Universidad de Oviedo

DEPARTAMENTO DE INFORMÁTICA

Metaheurísticas para problemas de scheduling con múltiples recursos

Tesis Doctoral

Programa de Doctorado en Ingeniería Informática

Autor
Raúl Mencía Cascallana

Director
José Ramiro Varela Arias

Febrero 2017

La memoria titulada ”Metaheurísticas para problemas reales de scheduling con múltiples recursos”, que presenta D. Raúl Mencía Cascallana para optar al grado de doctor, ha sido realizada dentro del programa de doctorado en Informática de la Universidad de Oviedo bajo la dirección del doctor D. José Ramiro Varela Arias.

La tesis ha sido financiada a través de los proyectos TIN2013-46511-C2-2-P y TIN2016-79190-R del Ministerio de Economía y Competitividad, y por el contrato de transferencia FUO-EM-331-15 firmado entre la Universidad de Oviedo y la empresa INFORCYL, S.A.

El director,

José Ramiro Varela Arias

Gijón, 10 de Enero de 2017

A mis padres y a mis abuelos

Agradecimientos

Estoy muy agradecido a mi director de tesis, Ramiro Varela Arias, por su gran dedicación a este proyecto, sin la cual esta tesis no hubiera sido posible. También quiero agradecer todo el apoyo y los consejos de mi hermano, Carlos Mencía, que siempre me sirvió de ejemplo durante mis estudios. No quiero olvidarme de agradecer a mis compañeros del Grupo de Investigación en Optimización y Scheduling Inteligentes de la Universidad de Oviedo su ayuda prestada en los últimos años.

Quiero agradecer el apoyo de mi familia y mis amigos.

Esta tesis doctoral ha sido financiada a través de los proyectos TIN2013-46511-C2-2-P y TIN2016-79190-R del Ministerio de Economía y Competitividad, y por el contrato de transferencia FUO-EM-331-15 firmado entre la Universidad de Oviedo y la empresa INFORCYL, S.A.

Resumen

Los problemas de *scheduling* consisten en organizar en el tiempo un conjunto de operaciones que compiten por el uso de un conjunto limitado de recursos, todo ello sujeto a una serie de restricciones y tratando de optimizar una o varias funciones objetivo. Se trata de problemas que aparecen con profusión en muchos entornos sociales y productivos, y en general presentan una alta complejidad computacional. Por estas razones, los problemas de scheduling tienen un gran interés tanto desde el punto de vista económico como desde el punto de vista académico, por lo que han sido objeto de investigación intensiva durante las últimas décadas por parte de las comunidades de Investigación Operativa e Inteligencia Artificial.

Esta tesis tiene como objetivo avanzar en el estudio y resolución de problemas de scheduling en los que las tareas requieren el uso de varios tipos de recursos, materiales y humanos. Concretamente, consideramos dos familias de problemas: en primer lugar algunas variantes de uno de los problemas más conocidos en la literatura como es el job shop scheduling (JSP), en las que cada tarea requiere no solamente una máquina para realizarse, sino también un operario que la asista durante todo su tiempo de procesamiento. El segundo problema es un caso particular del problema de asignación de vehículos y conductores a rutas de transporte de viajeros, conocido en la literatura como VCSP (Vehicle and Crew Scheduling Problem). Dada la alta complejidad de estos problemas y el tamaño de las instancias de interés, consideramos que las metaheurísticas constituyen una aproximación razonable. Así, utilizaremos metaheurísticas híbridas que combinen métodos basados en poblaciones, como los algoritmos genéticos, en la fase de diversificación con métodos de búsqueda, como algoritmos de escalada o búsqueda tabú, en la fase de intensificación.

Como metodología de trabajo, proponemos el uso de modelos gráficos disyuntivos para representar instancias de los problemas y sus soluciones. Estos modelos son útiles para estudiar las propiedades de los problemas, para diseñar algoritmos de generación de planificaciones o schedules, y también para diseñar estructuras de vecindad. Un generador de schedules es un algoritmo no determinista que se puede utilizar en distintos contextos para el cálculo de soluciones; las características más importantes de un generador de schedules que deben ser objeto de estudio son el tamaño del espacio de búsqueda que genera, la calidad media de las soluciones de este espacio y sus propiedades de dominancia. Una estructura de vecindad es una función que dada una solución devuelve un conjunto de soluciones próximas en el espacio de búsqueda a la solución de partida; en este caso, las características que se deben estudiar son el tamaño de la vecindad y la probabilidad de que esta vecindad contenga soluciones que mejoren a la solución de partida. Los generadores de schedules se utilizarán como evaluadores de cromosomas en los algoritmos genéticos y las estructuras de vecindad como generadores de nuevas soluciones en la búsqueda local.

El primero de los problemas abordados es el job shop scheduling con operarios (JSO). Este es un problema propuesto recientemente en la literatura. En este caso partimos de un modelo disyuntivo y de un generador de schedules, denominado $OG\&T$, propuestos en trabajos previos del grupo de investigación. La principal contribución en este problema es el diseño de una estructura de vecindad a partir de la cual se desarrolló un

algoritmo memético que combina un algoritmo genético con un algoritmo de búsqueda local. Este algoritmo mejora los resultados de los métodos anteriores y resulta incluso muy competitivo con una implementación sobre el solver CPLEX CP Optimizer. Si tenemos en cuenta que este solver está especialmente diseñado para tratar con las características propias del JSO, como son las restricciones disyuntivas y los recursos acumulativos, se trata sin duda de un resultado muy interesante. En el algoritmo memético el generador de schedules $OG\&T$ se utiliza como decodificador, y la estrategia de vecindad que explota el algoritmo de búsqueda local se define a partir de inversiones de arcos en el grafo solución.

El segundo problema que consideramos es una extensión del JSO en la que no todos los operarios tienen las mismas habilidades para atender a las tareas y además las restricciones de precedencia entre las tareas no definen un conjunto de trabajos como ocurre en el JSO o en el JSP. De este modo, el problema modela con más precisión muchas situaciones reales. Es también un problema propuesto recientemente en la literatura con la denominación JSSO (Job Shop Scheduling with Skilled Operators). Nuestra principal contribución para este problema es el diseño de un modelo disyuntivo y un generador de schedules denominado $SOG\&T$. Hemos estudiado las propiedades de dominancia de distintos espacios de búsqueda generados por este algoritmo y lo hemos utilizado como decodificador en un algoritmo genético. Este algoritmo proporciona los mejores resultados conocidos hasta el momento en los bancos de ejemplos propuestos en la literatura. El desarrollo de estrategias de vecindad es un problema abierto que planteamos como trabajo futuro.

El tercero de los problemas es una versión real del VCSP. Este tipo de problemas viene siendo considerado en la literatura desde hace casi dos décadas y hasta el momento las principales soluciones utilizan métodos de programación matemática. En el caso más general, en este problema se parte de un conjunto de trayectos programados y de un conjunto de autobuses y conductores distribuidos en distintas bases. El objetivo es asignar vehículos y conductores a cada uno de los trayectos, sujeto a un gran número de restricciones espaciales, temporales y de normativa laboral; con el objetivo de optimizar el número de vehículos y conductores, además de racionalizar las jornadas laborales de los conductores y optimizar el consumo energético. Entre otras características propias del VCSP, consideramos opciones como realizar viajes sin pasajeros (deadheads), desplazar conductores como pasajeros o programar tareas de mantenimiento a lo largo del horizonte de planificación, por las que el problema presenta una altísima complejidad computacional. En este caso nuestra principal contribución es el estudio del problema en colaboración con expertos de una compañía de transporte de viajeros por carretera, y el resultado es un modelado del problema en el marco de los problemas de satisfacción de restricciones. Siguiendo la metodología utilizada con los problemas anteriores, hemos propuesto un modelo gráfico para representar instancias del problema y soluciones. El desarrollo de generadores de schedules y de otros algoritmos de resolución es un tema de trabajo actual en el grupo de investigación.

Summary

Scheduling problems consist in organizing over time a set of operations that compete for the use of a limited set of resources, all subject to a series of constraints and trying to optimize one or several objective functions. These problems appear profusely in many social and productive environments, and in general they have a high computational complexity. For these reasons, scheduling problems are of great interest from both economical and academical points of view, and have therefore been the object of intensive research during the last decades by the Operations Research and Artificial Intelligence communities.

This thesis has as main objective to advance in the study and resolution of scheduling problems in which tasks require the use of various types of resources, material and human. Specifically, we consider two families of problems: first, some variants of one of the most well-known problems in the literature, the job shop scheduling problem (JSP), where we consider that each task requires not only a machine to be processed, but also an operator who attends it throughout its processing time. The second problem is a particular case of the problem of assigning vehicles and drivers to passenger transport routes, known in the literature as VCSP (Vehicle and Crew Scheduling Problem). Given the high complexity of these problems and the size of the instances of interest, we consider that metaheuristics constitute a reasonable approximation. Thus, we will use hybrid metaheuristics that combine population-based methods, such as genetic algorithms, in the diversification phase with search methods, such as hill-climbing or tabu search, in the intensification phase.

As working methodology, we propose the use of disjunctive graphic models to represent problem instances and their solutions. These models are useful for studying the properties of the problems, for designing schedule builders, and also for designing neighborhood structures. A schedules builder is a non-deterministic algorithm that can be used in different contexts for calculating solutions; the most important characteristics of a schedules builder that must be studied are the size of the search space it generates, the average quality of the solutions in this space and its properties of dominance. A neighborhood structure is a function that given a solution, it returns a set of nearby solutions in the search space to the starting solution. In this case, the characteristics to be studied are the size of the neighborhood and the probability that this neighborhood contains solutions that improve the starting solution. The schedule builders will be used as evaluators of chromosomes in the genetic algorithms and the neighborhood structures as generators of new solutions in the local search.

The first of the addressed problems is the job shop scheduling with operators (JSO). This is a problem recently proposed in the literature. In this case we start with a disjunctive model and a schedule builder, called $OG\&T$, proposed in previous works of the research group. The main contribution in this problem is the design of a neighborhood structure from which a memetic algorithm was developed combining a genetic algorithm with a local search algorithm. This algorithm improves the results of the previous methods and is even very competitive with an implementation on the CPLEX CP Optimizer solver. If we consider that this solver is specially designed to deal with the characteristics of the JSO, such as disjunctive constraints and

cumulative resources, this is undoubtedly a very interesting result. In the memetic algorithm the schedule builder $OG\&T$ is used as a decoder, and the neighborhood structure exploited by the local search algorithm is defined from arc reversals in the solution graph.

The second problem we consider is an extension of the JSO in which not all operators have the same skills to assist the tasks and also the precedence relations between the tasks do not define a set of jobs on the contrary of what happens with the JSO and the JSP. This way, the problem models more precisely many real situations. It is also a problem recently proposed in the literature with the denomination JSSO (Job Shop Scheduling with Skilled Operators). Our main contribution to this problem is the design of a disjunctive model and a schedule builder called $SOG\&T$. We have studied the dominance properties of different search spaces generated by this algorithm and we have used it as a decoder in a genetic algorithm. This algorithm yields the best results known so far in the benchmarks proposed in the literature. The development of neighborhood strategies is an open problem that we consider as future work.

The third of the problems is a real version of the VCSP. This type of problems has been considered in the literature for almost two decades and until now the main solutions use mathematical programming methods. In the most general case, in this problem we start with a set of scheduled trips and a set of buses and drivers distributed in different bases or depots. The objective is to assign vehicles and drivers to each of the trips, subject to a large number of space, temporary and labor regulation constraints; with the objective of optimizing the number of vehicles and drivers, as well as rationalizing the working days of the drivers and optimizing the energy consumption. Among other characteristics of the VCSP, we consider options such as the possibility of doing trips without passengers (deadheads), displacing drivers as passengers or scheduling maintenance tasks throughout the planning horizon, by which the problem presents a very high computational complexity. In this case our main contribution is the study of the problem in collaboration with experts of a passenger transport company, and the result is a modeling of the problem in the framework of constraint satisfaction problems. Following the methodology used in the previous problems, we have devised a graphical model to represent problem instances and solutions. The development of schedule builders and other solvers is a current topic of work in the research group.

Índice de Contenidos

1. INTRODUCCIÓN **1**
1.1. Scheduling 1
1.2. Metaheurísticas 1
1.3. Job Shop Scheduling Problem (JSP) 2
1.4. Vehicle and Crew Scheduling Problem (VCSP) 4
1.5. Contenido de la tesis 4
1.6. Estructura de la memoria 4

2. OBJETIVOS **7**

3. DISCUSIÓN DE RESULTADOS **9**
3.1. El problema Job Shop Scheduling con Operarios 9
3.1.1. Antecedentes e hipótesis 9
3.1.2. Contribuciones y resultados 11
3.2. El problema de planificación de tareas y operarios con habilidades y restricciones de precedencia arbitrarias 18
3.2.1. Antecedentes e hipótesis 18
3.2.2. Contribuciones y resultados 19
3.3. El problema de asignación vehículos y conductores a rutas de transporte 25
3.3.1. Antecedentes e hipótesis 25
3.3.2. Contribuciones 25

4. CONCLUSIONES **29**
4.1. Aportaciones de la tesis 29
4.2. Trabajo futuro 30

5. COMPENDIO DE PUBLICACIONES **31**
5.1. A genetic algorithm for job-shop scheduling with operators enhanced by weak Lamarckian evolution and search space narrowing 31
5.2. Memetic Algorithms for the Job Shop Scheduling Problem with Operators 47
5.3. Schedule Generation Schemes and Genetic Algorithm for the Scheduling Problem with Skilled Operators and Arbitrary Precedence Relations 61

5.4. Genetic Algorithm for the scheduling problem with arbitrary precedence relations and skilled operators 73
5.5. Technical Report: A Real-Life Vehicle and Crew Scheduling Problem for Extra-Urban Bus Passenger Routes 93

6. INFORME SOBRE LA CALIDAD DE LAS PUBLICACIONES 109

Capítulo 1

INTRODUCCIÓN

1.1. Scheduling

Los problemas de scheduling consisten en la distribución temporal de un conjunto limitado de recursos para realizar una serie de tareas [53, 10, 36]. Sus soluciones deben optimizar una o varias funciones de evaluación a la vez de satisfacer un cierto número de restricciones. Este tipo de problemas aparecen con profusión en muchos sectores sociales y productivos por lo que su resolución tiene un interés económico indudable. Algunos problemas de este tipo son la planificación de turnos de trabajo y actividades en plantas de producción o centros educativos, la planificación de terminales marítimas de contenedores, o la asignación de vehículos y conductores a rutas de transporte de viajeros.

Los problemas de scheduling han generado gran interés desde hace décadas entre los investigadores de diferentes áreas, como las de Investigación Operativa e Inteligencia Artificial. Los primeros utilizan fundamentalmente técnicas de programación matemática para resolverlos [33]. Los segundos, por su parte, utilizan técnicas como programación con restricciones [35, 59, 61, 48, 6] o búsqueda heurística [39, 55]. Dada la alta complejidad computacional de estos problemas (en general son problemas NP-duros [18]) y el tamaño de las instancias en dominios de aplicación reales, todas estas técnicas resultan insuficientes en muchos casos y por esta razón en ocasiones los investigadores han optado por utilizar técnicas basadas en metaheurísticas [8, 9], que normalmente ofrecen soluciones de una calidad razonable en un tiempo aceptable.

1.2. Metaheurísticas

Las metaheurísticas son estrategias generales de diseño de algoritmos heurísticos y se consideran especialmente indicadas para resolver problemas difíciles de optimización combinatoria. En la literatura, se suelen distinguir dos grandes familias: las metaheurísticas basadas en poblaciones y las metaheuristicas de mejora iterativa o búsqueda local. Las primeras están generalmente inspiradas en la forma que tiene la naturaleza de resolver problemas. Un ejemplo paradigmático de estos métodos son los algoritmos genéticos (GA) [29] que se fundamentan en la teoría de la evolución. Estos algoritmos mantienen una población de soluciones que evoluciona en el tiempo bajo la aplicación de operadores de selección, recombinación y mutación, mediante los que se intenta obtener un equilibrio entre la exploración del espacio de búsqueda y la intensificación en las zonas más prometedoras. Otros ejemplos notables de este tipo de metaheurísticas son los sistemas de inteligencia de enjambre (PSO)[31], las colonias de hormigas (ACO) [16, 17] y otros algoritmos evolutivos como scatter search (SS) [20, 24].

Las metaheurísticas de mejora iterativa o búsqueda local, como por ejemplo el enfriamiento simulado (SA) [32] o la búsqueda tabu (TS) [21, 22, 23], se basan en mayor medida en la forma en que los seres humanos resolvemos problemas cotidianos. A rasgos generales, este tipo de algoritmos consisten en la aplicación repetida de pequeños cambios en una solución de partida que pueden mejorar su calidad, con lo que constituyen técnicas muy efectivas para intensificar la búsqueda en zonas concretas del espacio, por ejemplo en el entorno de una solución.

Todas las técnicas mencionadas cuentan con un registro importante de casos de éxito y en muchos casos se utilizan de forma combinada. Entre los casos de éxito se encuentran metaheurísticas híbridas utilizadas para resolver problemas de scheduling [37, 60, 25, 27].

La literatura sobre teoría y aplicación de metaheurísticas es realmente amplia. Entre los textos destaca el de E.G. Talbi [58]. Existen además revistas que publican trabajos de investigación sobre fundamentos y aplicaciones de metaheuristicas como Evolutionary Computation (http://www.mitpressjournals.org/loi/evco) o Applied Soft Computing (https://www.journals.elsevier.com/applied-soft-computing/), entre otras muchas.

1.3. Job Shop Scheduling Problem (JSP)

Un problema paradigmático de scheduling es el *Job Shop Scheduling Problem* (JSP) en el que se requiere planificar un conjunto de n trabajos $\{J_1,..., J_n\}$ en un conjunto de m máquinas o recursos $\{R_1,..., R_m\}$. Cada trabajo J_i consta de una secuencia de operaciones o tareas ($\theta_{i1},..., \theta_{im}$), donde cada tarea θ_{il} necesita el uso de una máquina $R_{\theta_{il}}$ durante un tiempo de procesamiento $p_{\theta_{il}}$. El objetivo es asignar a cada operación un tiempo de inicio $st_{\theta_{il}}$ de modo que se optimice una determinada función objetivo satisfaciendo tres tipos de restricciones: i) las operaciones de un mismo trabajo deben procesarse en el orden dado en la secuencia, ii) las máquinas tienen capacidad para procesar una única operación en un instante dado y iii) no se permite interrumpir el procesamiento de una operación una vez que ha comenzado. La función objetivo es habitualmente la minimización de una métrica de la planificación, como por ejemplo el *makespan* (o tiempo de finalización de todos los trabajos). En la notación $\alpha|\beta|\gamma$ propuesta en [28], la versión del JSP con minimización del makespan se denota como $J||C_{max}$.

Para resolver el JSP se han utilizado numerosas técnicas de búsqueda heurística exactas y aproximadas del campo de la inteligencia artificial. Entre las aproximaciones que proporcionan los mejores resultados están las metaheurísticas, los métodos de propagación de restricciones, así como algunos heurísticos específicos para el problema. Entre las técnicas basadas en metaheurísticas, destancan los algoritmos genéticos que emplean esquemas de codificación de cromosomas y operadores genéticos específicos para este problema [8, 51] y los métodos de búsqueda local [34, 15, 49, 62]. Entre los métodos exactos, merecen especial mención los algoritmos de búsqueda heurística [13, 11, 38, 40] o los métodos de razonamiento basado en restricciones [5], empleadas por el conocido solver comercial CPLEX CP Optimizer [35]. En cuanto a los heurísticos específicos podemos citar el conocido *shifting bottleneck* [4].

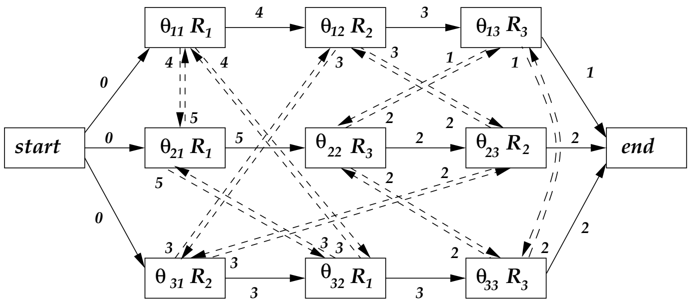


Figura 1.1: Grafo disyuntivo de una instancia con 3 trabajos y 3 máquinas.

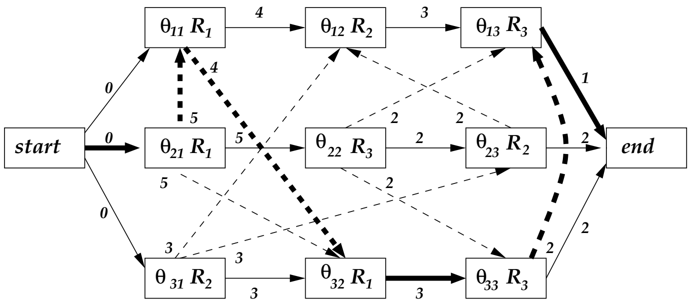


Figura 1.2: Grafo disyuntivo de una planificación factible para la instancia de la Figura 1.1.

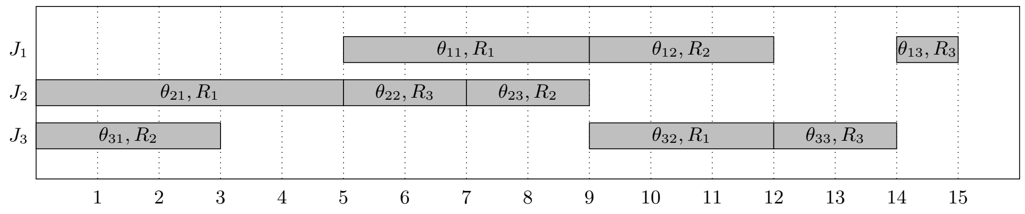


Figura 1.3: Diagrama de Gantt para la solución expresada por el grafo disyuntivo de la Figura 1.2.

El uso del modelo de grafo disyuntivo [54] está presente tanto en las propuestas aproximadas como en las exactas. En la Figura 1.1 se muestra un ejemplo para una instancia con tres trabajos y tres máquinas. En el grafo disyuntivo, hay un nodo por cada tarea, además de dos nodos ficticios (*start* y *end*) conectados a las primeras y a las últimas tareas de cada trabajo respectivamente. Hay dos tipos de arcos: los conjuntivos, que expresan las relaciones de precedencia entre tareas del mismo trabajo; y los disyuntivos, que conectan las tareas que utilizan la misma máquina. En la Figura 1.1 los arcos conjuntivos están representados por líneas continuas y los disyuntivos por líneas discontinuas. La Figura 1.2 es un ejemplo de grafo disyuntivo de una solución para la instancia representada en la Figura 1.1. Estos grafos han sido tomados de [38]. En la Figura 1.3 se puede ver un diagrama de Gantt que representa la misma solución que la expresada en la Figura 1.2.

Conviene mencionar que la OR-library (http://people.brunel.ac.uk/∼mastjjb/jeb/info.html) contiene un banco de ejemplos muy utilizado para comparar métodos para este problema [7]. También se pueden encontrar en la literatura revisiones de otros métodos propuestos para resolver el problema JSP [12].

1.4. Vehicle and Crew Scheduling Problem (VCSP)

El problema de asignación de vehículos y conductores a rutas de transporte de viajeros (Vehicle and Crew Scheduling Problem) consiste en asignar vehículos y conductores destribuidos en distintas bases a una serie de trayectos programados, de modo que se satisfagan un gran número de restricciones de distintos tipos y se optimicen una serie de funciones objetivo como pueden ser minimizar el número de vehículos y conductores requeridos. Este problema ha interesado a investigadores desde hace casi dos décadas y hasta el momento las principales soluciones utilizan métodos de programación matemática [30, 47, 57]. Sin embargo, otros métodos como la programación con restricciones o las metaheurísticas todavía no han sido muy utilizados para resolver problemas de esta familia.

1.5. Contenido de la tesis

En esta tesis consideramos dos versiones del JSP y un caso real del VCSP. El factor común de todos estos problemas es la existencia de un tipo recurso que representa la presencia de operarios humanos para asistir las tareas. En el caso del JSP, distinguimos dos situaciones, en la primera todos los operarios tienen capacidad para asistir a todas las tareas del problema y, en el segundo, cada operario puede asistir solo a un subconjunto de las tareas. Además, en esta versión las relaciones de precedencia entre las tareas son arbitrarias y por lo tanto no se organizan en trabajos como es habitual en los problemas de tipo job shop.

