–666.

[18] González-Rodríguez, I., Vela, C. R., Hernández-Arauzo, A., Puente, J. (2009). Improved local search for job shop scheduling with uncertain durations, in *Proceedings of the Nineteenth International Conference on Automated Planning and Scheduling* (ICAPS-2009), 154–161, Thesaloniki, AAAI Press.

[19] Dubois, D., Prade, H. (1986). Possibility Theory: An Approach to Computerized Processing of Uncertainty, *Plenum Press*, New York (USA).

[20] Nguyen, H. T., Walker, E. A. (2000). A First Course in Fuzzy Logic. *Chapman & Hall*.

[21] Heilpern, S. (1992). The expected value of a fuzzy number. *Fuzzy Sets and Systems*, 47, 81-86.

[22] Dubois, D., Fargier, H., Fortemps, P. (2003). Fuzzy scheduling: Modelling exible constraints vs. coping with incomplete knowledge. *European Journal of Operational Research*, 147, 231–252.

[23] Giffler, B.,Thompson, G. L. (1960). Algorithms for solving production scheduling problems. *Operations Research*, 8, 487–503.

[24] González Rodríguez, I., Puente, Varela, R. (2008). A new local search for the job shop problem with uncertain durations, in *Proc. of the Eighteenth International Conference on Automated Planning and Scheduling (ICAPS-2008)*, 124-131, Sidney. AAAI Press.

[25] Nakano, R., Yamada, T. (1991). Conventional Genetic Algorithm for Job Shop Problems.*Belew and Booker* (eds.), 474-479

[26] Marti, R., Laguna, M. (2003). Scatter Search: Diseño Básico y Estrategias Avanzadas, *Revista Iberoamericana de Inteligencia Artificial*, 19, 123–130.