En el caso del problema job shop scheduling con operarios, las propuestas en la literatura [1, 55] utilizan métodos de programación dinámica y algoritmos de búsqueda heurística. Por otro lado, en caso del problema de planificación de tareas y operarios con capacidades y restricciones de precedencia arbitrarias, los autores hacen uso de la heurística shifting bottleneck [2]. Finalmente, en el caso del problema VCSP las soluciones propuestas en la literatura utilizan principalmente técnicas de programación matemática [30, 47], y solamente en algunos casos se incorporan metaheurísticas para resolver alguna parte del problema, o bien versiones simplificadas del mismo [56].

La hipótesis de partida de esta tesis es que las metaurísticas pueden ser un marco de trabajo adecuado para conseguir buenas aproximaciones para los problemas anteriores. Así, las principales contribuciones están encaminadas al desarrollo de metaheurísticas para estos problemas. La metodología de trabajo propuesta consiste en el estudio formal de los problemas, su representación mediante modelos gráficos disyuntivos y el desarrollo de generadores de schedules y estructuras de vecindad que serán posteriormente incorporadas en metaheurístcas híbridas como algoritmos genéticos combinados con estrategias de búsqueda local. Los algoritmos propuestos son evaluados experimentalmente sobre bancos de ejemplos de uso común y comparados con el estado del arte.

1.6. Estructura de la memoria

Esta tesis doctoral se presenta como un compendio de publicaciones. El resto de la memoria se estructura de la siguiente forma: en el capítulo 2 se describen los objetivos de la presente tesis. El capítulo 3 contiene una discusión de las contribuciones realizadas y de los resultados obtenidos. En el capítulo 4 se presentan las conclusiones y se esbozan algunas líneas de trabajo futuro. El capítulo 5 contiene una copia original de los artículos presentados como compendio de publicaciones. En el capítulo 6 se presenta un informe sobre el factor de impacto de las publicaciones. Por último, se lista la bibliografía utilizada en esta memoria.

Capítulo 2

OBJETIVOS

El objetivo general de esta tesis es establecer un marco de trabajo para estudiar y resolver problemas de scheduling en los que las tareas requieren el uso de más de un tipo de recurso. Para ello, partimos de dos familias de problemas con estas características, pero a la vez con diferencias sustanciales entre ellas, como son los problemas de tipo job shop scheduling con operarios y los problemas de asignación de vehículos y conductores a rutas programadas. En el primer caso, las variables de decisión son los tiempos de inicio de las tareas y los operarios asignados a las mismas; mientras que en el segundo los tiempos de inicio de las tareas (trayectos) ya están programados y las variables de decisión son los vehículos y conductores asignados a las tareas.

Dadas la complejidad computacional de estos problemas y el tamaño de las instancias de interés real, consideramos como opción más razonable el uso de metaheurísticas para desarrollar métodos efectivos de resolución. Así pues, la metodología propuesta irá encaminada al desarrollo sistemático de metaheurísticas híbridas que permitan combinar estrategias generales con conocimiento específico del dominio del problema. Esta metodología incluye el modelado y estudio formal de instancias y soluciones, el desarrollo de generadores efectivos de soluciones y de estrategias de mejora, así como la explotación de estos elementos en el diseño de metaheurísticas.

De forma más concreta, los objetivos de esta tesis son los siguientes:

- Diseñar modelos gráficos disyuntivos para problemas de scheduling con operarios y para problemas de asignación de vehículos y conductores a rutas de transporte de pasajeros programadas. Estos modelos serán de utilidad para el estudio formal de las propiedades de los problemas y de sus soluciones.
- Diseñar generadores de planificaciones siguiendo una estrategia de cálculo de grafos solución como subgrafos del grafo disyuntivo. Estudiar las características de los espacios de búsqueda, como el tamaño, la calidad de las soluciones y las propiedades de dominancia.
- Diseñar estructuras de vecindad de las soluciones mediante modificaciones del grafo solución. Estudiar las propiedades de los espacios de vecindad como el tamaño y la posibilidad de mejora.
- Diseñar metaheurísticas híbridas combinando métodos basados en poblaciones con métodos de búsqueda local utilizando los generadores de schedules como algoritmos de decodificacion en la pri-

mera fase y las estructuras de vecindad en la segunda para obtener nuevas soluciones. Experimentar de forma sistemática con estas metaheurísticas para analizar la contribución de cada uno de los elementos y para compararlas con el estado del arte cuando sea posible.

Capítulo 3

DISCUSIÓN DE RESULTADOS

3.1. El problema Job Shop Scheduling con Operarios

Comenzamos con la descripción de las contribuciones que hemos realizado en esta tesis relacionadas con el estudio formal y experimental del problema Job Shop Scheduling con Operarios.

3.1.1. Antecedentes e hipótesis

El problema job shop scheduling con operarios (JSO) fue introducido en el año 2011 en [1] y es una extensión del problema job shop scheduling (JSP). La aplicación práctica del problema job shop scheduling es amplia, ya que muchas actividades se desarrollan en entornos de producción de este tipo. Además, las comunidades de investigadores en Inteligencia Artificial e Investigación Operativa han generado mucha literatura en versiones de este problema que prestan atención a determinados detalles del día a día para ajustar la definición de este problema a sus casos reales. Ejemplos de esto son el job shop scheduling con tiempos de setup [3, 60] o el job shop scheduling con incertidumbre [52], entre otros.

En esta tesis consideramos otra variante del problema JSP, en la cada tarea no solamente necesita una máquina determinada, sino que también necesita un operario que la asista durante su tiempo de procesamiento.

Formalmente, tenemos un conjunto de n trabajos $\{J_1,..., J_n\}$, un conjunto de m máquinas o recursos $\{R_1,..., R_m\}$, y un conjunto de p operarios $\{O_1, ..., O_p\}$. Un trabajo J_i es una secuencia de operaciones o tareas (θ_{i1},..., θ_{im}). Cada tarea θ_{il} tiene una duración entera $p_{\theta_{il}}$ y ha de ser planificada en una máquina específica. En una planificación o schedule factible, cada tarea θ_{il} tiene asignado un tiempo de inicio $st_{\theta_{il}}$ y un operario que la asiste durante todo su tiempo de procesamiento $p_{\theta_{il}}$ de modo que se satisfacen una serie de restricciones: (i) se respeta el orden de las tareas en sus respectivos trabajos, (ii) cada máquina puede procesar una sola tarea en cada momento, (iii) no se puede interrumpir el procesamiento de una tarea y (iv) un operario no puede asistir a más de una tarea al mismo tiempo. El objetivo es encontrar una planificación factible que minimice el tiempo de fin de todas las tareas, esto es, el *makespan*. En este problema se asume que todos los operarios tienen capacidad para asistir a todos los trabajos en todas las máquinas, por lo que si el número de operarios disponibles es igual o mayor que el mínimo entre el número de trabajos y el número

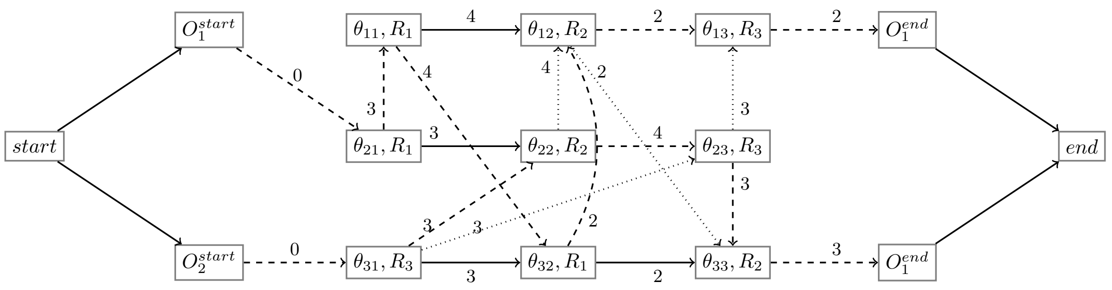


Figura 3.1: Un ejemplo de grafo solución.

de máquinas, el problema es equivalente al JSP.

Una planificación factible se puede representar con un grafo disyuntivo acíclico. Este grafo consta de un conjunto de nodos y cuatro conjuntos de arcos. El conjunto de nodos contiene nodos para representar las tareas, nodos para representar el inicio de la secuencia de tareas asistidas por el mismo operario y nodos ficticios $start$ y end. Además hay distintos tipos de arcos: arcos que unen tareas pertenecientes al mismo trabajo, arcos que unen operaciones que usan la misma máquina, arcos que unen tareas asistidas por el mismo operario y finalmente, arcos para conectar las operaciones ficticias $start$ y end con otros nodos. Un camino crítico es un camino de máximo coste entre el nodo $start$ y el nodo end. La Figura 3.1.1 muestra un ejemplo de grafo solución para un problema con 3 trabajos, 3 máquinas y 2 operarios.

El algoritmo $G\&T$ fue propuesto por Giffler y Thomson en [19] como esquema de generación de planificaciones para el JSP. Se trata de un algoritmo no determinista que es capaz de generar todas las planificaciones del conjunto de las planificaciones activas; conjunto que tiene la propiedad de dominancia, es decir que contiene al menos una solución óptima. Este algoritmo es la base sobre la que se diseñó el algoritmo $OG\&T$, para el problema JSO [55]. En el algoritmo $OG\&T$, se planifica una operación en cada iteración, asignándole un tiempo de comienzo y un operario, hasta que todas están planificadas. Como otros generadores de planificaciones, este algoritmo considera conjuntos de tareas candidatas que pueden ser elegidas para ser planificadas en cada iteración. En primer lugar, el conjunto A que contiene la primera operación sin planificar de cada uno de los trabajos que tienen al menos una operación sin planificar. Este conjunto se puede reducir sin perder la propiedad de dominancia. Así, se considera la tarea v^* de A con menor tiempo de finalización si fuese elegida en la iteración actual. A partir de esta tarea se calcula el conjunto A' que contiene las operaciones de A que pueden empezar antes del tiempo de finalización de la operación v^*. Se puede reducir aún más el conjunto de operaciones candidatas en cada iteración, teniendo en cuenta la disponibilidad de operarios en los tiempos mínimos de inicio de las tareas del conjunto A. El resultado es un nuevo conjunto B que reduce las opciones de A', manteniendo la propiedad de dominancia del espacio de búsqueda generado.

Dado que el problema JSO es una extensión del problema JSP y que los algoritmos genéticos y los métodos de búsqueda local han sido la base de muchos de los métodos más eficientes para el problema JSP, nuestra hipótesis es que extendiendo las ideas y fundamentos de estos algoritmos para el problema JSP podremos diseñar también métodos eficientes para el problema JSO.

3.1.2. Contribuciones y resultados

De acuerdo con la hipótesis de partida, nuesta primera propuesta fue un algoritmo genético [41, 42] con dos elementos esenciales: el esquema de permutaciones con repetición, propuesto por Bierwirth en [8] que se adapta perfectamente al JSO y el algoritmo $OG\&T$ como método de decodificación. Para resolver instancias de gran tamaño, este algoritmo genético se combinó con una estrategia de reducción del espacio de búsqueda y con un modelo de evolución que favorece la componente de intensificación de la búsqueda evolutiva.

Otra de nuestras propuestas fue el diseño de estrategias de vecindad para el problema JSO. Para ello, desarrollamos una generalización de la estrategia propuesta por van Laarhoven, Aarts y Lenstra [34] para el problema JSP. Esta generalización requirió la caracterización de arcos en el camino crítico cuya inversión genere planificaciones factibles y que a la vez tengan opciones de mejora. La estrategia de vecindad definida fue explotada en una búsqueda tabú que a su vez se incorporó como método de búsqueda local en el algoritmo genético. El resultado es un algoritmo memético que resultó ser muy competitivo con el estado del arte para el JSO.

Elementos del algoritmo genético para el JSO

Partimos en principio de un algoritmo genético con reemplazo generacional, selección aleatoria y reemplazo por torneo, e incorporamos algunos elementos propios de los problemas de scheduling: la codificación mediante permutaciones con repetición, operadores genéticos específicos para esta codificación, un método de reducción del espacio de búsqueda y evaluación de cromosomas mediante el algoritmo $OG\&T$.

El esquema de permutaciones con repetición fue propuesto en [8]. Siguiendo este esquema, un cromosoma es una permutación del conjunto de operaciones que representa un orden tentativo para planificarlas, estando cada operación representada por el índice de su trabajo. Por ejemplo, la secuencia (3 1 2 2 1 3 3 1 2) es un cromosoma válido para un problema con tres trabajos y tres máquinas, como el correspondiente a la solución que se muestra en la Figura 3.1.

Como operador de cruce el algoritmo genético usa el Job-based Order Crossover (JOX) propuesto en [51] para un esquema de codificación basado en matrices de máquinas. Este operador se puede utilizar también sobre permutaciones con repetición del siguiente modo: se selecciona un conjunto aleatorio de trabajos del primer padre y se copian sus genes en el cromosoma hijo en las mismas posiciones del primer padre; el resto de los genes se obtienen del segundo padre de modo que se conserve su orden relativo. Por ejemplo, si consideramos los dos siguientes padres:

- Padre 1: (**2** 1 1 3 **2** 3 1 **2** 3)
- Padre 2: (3 3 1 2 1 3 2 2 1)

Si el conjunto de trabajos considerado solo contiene el trabajo 2, el hijo generado es:

- Hijo: (**2** 3 3 1 **2** 1 3 **2** 1)

De este modo, el operador JOX transmite características relevantes de los padres a los hijos; en este caso órdenes de tareas en las máquinas. En concreto, mantiene para cada máquina una subsecuencia de operaciones en el mismo orden en el que están en el Padre 1, y otra subsecuencia en el orden en que aparecen en el Padre 2. El operador también introduce algunos órdenes nuevos, por lo que se considera que tiene un efecto de mutación importante. A diferencia de lo que ocurre cuando el operador JOX se aplica a matrices de máquinas, cuando se aplica a permutaciones con repetición los cromosomas son siempre factibles.

Para evaluar cromosomas, el algoritmo genético utiliza el generador de planificaciones $OG\&T$ de modo que la elección de la tarea del conjunto de opciones posibles considerado en cada iteración, A, A' o B, se realiza en función del cromosoma, tomando la operación de este conjunto que está más a la izquierda en el cromosoma.

El algoritmo $OG\&T$ es un generador de planificaciones que fue diseñado con el objetivo de que el espacio de búsqueda sea lo más reducido posible, sin perder la seguridad de que exista al menos una solución óptima. Aunque el conjunto de planificaciones definido por el algoritmo $OG\&T$ es mucho más pequeño que el conjunto de todas las planificaciones factibles, puede ser de un tamaño inmanejable a partir de un cierto tamaño de las instancias. Por esta razón, consideramos nuevas reducciones del conjunto de opciones en cada iteración con el objetivo de reducir el espacio de búsqueda efectivo [42], aunque con estas nuevas reducciones se pierda la propiedad de dominancia. Para esto, definimos un parámetro δ $(0 \leq \delta \leq 1)$, que establece el límite superior para el tiempo de comienzo de la siguiente operación a planificar, y de este modo controla el tiempo máximo que una máquina puede estar desocupada a la vez que una operación esté preparada para ser procesada en esa máquina y un operador puede asistir su procesamiento. Ocurre que cuanto menor sea el valor de δ, menores serán los periodos de tiempo en los que las máquinas estén desocupadas, por lo que podemos esperar que, en media, el makespan de las planificaciones que permanecen en el espacio de búsqueda tras la reducción será menor que el de todas las planificaciones del conjunto original. Por esto, partimos de la suposición de que las reducciones del espacio de búsqueda serán más eficaces cuanto más grandes sean las instancias.

Tradicionalmente, se consideran dos modelos de evolución para un algoritmo genético, evolución Baldwiniana y evolución Lamarckiana. En el modelo de evolución Baldwiniana se calcula el fitness de cada cromosoma pero no se traslada ninguna información al cromosoma tras decodificarlo, esto es, el cromosoma es el mismo antes y después de calcular su valor de fitness. Por el contrario, en la evolución Lamarckiana, las características de las planificaciones se trasladan al cromosoma, lo cual facilita que estas características sean heredadas por sus hijos. Para realizar este tipo de evolución, una vez que el algoritmo $OG\&T$ ha construido la planificación, el cromosoma original es reemplazado por uno nuevo en el que las operaciones aparecen en un orden compatible con sus tiempos de procesamiento en la planificación. Denominamos *evolución Lamarckiana débil* a este proceso porque la diferencia entre el nuevo y el viejo cromosoma probablemente será pequeña, al menos si la comparamos con la diferencia que se suele producir cuando la evolución Lamarckiana se utiliza en combinación con métodos de búsqueda local que pueden producir grandes cambios en las soluciones de partida.

El algoritmo genético propuesto se evaluó experimentalmente para analizar cada una de sus componentes y para estudiar la sinergia entre ellas, así como para compararlo con el estado del arte, que en su momento estaba constituido por los algoritmos heurísticos y de programación dinámica propuestos en [1] y el algoritmo de búsqueda de tipo A^* propuesto en [55]. En estos experimentos, el algoritmo genético superó claramente al resto de los métodos. Además, hicimos un estudio acerca de la idoneidad de las propuestas de reducción del espacio de búsqueda considerando diferentes valores para el parámetro δ, y de los modelos de evolución, comparando las versiones Baldwiniana y Lamarckiana del algoritmo. Estas pruebas nos llevaron a apuntar que la evolución Lamarckiana débil y un valor del parámetro de reducción $\delta = 0.75$ dan lugar a la mejor versión del algoritmo genético para el conjunto de instancias considerado.

Algoritmo memético para el JSO

Un algoritmo memético es una metaheurística híbrida que combina la capacidad de exploración de un algoritmo basado en poblaciones, como por ejemplo un algoritmo genético, con la capacidad de búsqueda intensiva en un entorno propia de las estrategias de búsqueda local. Cuando se diseñan adecuadamente y se adaptan al dominio del problema, los algoritmos meméticos pueden conseguir un equilibrio adecuado del binomio diversificación/intensificación que es precisamente uno de los principales objetivos que se persiguen con el diseño y uso de metaheurísticas.

La contribución más relevante que se hace en esta tesis para el problema JSO es el diseño de un algoritmo memético [43] que combina el algoritmo genético descrito en la sección anterior con un algoritmo de búsqueda local. La parte esencial del algoritmo de búsqueda local es la estructura de vecindad diseñada específicamente para el JSO, denominada $\mathcal{N}^{\mathcal{O}}$, que como ya hemos comentado es una extensión de la estructura $\mathcal{N}$ propuesta en [34] para el JSP. Estas dos estructuras se describen a continuación.

Estructura de vecindad $\mathcal{N}$ para el JSP

Los movimientos de la estructura $\mathcal{N}$ consisten en inversiones de arcos en el grafo solución que dan lugar a soluciones factibles y que a la vez tienen posibilidades de mejora con respecto a la solución de partida. La caracterización de estos arcos se hace a través de un estudio de las propiedades del problema que de forma resumida se concreta en las siguientes nociones y resultados.

- Un grafo solución es un grafo que representa el orden de procesamiento de las operaciones en las máquinas y en los trabajos. Se trata de una simplificación del grafo solución definido para el JSO (Figura 3.1) en el que se suprimen los arcos y nodos asociados a los operarios.
- Un camino crítico es un camino de coste máximo entre los nodos $start$ y end del grafo solución.
- Un bloque crítico es una subsecuencia maximal de tareas en el camino crítico que requieren la misma máquina.
- Un arco crítico es un arco de un camino crítico.
- La inversión de un arco crítico siempre da lugar a una solución factible.
- La inversión de un arco simple en el grafo solución solamente puede producir una solución mejor si el arco invertido es el primero o el último de un bloque crítico, salvo en los casos que sea el primero del primer bloque crítico o el último del último bloque crítico, y que estos bloques tengan longitud mayor que 2.

Los movimientos descritos en el último punto son los que definen la estructura $\mathcal{N}$ para el JSP. Además, para estimar el coste de las soluciones resultantes de estos movimientos, se definen los conceptos de cabeza y cola de una operación:

- La cabeza de una operación u en el grafo solución es el coste del camino de coste máximo desde el nodo $start$ hasta el nodo asociado a u.
- La cola de una operación u es la suma de los costes de las tareas posteriores a u en un camino de coste máximo desde u al nodo end.

Las cabezas y colas de las operaciones después de una inversión se pueden calcular en tiempo constante, y a partir de estos valores se pueden establecer cotas inferiores del makespan de la nueva solución. Obviamente, si el valor de esta cota inferior es mayor o igual al coste de la solución de partida, el movimiento no producirá mejora alguna y se puede descartar sin necesidad de hacer un cálculo exacto del makespan.

Estructura de vecindad $\mathcal{N}^{O}$ para el JSO

La estructura $\mathcal{N}^{O}$ extiende $\mathcal{N}$ considerando los nuevos arcos que aparecen en el grafo solución como consecuencia de los operarios. Así para definir $\mathcal{N}^{O}$ debemos considerar cinco tipos de arcos:

- Un arco conjuntivo (J) conecta dos operaciones consecutivas en el mismo trabajo.
- Un arco disyuntivo (M) conecta dos operaciones que utilizan la misma máquina de forma consecutiva.
- Un arco de operario (O) conecta dos operaciones que son asistidas por el mismo operario de forma consecutiva.
- Un arco de trabajo y operario (JO) conecta dos operaciones consecutivas en el mismo trabajo y que son asistidas por el mismo operario también de forma consecutiva.
- Un arco de máquina y operario (MO) conecta dos operaciones que usan la misma máquina y son asistidas por el mismo operario de forma consecutiva.

En la Figura 3.1, los arcos O, MO y JO se representan mediante líneas discontinuas, los de tipo M mediante líneas de puntos y los de tipo J mediante líneas continuas.

Es evidente que si invirtiésemos arcos de tipo J o JO tendríamos planificaciones no factibles ya que el orden de las operaciones en los trabajos no puede ser alterado. Sin embargo, si invertimos arcos de tipo MO, O o M tendremos soluciones factibles, siempre y cuando no haya ciclos en el grafo solución resultante. Así pues, estos son los candidatos para definir los movimientos de nuestra estructura de vecindad. Para que esta estructura sea eficiente, esto es, tenga un vecindario de un tamañoo reducido y una probabilidad de mejora alta, establecemos algunas condiciones de factibilidad y de no mejora para crear vecinos candidatos. Para esto, definimos algunos conceptos que extienden a los correspondientes conceptos definidos para el JSP:

- Un bloque crítico es una subsecuencia maximal de operaciones en un camino crítico que usan
  - la misma máquina (bloque crítico de míquina), o bien
  - el mismo operario (bloque crítico de operario).

Nótese que los arcos en un bloque critico de máquina son de tipo M o MO, mientras que los arcos de un bloque crítico de operario son de tipo O, MO o JO, por lo que un arco crítico puede estar en un bloque crítico de máquina y en un bloque crítico de operario al mismo tiempo, y estar en el interior de uno y en el exterior o extremo del otro.

La estructura $\mathcal{N}^{\mathcal{O}}$ se fundamenta en los siguientes resultados:

- Un arco crítico (u, v) es el único camino de u a v en el grafo solución.
- Si un arco no es crítico, su inversión no puede dar lugar a mejora alguna.
- Si se invierte un arco interno a un bloque crítico, ya sea de máquina o de operario, la solución resultante no puede ser mejor que la de partida.
- La inversión del primer arco del primer bloque crítico o del último arco del último bloque crítico no puede producir mejora alguna, salvo que el bloque critico solo tenga dos operaciones.

De acuerdo con todo lo anterior, la única opción de mejora mediante la inversión de un arco simple es invertir un arco de tipo O, M o MO que esté en el borde de un bloque crítico, con las excepciones del primer (último) arco de del primer (último) bloque, a no ser que el bloque tenga solo dos operaciones.

Aunque el grafo de una solución vecina es muy similar al grafo de la solución de partida, el cálculo exacto del makespan requiere un recorrido completo del grafo resultante, lo cual es costoso computacionalmente. Para evitar este coste, hemos adaptado el método de cálculo de cabezas y colas después de la inversión para cada uno de los tipos de arcos O, M y MO. Con estos valores se pueden hacer estimaciones del makespan de la nueva solución en tiempo constante.

Búsqueda local para el JSO

Consideramos dos estrategias de búsqueda local bien conocidas, hill-climbing simple (también conocida como *First*) y búsqueda tabú.

Cuando utilizamos hill-climbing simple, se calculan los vecinos de la solución actual uno a uno. Se realiza una estimación del makespan para cada vecino. Si esta es igual o mayor que el makespan de la planificación actual, se descarta; si no, se calcula el makespan real del vecino, si este es menor que el de la solución actual, el vecino reemplaza a la solución actual y se pasa a calcular su vecindario. La búsqueda local finaliza cuando no hay ningún vecino que mejore la solución de la que parte.

En cuanto a la búsqueda tabú, utilizamos una estrategia similar a la propuesta en [21, 23]. En cada iteración se crea una lista de movimientos candidatos dados por la estructura de vecindad $\mathcal{N}^{\mathcal{O}}$ y se estima el makespan de cada uno. Los movimientos que son tabú y no satisfacen el criterio de aspiración, esto es, que no tienen una estimación mejor que la solución actual, son descartados. Finalmente, se escoge el candidato con la mejor estimación. El algoritmo termina después de $Maxiter$ iteraciones sin producir mejora o cuando se llega a una situación en la que no hay vecinos posibles (o todos los vecinos han sido descartados). La búsqueda tabú mantiene una lista tabú para evitar ciclos. Cada arco invertido se mantiene en la lista tabú durante $TabuLength$ iteraciones. El valor de $TabuLength$ varía entre 1 y $Maxiter$ y se actualiza dependiendo de la calidad del vecino escogido en cada iteración. Si es mejor que el de la iteración anterior, $TabuLength$ se incrementa en uno; si no, $TabuLength$ se reduce en uno (a no ser que $TabuLength$ =

1). Finalmente, $TabuLength$ es 1 si la nueva solución es mejor que la mejor solución encontrada hasta el momento.

Combinando el algoritmo genético con la búsqueda local

En el algoritmo memético la búsqueda local se aplica a las soluciones resultantes de la decodificación de los cromosomas. Para mantener un equilibrio razonable entre diversificación e intensificación, la búsqueda local sólo se le aplica a una fracción de los cromosomas de la población. Si la búsqueda local devuelve una solución mejor que la que partida, el cromosoma se recodifica de modo que los órdenes de las tareas en la solución se trasladan al cromosoma, es decir, se sigue un modelo de evolución Lamarckiana. Cuando no se aplica búsqueda local, se realiza la recodificación del cromosoma (que denominamos evolución Lamarckiana débil).

Nótese que el algoritmo $OG\&T$ se utiliza con los conjuntos de opciones A y A', y no con el conjunto B, en el algoritmo memético. Esto se hace así para aumentar diversificación y compensar el efecto de la intensificación introducida por la búsqueda local.

Estudio experimental para el JSO

Para analizar el rendimiento de los algoritmos propuestos y la contribución de los distintos elementos, como el generador de planificaciones $OG\&T$ y la búsqueda local, se realizó un estudio experimental sobre distintos bancos de ejemplos, unos tomados de la literatura y otros adaptando instancias de uso común del problema JSP.

En la experimentación con el algoritmo genético, se consideraron distintas versiones del decodificador $OG\&T$; en primer lugar tomando B como conjunto de opciones en cada iteración, pero también considerando distintas reducciones del espacio de búsqueda a través de distintos valores del parámetro δ (0, 0.25, 0.5 y 0.75). Asimismo, se estudió la influencia de la evolución Lamarkiana débil. Los resultados del algoritmo genético se compararon con las propuestas de la literatura, a saber el algoritmo propuesto en [55] que combina un algoritmo A^* con un algoritmo voraz que se ejecuta desde un porcentaje de nodos expandidos para obtener soluciones completas, y los algoritmos propuestos en [1], un algoritmo exacto de programación dinámica y dos algoritmos heurísticos aproximados.

Las principales conclusiones del estudio experimental con el algoritmo genético son las siguientes: la mejor opción para el algoritmo genético es utilizar evolución Lamarkiana y un valor del parámetro $\delta = 0.75$. Con esta configuración, el algoritmo genético es muy superior a los métodos heurísticos propuestos en [1] ya que encuentra soluciones de mucha más calidad en mucho menos tiempo. Con respecto al algoritmo de programación dinámica propuesto en [1] no es estrictamente comparable ya que este algoritmo es capaz de resolver óptimamente muchas de las instancias, mientras que el algoritmo genético no lo es. Pero los tiempos de ejecución difieren en varios órdenes de magnitud; mientras que el algoritmo de programación dinámica tarda hasta 4 horas en resolver alguna instancia (y alguna de ellas incluso no se resuelve en este tiempo), el algoritmo genético obtiene soluciones con un error relativo inferior al $4\,\%$ en menos de 1 segundo. Por último, con respecto al algoritmo propuesto en [55], el algoritmo genético es competitivo en instancias de tamaño medio y grande.

En la experimentación con el algoritmo memético, para contrarrestar el efecto de la búsqueda local en la

Artículo	Contribución	Notas
NACO	Algoritmo genético para el JSO.	Adaptación de operadores genéticos al problema JSO.
	Incorporación de algoritmo $OG\&T$ como generador de schedules de nuestro AG.	Utilizamos el cromosoma para elegir una tarea entre el conjunto de opciones del $OG\&T$ en cada iteración.
	Modificaciones del $OG\&T$	Se reduce el espacio de búsqueda del $OG\&T$ limitando el tiempo de inactividad de las máquinas y operarios, obteniendo espacios de búsqueda que en media tienen mejor makespan, a costa de la propiedad de dominancia.
	Evolución Lamarckiana.	Se recodifican los cromosomas tras su decodificación poniendo los genes que representan cada tarea en el orden en el que se planifican.
ASOC	Algoritmo memético para el JSO	Se usa el algoritmo genético de la parte anterior y se le añade una búsqueda local.
	Estructura de vecindad para el JSO	Adaptación de la estructura de vecindad $\mathcal{N}$ para este problema, obteniendo la estrucutra $\mathcal{N}^{\mathcal{O}}$.
	Estrategias de búsqueda local para el JSO	Hill-climbing y búsqueda tabú,
	Comparación entre generadores de schedules	Se usa el $OG\&T$ con los conjuntos A y A' con el objetivo de encontrar el que nos de un mejor balance entre diversificación e intensificación.

Tabla 3.1: Tabla resumen de la sección 3.1.

disminución de la diversidad de la población, la decodificación de los cromosomas se realizó considerando los conjuntos de opciones A y A' en cada iteración del algoritmo $OG\&T$. En principio, el uso del conjunto A ofrece soluciones más diversas que A', pero al mismo tiempo la calidad media de las soluciones obtenidas con A' es superior a la media obtenida con A. Asimismo, se consideraron tres opciones de búsqueda local: sin búsqueda local, con escalada o hill-climbing y búsqueda tabú; en todos los casos considerando evolución Lamarckiana.

Los resultados muestran claramente que es mejor utilizar el conjunto de opciones A' en la decodificación de cromosomas y además que la búsqueda tabú es mejor que la búsqueda con escalada simple y esta a su vez es mejor que la ausencia de búsqueda local. Además, en relación con el resto de métodos de la literatura, el algoritmo memético es claramente superior, sobre todo en la resolución de instancias de gran tamaño.

Conclusiones

Las principales conclusiones que podemos extraer del estudio teórico y experimental realizado sobre el problema JSO es que la combinación adecuada de distintos métodos y el uso de conocimiento específico del problema permiten obtener soluciones de calidad para este problema. En esta tesis hemos utilizado algoritmos genéticos convencionales y métodos de búsqueda local que explotan una estructura de vecindad concreta $\mathcal{N}^{\mathcal{O}}$, la cual es una extensión de la estrategia $\mathcal{N}$ definida para el JSP. Dado que existen otras metaheurísticas que han demostrado ser más efectivas en otros problemas, como por ejemplo scatter search combinada con path relinking [26, 24] es posible que estas estrategias proporcionen mejores resultados también para el JSO. Además, el problema JSO ofrece otras opciones para definir nuevas estructuras de vecindad, algunas mediante extensiones de otras definidas para el JSP, pero la presencia de los operarios hace que el problema tenga una estructura diferente de modo que posiblemente se puedan obtener estructuras enfocadas a cambios en la asignación de operarios a las tareas. Así pues, el uso de otras metaheurísticas y el desarrollo de nuevas estructuras de vecindad se plantean como líneas de trabajo futuro.

La Tabla 3.1 contiene un resumen de los contenidos de esta sección.

En el capítulo 5 se incluyen las siguientes publicaciones dónde se detallan las contribuciones discutidas en esta sección:

- Raúl Mencía, María R. Sierra, Carlos Mencía, Ramiro Varela. A genetic algorithm for job-shop scheduling with operators enhanced by weak Lamarckian evolution and search space narrowing. *Natural Computing*. 13(2): 179-192. 2014. DOI: 10.1007/s11047-013-9373-x.

- Raúl Mencía, María R. Sierra, Carlos Mencía, Ramiro Varela. Memetic Algorithms for the Job Shop Scheduling Problem with Operators. *Applied Soft Computing*, 2015, 34: 94-105. 2015. DOI: 10.1016/j.asoc.2015.05.004.

3.2. El problema de planificación de tareas y operarios con habilidades y restricciones de precedencia arbitrarias

En esta sección describimos las principales contribuciones de esta tesis en relación con el problema de planificación de tareas y operarios en el que estos tienen distintas habilidades y por lo tanto solamente son capaces de asistir a algunas de las tareas [44, 46, 45]. Además, en este problema se consideran restricciones de precedencia arbitrarias entre las tareas de modo que no están organizadas en un conjunto de trabajos. Este problema fue propuesto en [2] donde se denomina JSSO (*Job Shop Scheduling with Skilled Operators*).

3.2.1. Antecedentes e hipótesis

En el problema JSSO, tenemos un conjunto $\mathcal{M}$ de q máquinas, un conjunto $\mathcal{O}$ de p operarios y un conjunto $\mathcal{T}$ de n operaciones o tareas. Durante su tiempo de procesamiento p_u, la tarea u necesita dos recursos, una máquina en particular $m_u \in \mathcal{M}$ y un operario $o \in \mathcal{O}_u \subseteq \mathcal{O}$ con la habilidad de asistirla (que pertenece al subconjunto $\mathcal{O}_u$ de los operarios con esta habilidad). Asumimos que el tiempo de procesamiento p_u de la tarea es independiente al operario que la asista. Además, se da un grafo de tareas $(\mathcal{T}, E)$ que contiene las relaciones de precedencia (arbitrarias) entre ellas. En este grafo, cada nodo representa una tarea y cada arco $(u, v) \in E$ representa que u debe ser procesada completamente antes de que v comience. El objetivo es asignar a cada tarea $u \in \mathcal{T}$ un tiempo de inicio st_u y un operario o_u de modo que se minimice el makespan y se satisfagan las siguientes restricciones:

- Las tareas deben ser procesadas siguiendo el orden expresado por el grafo de tareas.

- Cada tarea debe ser asistida por operario con la habilidad para asistirla durante todo su tiempo de procesamiento.

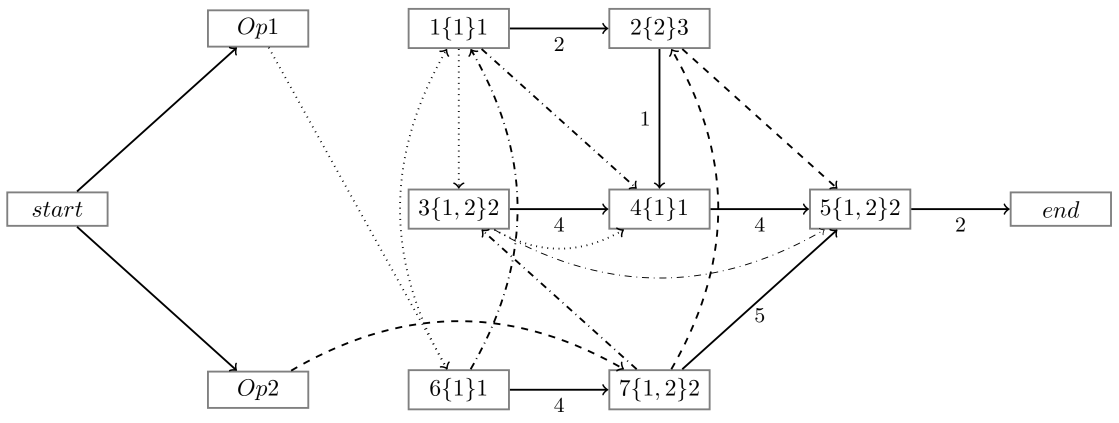


Figura 3.2: Un ejemplo de grafo solución para el problema JSSO con 7 tareas y 2 operarios con diferentes habilidades.

- Solo se puede ejecutar una tarea en cada máquina en cada momento y cada operario solo puede asistir una operación en cada momento.
- No se puede interrumpir el tiempo de procesamiento de ninguna tarea.

Dado que también se trata de una extensión de otros problemas de scheduling como el JSP o el JSO, nuestra hipótesis de trabajo es que el diseño de generadores de planificaciones y de estrategias de vecindad propias para el problema JSSO, generalizando las ideas utilizadas en los problemas anteriores, nos permitirá desarrollar algoritmos eficientes para calcular buenas soluciones para el JSSO.

3.2.2. Contribuciones y resultados

De acuerdo con la hipótesis de partida, proponemos en primer lugar un modelo disyuntivo para representar instancias del problema y soluciones. Este modelo es la base para el estudio de las propiedades del problema y para el desarrollo de generadores de schedules. Con estos generadores, se desarrolló un algoritmo genético que proporciona resultados muy competitivos para el problema JSSO. En las secciones siguientes detallamos estas contribuciones.

Modelo disyuntivo para el JSSO

El modelo disyuntivo para el JSSO es una generalización del modelo propuesto para el JSO. Cada nodo del grafo incluye el conjunto de operarios que pueden asistir a la tarea correspondiente. Hay varios tipos de arcos para expresar las restricciones de precedencia entre las tareas y las restricciones disyuntivas entre las tareas que utilizan la misma máquina o que pueden ser asistidas por el mismo operario. La Figura 3.2 muestra un ejemplo de grafo solución para una instancia del problema con 7 tareas y 2 operarios. [54]

El algoritmo $SOG\&T$

Este algoritmo tiene una estructura muy similar al algoritmo $OG\&T$. Así, el algoritmo planifica las tareas de una en una un siguiendo un orden compatible con el grafo de tareas. Cuando una tarea se planifica, se le asigna un tiempo de inicio y un operario que tiene capacidad para asistir a la tarea. La diferencia esencial con el algoritmo $OG\&T$ está en la determinación del conjunto de opciones en cada iteración debido a que no todos los operarios que pueden asistir a una determinada tarea están disponibles en el mismo instante de tiempo.

El conjunto de operaciones elegibles para ser planificadas en la iteración actual está formado por todas las operaciones cuyas predecesoras en el grafo de tareas ya han sido planificadas. Dado que cada una de estas tareas puede a su vez ser asistida por distintos operarios, el conjunto de opciones de planificación $\mathcal{A}$ está formado por una serie de opciones de la forma (u, o). En general, para cada tarea hay varias opciones y si se elige la opción (u, o), es decir la tarea u es asistida por el operario o, el tiempo de inicio de la operación vendrá dado por el máximo de los tiempos de fin de sus tareas predecesoras (en el grafo de tareas, en la máquina y en el operario). Con lo cual cada tarea puede ser planificada en tiempos de inicio distintos en una determinada iteación, lo que dificulta la definición de espacios de búsqueda dominantes.

El Algoritmo 3.1 muestra la estructura general del generador de planificaciones $SOG\&T$. En cada iteración, se escoge una opción (u, o), de modo no determinista, de un subconjunto $\mathcal{X}$ del conjunto de opciones de planificación $\mathcal{A}$.

Algoritmo 3.1 Generador de planificaciones $SOG\&T$. Dada una instancia $\mathcal{P}$ del problema JSSO, produce una planificación factible para $\mathcal{P}$ en n pasos, siendo n el número de taras.

```
Data: A SPSO problem instance $\mathcal{P}$
Result: A feasible schedule for $\mathcal{P}$
$A = \{u \in \mathcal{T}, \neg\exists(w, u) \in E\}$;
for i=1 to n do
    $\mathcal{A} = \{(u, o), u \in A, o \in \mathcal{O}_u\}$;
    $\mathcal{X} =$ a subset of $\mathcal{A}$;
    choose $(u, o) \in \mathcal{X}$ non deterministically;
    set $st_u = st_u(o)$ and $o_u = o$;
    add $u$ to $SC$ and update $G_{SC}$;
    $A = \{v, v \notin SC, P(v) \subseteq SC\}$;
end
return the built schedule $G_{SC}$;
```

Espacios de búsqueda

Hemos definido distintos conjuntos de opciones de planificación, algunos de los cuales dan lugar a espacios de búsqueda dominantes mientras que otros dan lugar a espacios no dominantes. Si tomamos $\mathcal{X} = \mathcal{A}$, tenemos un espacio de búsqueda dominante. Sin embargo, este espacio de búsqueda puede ser demasiado grande, incluso para instancias pequeñas. Afortunadamente, podemos reducir el espacio de búsqueda, sin perder la dominancia, del siguiente modo. La idea es considerar en principio la opción (v^*, o^*) de $\mathcal{A}$ tal que con esta opción el tiempo de fin C^* de la tarea v^* es el mínimo de todas las opciones en $\mathcal{A}$. A partir de C^*, definimos los conjuntos de opciones $\mathcal{A}'$ y $\mathcal{B}$, tales que:

- $\mathcal{A}'$ está formado por las opciones de $\mathcal{A}$ que pueden empezar antes de C^*.
- $\mathcal{B}$ está formado por las opciones de $\mathcal{A}'$ que utilizan la misma máquina que v^* o son asistidas por el operario o^*.

Nótese que cada opción de $\mathcal{A}'$ establece un tiempo de inicio para una tarea en el intervalo $[T_{\mathcal{A}}, C^*)$ y que $\mathcal{B}$ restringe más este intervalo a tiempos de inicio en $[T_{\mathcal{B}}, C^*)$ siendo $T_{\mathcal{B}} \geq T_{\mathcal{A}}$, por lo que es probable que en media, las opciones de $\mathcal{A}'$ tengan un makespan menor que las de $\mathcal{B}$, aunque en $\mathcal{B}$ haya menos opciones. Se puede demostrar que los conjuntos $\mathcal{B}$, $\mathcal{A}$ y $\mathcal{A}'$ dan lugar a espacios de búsqueda dominantes.

Los siguientes conjuntos de opciones no contienen en general al conjunto $\mathcal{B}$ y, en consecuencia, los espacios de búsqueda que generan pierden la propiedad de dominancia. No obstante, estos espacios pueden ser interesantes si son pequeños y si además contienen una buena proporción de soluciones de alta calidad. Como utilizamos algoritmos genéticos, el hecho de no poder garantizar la existencia de al menos una solución óptima en el espacio de búsqueda no es un gran problema, ya que un algoritmo genético no puede garantizar que se alcance la mejor solución del espacio de búsqueda en un tiempo finito.

Para generar espacios de búsqueda no dominantes consideramos los conjuntos de opciones $\mathcal{A}$, $\mathcal{A}'$ y $\mathcal{B}$ y en los tres casos descartamos opciones que dan lugar a los mayores periodos de inactividad de las máquinas y los operarios. Hacemos esto introduciendo un parámetro δ $(0 < \delta \leq 1)$ y así obtenemos los espacios de búsqueda $\mathcal{A}(\delta)$, $\mathcal{A}'(\delta)$ y $\mathcal{B}(\delta)$. δ regula el tiempo que las máquinas o los operarios pueden estar desocupados mientras haya una tarea que pueda hacer uso de ellos. Cuanto menor sea δ, menor será este tiempo. Estos conjuntos de opciones dan lugar a distintas versiones del generador de planificaciones $SOG\&T$. Todas ellas han sido utilizadas como esquema de decodificación de un algoritmo genético que hemos desarrollado para resolver el problema JSSO y que se describe a continuación.

Elementos del algoritmo genético para el problema JSSO

Como en el caso del problema JSO, utilizamos un algoritmo genético convencional y lo adaptamos al problema JSSO. En primer lugar definimos un esquema de codificación en el que un cromosoma está formado por dos permutaciones de símbolos denominadas respectivamente la secuencia de tareas y la secuencia de operarios.

La secuencia de tareas es una permutación de números $1, \ldots, n$, y representa ordenes de tareas, mientras que la segunda es la secuencia de operarios y viene dada por una permutación con repeticiones de los símbolos $1, \ldots, p$ de tamaño n y representa preferencias de operarios. Si la tarea u aparece antes que la tarea v en la primera permutación, entonces u se planificará preferentemente antes que v. Al mismo tiempo, la segunda permutación representa prioridades para la asignación de operarios a tareas. Para evitar que se considere para todas las tareas el mismo orden de operarios, el primer operario para una tarea será el que esté en la misma posición que la tarea en la primera permutación y el resto de los operarios se tomarán siguiendo el cromosoma como estructura circular. La ordenación de las tareas y la asignación de los operarios no dependen únicamente del cromosoma, sino también del algoritmo de decodificación; por esa razón los órdenes expresados en el cromosoma se consideran tentativos.

Esta codificación tiene ciertas características que la hacen atractiva. Por ejemplo, es simple de forma que permite el diseño de algoritmos eficientes para el cruce y la mutación. El principal inconveniente es que el número de réplicas de un símbolo en la secuencia de operarios es variable y por lo tanto un operario podría desaparecer del cromosoma. Por esa razón hemos tenido que introducir un esquema de reparación heurística en el algoritmo de decodificación.

Como algoritmo de decodificación se utiliza el generador $SOG\&T$ considerando los órdenes de tareas y operarios del cromosoma para la selección de la opción del conjunto $\mathcal{X}$ en cada iteración. Así, este algoritmo elige en cada iteración la opción (u, o) de $\mathcal{X}$ que esté más a la izquierda en el cromosoma, esto es, que satisface los dos siguientes criterios:

- u es la tarea que está más a la izquierda en la secuencia de tareas del cromosoma de todas las tareas que aparecen en $\mathcal{X}$, y
- o es el primer operario que puede asistir a u en la lista de preferencias definida por la secuencia de operarios de todos los operarios que aparecen en alguna opción de $\mathcal{X}$ para asistir a u. Si ninguno de los operarios que pueden asistir a la tarea u está presente en la secuencia de operarios en el cromosoma, entonces se selecciona el operario presente en $\mathcal{X}$ que permita que u comience lo antes posible.

Operadores genéticos para el JSSO

Como operador de cruce, usamos cruce de orden (OX) [14, 50] con ambas secuencias de los cromosomas a la vez. Este algoritmo traslada la subsecuencia de símbolos entre dos puntos de corte de un padre a un hijo y el orden relativo de los valores restantes del otro padre. OX puede ser una buena opción para el JSSO debido al hecho de que algunas características importantes, como el orden de procesamiento de las tareas sobre las máquinas o las prioridades de los operarios para asistir las tareas, puede ser transmitidas de los padres a los hijos. Para evitar un efecto disruptivo fuerte del cruce, los puntos de corte serán los mismos en ambas secuencias de los cromosomas (tareas y operarios).

Asimismo, consideramos mutación simple (SM) que consiste en intercambiar dos posiciones consecutivas del cromosoma. Pensamos que es apropiado porque puede producir cambios pequeños. Como antes, las mismas posiciones serán intercambiadas en ambas secuencias del cromosoma. También usamos mutación de operarios (OM) para obtener diferentes distribuciones de preferencias de operarios. OM cambia aleatoriamente un valor de la secuencia de operarios. Cuando un cromosoma es mutado, se escoge SM o OM con una probabilidad de 0,5.

Para el resto de los componentes, el algoritmo genético sigue la misma estructura que el algoritmo genético utilizado para el JSO y presentado en secciones anteriores de esta tesis. Los cromosomas de la población inicial se generan aleatoriamente, siguiendo una distribución uniforme. Se utiliza también el modelo de evolución Lamarckiana débil comentado en la sección anterior. Esto significa que se recodifica el cromosoma a partir de los órdenes de las tareas en la solución.

Estudio experimental para el JSSO

Realizamos un estudio experimental con el objetivo de evaluar las características de los espacios de búsqueda propuestos, así como analizar el comportamiento del algoritmo genético y compararlo con el estado del arte. Para esto, hicimos uso de un banco de instancias de diferentes tamaños y características. Para evaluar los espacios de búsqueda, primero generamos soluciones aleatorias en cada espacio de búsqueda y analizamos la calidad y la distribución de los valores de makespan. Después, analizamos la evolución del algoritmo en cada espacio y, en particular, la influencia del modelo evolutivo (con o sin recodificación) en la convergencia del algoritmo. Este amplio estudio experimental se realizó sobre tres bancos de instancias; las instancias utilizadas en [2], instancias derivadas de instancias para el JSP e instancias derivadas de instancias para el RCPSP (*Resource Constrained Project Scheduling Problem*).

Análisis de los espacios de búsqueda

Para hacernos una idea inicial sobre la calidad de las soluciones de los diferentes espacios de búsqueda considerados, hicimos un experimento en el que generamos 1000 planificaciones aleatorias en cada uno de ellos y analizamos la calidad de estas soluciones. En los resultados observamos que las soluciones generadas considerando en conjunto de opciones $\mathcal{A}$ son mucho más diversas que las que generan los otros dos conjuntos A' y B, pero también mucho peores en términos de makespan, por lo que $\mathcal{A}$ no parece una buena opción. $\mathcal{A}'$ y $\mathcal{B}$, por otro lado, obtienen resultados muy similares, siendo un poco mejores los de $\mathcal{A}'$.

Realizamos también experimentos similares con los espacios no dominantes considerando diferentes valores del parámetro δ. Observamos que, en general, cuanto menor sea el valor de δ, mejores son los resultados. De nuevo pudimos observar que $\mathcal{A}(\delta)$ obtene mucho peores resultados de $\mathcal{A}'(\delta)$ y $\mathcal{B}(\delta)$. Ocurre también que con $\mathcal{A}'(\delta)$ los resultados son algo mejores que con $\mathcal{B}(\delta)$. Todo esto nos llevó a descartar $\mathcal{A}(\delta)$ en los siguientes experimentos.

Análisis de la convergencia del algoritmo genético

Con respecto a la convergencia del algoritmo genético, es bien conocido que los mecanismos que hacen que los algoritmos genéticos busquen en espacios con soluciones de buena calidad media, pueden dar lugar a convergencia prematura. En nuestros trabajos introducimos mecanismos de este tipo, como son la recodificación y la definición de diferentes espacios de búsqueda. Por lo tanto, es procedente analizar cómo pueden estos elementos afectar la convergencia del algoritmo.

Primero consideramos el efecto de la recodificación en los espacios de búsqueda $\mathcal{A}'$ y $\mathcal{B}$. En todos los resultados se observa que la recodificación mejora la convergencia del algoritmo y que $\mathcal{A}'$ obtene mejores resultados que $\mathcal{B}$. Analizamos también la convergencia del algoritmo genético para los espacios $\mathcal{A}'(\delta)$ y $\mathcal{B}(\delta)$. Para esto, consideramos 5 valores diferentes para δ: $\sim$0, 0.25, 0.5, 0.75 y 1, e instancias de diferentes tamaños, llegando a la conclusión de que cuanto más grande es el tamaño de la instancia, el valor más adecuado para δ debe ser más pequeño. A partir de estos experimentos preliminares consideramos que espacio de búsqueda más adecuado es el que se genera a partir del conjunto $\mathcal{A}'(\delta)$.

Evaluación del algoritmo genético

Para la evaluación del algoritmo genético, considerando el espacio de búsqueda $\mathcal{A}'(\delta)$, la opción de recodificación y diferentes valores para δ, utilizamos dos grupos de instancias. El primer grupo está formado por instancias de tamañoo medio y el segundo por instancias de tamaño grande. Para el primer grupo, consideramos valores para δ en $\sim$0, 0.25, 0.5, 0.75, 1 y para el segundo $\sim$0 y 1. Los resultados de estas pruebas nos llevaron a concluir que nuestras sospechas previas eran ciertas, cuanto mayor sea la instancia, menor tendrá que ser el valor de δ.

Comparación con otros métodos

En [2], los autores proponen dos métodos heurísticos MSB-DOS y STA-OMSB y también proponen una formulación MILP que se implementa en Cplex 12.2. En su estudio muestran que el mejor de sus métodos es MSB-DOS. Los resultados de nuestro estudio muestran que el algoritmo genético es mucho más rápido que estos métodos y que ofrece soluciones mucho mejores que el heurístico MSB-DOS.

Conclusiones

A raíz de nuestro estudio experimental concluimos que combinar un generador de planificaciones no trivial como el $SOG\&T$ con un algoritmo genético permite encontrar buenas soluciones para el problema JSSO propuesto en [2]. Asimismo, observamos que entre los espacios de búsqueda considerados, el más adecuado depende del tamaño de la instancia del problema; concretamente para instancias grandes merece la pena buscar en espacios de búsqueda no dominantes pero con soluciones de alta calidad media, mientras que para instancias pequeñas es preferible explotar espacios de búsqueda dominantes.

La Tabla 3.2 contiene un resumen de los contenidos de esta sección.

Artículo	Contribución	Notas
ICAPS	Algoritmo genético para el JSSO.	Adaptación de operadores genéticos al problema JSSO.
	Algoritmo $SOG\&T$	Diseñamos un algoritmo que en cada iteración planifica una opción de tarea y operario. Este algoritmo se basa en el algoritmo $OG\&T$ para el JSP.
ICAE	Modificaciones al $SOG\&T$	Se reduce el espacio de búsqueda del $SOG\&T$, obteniendo espacios de búsqueda que en media tienen un makespan menor, a costa de perder la dominancia del espacio de búsqueda.
	Experimentación	Se realizó una experimentación extensa que nos ayudó a discernir qué configuración del algoritmo es más adecuada en cada situación.

Tabla 3.2: Tabla resumen de la sección 3.2.

En el capítulo 5 se incluyen las siguientes publicaciones donde se detallan las contribuciones discutidas en esta sección:

- Raúl Mencía, María R. Sierra, Carlos Mencía, Ramiro Varela. Schedule Generation Schemes and Genetic Algorithm for the Scheduling Problem with Skilled Operators and Arbitrary Precedence Relations. *Twenty-Fifth International Conference on Automated Planning and Scheduling. (ICAPS 2015).* 165-173.

- Raúl Mencia, María R. Sierra, Carlos Mencía, Ramiro Varela. Genetic Algorithm for the scheduling problem with arbitrary precedence relations and skilled operators. *Integrated Computer-Aided Engineering*. 23(3): 269-285. 2016. DOI: 10.3233/ICA-160519.

3.3. El problema de asignación vehículos y conductores a rutas de transporte

El tercero de los problemas que hemos estudiado en esta tesis es una versión real del problema asignación de vehículos y conductores a rutas de transporte de viajeros, conocido en la literatura como VCSP (*Vechicle and Crew Scheduling Problem*) En el VCSP se trata de asignar vehículos y conductores a rutas de transporte programadas, de modo que se satisfagan una serie de restricciones y se optimicen una serie de objetivos.

3.3.1. Antecedentes e hipótesis

El problema VCSP ha sido tratado en la literatura durante las dos últimas décadas [30, 47]. Existe un banco de ejemplos de uso común, propuesto en [30], y además los distintos autores consideran casos reales de compañías particulares. Para resolver este problema se han utilizado técnicas exactas de programación matemática que requieren un tiempo de computación muy elevado; nuestra hipótesis es que se pueden utilizar metaheurísticas para obtener soluciones aceptables en un tiempo razonable. Para el diseño de metaheurísticas, proponemos seguir una metodología de trabajo similar a la utilizada en la resolución de otros problemas de scheduling como el JSO o el JSSO, entre otros.

3.3.2. Contribuciones

Nuestra contribucion en este caso es el analisis y formalización de un problema real en colaboración con expertos de una compañía de transporte de viajeros por carretera y el diseño de un modelo gráfico para representar instancias y soluciones.

Formulación del problema de asignación de conductores y vehículos a rutas de transporte

De forma resumida, la formalización del problema es la que se indica a continuación.

Los datos del problema son los siguientes:

- Un conjunto de *depósitos* o *bases*. Un subconjunto de estos disponen de talleres de reparación.
- Un conjunto ordenado de *clases de vehículos*.
- Un horizonte de planificación $[0, \mathbf{H}]$.
- Un conjunto de *vehículos*, donde cada vehículo pertenece a una base y a una clase.
- Un conjunto de *conductores*, donde cada conductor pertenece a una base y está capacitado para conducir vehículos de un subconjunto de clases de vehículos.
- Un conjunto de *puntos de relevo*, que son localizaciones (ciudad, pueblo, base, ...) donde un vehículo puede ser aparcado durante un tiempo sin ser atendido y un conductor puede descansar durante un tiempo y cambiar de vehículo.

- Un conjunto de *rutas* o *expediciones* fijadas a lo largo del horizonte de planificación, donde cada expedición, está definida por una clase y una secuencia de trayectos. Un trayecto está definido por su origen y su destino, además de sus tiempos de salida y llegada.

- Un conjunto de *tareas de revisión* que tienen que ser realizadas sobre un subconjunto de los vehículos. Para cada uno de estos vehículos hay unas ventanas de tiempo y de kilómetros recorridos que determinan cuándo se debe realizar la revisión.

- Un conjunto de *deadheads* candidatos. Cada deadhead representa un viaje de un vehículo sin pasajeros entre dos puntos de relevo. Hay diferentes tipos de deadheads. Los deadheads pueden ser de distintas clases dependiendo de si el vehículo se dirige al principio de una expedición, a la base para finalizar el servicio en el horizonte de planificación, o bien a una operación de revisión.

- Una serie de normas laborales que especifican las conduciones de trabajo de los conductores (descansos, jornadas, etc).

- Un conjunto de reglas que especifican los periodos de tiempo en los que un conductor puede incurrir en gastos de dietas.

- Una serie de parámetros que se utilizan en la evaluación de las restricciones, como el número máximo de vehículos en un taller, el número máximo de deadheads en la solución, etc.

Las restricciones principales establecen que la secuencia de tareas (trayectos y deadheads) y descansos asignadas a los conductores, así como las tareas y períodos de revisión asignados a los vehículos deben ser factibles. En particular todos los vehículos y conductores deben regresar a sus bases al final del horizonte de planificación. Además, hay restricciones que limitan los costes máximos de dietas y los debidos a deadheads.

La función objetivo principal es la suma ponderada de vehículos y conductores que debe ser minimizada. Hay otras funciones secundarias que tienen que ver con la racionalización de los períodos de descanso de los conductores.

Modelo gráfico

Una vez formulado el problema, diseñamos un modelo gráfico, con el objetivo de representar instancias y soluciones del problema. El objetivo es que este modelo sea de utilidad para estudiar las propiedades del problema y en particular para diseñar esquemas de generación de planificaciones y estructuras de vecindad, de la misma forma que lo han sido los modelos disyuntivos definidos para distintas versiones del problema job shop scheduling.

En este modelo, un grafo problema es una estructura (V, A), donde el conjunto V incluye nodos para representar bases y tareas asociadas a trayectos, deadheads y tareas de reparación; y el conjunto A incluye arcos para representar compatibilidades entre tareas consecutivas para vehículos y conductores, así como movimientos posibles de vehículos y conductores entre bases y trayectos.

Un grafo solución es un subgrafo del grafo que representa al problema que incluye todos los nodos asociados a bases y trayectos, y un subconjunto de los nodos asociados a deadheads y tareas de reparación.

Asimismo, incluye un subcojunto de arcos que conecta a los nodos seleccionados. Tanto los nodos como los arcos están etiquetados con vehículos y conductores asociados a las tareas correspondientes.

La Figura 3.3 es un grafo problema para una instancia pequeña de este problema.

La Tabla 3.3 contiene un resumen de los contenidos de esta sección.

Artículo	Contribución	Notas
Informe técnico	Formulación del VCSP de una compañia real de transporte.	Para su realización fueron necesarias multiples runiones con miembros de dicha compañía.
	Modelo gráfico para el problema	Diseñamos un modelo gráfico para el problema de asignación de vehículos y conductores de la compañía.

Tabla 3.3: Tabla resumen de la sección 3.3.

En el capítulo 5 se incluye el siguiente informe técnico que describe la formulación del problema real y el modelo gráfico.

- Raúl Mencía, María Sierra, Carlos Mencía, Jorge Puente, Ramiro Varela. Technical Report: A Real-Life Vehicle and Crew Scheduling Problem for Extra-Urban Bus Passenger Routes.

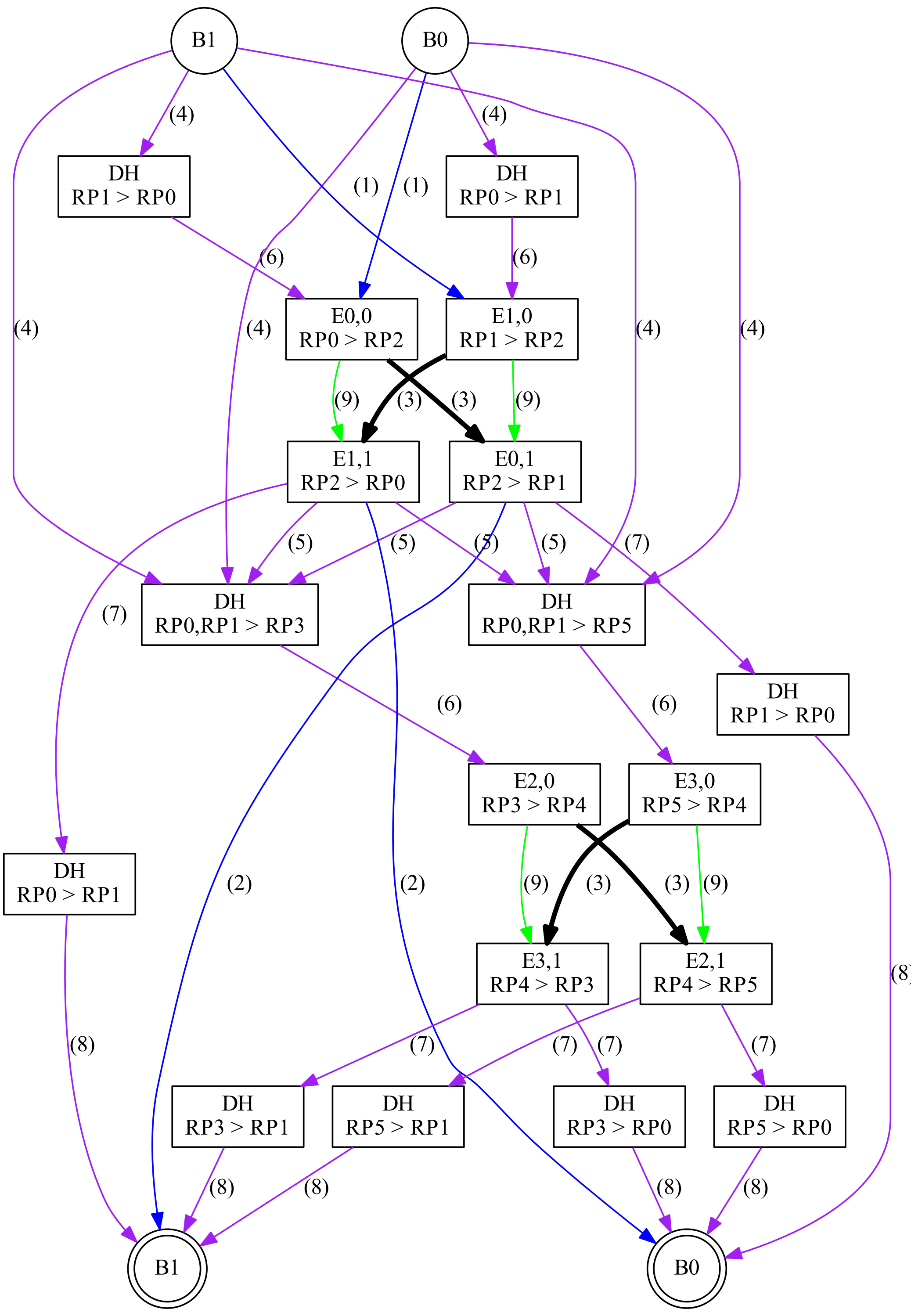


Figura 3.3: Un ejemplo de grafo problema para una instancia pequeña de VCSP.

Capítulo 4

CONCLUSIONES

En este capítulo se resumen las principales aportaciones de la tesis y se describen brevemente algunas líneas de trabajo futuro.

4.1. Aportaciones de la tesis

El objetivo principal planteado al inicio de esta tesis fue el desarrollo de algoritmos para resolver problemas de scheduling con múltiples recursos, considerando en principio dos campos de aplicación: en primer lugar problemas formales de la familia job shop scheduling con operarios, y en segundo lugar problemas reales de asignación de vehículos y conductores a rutas de transporte de viajeros. En líneas generales, podemos decir que con respecto a la primera familia de problemas el objetivo se ha alcanzado de forma muy satisfactoria ya que hemos desarrollado algoritmos que compiten o mejoran al estado de arte, mientras que en el segundo caso el objetivo se ha cubierto solamente de forma parcial. En este caso hemos analizado un problema real contando con la colaboración de expertos, hemos dado una definición formal del mismo y propuesto un modelo gráfico de representación de instancias y soluciones, dejando el diseño de algoritmos efectivos como tema de trabajo futuro.

De forma más detallada, hemos desarrollado un algoritmo genético para el problema job shop scheduling con operarios, una versión de clásico job shop scheduling en la que aparece un nuevo recurso, un operario humano que debe atender las operaciones de los trabajos. Este algoritmo utiliza como decodificador al generador de planificaciones $OG\&T$, propuesto en [55], que está basado en el esquema de generación de planificaciones $G\&T$, propuesto en [19]. Hemos considerado algunas modificaciones de este algoritmo que producen una clara mejora en la calidad de las soluciones. En primer lugar una estrategia para reducir el espacio de búsqueda, a costa de perder la propiedad de dominancia, y en segundo lugar un modelo de evolución lamarckiana débil, que consiste en trasladar al cromosoma los órdenes de las tareas en la solución. Posteriormente, desarrollamos un algoritmo memético añadiendo una estrategia de búsqueda local. Para esto diseñamos una estructura de vecindad para el problema, denominada $\mathcal{N}^{\mathcal{O}}$, que es una extensión de la estructura $\mathcal{N}$ propuesta en [34] para el job shop scheduling. Consideramos dos estrategias de búsqueda local, hill-climbing y búsqueda tabú.

Otro problema en el que hemos hecho aportaciones importantes es el problema de planificación de tareas y operarios con habilidades y restricciones de precedencia arbitrarias, propuesto en [2] donde se denomina *Job Shop Scheduling with Skilled Operators* (JSSO). Proponemos en primer lugar un modelo disyuntivo para representar instancias del problema y soluciones. Luego, para generar planificaciones diseñamos el algoritmo $SOG\&T$, que tiene una estructura muy similar al algoritmo $OG\&T$. Con distintas variantes de este algoritmo definimos espacios de búsqueda dominantes y espacios no dominantes. Luego diseñamos un algoritmo genético para el JSSO. Para esto definimos un esquema de codificación y operadores genéticos específicos para este problema. Este algoritmo genético utiliza el algoritmo $SOG\&T$ en la decodificación de cromosomas y sigue el modelo de evolución lamarckiana débil.

Por último, en colaboracion con expertos de una compañía de transporte de viajeros por carretera, hemos analizado y formalizado un problema real de asignacion de vehiculos y conductores a rutas programadas de transporte. En este caso nuestra principal aportacion es la defincion formal del problema y el diseño de un modelo gráfico para representar instancias y soluciones. Este modelo servirá de base para el diseño de generadores de planificaciones y estrategias de vecindad que a su vez se utilizaran para desarrollar metaheurísticas híbridas como algoritmos meméticos.

4.2. Trabajo futuro

Esta tesis deja abiertas varias líneas de investigación. En primer lugar tenemos como objetivo el desarrollo de algoritmos efectivos para distintas versiones del problema de asignación de vehículos y conductores a rutas de transporte. Para ello estamos trabajando en el desarrollo de un generador de planificaciones siguiendo la idea de los generadores $OG\&T$ y $SOG\&T$. Este generador se utilizará para desarrollar algoritmos exactos y metaheurísticas. Asimismo, tenemos como objetivo el desarrollo de estructuras de vecidad inspiradas en modificaciones del grafo solución que serán utilizadas en distintas metaheurísticas híbridas como algoritmos meméticos o GRASP, por ejemplo. Estos algoritmos se adaptarán a distintas versiones del problema real y también a distintas versiones consideradas en la literatura con el propósito de establecer comparaciones con otros métodos.

Con respecto a los problemas académicos de scheduling con múltiples recursos, la principal línea de trabajo irá encaminada al diseño de estrategias de vecindad para el problema de planificación de tareas y operarios con habilidades y restricciones de precedencia arbitrarias. Además, proponemos avanzar en este tipo de problemas considerando nuevas restricciones derivadas de las normativas laborales, como por ejemplo tiempos máximos de trabajo y tiempos de descanso de los operarios, entre otras.

Capítulo 5

COMPENDIO DE PUBLICACIONES

En este capítulo se incluyen los artículos que dan cuerpo a la tesis doctoral.

5.1. A genetic algorithm for job-shop scheduling with operators enhanced by weak Lamarckian evolution and search space narrowing

Se presenta la siguiente publicación:

- Raúl Mencía, María R. Sierra, Carlos Mencía, Ramiro Varela. A genetic algorithm for job-shop scheduling with operators enhanced by weak Lamarckian evolution and search space narrowing. *Natural Computing*. 13(2): 179-192. 2014. DOI: 10.1007/s11047-013-9373-x.
    - Estado: **Publicado.**

Nat Comput (2014) 13:179–192
DOI 10.1007/s11047-013-9373-x

A genetic algorithm for job-shop scheduling with operators enhanced by weak Lamarckian evolution and search space narrowing

Raúl Mencía · María R. Sierra · Carlos Mencía · Ramiro Varela

Published online: 16 May 2013

Abstract The job-shop scheduling problem with operators is a very interesting problem that generalizes the classic job-shop problem in such a way that an operation must be algorithm to solve this problem considering makespan minimization. The genetic algorithm uses permutations with repetition to encode chromosomes and a schedule generation scheme, termed *OG&T*, as decoding algorithm. This combination guaranties that at least one of the chromosomes represents and optimal schedule and, at the samhat machines and operators are idle while an operation is available to be processed. To improve the quality of the schedules for large instances, we use Lamarckian evolution and modify the *OG&T* algorithm to further reduce the idle time of the machines and operators, in this case at the risk of leaving all optimal schedules out of the search space. We conducted a large experimental study showing that these improvements allow the genetic algorithm to reach high quality solutions in very short time, and so it is quite competitive with the current state-of-the-art methods.

Keywords Job shop scheduling problem with operators · Genetic algorithms · Lamarckian evolution · Schedule generation schemes

1 Introduction

In this paper we propose a genetic algorithm (GA) to solve the job-shop scheduling problem with operators. This problem has been recently proposed in (Agnetis et al. 2011) and it is motivated by manufacturing processes in which part of the work is done by human operators sharing the same set of tools. The problem is formalized as a classical job-shop problem in which the processing of an operation on a given machine requires the assistance of one of the *p* available operators. In (Agnetis et al. 2011) the problem is studied and the minimal *NP*-hard cases are established. Also, a number of exact and approximate algorithms to cope with this problem are proposed and evaluated on a set of instances generated from that minimal relevant cases. The result of their experimental study makes it clear that instances with 3 jobs, 3 machines, 2 operators and a number of 30 operations per job are hard to solve to optimality.

The proposed GA exploits the permutation with repetition schema proposed in (Bierwirth 1995) for the classic job-shop scheduling problem. In order to get feasible schedules for the problem with operators, we use the schedule generation scheme, termed *OG&T*, proposed in (Sierra et al. 2013) to design a decoding algorithm. Combining these two elements, it is guaranteed that at least one chromosome represents an optimal schedule and, at the same time, the periods of time that machines and operators are idle while they can be used to process some operation are limited. So the GA searches for solutions over a reduced space with good solutions in average.

In order to improve the performance of the GA, we introduce two more elements. On the one hand, we exploit Lamarckian evolution. This kind of evolution is often used when a local searcher is combined with a genetic algorithm as it allows for the transmission of the learned characteristics from parents to offsprings. This technique was shown efficient, for example, in (González et al. 2012) where the authors proposed a memetic algorithm for a job-shop

R. Mencía · M. R. Sierra (✉) · C. Mencía · R. Varela
Department of Computer Science, University of Oviedo, Campus de Viesques s/n, Gijón 33271, Spain
e-mail: sierramaria@uniovi.es
URL: http://www.di.uniovi.es/iscop

scheduling problem with setup times. Even though we do not use any local searcher herein, we can exploit Lamarckian evolution due to the fact that the starting times of the operations in the schedule built by the *OG&T* decoder do not keep the same order as the operations in the chromosome encoding. So, the order given by the starting times may be coded back in the chromosome and so it can be easily transmitted to the offsprings by crossover an mutations. We call weak Lamarckian evolution to this procedure due to the fact that it is expected to produce much less changes in the chromosome than a classical local searcher.

On the other hand, we modified the strategy of the *OG&T* algorithm in order to further reduce the idle times of the machines and operators. In this way, we restrict the search to a smaller space in which the schedules are better in average. However, we have the risk of leaving all optimal schedules out of the search space. This search space narrowing strategy is similar to that used in (Bierwirth and Mattfeld 1999) for the classic job-shop scheduling problem and it is implemented by introducing a parameter $\delta \in [0, 1]$ to control the idle time intervals.

To assess the performance of the proposed GA, we have conducted a thorough experimental study across instances of different sizes and characteristics. The results of this study show that the genetic algorithm outperforms the approximate state-of-the-art algorithms, that as far as we know are those proposed in (Agnetis et al. 2011), and compares favorably with the exact methods proposed in (Agnetis et al. 2011) and (Sierra et al. 2013). The study also shows that the two new elements proposed allow GA to perform much more effectively.

The remaining of the paper is organized as follows. In Sect. 2 we define the job-shop scheduling problem with operators and give a disjunctive model to represent instances and schedules. In Sect. 3 we describe the original *OG&T* algorithm and how it may be adapted to search on a restricted space. Section 4 presents the main components of the GA used and discuss the two evolution models considered. Section 5 is devoted to the experimental study. Finally, we summarize the main conclusions of the paper and propose some ideas for future research in Sect. 6.

2 Description of the problem

Formally the job-shop scheduling problem with operators can be defined as follows. We are given a set of n jobs $\{J_1, \ldots, J_n\}$, a set of m resources or machines $\{R_1, \ldots, R_m\}$ and a set of p operators $\{O_1, \ldots, O_p\}$. Each job J_i consists of a sequence of v_i operations or tasks $(\theta_{i1}, \ldots, \theta_{iv_i})$. Each task θ_{il} has a single resource requirement $R_{\theta_{il}}$, an integer duration $p_{\theta_{il}}$ and a start time $st_{\theta_{il}}$ to be determined. A feasible schedule is a complete assignment of starting times and operators to operations that satisfies the following constraints: (i) the operations of each job are sequentially scheduled, (ii) each machine can process at most one operation at any time, (iii) no preemption is allowed and (iv) each operation is assisted by one operator and one operator cannot assist more than one operation at a time. The objective is finding a feasible schedule that minimizes the completion time of all the operations, i.e. the makespan. This problem was first defined in (Agnetis et al. 2011) and is denoted as $JSO(n, p)$. The significant cases of this problem are those with $p < min(n, m)$, otherwise the problem is a standard job-shop problem denoted as $J||C_{max}$.

We use the following disjunctive model for the $JSO(n, p)$. A problem instance is represented by a directed graph $G = (V, A \cup E \cup I \cup O)$. Each node in the set V represents either an actual operation, or any of the fictitious operations introduced with the purpose of giving the graph a particular structure: starting and finishing operations for each operator i, denoted O_i^{start} and O_i^{end} respectively, and the the dummy operations *start* and *end*.

The arcs in A are called *conjunctive arcs* and represent precedence constraints among operations of the same job. The arcs in E are called *disjunctive arcs* and represent capacity constraints. E is partitioned into subsets E_i with $E = \cup_{\{i=1,\ldots,M\}} E_i$. E_i includes an arc (v, w) for each pair of operations requiring the resource R_i. The set O of *operator arcs* includes three types of arcs: one arc (u, v) for each pair of operations of the problem, and arcs (O_i^{start}, u) and (u, O_i^{end}) for each operator node and operation. The set I includes arcs connecting node *start* to each node O_i^{start} and arcs connecting each node O_i^{end} to node *end*. The arcs are weighted with the processing time of the operation at the source node.

From this representation, building a solution can be viewed as a process of fixing disjunctive and operator arcs. A disjunctive arc between operations u and v gets fixed when one of (u, v) or (v, u) is selected and consequently the other one is discarded. An operator arc between u and v is fixed when (u, v), (v, u) or none of them is selected, and fixing the arc (O_i^{start}, u) means discarding (O_i^{start}, v) for any operation v other than u. Analogously for (u, O_i^{end}).

Therefore, a feasible schedule S is represented by an acyclic subgraph of G, of the form $G_S = (V, A \cup H \cup I \cup Q)$, where H expresses the processing order of operations on the machines and Q expresses the sequences of operations that are assisted by each operator. The makespan is the cost of a *critical path* in G_S. A critical path is a longest cost path from node *start* to node *end*.

Figure 1 shows a solution graph for an instance with 3 jobs, 3 machines and 2 operators. Discontinuous arrows

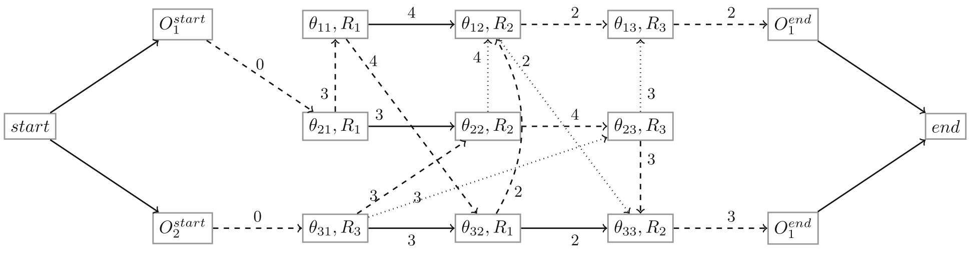


Fig. 1 A feasible schedule to a problem with 3 jobs, 3 machines and 2 operators

represent operator arcs. So, the sequences of operations assisted by operators O_1 and O_2 are (θ_{21}, θ_{11}, θ_{32}, θ_{12}, θ_{13}) and (θ_{31}, θ_{22}, θ_{23}, θ_{33}) respectively. In order to simplify the picture, only the operator arc is drawn when there are two arcs between the same pair of nodes. Continuous arrows represent conjunctive arcs and doted arrows represent disjunctive arcs; in these cases only arcs not overlapping with operator arcs are drawn. In this example, the critical path is given by the sequence (θ_{21}, θ_{11}, θ_{32}, θ_{12}, θ_{33}), so the makespan is 14.

In order to simplify expressions, we define the following notation for a feasible schedule. The *head* r_v of an operation v is the cost of the longest path from node *start* to node v, i.e. it is the value of st_v. The *tail* q_v is defined so as the value $q_v + p_v$ is the cost of the longest path from v to *end*. Hence, $r_v + p_v + q_v$ is the makespan if v is in a critical path, otherwise, it is a lower bound. PM_v and SM_v denote the predecessor and successor of v respectively on the machine sequence, PJ_v and SJ_v denote the predecessor and successor operations of v respectively on the job sequence and PO_v and SO_v denote the predecessor and successor operations of v respectively on the operator of v.

A partial schedule is given by a subgraph of G where some of the disjunctive and operator arcs are not fixed yet. In such a schedule, heads and tails can be estimated as

$$r_v = \max\left\{ \max_{J\subseteq P(v)} \left\{ \min_{j\in J} r_j + \sum_{j\in J} p_j \right\}, \max_{J\subseteq PO(v)} \left\{ \min_{j\in J} r_j + \sum_{j\in J} p_j \right\}, r_{PJ_v} + p_{PJ_v} \right\} \tag{1}$$

$$q_v = \max\left\{ \max_{J\subseteq S(v)} \left\{ \sum_{j\in J} p_j + \min_{j\in J} q_j \right\}, \max_{J\subseteq SO(v)} \left\{ \sum_{j\in J} p_j + \min_{j\in J} q_j \right\}, p_{SJ_v} + q_{SJ_v} \right\} \tag{2}$$

with $r_{start} = q_{end} = r_{O_i^{start}} = q_{O_i^{end}} = 0$ and where $P(v)$ denotes the disjunctive predecessors of v, so as for all $w \in P(v)$, $R_w = R_v$ and the disjunctive arc (w, v) is already fixed (analogously, $S(v)$ denotes the disjunctive successors of v). $PO(v)$ denotes the operator predecessors of v, i.e $w \in PO(v)$ if it is already established that $O_w = O_v$ and w is processed before v, so as the operator arc (w, v) is fixed (analogously, $SO(v)$ are the operator successors of v).

3 Schedule generation schemes

In (Giffler and Thompson 1960), the authors proposed a schedule generation scheme for the $J||C_{max}$ problem termed *G&T* algorithm. This algorithm has been used in a variety of settings for job-shop scheduling problems. For example, in (Brucker et al. 1994; Mencía et al. 2012; Mencía et al. 2013) it was used to devise a greedy algorithm, in (Bierwirth 1995) and (Mattfeld 1995) it was exploited as a decoder for genetic algorithms, and in (Sierra and Varela 2010) it was used to define the state space for a best-first search algorithm. Unfortunately, for other scheduling problems the original *G&T* algorithm is not longer suitable, but it has inspired the design of similar schedule builders such as the *EG&T* algorithm proposed in (Artigues et al. 2005) for the job-shop scheduling problems with sequence dependent setup times. In this section we propose to use a schedule generation scheme for the $JSO(n, p)$ which is also inspired in the *G&T* algorithm. This scheme is termed *OG&T* and it has been proposed in (Sierra et al. 2013). In the remainder of this section, we summarize the main characteristics of the *OG&T* algorithm.

As it is done by the *G&T* algorithm, the operations are scheduled one at a time in sequential order within each job. When an operation u is scheduled, it is assigned a starting time st_u and an operator O_i, $1 \le i \le p$. Let SC be the set of scheduled operations at a given time and G_{SC} the partial solution graph built so far. For each operation u in SC, all operations in $P(u)$ or $PO(u)$ are scheduled as well. So, there is a path in G_{SC} from the node O_i^{start} to u through all operations in $PO(u)$ and a path from *start* to u through all operations in $P(u)$. Let A be the set that includes the first unscheduled operation of each job that has at least one unscheduled operation, i.e.

$$A = \{u \notin SC; \nexists PJ_u \vee PJ_u \in SC\} \tag{3}$$

For each operation u in A, r_u is the starting time of u if u is selected to be scheduled next. In accordance with expression (1), r_u is greater or at least equal than both the completion time of PJ_u and the completion time of the last operation scheduled on the machine R_u. Moreover, the value of r_u also depends on the availability of operators at this time, hence r_v is not lower than the earliest time an operator is available. Let t_i, $1 \leq i \leq p$, be the time at which the operator i is available, then

$$r_u = \max\left\{ r_{PJ_u} + p_{PJ_u}, r_v + p_v, \min_{1 \leq i \leq p} t_i \right\} \tag{4}$$

where v denotes the last operation scheduled having $R_v = R_u$. In general, a number of operations in A could be scheduled simultaneously at their current heads, however it is clear that not all of them could start processing at these times due to both capacity constraints and operators availability. So, a straightforward schedule generation scheme is obtained if each one of the operations in A is considered as candidate to be scheduled next.

If the selected operation is u, it is scheduled at the time $st_u = r_u$ and all the disjunctive arcs of the form (u, v), for all $v \notin P(u)$, get fixed. Operator arcs should also be fixed from the set of scheduled operations assisted by any of the operators. In principle, any operator i with $t_i \leq r_u$ can be selected for the operation u. So, if we start from the set A containing the first operation of each job and iterate until all the operations get scheduled, no matter what operation is selected in each step, we will finally have a feasible schedule, and there is a sequence of choices eventually leading to an optimal schedule.

Let us now consider the operation v^* with the earliest completion time if it is selected from the set A, i.e.

$$v^* = \arg\min\{r_u + p_u; u \in A\} \tag{5}$$

and the set A' given by the operations in A that can start before the earliest completion of v^*, i.e.

$$A' = \{u \in A; r_u < r_{v^*} + p_{v^*}\} \tag{6}$$

If we restrict the selection, in each step of the previous scheduling algorithm, to an operation in A', at least one of the schedules that may be eventually reached is optimal. The reason for this is that for every operation u in A not considered in A', the possibility of being scheduled at a time $st_u = r_u^A$ remains in the next step.

Moreover, if the number of operators available is large enough, it is not necessary to take all the operations in the set A' as candidate selections. Let $[T, C)$ be a time interval, where $C = r_{v^*} + p_{v^*}$ and $T = min\{r_u; u \in A'\}$, and the set of machines $R_{A'} = \{R_u; u \in A'\}$. If we consider the simplified situation where $r_u = T$, for all $u \in A'$ we can do the following reasoning: if, for instance, the number of machines in $R_{A'}$ is two and there are two or more operators available along $[T, C)$, then the set A' can be reduced to the operations requiring the machine R_{v^*}. In other words, we can do the same as it is done in the *G&T* algorithm for the classical job-shop problem. The reason for this is that after selecting an operation v requiring R_{v^*} to be scheduled, every operation $u \in A'$ requiring the other machine can still be scheduled at the same starting time as if it were scheduled before v, so as this machine may not be considered in the current step. However, if there is only one operator available along $[T, C)$ then A' may not be reduced, otherwise the operations removed from A' will no longer have the possibility of being processed at their current heads.

The reasoning above can be extended to the case where p' operators are available along $[T, C)$ and the number of machines in $R_{A'}$ is $m' \geq p'$. In this case A' can be reduced to maintain the operations of only $m' - p' + 1$ machines in order to guarantee that all the operations in A' have the opportunity to get scheduled at their heads in G_{SC}.

In general, the situation is more complex as the heads of operations in A' are distributed along the interval $[T, C)$ as well as the times at which the operators get available. Let $\tau_0 < \ldots < \tau_{p'}$ be the times given by the head of some operation in A' or the time at which some operator becomes available along the interval $[T, C)$. It is clear that $\tau_0 = T$ and $\tau_{p'} < C$. Let $NO_{\tau_i}, i \leq p'$, be the number of operators available in the interval $[\tau_i, \tau_{i+1})$, with $\tau_{p'+1} = C, R_{\tau_i}$ the set of machines that are required before τ_{i+1}, i.e $R_{\tau_i} = \{R_u; r_u \leq \tau_i\}$ and NR_{τ_i} be the number of machines in R_{τ_i}.

We now consider the time intervals from $[\tau_{p'}, C)$ backwards to $[T, \tau_1)$. If $NO_{\tau_i} \geq NR_{\tau_i}$ then only the operations of just one of the machines in R_{τ_i} should be maintained in A' due to the interval $[\tau_i, \tau_{i+1})$. Otherwise, i.e. if $NO_{\tau_i} < NR_{\tau_i}$, then the operations from at least $NR_{\tau_i} - NO_{\tau_i} + 1$ machines must be maintained from A' in order to guarantee that any operation requiring a machine in R_{τ_i} could be eventually processed along the interval $[\tau_i, \tau_{i+1})$. In other words, in this way we guarantee that any combination of NO_{τ_i} operations requiring different machines can be processed along the interval $[\tau_i, \tau_{i+1})$, as at least one operation requiring each of the $NR_{\tau_i} - NO_{\tau_i} + 1$ machines will appear in such combination of operations.

Here, it is important the following remark. As it was pointed, in principle any operator available at time r_u is suitable for u. However, we now have to be aware of leaving available operators for the operations that are removed from A'. In order to do that we select an operator i available at the latest time t_i such that $t_i \leq r_u$. In this way, we maximize the availability of operators for the operations to be scheduled in the next steps. To be more precise, we guarantee that any operation removed from A' can be scheduled at its current head in a subsequent step.

From the interval $[\tau_{p'}, C)$ at least the operation v^* is maintained and then some new operations are added from the remaining intervals. The set of operations obtained in this way is termed B and it is clear that $|B| \leq |A'| \leq |A|$. An important property of this schedule generation scheme is that if the number of operators is large enough, in particular if $p \geq \min(n, m)$ so as $JSO(n, p)$ becomes $J||C_{max}$, it is equivalent to the *G&T* algorithm. So, we call this new algorithm *OG&T* (Operators *G&T*). From the reasoning above the following result can be established.

Theorem 1 *Let $\mathcal{P}$ be a JSO(n, p) instance. The set $\mathcal{S}$ of schedules that can be obtained by the OG&T algorithm to $\mathcal{P}$ is dominant, i.e. $\mathcal{S}$ contains at least one optimal schedule.*

3.1 Narrowing of the search space

In its original formulation, the *G&T* algorithm was shown to be able to generate all the possible active schedules for the classic job-shop problem. In these schedules, at least one operation has to be delayed in order to start the processing of another operation earlier, i.e., there are no machines idle along a whole period where an operation could be completely processed. Active schedules constitute a subset of the feasible schedules that contains at least one optimal solution.

The concept of active schedule has not yet been formalized for the $JSO(n, p)$ problem, and it is not trivial due to the limited number of assisting operators. Nevertheless, the reduction made by the *OG&T* algorithm from the set A to A', taking only the operations that can start before the earliest possible completion time of v^*, guarantees that no operation can be processed earlier in a schedule without delaying at least other operation.

Although the set of schedules defined by the *OG&T* algorithm is smaller than the set of the feasible schedules, it may be still huge for large instances. For this reason, we propose extending this algorithm in order to achieve further reductions, yet loosing the possibility of finding an optimal solution. Concretely, we reduce B to the B' defined as

$$B' = \{u \in B; r_u \leq \min\{r_w; w \in B\} + \delta \times (r_{v^*} + p_{v^*} - \min\{r_w; w \in B\})\} \quad (7)$$

where δ is a real parameter such that $0 \leq \delta \leq 1$.

Hence, the parameter δ establishes the upper limit for the starting time of the operation to be scheduled next, and so it controls the maximum time that a machine can be idle when an operation is ready to be processed on that machine and an operator is available to assist that processing. It is clear that the smaller value of δ the lower idle time periods of the machines. So, it can be expected that the average makespan is lower for the schedules that remain in the search space after this reduction than it is for the whole set of schedules in the original search space. At the same time, it is clear that for $\delta < 1$ we can no longer guarantee that at least one of the chromosomes represent an optimal schedule. From all of this, it is our hypothesis that for large instances where the search space is huge it may be worthwhile to restrict the search to a reduced subspace with good solutions in average by setting δ to a small value, even though this subspace may not contain any optimal schedule. However, for small instances it may be better to set $\delta = 1$ so as the genetic algorithm can search over the whole space.

4 Genetic algorithm for the JSO(*n*,*p*)

The GA used here is inspired in that proposed in (González et al. 2008). The coding schema is based on permutations with repetition as it was proposed in (Bierwirth 1995). A chromosome is a permutation of the set of operations that represents a tentative ordering to schedule them, each one being represented by its job number. For example, the sequence (2 1 1 3 2 3 1 2 3) is a valid chromosome for a problem with 3 jobs and 3 machines. As it was shown in (Varela et al. 2005), this encoding has a number of interesting properties for the classic job-shop scheduling problem; for example, it tends to represent orders of operations as they appear in good solutions. So, it is expected that these characteristics are to be good for the $JSO(n, p)$ as well.

For chromosome mating, the GA uses the Job-based Order Crossover (JOX) described in (Bierwirth 1995). Given two parents, JOX selects a random subset of jobs and copies their genes to the offspring in the same positions as they are in the first parent, then the remaining genes are taken from the second parent so as they maintain their relative ordering. We clarify how JOX works in the next example. Let us consider the following two parents

Parent1 (**2** 1 1 3 **2** 3 1 **2** 3) Parent2 (3 3 1 2 1 3 2 2 1)

If the selected subset of jobs from the first parent just includes the job 2, the generated offspring is

Offspring (**2** 3 3 1 **2** 1 3 **2** 1).

Hence, operator JOX maintains for each machine a subsequence of operations in the same order as they are in Parent1 and the remaining in the same order as they are in Parent2.

To evaluate chromosomes, the GA uses the algorithm *OG&T* described in the previous section, so that the non-deterministic choice is done by looking at the chromosome: the operation in B which is in the leftmost position in the chromosome sequence is selected to be scheduled next.

The remaining elements of GA are rather conventional. To create a new generation, all chromosomes from the current one are organized into couples which are mated and then mutated to produce two offsprings in accordance with the crossover and mutation probabilities respectively. Finally, tournament replacement among every couple of parents and their offsprings is done to obtain the next generation.

4.1 Evolution models

Regarding the evolution model of a genetic algorithm, we can consider either Baldwinian or Lamarckian evolution. The genetic algorithm described above follows the Baldwinian model, as no information but the fitness from the phenotypes is translated back to the genotypes after the decoding algorithm is applied to the chromosomes. According to this evolution model, no characteristics learned during the life time of an individual are passed down to its offspring, as it happens in natural evolution. In this model, a trait would remain in the population as a result of selection pressure focusing towards individuals more capable of acquiring the trait.

On the contrary, in Lamarckian evolution the changes of an individual during its life are inherited by its offspring. This would happen if the characteristics acquired by the individual were coded back into the chromosome structure. It is well known that this effect does not occur in natural evolution, but it is usually implemented in artificial evolution as it allows the characteristics of the parents to be immediately transmitted to the offspring. So, a genetic algorithm exploiting this model would exhibit more selection pressure, so that the search may be biased towards better regions of the solution space. As a drawback, Lamarckian evolution may have negative effects on the diversity of the genetic material in the population, and thus causing premature convergence.

We propose a Lamarckian version of our genetic algorithm, in which the characteristics of the solutions are coded back into the chromosomes. To do so, after a schedule is built by the *OG&T* algorithm, the original chromosome is replaced in the population by a new one in which the operations appear in a compatible order with their starting times in the schedule, i.e., for any two operations u and v requiring the same machine, if $st_u < st_v$ then the gene representing u is before than that representing v in the chromosome sequence. This way, the new chromosome will hold all the information about the solution it represents, and not only the fitness value. Clearly, the differences between the new and the old chromosomes are not likely to be large, or at least they are expected to be much lower than the differences between them if a conventional local search were applied from the original chromosome. For this reason, as we have pointed out, we consider this as a weak Lamarckian evolution model. In the next section, both evolution models are empirically evaluated.

5 Experimental study

We have conducted a thorough experimental study aimed at evaluating our proposals and making a comparison with the state-of-the-art methods which, to the best of our knowledge, are the dynamic programming and the heuristic algorithms given in (Agnetis et al. 2011), and the A* search algorithm proposed in (Sierra et al. 2013).

For this purpose, we have experimented with both the Baldwinian (GA) and the Lamarckian (LGA) versions of the genetic algorithm, and have considered five different values of δ (0, 0.25, 0.5, 0.75, 1) for each of them. Among the 10 possible configurations, we will consider GA and $\delta = 1$ as a base method, as it does not present any difference with the original genetic algorithm proposed in (Mencía et al. 2011).

The experiments were carried out across two sets of instances. The first one was proposed in (Agnetis et al. 2011); all these instances have $n = 3$ and $p = 2$ and are characterized by the number of machines (m), the number of operations per job (v_{max}) and the range of processing times (p_i). A set of small instances was generated combining the values $m = 3, 5, 7$; $v_{max} = 5, 7, 10$ and $p_i = [1, 10], [1, 50], [1, 100]$ and a set of larger instances was generated with $m = 3$, combining $v_{max} = 20, 25, 30$ and $p_i = [1, 50], [1, 100], [1, 200]$; 10 instances were considered from each combination. The sets of small instances are identified by numbers from 1 to 27: set 1 corresponds to triplet $3 - 5 - 10$, the second is $3 - 5 - 50$ and so on. The large instances are identified analogously by labels from $L1$ to $L9$. There are 360 instances in all.

The second benchmark set is derived from a number of instances taken from the *OR*-library (Beasley 1990). Firstly, small instances with $m = 5$ and different number of jobs: $LA01 - 05$ with $n = 10$, $LA06 - 10$ with $n = 15$ and $LA11 - 15$ and $FT20$ with $n = 20$. Then larger instances with $m = 10$ and different number of jobs: $FT10$, $LA16 - 20$, $ABZ5, 6$ and $ORB01 - 10$ with $n = 10$, $LA21 - 25$ with $n = 15$, $LA26 - 30$ with $n = 20$ and $LA31 - 35$ with $n = 30$. Finally, the instances $LA36 - 40$ with $n = m = 15$. For each instance, all values of p in $[1, \min(n, m)]$ are considered. So, this set contains 480 instances in all.

In all the experiments, the genetic algorithms were parameterized with a population of 100 chromosomes, a number of 140 generations, crossover probability of 0.7 and mutation probability of 0.2, so that about 10,000 chromosomes are evaluated in each run. Each algorithm was run 30 times for each instance. The algorithms were coded in C++ and the target machine was Intel Xeon 2.26 GHz. 24 GB RAM.

In the remainder of this section, we first provide a comparison between the base GA and the state-of-the-art methods. Then, we analyse the effects that reducing the search space and incorporating a Lamarkian evolution scheme have on the genetic algorithm.

5.1 Comparison between GA and the state-of-the-art methods

In the first part of the experimental study, we have compared the original genetic algorithm (GA and $\delta = 1$) with the methods proposed in (Agnetis et al. 2011) and (Sierra et al. 2013).

For all the experiments, we report the mean relative error in percentage terms of the best and average solutions (*Ebest* and *Eavg* respectively) reached by GA, w.r.t. the best lower bounds given by the A^* algorithm proposed in (Sierra et al. 2013). This algorithm was able to solve optimally all the instances taken from (Agnetis et al. 2011) and a number of the instances from the second set. For the remaining ones it has given a suboptimal solution (as it combines best-first search with a greedy algorithm that is issued from a number of expanded states to obtain upper bounds) and a lower bound in a time limit of 3,600 s, or after the memory of the target machine got exhausted. We also report the time taken in average ($T(s)$) by GA and the Pearson coefficient of variation in percentage terms of the solutions (CV) computed as the ratio of standard deviation and the average of the solutions across the 30 runs.

Table 1 reports the results obtained by GA for the first set, together with the results from the best exact and approximate algorithms given in (Agnetis et al. 2011), namely the dynamic programming algorithm (DP) and the heuristic algorithms (*Heur* and *Heur+*). DP was able to solve optimally all but a number of instances from sets L4-9. In these cases, DP was stopped after 3 h, and the incumbent solution is taken as reference for calculating the gap of the heuristic algorithms. In the other cases the gap is defined as the error in percentage w.r.t. the optimal solution, averaged for all instances. For the sets L4-9, the instances that were not solved by DP are not considered in the average error computed for the methods proposed in (Agnetis et al. 2011). For GA, we report *Ebest*, *Eavg*, CV and $T(s)$. In all cases the results are averaged for subsets of instances with the same number of operations per job v_{max}.

The first conclusion we can draw from these experiments is that GA clearly outperforms the heuristic methods *Heur* and *Heur+* ; not only the error is much lower for GA than it is for both heuristic methods, but also the time taken is much lower for GA, in particular for the largest instances. A comparison between GA and DP is not so straightforward. As we can see, GA is not able to solve optimally all these instances in the time given whereas DP solves small instances easily. At the same time, GA is able to produce high-quality solutions in very short time regardless of the size of the instances, while DP performs

Table 1 Summary of results from instances with 3 jobs and 2 operators

		GA				T.(s)			Gap	
Instances		*Ebest*	*Eavg*	*CV*	T.(s)	*DP*	*Heur*	*Heur+*	*Heur*	*Heur+*
Small	1–9	4.50	4.51	0.03	0.18	0.03	0.01	0.04	15.33	11.22
	10–18	1.55	1.88	0.12	0.18	1.29	0.04	0.47	14.81	10.98
	19–27	1.07	1.28	0.23	0.26	14.93	0.47	4.86	15.31	11.12
Large	L1-L3	1.43	2.37	0.50	0.50	654.10	9.40	91.47	17.80	12.90
	L4-L6	1.82	2.97	0.58	0.64	1683.60	35.77	366.73	15.40	11.97
	L7-L9	2.53	3.96	0.68	0.73	4351.00	97.00	823.97	16.50	11.57

Table 2 Summary of results from instances with 5 machines and 10, 15 and 20 jobs

	10 *jobs*			15 *jobs*			20*jobs*		
	(T=0.4s, CV=0.2)			(T=0.8s, CV=0.1)			(T=1.1s, CV=0.1)		
#Op.	*Ebest*	*Eavg*	EA^*	*Ebest*	*Eavg*	EA^*	*Ebest*	*Eavg*	EA^*
1	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
2	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
3	0.00	0.18	0.00	0.00	0.07	0.00	0.01	0.12	0.05
4	1.20	2.20	0.24	0.20	0.71	0.00	0.98	1.49	1.27
5	0.30	1.27	0.00	0.00	0.00	0.00	0.70	1.14	0.00
Avg.	0.30	0.73	0.05	0.04	0.16	0.00	0.34	0.55	0.26

poorly on large instances, either taking a long time to solve them or not even being able to come up with a single solution by a time limit of 3 h. So, DP is well suited for solving small instances and GA may be a better option for large ones. It is also remarkable that GA seems to be very stable, as the value of *CV* is very low in all cases and increases very slightly with the size of the problem.

The second set of instances is more interesting than the first one, as it allows us to analyze the behavior of the algorithms in a broader range of settings regarding the numbers of machines, jobs and available operators. Here, we will only consider GA and A*, as there are no results available in the literature from Agnetis et al's methods. Besides, these methods were shown to be clearly outperformed by the A* algorithm in (Sierra et al. 2013). Tables 2, 3, and 4 are devoted to this set, and share a similar structure: they show the errors in percentage obtained by GA (*Ebest* and *Eavg*) and by A* (*EA**) averaged for subsets of instances with the same size and number of operators. They also report, for GA, the time taken (*T*(*s*)) and *CV* averaged for subsets of instances with the same size.

One important remark about these experiments is that A* was given more time and space than GA, so a strict comparison between the two methods is not possible. Nevertheless, these results provide an insight into the effectiveness that can be achieved by GA with respect to the best known solutions to the problem.

Table 2 reports the results across instances with 5 machines and 10, 15 and 20 jobs. These are small instances and so they were almost all solved to optimality by the A* algorithm. As we can see, GA is able to reach very good solutions, finding an optimum in many cases. Even for some of the largest instances with 20 jobs that A* did not solve optimally, GA reached better solutions in less than one second.

Table 3 shows the results from larger instances with 10 machines and 10, 15, 20 and 30 jobs, which are really hard to solve. First of all, we can note that both GA and A* solved to optimality all the instances with 1 or 2 operators, regardless of the number of jobs. In the remaining cases, there seems to be a trend that GA performs better than A* as long as the size of the instances grows. A* reaches better

Table 4 Summary of results from instances with 15 jobs and 15 machines

	GA		
	(T=4.3s, CV=0.4)		
#Op.	*Ebest*	*Eavg*	*EA**
1	0.00	0.00	0.00
2	0.00	0.00	0.00
3	0.00	0.03	0.00
4	0.07	0.17	0.00
5	0.25	0.45	1.59
6	0.67	1.01	2.11
7	1.42	2.09	3.11
8	3.34	4.58	4.89
9	7.80	9.66	8.46
10	10.02	13.17	13.02
11	10.20	12.94	10.63
12	9.09	12.65	9.76
13	9.96	12.62	9.75
14	9.98	12.77	9.68
15	9.61	12.64	9.64
Avg.	4.83	6.32	5.51

Table 3 Summary of results from instances with 10 machines and 10, 15, 20 and 30 jobs

	10 *jobs*			15 *jobs*			20 *jobs*			30 *jobs*		
	(T=1.1s, CV=0.4)			(T=2.4s, CV=0.3)			(T=3.8s, CV=0.3)			(T=7.1s, CV=0.1)		
#Op.	*Ebest*	*Eavg*	*EA**	*Ebest*	*Eavg*	*EA**	*Ebest*	*Eavg*	*EA**	*Ebest*	*Eavg*	*EA**
1	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
2	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
3	0.03	0.16	0.21	0.00	0.04	0.00	0.00	0.02	0.00	0.00	0.01	0.00
4	0.69	1.24	0.65	0.07	0.24	0.00	0.04	0.14	0.26	0.01	0.06	0.32
5	3.01	4.20	3.26	0.36	0.73	1.41	0.21	0.43	1.63	0.10	0.20	0.80
6	6.69	8.92	6.40	1.17	1.89	5.71	0.60	1.01	3.45	0.26	0.45	3.25
7	5.17	7.46	4.14	3.30	4.76	3.12	1.71	2.57	2.92	0.67	1.01	1.96
8	2.54	4.94	1.05	7.70	9.95	7.58	4.82	6.52	4.83	1.76	2.55	2.94
9	2.52	4.80	0.80	7.21	10.19	9.04	8.14	10.40	9.45	1.91	3.43	4.48
10	2.67	4.89	0.89	6.74	9.48	6.91	7.37	9.62	7.44	0.49	1.46	2.48
Avg.	2.33	3.66	1.74	2.66	3.73	3.38	2.29	3.07	3.00	0.52	0.92	1.62

solutions than GA for most of the smallest instances with 10 jobs. For instances with 15 and 20 jobs, the best solutions reached by GA are better in average than those obtained by A*. Moreover, regarding the average makespan, GA outperforms A* in most of the cases having an intermediate numbers of available operators: instances with 5, 6 and 7 available operators and 15 jobs and instances with 4, 5, 6 and 7 operators and 20 jobs. For the largest and hardest instances in the whole experimental study, with 30 jobs and 10 machines, GA outperforms A* in all cases, reducing the average error by about 43.21 % in average and 67.90 % in the best case.

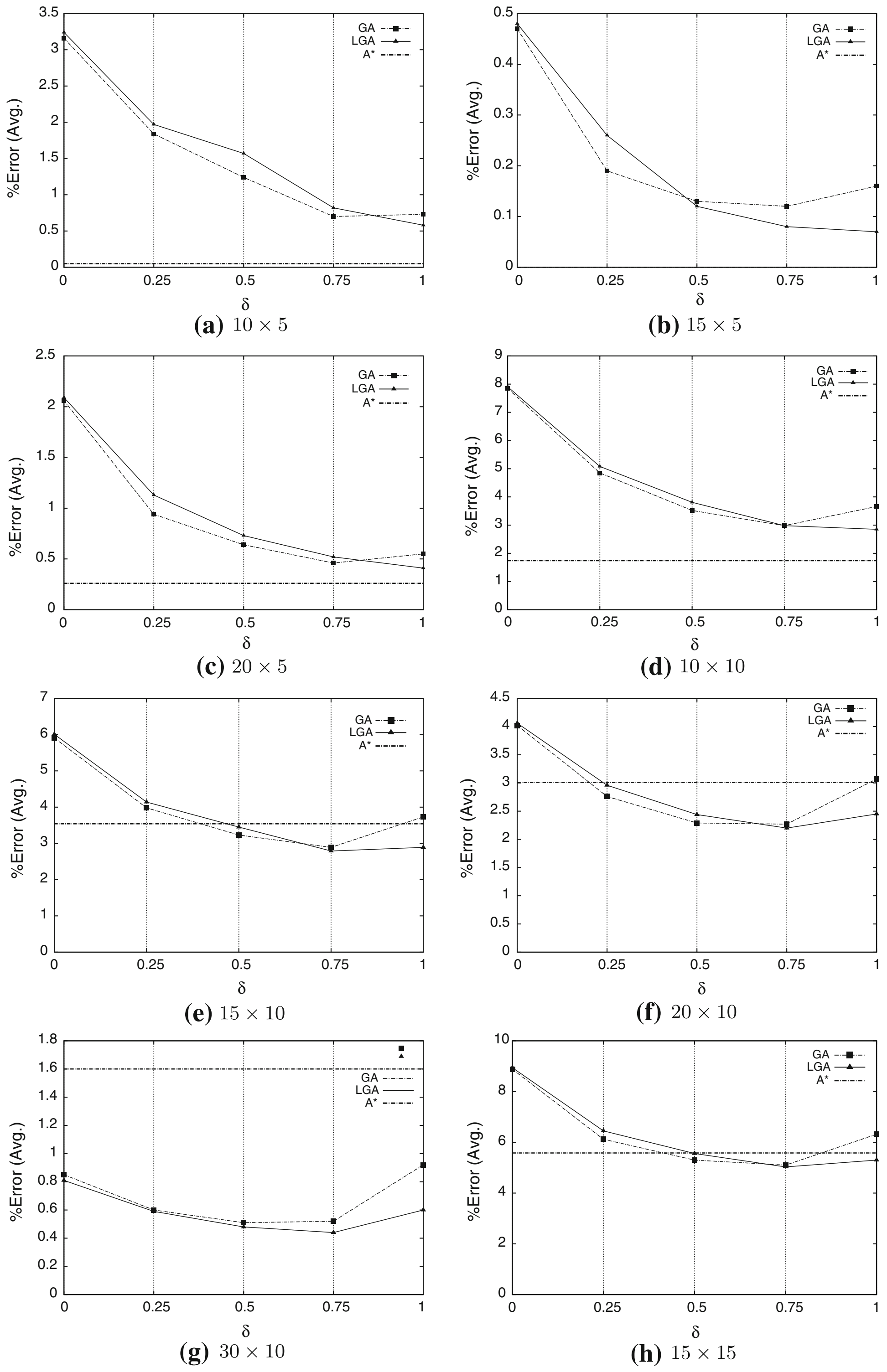


Fig. 2 Errors in percentage from GA, LGA and A* across the second set averaged for instances with the same size and number of operators

Finally, Table 4 shows results from instances with 15 jobs and 15 machines. The results are fairly similar to those from instances with 20 jobs and 10 machines. Overall, the best solutions computed by GA are better than those returned by A*. Also, GA outperforms A* in average solving instances with an intermediate number of operators (5, 6, 7 and 8). In the remaining cases, A* reaches the best solutions.

We can also observe that GA is able to solve all the instances very quickly. Indeed, it only takes an average time of 7.1 seconds to solve the largest ones. GA is also very stable, as shown by the very low values of *CV* in all cases.

5.2 Evaluation of the search space narrowing and the Lamarckian evolution

This part of the experimental study is intended to evaluate the two new elements used in the genetic algorithm with the aim of improving its effectiveness: the strategy for narrowing the search space and the Lamarckian evolution model. It has been carried out over the second set of instances presented at the beginning of the section.

Figure 2 shows the average error in percentage terms obtained by the different versions of the genetic algorithms and the A* algorithm, w.r.t. the best lower bounds computed by A*. The genetic algorithms were run 30 times for each instance and value of δ. GA and LGA denote the Baldwinian and the Lamarckian genetic algorithms respectively. In the experiments, we have considered five values for the parameter δ (0, 0.25, 0.5, 0.75, 1); which are represented in the x-axis of the graphs. The y-axis represent the errors of the algorithms, which are averaged for all the instances with the same number of jobs and machines.

Let us firstly focus on the reduction of the search space. Looking at GA in Figure 2, we can observe the expected behavior. Overall, the reductions achieved with $\delta \in \{0.5, 0.75\}$ seem to be the best options to reach a tradeoff between the size of the search space and the quality of the solutions that can be computed by GA. Note that the average solutions obtained by these versions of GA are better than those obtained by A* in all the subsets with the largest instances (Figure 2d, e, f, g) whereas the original GA ($\delta = 1$) is only able to do so for one of these subsets. At the same time, reducing the space to a great extent with $\delta \in \{0, 0.25\}$ often turns out in poor results. For the smallest instances (Figure 2a, b and c), using $\delta = 1$ is usually better than $\delta = 0.5$, and a small reduction of $\delta = 0.75$ is always the best option. This is quite reasonable as the search spaces for these instances are not too large. In the other cases, a value of $\delta = 0.5$ produces either fairly similar results as $\delta = 0.75$ or it is even better, as it happens for the largest instances with 30 jobs and 10 machines (Figure 2g). It is also clear that as the size of the instances grows, not reducing the search space at all ($\delta = 1$) yields worse results compared to those obtained by other options, being even outperformed by $\delta = 0$ (the greatest possible reduction) on the largest instances (Figure 2g).

Regarding the Lamarckian version of the genetic algorithm (LGA), we can observe that the results showed in Figure 2 do not contradict the hypotheses. Comparing both evolution models over the original search space ($\delta = 1$) LGA clearly outperforms GA in all cases, regardless of the size of the instances. In this case, LGA looks for solutions in an unconstrained and (often) large search space, so increasing the selection pressure is worthwhile. Note that the Lamarckian model itself allows LGA to outperform A* in all the subsets of largest instances (cases e, f, g, h). Combining LGA with the strategy for reducing the search space does not always produce good results, as it was expected. In the first four subsets with the smallest instances (cases a, b, c and d), LGA reaches better solutions with $\delta = 1$ than with $\delta < 1$; getting especially worse with $\delta < 0.75$. As the search space for these instances is not too large, LGA is able to reach a near-optimal solution. On the other hand, for instances with very large solution spaces, it seems that LGA is not able to explore the whole search space and so the best option is to perform a small reduction using $\delta = 0.75$. This combination reaches a good tradeoff and yields the best results in the whole experimental study for the cases e, f, g and h. It also seems that for values of $\delta < 0.75$, GA usually performs better than LGA, with the exception of the largest instances with 30 jobs and 10 machines (case g). This might be due to a premature convergence of LGA.

One important note here is that we have not observed significant differences regarding the time taken and the variation coefficient among the different versions of the genetic algorithm.

Table 5 Average ranking of the algorithms for instances with up to 100 operations

Algorithm	Ranking
LGA($\delta = 0.75$)	3.44
LGA($\delta = 1$)	3.48
GA($\delta = 0.75$)	4.44
LGA($\delta = 0.5$)	5.04
GA($\delta = 0.5$)	5.37
GA($\delta = 1$)	5.99
GA($\delta = 0.25$)	6.15
LGA($\delta = 0.25$)	6.35
GA($\delta = 0$)	7.31
LGA($\delta = 0$)	7.43

Table 6 Adjusted p-values. Instances with up to 100 operations

i	Hypothesis	Unadjusted p	p_{Holm}	p_{Shaf}
1	LGA(δ = 0) vs .LGA(δ = 0.75)	4.36E−51	1.96E−49	1.96E−49
2	LGA(δ = 0) vs .LGA(δ = 1)	4.29E−50	1.89E−48	1.54E−48
3	GA(δ = 0) vs .LGA(δ = 0.75)	3.86E−48	1.66E−46	1.39E−46
4	GA(δ = 0) vs .LGA(δ = 1)	3.54E−47	1.49E−45	1.27E−45
5	GA(δ = 0.75) vs .LGA(δ = 0)	1.46E−29	6.00E−28	5.27E−28
6	LGA(δ = 0.25) vs .LGA(δ = 0.75)	7.80E−28	3.12E−26	2.81E−26
7	GA(δ = 0) vs .GA(δ = 0.75)	2.37E−27	9.25E−26	8.54E−26
8	LGA(δ = 0.25) vs .LGA(δ = 1)	4.12E−27	1.57E−25	1.48E−25
9	GA(δ = 0.25) vs .LGA(δ = 0.75)	1.90E−24	7.03E−23	6.84E−23
10	GA(δ = 0.25) vs .LGA(δ = 1)	9.00E−24	3.24E−22	3.24E−22
11	GA(δ = 1) vs .LGA(δ = 0.75)	9.59E−22	3.36E−20	2.78E−20
12	GA(δ = 1) vs .LGA(δ = 1)	4.13E−21	1.41E−19	1.20E−19
13	LGA(δ = 0) vs .LGA(δ = 0.5)	1.94E−19	6.42E−18	5.64E−18
14	GA(δ = 0) vs .LGA(δ = 0.5)	1.13E−17	3.61E−16	3.27E−16
15	GA(δ = 0.5) vs .LGA(δ = 0)	6.57E−15	2.04E−13	1.91E−13
16	GA(δ = 0) vs .GA(δ = 0.5)	2.20E−13	6.59E−12	6.37E−12
17	GA(δ = 0.5) vs .LGA(δ = 0.75)	4.42E−13	1.28E−11	1.28E−11
18	GA(δ = 0.75) vs .LGA(δ = 0.25)	6.41E−13	1.80E−11	1.80E−11
19	GA(δ = 0.5) vs .LGA(δ = 1)	1.34E−12	3.62E−11	3.22E−11
20	GA(δ = 0.25) vs .GA(δ = 0.75)	1.05E−10	2.72E−09	2.51E−09
21	LGA(δ = 0.5) vs .LGA(δ = 0.75)	1.76E−09	4.41E−08	4.23E−08
22	LGA(δ = 0.5) vs .LGA(δ = 1)	4.46E−09	1.07E−07	1.07E−07
23	GA(δ = 0.75) vs .GA(δ = 1)	5.31E−09	1.22E−−07	1.17E−07
24	GA(δ = 1) vs .LGA(δ = 0)	4.94E−08	1.09E−06	1.09E−06
25	GA(δ = 0) vs .GA(δ = 1)	5.82E−07	1.22E−05	1.22E−05
26	LGA(δ = 0.25) vs .LGA(δ = 0.5)	8.77E−07	1.75E−05	1.75E−05
27	GA(δ = 0.25) vs .LGA(δ = 0)	1.36E−06	2.59E−05	2.45E−05
28	GA(δ = 0) vs .GA(δ = 0.25)	1.22E−05	2.19E−04	2.19E−04
29	GA(δ = 0.25) vs .LGA(δ = 0.5)	2.84E−05	4.83E−04	4.83E−04
30	LGA(δ = 0) vs .LGA(δ = 0.25)	4.15E−05	6.64E−04	6.64E−04
31	GA(δ = 0.75) vs .LGA(δ = 0.75)	1.81E−04	2.72E−03	2.72E−03
32	GA(δ = 0.5) vs .LGA(δ = 0.25)	2.21E−04	3.10E−03	3.10E−03
33	GA(δ = 0) vs .LGA(δ = 0.25)	2.70E−04	3.51E−03	3.51E−03
34	GA(δ = 0.75) vs .LGA(δ = 1)	3.28E−04	3.94E−03	3.94E−03
35	GA(δ = 1) vs .LGA(δ = 0.5)	3.67E−04	4.03E−03	4.03E−03
36	GA(δ = 0.5) vs .GA(δ = 0.75)	4.69E−04	4.69E−03	4.69E−03
37	GA(δ = 0.25) vs .GA(δ = 0.5)	3.06E−03	2.75E−02	2.75E−02
38	GA(δ = 0.5) vs .GA(δ = 1)	1.93E−02	1.55E−01	1.55E−01
39	GA(δ = 0.75) vs .LGA(δ = 0.5)	2.30E−02	1.61E−01	1.61E−01
40	GA(δ = 1) vs .LGA(δ = 0.25)	1.76E−01	1.05E+00	1.05E+00
41	GA(δ = 0.5) vs .LGA(δ = 0.5)	2.21E−01	1.10E+00	1.10E+00
42	GA(δ = 0.25) vs .LGA(δ = 0.25)	4.65E−01	1.86E+00	1.86E+00
43	GA(δ = 0.25) vs .GA(δ = 1)	5.33E−01	1.86E+00	1.86E+00
44	GA(δ = 0) vs .LGA(δ = 0)	6.48E−01	1.86E+00	1.86E+00
45	LGA(δ = 0.75) vs .LGA(δ = 1)	8.79E−01	1.86E+00	1.86E+00

5.3 Statistical analysis

The results reported so far rely on descriptive statistics, so they are only suitable for comparing the algorithms' performance over the actual instances considered in the experimental study. In order to infer conclusions for a broader class of problem instances we have made a series of statistical inference tests. Concretely, we have divided the second set of instances in two subsets: One containing instances with up to 100 operations (sizes 10 × 5, 15 × 5, 20 × 5 and 10 × 10) and another with larger instances (sizes 15 × 10, 20 × 10, 30 × 10 and 15 × 15). In a preliminary analysis, the Shapiro-Wilk test rejected the hypotheses of normality so, following (García et al. 2010), we have used non-parametric statistical techniques[1]. Concretely, for each subset we made Friedman and Iman-Davenport tests, so establishing a ranking among the algorithms and deducing whether there are significant differences among the algorithms or not. Then, as recommended in (Trawiński et al. 2012) for multiple comparisons of regression algorithms, we carried out Holm's and Shaffer's post-hoc procedures for multiple comparisons so as to establish solid pairwise comparisons between the algorithms, and determine the soundness of the ranking.

Table 5 shows the average ranking computed by the Friedman test for the subset of instances with up to 100 operations. In this case, Friedman and Iman-Davenport tests yielded p-values of 1.72E-10 and 2.22E-16 respectively, meaning there are significant differences among some of the algorithms. Table 6 reports, for each hypothesis comparing any two methods, the unadjusted *p*-value, and the adjusted *p*-values given by Holm's (*pHolm*) and Shaffer's (*pShaf*) post-hoc procedures. Roughly speaking, the hypothesis *method1 versus method2* states that there are no significant differences between the two methods. If *pHolm* <0.05 (resp. *pShaf* < 0.05) allows us to reject that hypothesis via Holm's (resp. Shaffer's) procedure, concluding that the method that appears first in the ranking (*method1* or *method2*) is more effective than the other one. As we can observe, 37 out of the 45 hypotheses (i = 1..37) are rejected by both Holm's and Shaffer's procedures. It is remarkable that LGA(δ = 0.75) and LGA (δ = 1) outperform all the other algorithms (although nothing can be said about a comparison between both). On the other hand, GA(δ = 0) and LGA(δ = 0) are outperformed by any other configuration of the genetic algorithm. Regarding the original GA (δ = 1), the only conclusion we can draw is that it outperforms GA(δ = 0) and LGA(δ = 0).

The average ranking given by the Friedman test for the subset of largest instances is reported in Table 7. For these instances, Friedman and Iman-Davenport tests returned *p*-values of 1.27E-10 and 3.33E-16 respectively, so there are differences among some algorithms. We computed Holm's and Shaffer's post-hoc procedures as well, whose adjusted p-values are shown in Table 8. Both procedures allow us to reject the same 28 hypotheses (i = 1..28). It is clear that LGA(δ = 0.75) is the best algorithm, as outperforms all the other methods. This algorithm is followed by LGA (δ = 1), that performs better than every other method under it in the ranking with the exception of LGA(δ = 0.5) and LGA(δ = 0.75). In these two cases there are not significant statistical differences among them. Finally, it is quite remarkable that LGA(δ = 0), GA(δ = 0) and the original GA(δ = 1) are worse than all the other genetic algorithms. This last result regarding the base genetic algorithm strongly supports the idea that the elements introduced in GA can improve its effectiveness to a great extent.

Table 7 Average rankings of the algorithms for large instances

Algorithm	Ranking
LGA(δ = 0.75)	3.42
LGA(δ = 1)	4.44
GA(δ = 0.75)	4.97
LGA(δ = 0.5)	5.22
GA(δ = 0.5)	5.46
GA(δ = 0.25)	5.60
LGA(δ = 0.25)	6.02
GA(δ = 0)	6.53
GA(δ = 1)	6.55
LGA(δ = 0)	6.79

Summing up, the results have shown that a genetic algorithm can take profit from the proposed improvements, but we have to be aware of the importance of reaching a good tradeoff between the selection pressure and the diversity of the population. Overall, for instances up to 100 operations the best algorithm is LGA using $\delta \in \{0.75, 1\}$ whereas for larger instances LGA using δ = 0.75 reaches the best solutions.

6 Conclusions

We have seen that the job-shop scheduling problem with operators proposed in (Agnetis et al. 2011) can be efficiently solved by means of a genetic algorithm. Two key points of this algorithm are the permutations with repetition encoding and the decoding algorithm designed from the *OG&T* schedule builder proposed in (Sierra et al. 2013). Also, the two enhancements proposed in this paper, namely the weak Lamarckian evolution and the narrowing of the search space, contribute greatly to improve the performance of the algorithm, as shown in the experimental evaluation.

As future work, we plan to combine the proposed genetic algorithm with local search and to apply other metaheuristics such as scatter search and path relinking to the same problem. To do that, we need to devise efficient

[1] To this aim, we have used the software developed by the Research Group on Soft Computing and Intelligent Information Systems at the University of Granada, which is available at http://sci2s.ugr.es/sicidm.

Table 8 Adjusted p-values. Large instances

i	Hypothesis	Unadjusted p	p_{Holm}	p_{Shaf}
1	LGA(δ = 0) vs .LGA(δ = 0.75)	3.17E-32	1.42E-30	1.42E-30
2	GA(δ = 1) vs .LGA(δ = 0.75)	4.53E-28	1.99E-26	1.63E-26
3	GA(δ = 0) vs .LGA(δ = 0.75)	1.17E-27	5.01E-26	4.20E-26
4	LGA(δ = 0.25) vs .LGA(δ = 0.75)	6.74E-20	2.83E-18	2.43E-18
5	LGA(δ = 0) vs .LGA(δ = 1)	1.77E-16	7.27E-15	6.38E-15
6	GA(δ = 0.25) vs .LGA(δ = 0.75)	2.36E-14	9.44E-13	8.50E-13
7	GA(δ = 1) vs .LGA(δ = 1)	1.33E-13	5.17E-12	4.77E-12
8	GA(δ = 0) vs .LGA(δ = 1)	2.52E-13	9.57E-12	9.07E-12
9	GA(δ = 0.5) vs .LGA(δ = 0.75)	9.41E-13	3.48E-11	3.39E-11
10	GA(δ = 0.75) vs .LGA(δ = 0)	1.91E-10	6.89E-09	6.89E-09
11	LGA(δ = 0.5) vs .LGA(δ = 0.75)	2.73E-10	9.54E-09	7.90E-09
12	LGA(δ = 0.25) vs .LGA(δ = 1)	2.85E-08	9.67E-07	8.25E-07
13	GA(δ = 0.75) vs .GA(δ = 1)	3.11E-08	1.03E-06	9.02E-07
14	LGA(δ = 0) vs .LGA(δ = 0.5)	3.71E-08	1.19E-06	1.08E-06
15	GA(δ = 0) vs .GA(δ = 0.75)	5.05E-08	1.57E-06	1.46E-06
16	GA(δ = 0.75) vs .LGA(δ = 0.75)	5.05E-08	1.57E-06	1.46E-06
17	GA(δ = 0.5) vs .LGA(δ = 0)	2.89E-06	8.37E-05	8.37E-05
18	GA(δ = 1) vs .LGA(δ = 0.5)	3.00E-06	8.39E-05	8.39E-05
19	GA(δ = 0) vs .LGA(δ = 0.5)	4.53E-06	1.22E-04	1.09E-04
20	GA(δ = 0.25) vs .LGA(δ = 0)	2.81E-05	7.31E-04	6.74E-04
21	GA(δ = 0.25) vs .LGA(δ = 1)	5.16E-05	1.29E-03	1.24E-03
22	GA(δ = 0.5) vs .GA(δ = 1)	1.20E-04	2.88E-03	2.88E-03
23	GA(δ = 0) vs .GA(δ = 0.5)	1.70E-04	3.91E-03	3.74E-03
24	GA(δ = 0.75) vs .LGA(δ = 0.25)	2.31E-04	5.09E-03	5.09E-03
25	LGA(δ = 0.75) vs .LGA(δ = 1)	3.42E-04	7.19E-03	7.19E-03
26	GA(δ = 0.5) vs .LGA(δ = 1)	3.74E-04	7.48E-03	7.48E-03
27	GA(δ = 0.25) vs .GA(δ = 1)	7.93E-04	1.51E-02	1.43E-02
28	GA(δ = 0) vs .GA(δ = 0.25)	1.08E-03	1.94E-02	1.94E-02
29	LGA(δ = 0.25) vs .LGA(δ = 0.5)	4.83E-03	8.21E-02	8.21E-02
30	LGA(δ = 0.5) vs .LGA(δ = 1)	6.28E-03	1.01E-01	1.01E-01
31	LGA(δ = 0) vs .LGA(δ = 0.25)	7.24E-03	1.09E-01	1.09E-01
32	GA(δ = 0.25) vs .GA(δ = 0.75)	2.93E-02	4.10E-01	4.10E-01
33	GA(δ = 0.5) vs .LGA(δ = 0.25)	4.63E-02	6.01E-01	6.01E-01
34	GA(δ = 0.75) vs .LGA(δ = 1)	6.17E-02	7.40E-01	7.40E-01
35	GA(δ = 1) vs .LGA(δ = 0.25)	6.39E-02	7.40E-01	7.40E-01
36	GA(δ = 0) vs .LGA(δ = 0.25)	7.72E-02	7.72E-01	7.72E-01
37	GA(δ = 0.5) vs .GA(δ = 0.75)	9.12E-02	8.20E-01	8.20E-01
38	GA(δ = 0.25) vs .LGA(δ = 0.25)	1.33E-01	1.06E+00	1.06E+00
39	GA(δ = 0.25) vs .LGA(δ = 0.5)	1.88E-01	1.32E+00	1.32E+00
40	GA(δ = 0) vs .LGA(δ = 0)	3.58E-01	2.15E+00	2.15E+00
41	GA(δ = 0.75) vs .LGA(δ = 0.5)	3.88E-01	2.15E+00	2.15E+00
42	GA(δ = 1) vs .LGA(δ = 0)	4.05E-01	2.15E+00	2.15E+00
43	GA(δ = 0.5) vs .LGA(δ = 0.5)	4.09E-01	2.15E+00	2.15E+00
44	GA(δ = 0.25) vs .GA(δ = 0.5)	6.24E-01	2.15E+00	2.15E+00
45	GA(δ = 0) vs .GA(δ = 1)	9.32E-01	2.15E+00	2.15E+00

neighborhood structures. As it was done, for example, in (González et al. 2008) for the job-shop scheduling problem with sequence-dependent setup times or in (González Rodríguez et al. 2008) for the job-shop scheduling with uncertain durations, we will look for inspiration in the structures proposed in (Van Laarhoven et al. 1992, Dell' Amico and Trubian 1993) for the classical job-shop scheduling problem.

Acknowledgments This work has been supported by the Spanish Ministry of Science and Innovation under research project MICINN-FEDER TIN2010-20976-C02-02 and by FICYT under Grant BP09105.

References

Agnetis A, Flamini M, Nicosia G, Pacifici A (2011) A job-shop problem with one additional resource type. J Sched 14(3):225–237

Artigues C, Lopez P, Ayache P (2005) Schedule generation schemes for the job shop problem with sequence-dependent setup times: Dominance properties and computational analysis. Ann Oper Res 138:21–52

Beasley JE (1990) Or-library: distributing test problems by electronic mail. J Oper Res Soc 41(11):1069–1072, http://www.jstor.org/stable/2582903

Bierwirth C (1995) A generalized permutation approach to jobshop scheduling with genetic algorithms. OR Spectrum 17:87–92

Bierwirth C, Mattfeld DC (1999) Production scheduling and rescheduling with genetic algoritms. Evol Comput 7:1–17

Brucker P, Jurisch B, Sievers B (1994) A branch and bound algorithm for the job-shop scheduling problem. Discret Appl Math 49:107–127

Dell' Amico M., Trubian M. (1993) Applying tabu search to the job-shop scheduling problem. Ann Oper Res 41:231–252

García S, Fernández A, Luengo J, Herrera F (2010) Advanced nonparametric tests for multiple comparisons in the design of experiments in computational intelligence and data mining: Experimental analysis of power. Inf Sci 180:2044–2064

Giffler B, Thompson GL (1960) Algorithms for solving production scheduling problems. Oper Res 8:487–503

González MA, Vela CR, Varela R (2008) A new hybrid genetic algorithm for the job shop scheduling problem with setup times. In: Proceedings of the Eighteenth International Conference on Automated Planning and Scheduling (ICAPS-2008). AAAI Press, Sidney

González MA, Vela CR, Varela R (2012) A competent memetic algorithm for complex scheduling. Nat Comput 11:151–160

González Rodríguez I, Vela CR, Puente J, Varela R (2008) A new local search for the job shop problem with uncertain durations. In: Proceedings of the Eighteenth International Conference on Automated Planning and Scheduling (ICAPS-2008). AAAI Press, Sidney

Mattfeld DC (1995) Evolutionary search and the job shop investigations on genetic algorithms for production scheduling. Springer, Berlin

Mencía C, Sierra MR, Varela R (2012) Depth-first heuristic search for the job shop scheduling problem. Ann Oper Res. doi:10.1007/s10479-012-1296-x

Mencía C, Sierra MR, Varela R (2013) Intensified iterative deepening A* with application to job shop scheduling. J Intell Manuf. doi:10.1007/s10845-012-0726-6

Mencía R, Sierra M, Mencía C, Varela R (2011) Genetic algorithm for job-shop scheduling with operators. Lect Notes Comput Sci 6687(2):305–314

Sierra MR, Mencía C, Varela R (2013) Searching for optimal schedules to the job-shop problem with operators. Technical report. iScOp Research Group. University of Oviedo, Oviedo

Sierra MR, Varela R (2010) Pruning by dominance in best-first search for the job shop scheduling problem with total flow time. J Intell Manuf 21(1):111–119

Trawiński B, Smetek M, Telec Z, Lasota T (2012) Nonparametric statistical analysis for multiple comparison of machine learning regression algorithms. Int J Appl Math Comput Sci 22(4):867–881

Van Laarhoven P, Aarts E, Lenstra K (1992) Job shop scheduling by simulated annealing. Oper Res 40:113–125

Varela R, Serrano D, Sierra M (2005) New codification schemas for scheduling with genetic algorithms. Proceedings of IWINAC 2005. Lect Notes Comput Sci 3562:11–20

5.2. Memetic Algorithms for the Job Shop Scheduling Problem with Operators

Se presenta la siguiente publicación:

- Raúl Mencía, María R. Sierra, Carlos Mencía, Ramiro Varela. Memetic Algorithms for the Job Shop Scheduling Problem with Operators. *Applied Soft Computing*, 2015, 34: 94-105. 2015. DOI: 10.1016/j.asoc.2015.05.004.
    - Estado: **Publicado.**

Applied Soft Computing 34 (2015) 94–105

Contents lists available at ScienceDirect

Applied Soft Computing

journal homepage: www.elsevier.com/locate/asoc

Memetic algorithms for the job shop scheduling problem with operators

Raúl Mencía[a], María R. Sierra[a], Carlos Mencía[b], Ramiro Varela[a,*]

[a] Department of Computing, University of Oviedo, Campus of Gijón, 33204 Gijón, Spain
[b] CASL, University College Dublin, Ireland

ARTICLE INFO

Article history:
Received 9 May 2014
Received in revised form 4 March 2015
Accepted 2 May 2015
Available online 12 May 2015

Keywords:
Metaheuristics
Scheduling with operators
Local search
Memetic algorithm

ABSTRACT

The job-shop scheduling problem with operators (JSO) is an extension of the classic job-shop problem in which an operation must be assisted by one of a limited set of human operators, so it models many real life situations. In this paper we tackle the JSO by means of memetic algorithms with the objective of minimizing the makespan. We define and analyze a neighborhood structure which is then exploited in local search and tabu search algorithms. These algorithms are combined with a conventional genetic algorithm to improve a fraction of the chromosomes in each generation. We also consider two different schedule builders for chromosome decoding. All these elements are combined to obtain memetic algorithms which are evaluated over an extensive set of instances. The results of the experimental study show that they reach high quality solutions in very short time, comparing favorably with the state-of-the-art methods.

1. Introduction

In this paper we propose some memetic algorithms (MAs) to solve the job-shop scheduling problem with operators (JSO). This problem was proposed in [1] and is motivated by manufacturing processes in which part of the work is done by human operators sharing the same set of tools. The problem is formalized as a classical job shop scheduling problem (JSP) in which the processing of an operation on a given machine requires the assistance of one operator. The number of operators is limited and so some tasks will need to be delayed until an operator is available. In [1], the authors propose a number of exact and heuristics algorithms to solve the JSO. These algorithms are outperformed by the genetic algorithm (GA) proposed in [2] and by the exact best-first search algorithm proposed in [3].

Starting from the GA proposed in [2] for the JSO, we introduce herein new versions of the decoding operator, derived from the schedule generation scheme termed $OG\&T$ in [3], which allow the GA to search in different spaces. We then turn the GA into a MA by adding a local search procedure. To this aim, we propose and formalize a neighborhood structure for the JSO which is inspired in a structure for the classic JSP. The essential virtue of this structure is that it avoids moves that will not improve the current solution without computing the makespan of the candidate neighbors, thus saving a lot of computation time. This structure is exploited in two local search strategies: hill climbing and tabu search. To assess the performance of the proposed MA, we have conducted a thorough experimental study across instances of different sizes and characteristics. The results of this study show that our approach is among the best performing methods for the JSO.

Summarizing, the main contributions of this paper are the following: (1) we define a number of new decoding algorithms for a previous genetic algorithm designed to solve the JSO, (2) we define and formally analyze a new neighborhood structure for the JSO, (3) we design a memetic algorithm that combines the previous genetic algorithm with a tabu search algorithm which incorporates the neighborhood structure in its core. This algorithm outperforms the current state of the art for the JSO, and (4) from the formal and experimental studies carried out on the JSO, we obtained insights that may be useful to devise new algorithms for this problem and some of its variants.

The remainder of the paper is organized as follows. In Section 2 we review the literature on memetic algorithms and metaheuristics applied to JSP and some of its extensions, such as the JSO. In Section 3 we formally define the JSO and give a disjunctive model to represent schedules. In Section 4 we describe the $OG\&T$ schedule generation scheme. Section 5 is devoted to explain and study the proposed neighborhood structure. Section 6 describes the local search algorithms that exploit this neighborhood structure. Section 7 presents the main components of the GA and discuss how to

* Corresponding author. Tel.: +34 985182508.
E-mail address: ramiro@uniovi.es (R. Varela).

http://dx.doi.org/10.1016/j.asoc.2015.05.004
1568-4946/© 2015 Elsevier B.V. All rights reserved.

combine the local search algorithms with the GA and the decoding algorithms. Section 8 reports the design and results of the experimental study. Finally, we summarize the main conclusions of the paper and propose some ideas for future research in Section 9.

2. Literature review

In this section we review the main notions regarding memetic algorithms and some of the most relevant metaheuristics that have been applied to solve the JSP and some related problems.

2.1. Memetic algorithms

The term memetic algorithm (MA) was coined by Moscato in [4] to refer to the combination of evolutionary algorithms with local search. These methods are inspired by natural systems that combine the evolutionary adaptation of a population with individual learning over the lifetime of its members. The term memetic comes from the concept of *meme*, defined [5] as a unit of cultural evolution that can exhibit local refinement. In the context of evolutionary optimization, a meme is a local refinement strategy that may improve individuals. Other terms as hybrid genetic algorithms or genetic local searchers are often used in the literature to name MAs.

MAs are a class of hybrid metaheuristics [6] have been successfully used to solve complex optimization problems in discrete and continuous domains in areas such as operations research and artificial intelligence. In particular, they have a large track of success in constrained optimization problems such as scheduling [7]. MAs combine the capability of evolutionary algorithms, such as genetic algorithms, to explore solutions over the whole search space with the capability of local searchers to intensify the search in some promising areas. For this reason, designing a competent MA requires a proper combination of diversification and intensification search efforts. One of the most common approaches to combining them consists in applying local search to chromosomes just after they are generated, as it is pointed in [8], where the authors present an extensive taxonomy of MAs and discuss the main design issues. Some of these important issues are the chromosomes local search should be applied to, how often or for how long [7]. Another important decision is whether Lamarckian evolution is preferred or not. In Lamarckian evolution the characteristics learned during the local search are coded back into the chromosome. This way, these characteristics may better preserved for subsequent generations, but at the same time it may contribute to premature convergence. These and other issues are well discussed in [4,6–8].

2.2. Solving scheduling problems with metaheuristics

Hybrid metaheuristics have been applied to scheduling problems over the last decades. Arguably, one of the problems that has attracted most interest is the classic JSP with makespan minimization. The reason for this may that the JSP can model real life situations [9] and at the same time it is very hard to solve; its decision version is NP-complete in the strong sense and it is considered as one of the hardest problems in this class. As a result, a good number of approaches can be found in the literature, hybrid metaheuristics standing out among them. Building on seminal ideas proposed by Van Laarhoven et al. [10] and Dell'Amico and Trubian [11], sophisticated neighborhood structures have been designed which constitute the core of current state-of-the-art techniques as, for example, the tabu search algorithms proposed in [12,13], the hybrid genetic algorithms proposed in [14], or the recent approach in [15] which combines a differential algorithm with the well-known *i*-TSAB procedure proposed in [12].

In addition, several extensions to the JSP have been proposed in the literature, considering additional characteristics and constraints from real-world scenarios. In these settings, hybrid metaheuristics have often been shown to perform remarkably well, representing the state of the art in many cases. For example, the JSP with sequence dependent setup times was solved in [16] with a memetic algorithm; the JSP with maintenance considerations was considered in [17,18], and solved by hybrid evolutionary and chemical-reaction optimization algorithms respectively. The fuzzy flexible JSP in which the processing times of the tasks are uncertain and each task can be processed in different machines is considered in [19,20]; the processing times are modeled with fuzzy numbers and in both cases the problem is solved by means of hybrid metaheuristics. The JSO is also an extension of the JSP introduced in [1], where the authors propose an exact algorithm based on dynamic programming and some heuristic algorithms. These methods were outperformed by the genetic algorithms proposed in [2,21] and by the best-first search algorithm proposed in [3]. In this last paper, the authors propose the $OG\&T$ schedule generation schema which is used to generate the search space of their best-first search algorithm. The GA proposed in [21] exploits the $OG\&T$ as decoder. In [2], the authors introduce some improvements to this GA, such as the concept of weak Lamarckian evolution and a way to reduce the search space.

3. Description of the problem

In the JSO, we are given a set of n jobs $\{J_1, \ldots, J_n\}$, a set of m resources or machines $\{R_1, \ldots, R_m\}$ and a set of p operators $\{O_1, \ldots, O_p\}$. The job J_i consists of a sequence of v_i operations or tasks $(\theta_{i1}, \ldots, \theta_{iv_i})$. The task v has a single resource requirement R_v and a positive integer duration p_v. A feasible schedule is an assignment of starting times st_v and operators O_v for all operations v that satisfies the following constraints: (i) the operations of each job are sequentially scheduled, (ii) each machine can process at most one operation at any time, (iii) no preemption is allowed and (iv) each operation is assisted by one operator and one operator cannot assist more than one operation at a time. The objective is finding a feasible schedule that minimizes the completion time of all the operations, i.e. the makespan. The significant cases of this problem are those with $p < min(n, m)$, otherwise the problem is equivalent to the classic JSP.

A feasible schedule $\mathcal{S}$ can be represented by an acyclic graph of the form $G_{\mathcal{S}} = (V, K \cup H \cup I \cup Q)$, where the set V includes one node for each operation, nodes that represent the beginning of the sequence of operations assisted by the operator O_i, O_i^{start}, and the dummy nodes *start* and *end*. K represents precedence constraints among operations of the same job. I includes arcs of the form $(start, O_i^{start})$, $i = 1, \ldots, p$, and arcs $(start, \theta_{i1})$ and (θ_{ik_i}, end), $i = 1, \ldots, n$. H expresses the processing order of operations on the machines and Q expresses the sequences of operations assisted by each operator. The makespan is the cost of a *critical path* in $G_{\mathcal{S}}$. A critical path is a longest cost path from node *start* to node *end*.

Fig. 1 shows a solution graph for an instance with 3 jobs, 3 machines and 2 operators. Discontinuous arrows represent arcs in Q. So, the sequences of operations assisted by operators O_1 and O_2 are $(\theta_{21}, \theta_{11}, \theta_{32}, \theta_{12}, \theta_{13})$ and $(\theta_{31}, \theta_{22}, \theta_{23}, \theta_{33})$ respectively. In order to simplify the picture, if there are two arcs between the same pair of nodes, only the operator arc in Q is drawn. Continuous arrows represent arcs in K and doted arrows represent arcs in H. In this example, the critical path is given by the sequence $(\theta_{21}, \theta_{11}, \theta_{32}, \theta_{12}, \theta_{33})$, so the makespan is 14. Fig. 2 is a Gantt chart of the schedule represented by Fig. 1.

In order to simplify expressions, we define the following notation for a feasible schedule. The *head* r_v of an operation v is the cost

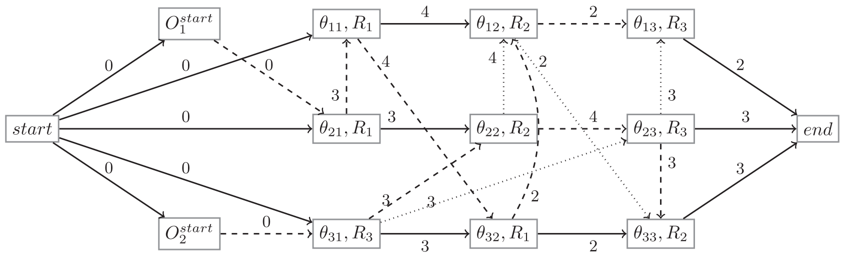


Fig. 1. A feasible schedule to a problem with 3 jobs, 3 machines and 2 operators.

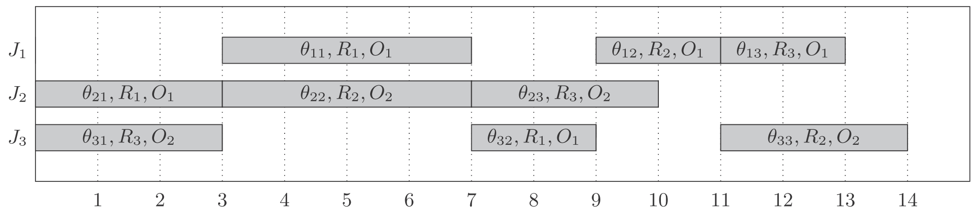


Fig. 2. Gantt chart of the schedule represented in Fig. 1.

of the longest path from node *start* to node v, i.e. it is the value of st_v. The *tail* q_v is defined so that the value $q_v + p_v$ is the cost of the longest path from v to *end*. Hence, $r_v + p_v + q_v$ is the makespan if v is in a critical path, otherwise it is a lower bound. PM_v and SM_v denote the immediate predecessor and successor of v respectively on the machine sequence, PJ_v and SJ_v denote those on the job sequence and PO_v and SO_v on the operator of v.

4. Schedule generation schemes

In [22], the authors proposed a schedule generation scheme for the JSP termed *G*&*T* algorithm. This algorithm has been used in a variety of settings for job shop scheduling problems. For example, in [23–25] it was used as a greedy algorithm, in [26,14] it was exploited as a decoder for genetic algorithms, and in [27] it was used to define the state space for a best-first search algorithm. Furthermore, the *G*&*T* algorithm has inspired the design of similar schedule builders such as the *EG*&*T* algorithm proposed in [28] for the job-shop scheduling problem with sequence dependent setup times, or the *OG*&*T* algorithm for the JSO proposed in [3]. We use here this last algorithm as the basis of the decoding algorithms, so in the remainder of this section, we summarize the characteristics of this algorithm that are relevant for our purpose. For a full description of the *OG*&*T* algorithm and the proofs of its formal properties, we refer the interested readers to [3].

In the *OG*&*T* algorithm, the operations are scheduled one at a time, in sequential order within each job. When an operation u is scheduled, it is assigned a starting time st_u and an operator $O_u \in \{O_1, \ldots, O_p\}$. Let SC be the set of scheduled operations at a given time and G_{SC} the partial solution graph built so far[1], and let $PM(u)$ and $PO(u)$ be the sets of operations preceding $u \in SC$ in the machine sequence and the operator sequence respectively. As all operations in $PM(u)$ or $PO(u)$ were already scheduled, if u was assigned operator O_i there is a path in G_{SC} from the node O_i^{start} to u through all operations in $PO(u)$ and a path from *start* to u through all operations in $PM(u)$. Let A be the set that includes the first unscheduled operation of each job that has at least one unscheduled operation, i.e.

$$A = \{u \notin SC; (\nexists PJ_u) \vee (PJ_u \in SC)\} \tag{1}$$

[1] If $SC = \emptyset$ then G_{SC} contains neither machine nor operator arcs, i.e., it only contains arcs in sets K and I.

For each operation u in A, let r_u denote the starting time of u if u were selected to be scheduled next. Clearly, r_u is greater than or at least equal to both the completion time of PJ_u and the completion time of the last operation scheduled on the machine R_u; let v be this operation. Moreover, the value of r_u also depends on the availability of operators at this time, hence r_u is not lower than the earliest time an operator is available. Let t_i, $1 \le i \le p$, be the time at which the operator O_i is available, then

$$r_u = \max \left\{ r_{PJ_u} + p_{PJ_u}, r_v + p_v, \min_{1 \le i \le p} t_i \right\} \tag{2}$$

abusing notation we consider $PJ_u = start$ if u is the first operation in its job and $v = start$ if no operation has been scheduled on the machine R_u yet.

In general, a number of operations in A could be scheduled simultaneously at their current heads, however it is clear that not all of them could start processing at these times due to both capacity constraints and operators availability. So, a straightforward schedule generation scheme is obtained if each one of the operations in A is considered as candidate to be scheduled next.

If the selected operation is $u \in A$, it is scheduled at a time $st_u = r_u$, u is added to SC and the machine arc (v, u), such that v is the last operation scheduled on the machine R_u, is included in G_{SC}. Also $O_u = O_j$, being O_j the last operator available at a time before r_u, i.e.,

$$t_j = arg\max\{t_i; t_i \le r_u, 1 \le i \le p\} \tag{3}$$

Then, the operator arc (v, u), such that v is the last operation assigned to the operator O_j, is also included in G_{SC}. If $PO(u) = \emptyset$, then the arc (O_j^{start}, u) is added to G_{SC}.

So, if we start from the set A containing the first operation of each job and iterate until all the operations get scheduled, no matter what operation is selected in each step, we will finally have a feasible schedule, and there is a sequence of choices eventually leading to an optimal schedule.

Let us now consider the operation v^* in A with the earliest completion time if it were selected to be scheduled next, i.e.,

$$v^* = arg\min\{r_u + p_u; u \in A\} \tag{4}$$

and the set A' given by the operations in A that can start before the earliest completion of v^*, i.e.,

$$A' = \{u \in A; r_u < r_{v^*} + p_{v^*}\} \tag{5}$$

If we restrict the selection, in each step of the previous scheduling algorithm, to operations in A', there still exists a sequence of choices leading to an optimal schedule. In doing so, the search space is reduced to a subset of the previous space. Furthermore, the starting time of the next operation scheduled is in the interval $[T, C)$, where $C = r_{v^*} + p_{v^*}$ and $T = min\{r_u ; u \in A\}$, if this operation is selected from A', while an operation selected from A may be scheduled at a time equal or larger than C.

So, using the set A may produce larger idle intervals of available resources, machines or operators, than using set A' and then one may expect that the average value of schedules generated from A is to be larger than that generated from A'. At the same time, the search space potentially generated from A will be larger than that generated from A'.

5. Neighborhood structure for the JSO

As far as we know, no neighborhood structure has been proposed for the JSO. In this section, we propose a neighborhood structure for the JSO, termed $\mathcal{N}^{\mathcal{O}}$, which is an extension of the well-known $\mathcal{N}$ structure proposed in [29] for the classic JSP. These structures rely on reversing arcs in a critical path of a solution and one their key points is avoiding moves that cannot improve the makespan of the current schedule. We start reviewing the $\mathcal{N}$ structure and then we will see how to extend it to the JSO.

5.1. The neighborhood structure $\mathcal{N}$ for the JSP

This structure relies on the concepts of critical path and critical block, and it is based on various results proved in [10,29,30]. Given a schedule $\mathcal{S}$, it considers one critical path in the solution graph[2] that represents $\mathcal{S}$ and a set of moves called "interchange near the borderline of blocks on a single critical path", what means swapping pairs of operations only at the beginning or at the end of a critical block.[3] Swapping these operations yields a proven feasible schedule. The transformation rules of $\mathcal{S}$ are defined as follows:

Definition 1 *($\mathcal{N}$).* Let (v, w) be the first or the last two operations of a critical block B. Reversing the arc (v, w) is considered as a neighboring solution. For the first block in the critical path only (v, w) at the end of the block is considered whereas for the last block only (v, w) at the beginning of the block is considered.

5.2. The neighborhood structure $\mathcal{N}^{\mathcal{O}}$ for the JSO

The structure $\mathcal{N}^{\mathcal{O}}$ extends $\mathcal{N}$ by considering the arcs of the solution graph due to the operators. So, to formalize the new structure, we establish the following types of arcs:

- (v, w) is a conjunctive arc or an arc of type J if $J_v = J_w$, $O_v \neq O_w$ and $M_v \neq M_w$.
- (v, w) is an arc of machine or an arc of type M if $M_v = M_w$, $O_v \neq O_w$ and $J_v \neq J_w$.
- (v, w) is an arc of operator or an arc of type O if $O_v = O_w$, $J_v \neq J_w$ and $M_v \neq M_w$.
- (v, w) is an arc of job and operator or an arc of type JO if $J_v = J_w$, $O_v = O_w$ and $M_v \neq M_w$.
- (v, w) is an arc of machine and operator or an arc of type MO if $M_v = M_w$, $O_v = O_w$ and $J_v \neq J_w$.

[2] This graph is a simplification of the solution graph defined here for the JSO. It does not contain nodes and arcs associated to operators.

[3] A critical block is maximal subsequence of operations of a critical path that use the same machine.

Clearly, reversing arcs of type J and JO gives always rise to unfeasible schedules as the order of two operations in a job cannot be altered. Reversing arcs of types MO, M, and O will lead to feasible schedules if the resulting solution graph has no cycles. So these are the candidates to define moves of the new structure. In order to obtain an efficient structure, i.e., a neighborhood with reduced size and a large chance of improvement, we will establish some conditions for feasibility and non-improvement for candidate neighbors. To do that, we need to introduce the following concepts.

Definition 2. (Critical arc) An arc that is in a critical path. It may be of any type (J, JO, M, O or MO).

Definition 3. (Critical block) Maximal subsequence of operations of a critical path that use

- the same machine (machine critical block), or
- the same operator (operator critical block).

Note that the arcs in a machine critical block are of type M or MO, and the arcs in an operator critical block are of type O, MO or JO. It is also important to remark that a critical arc (v, w) may belong to a machine critical block MB and to an operator critical block OB at the same time and that, for example, the arc (v, w) may be interior to MB and in an extreme of OB. So, according to the definitions above, a critical path may not be a sequence of disjoint critical blocks, as it is in simpler versions of the problem, such as the classic JSP.

Let $\mathcal{S}$ be a schedule and $G_{\mathcal{S}}$ its solution graph. The following propositions establish the foundations of this neighborhood structure.

Proposition 1. *If an arc (v, w) of $G_{\mathcal{S}}$ is critical, then the only path from v to w in $G_{\mathcal{S}}$ is the arc (v, w).*

Proof. Any other path from v to w has at least two arcs (v, x) and (x, w). The arc (v, x) has the same cost as the arc (v, w), as the source is v in both cases. Thus the cost of this path would be larger than the cost of the arc (v, w) and in that case (v, w) would not be in the critical path. □

Corollary 1. *If (v, w) of type M, O, or MO is critical in $G_{\mathcal{S}}$, then the graph $G_{\mathcal{S}prime}$ which is obtained from $G_{\mathcal{S}}$ by reversing (v, w) does not have cycles and thus represents a feasible schedule.*

Proposition 2. *If an arc (v, w) that is not critical in $G_{\mathcal{S}}$ is reversed and the resulting graph $G_{\mathcal{S}'}$ has no cycles, the obtained solution $\mathcal{S}'$ is not better than $\mathcal{S}$.*

Proof. All critical paths of $G_{\mathcal{S}}$ remains in $G_{\mathcal{S}'}$ after reversing the non-critical arc. □

Proposition 3. *If a critical arc (v, w) is reversed in $G_{\mathcal{S}}$ to obtain $G_{\mathcal{S}'}$, and (v, w) is internal (neither the first nor the last) to a machine critical block or internal to an operator critical block, the resulting solution $\mathcal{S}'$ is not better than $\mathcal{S}'$.*

Proof. If (PO_v, v, w, SO_w) are successive operations in an operator critical path, then $r_{SO_w} = r_{PO_v} + p_v + p_w$. So, after reversing (v, w) the starting time of SO_w, r_{SO_w} would be at least $r_{PO_v} + p_v + p_w$. Therefore this reversal cannot lead to an improvement. Analogously, if (PM_v, v, w, SM_w) are successive operations in a machine critical path, after reversing (v, w), r_{SM_w} would be at least $r_{PM_v} + p_v + p_w$. Therefore this reversal cannot lead to an improvement. □

Proposition 4. *If (v, w) is the first arc of the first critical block, or the last arc of the last critical block, then the reversal of (v, w) cannot produce any improvement (unless the block has only two operations).*

Proof. If (v, w) is the first arc of the first critical block and this is a machine critical block with at least three operations, (v, w, SM_w), then after reversing (v, w) the head of SM_w would be at least $(p_w + p_v)$, so there would be a path in the resulting solution graph with at least the same length than the original critical path. By analogous reasoning if the first critical block is a machine critical block, or (v, w) is the last arc of the last critical block (operator or machine) with length larger than 2, it can be proved that reversing (v, w) cannot lead to a schedule with lower makespan than that of the original schedule. □

According to the above results, the only option to improve, with a simple reversal of an arc, is reversing one arc of type O, M, or MO, in the border of a critical block; with the exception of the first (last) arc of the first (last) block unless this block has only two operations. So, we define the neighborhood as follows:

Definition 4. ($\mathcal{N}^O$) Given a schedule $\mathcal{S}$, the neighborhood $\mathcal{N}^O(\mathcal{S})$ is defined as the set of schedules obtained from $G_{\mathcal{S}}$ by reversing an arc (v, w) that satisfies the following four conditions:

(i) is critical,
(ii) is of type O, M or MO,
(iii) is not in the interior of a machine or operator critical block,
(iv) is neither the first nor the last arc of a critical path, or it belongs to a machine or operator critical block with length 2.

5.3. Makespan estimation

Even though a neighboring solution is very similar to the original solution, evaluating its exact makespan requires searching across the new solution graph. This search may be very time consuming and at the same time the exact value is not usually necessary for a local search strategy. For these reasons, some approximate method is often used. In [31], a estimation procedure for the classic JSP was proposed that returns a tight lower bound of the makespan after reversing some arcs. This strategy can be adapted to the JSO considering the differences due to the operators. This method is based on the following ideas.

Given a schedule $\mathcal{S}$, for every node v in $G_{\mathcal{S}}$, the value $r_v + p_v + q_v$ is a lower bound on the makespan. Moreover, it is the actual makespan if node v belongs to a critical path.

The makespan estimation is based on calculating heads and tails after a move. Let r'_v, r'_w, q'_v and q'_w be the heads and tails of the operations v and w after the reversal of the arc (v, w). The makespan of the neighboring solution can be estimated as:

$$C'_{\max} = \max(r'_w + p_w + q'_w, r'_v + p_v + q'_v) \tag{6}$$

For the JSO, the exact values of r'_v, r'_w, q'_v and q'_w can be calculated for each type of arc as follows:

- For an arc of type MO

$$\begin{aligned} r'_w &= \max(r_{PM_v} + p_{PM_v}, r_{PJ_w} + p_{PJ_w}, r_{PO_v} + p_{PO_v}) \\ r'_v &= \max(r'_w + p_w, r_{PJ_v} + p_{PJ_v}) \\ q'_v &= \max(q_{SM_w} + p_{SM_w}, q_{SJ_v} + p_{SJ_v}, q_{SO_w} + p_{SO_w}) \\ q'_w &= \max(q'_v + p_v, q_{SJ_w} + p_{SJ_w}) \end{aligned} \tag{7}$$

- For an arc of type M

$$\begin{aligned} r'_w &= \max(r_{PM_v} + p_{PM_v}, r_{PJ_w} + p_{PJ_w}, r_{PO_w} + p_{PO_w}) \\ r'_v &= \max(r'_w + p_w, r_{PJ_v} + p_{PJ_v}, r_{PO_v} + p_{PO_v}) \\ q'_v &= \max(q_{SM_w} + p_{SM_w}, q_{SJ_v} + p_{SJ_v}, q_{SO_v} + p_{SO_v}) \\ q'_w &= \max(q'_v + p_v, q_{SJ_w} + p_{SJ_w}, q_{SO_w} + p_{SO_w}) \end{aligned} \tag{8}$$

- For an arc of type O

$$\begin{aligned} r'_w &= \max(r_{PM_w} + p_{PM_w}, r_{PJ_w} + p_{PJ_w}, r_{PO_v} + p_{PO_v}) \\ r'_v &= \max(r'_w + p_w, r_{PJ_v} + p_{PJ_v}, r_{PM_v} + p_{PM_v}) \\ q'_v &= \max(q_{SM_v} + p_{SM_v}, q_{SJ_v} + p_{SJ_v}, q_{SO_w} + p_{SO_w}) \\ q'_w &= \max(q'_v + p_v, q_{SJ_w} + p_{SJ_w}, q_{SM_w} + p_{SM_w}) \end{aligned} \tag{9}$$

So, the estimation given by Eq. (6) is a lower bound on the makespan of the neighboring solution.

6. Local search strategies

We consider two classic local search strategies, simple hill-climbing (HC) and tabu search (TS). In the first case, the neighbors of the current solution are calculated one at a time. The makespan of each neighbor is estimated with the algorithm described in Section 5.3. If the estimation returns a value greater than or equal to the makespan of the current solution, the neighbor is discarded, otherwise the neighbor replaces the current solution and its actual makespan is calculated. The local search finishes when all the neighbors of the current solution are non-improving ones, in accordance with the makespan estimation.

Algorithm 1. Tabu search for the JSO

```
Require: An initial solution $s_0$ for a problem instance $P$
Ensure: A (hopefully) improved solution $s_B$ for problem instance $P$
  Set current solution $s = s_0$ and current solution $s_B = s$;
  Set NoImpIter=0. Empty the tabu list;
  while no stoping criteria has been reached do
    Generate list of candidate arcs of the current solution $s$ using the neighborhood structure;
    if there is one or more candidate arcs that are not tabu or satisfy the aspiration criterion then
      Let $(u, w)$ be the arc with the best estimation. Calculate the neighbor $s^*$ reversing $(u, w)$, update the tabu list accordingly and let $s = s^*$;
      if $s^*$ is better than $s_B$ then
        Set $s_B = s^*$;
        Set NoImpIter=0;
      else
        Set NoImpIter=NoImpIter+1;
    else
      Stopping criteria reached;
  return The solution $s_B$;
```

Algorithm 1 shows the TS algorithm used herein, which is very similar to that proposed in [11]. In each iteration, the list of candidate moves given by $\mathcal{N}^O$, i.e., the arcs that can be reversed in the current solution, is obtained and their makespan estimated. Moves that are tabu and do not fulfill the aspiration criterion, i.e., the estimation of the neighbor is not better than the makespan of the current solution, are discarded. Finally, the candidate with the best estimation is selected. The algorithm stops after running *Maxiter* iterations without improving the best solution found, or when there are no neighbors or all of them are discarded. TS uses a tabu list to keep track of the evolution of the algorithm in order to avoid cycles and find good solutions. After an arc (u, w) is reversed, it remain in the tabu list for the next *TabuLength* iterations. The value of *TabuLength* varies between 1 and *Maxiter* and it is updated depending on the quality of the new neighbor each iteration. If the new neighbor is better than the last one, *TabuLength* is increased by one. If not, *TabuLength* is decreased by one (unless *TabuLength* = 1). Finally, *TabuLength* is set to 1 if the new solution is better than the best solution found so far.

7. Memetic algorithm for the JSO

As we have mentioned, we start from a GA inspired in that proposed in [2]. The coding schema is based on permutations with repetition as it was proposed in [26]. A chromosome is a permutation of the set of operations that represents a tentative ordering to schedule them, each one being represented by its job number. For example, the sequence (2 1 1 3 2 3 1 2 3) is a valid chromosome for a problem with 3 jobs and 3 machines. For chromosome mating, the GA uses the job-based order crossover (JOX) described in [26]. Given two parents, JOX selects a random subset of jobs and copies their genes to the offspring in the same positions as they are in the first parent, then the remaining genes are taken from the second parent so that they maintain their relative ordering.

The remaining elements of GA are rather conventional. To create a new generation, all chromosomes from the current one are organized into couples which are mated and then mutated to produce two offsprings in accordance with the crossover and mutation probabilities respectively. Finally, tournament replacement among every couple of parents and their offsprings is done to obtain the next generation. Algorithm 2 shows the general structure of the GA.

Algorithm 2. Genetic algorithm

```
Require: A JSO problem instance $\mathcal{P}$ and a set of parameters ($P_c, P_m, \#gen, \#popsize, \mathcal{X}$).
Ensure: A feasible schedule for $\mathcal{P}$.
  Generate and evaluate the initial population $P(0)$;
  for t=1 to #gen-1 do
    selection: organize the chromosomes in $P(t-1)$ into pairs at random;
    recombination: mate each pair of chromosomes and mutate the two offsprings in accordance with $P_c$ and $P_m$;
    evaluation: evaluate the resulting chromosomes considering the search space $\mathcal{X}$ and code back the schedules into the chromosomes;
    replacement: make a tournament selection among every two parents and their offsprings to generate $P(t)$;
  return The best schedule built so far;
```

7.1. Decoding algorithm

We consider two decoding algorithms derived from the *OG*&*T* algorithm described in Section 4; more precisely from the sets A and A' calculated at each step of the algorithm. So, a schedule is built by means of *OG*&*T* and, in the first decoding algorithm, the next operation scheduled is that in the set A which is the leftmost in the chromosome. For the second decoder, the set A' is considered. Abusing language, we call these decoders A and A'.

These decoders build solutions in different search spaces, being the space from A' a subspace of that from A. This fact may have some influence on the diversification-intensification tradeoff and then on the evolution of the algorithms; not only on the evolution of GA working alone but also in that of the MA obtained by combining the GA with any of the local searchers, HC or TS. In these last cases, starting from a solution in the corresponding space, the local search may lead to a new solution out of this space.

7.2. Combining GA with local search

As it is usual, local search is applied to the chromosomes in each generation after crossover and mutation. In order to keep a reasonable balance between intensification and diversification, local search is applied to a fraction of the chromosomes in the population. If the local search returns an improved solution, this solution is coded back in the chromosome, what is known as Lamarckian evolution. In this way, the learned characteristics during the local search are better translated to the new offsprings in subsequent matings. Also, when a chromosome does not undergo local search, the schedule built by the decoder is also coded back in the chromosome, this was called *weak Lamarckian evolution* in [2] as it usually makes small changes in the chromosome structure.

8. Experimental study

We have conducted and experimental study aimed at analyzing the proposed algorithms and comparing them with the state of the art. In the experiments we considered two benchmark sets, $B1$ and $B2$. Due to the differences in their sizes, the instances in $B1$ can be expected to be harder to solve than the instances in $B2$.

$B1$ consists of a number problem instances derived from well-known JSP benchmarks (some medium-size and large instances taken from the OR-library [32], and some large ones from [33]). For an instance with n jobs and m machines, i.e. a $n \times m$ instance, we generated instances with number of operators p ranging in $\{\lfloor \min(n, m)/2 \rfloor, \ldots, \min(n, m)\}$; the reason for this is that, in our experience, instances with less operators are in general much easier to solve. The JSP instances considered are: *LA*21-25 (15 × 10), *LA*26-30 (20 × 10), *LA*31-35 (30 × 10), *LA*36-40 (15 × 15), *ta*11-20 (20 × 15), *ta*21-30 (20 × 20) and *ta*31-40 (30 × 15). Then, we have 425 instances in all.

$B2$ contains the 360 instances proposed in [1]. All these instances have $n = 3$ and $p = 2$ and are characterized by the number of machines (m), the maximum number of operations per job ($\upsilon_{\max}$) and the range of processing times (p_i). A set of small instances was generated combining three values of each parameter: m in {3, 5, 7}; $\upsilon_{\max}$ in {5, 7, 10} and p_i ranging in each of the intervals {[1, 10], [1, 50], [1, 100]}. Also, a set of larger instances was generated with $m = 3$, with $\upsilon_{\max}$ in {20, 25, 30} and p_i in {[1, 50], [1, 100], [1, 200]}. The sets of small instances are identified by numbers from 1 to 27 and the sets of large instances are identified by labels from $L1$ to $L9$. Additionally, $B2$ includes a set of 60 larger instances named L4J in [3]. They were created from 15 JSP instances with $n = 4$ and $m = 30$, considering a number of operators p ranging from 1 to 4.

The algorithms were coded in C++ and the target machine was Intel Xeon 2.26 GHz. 24 GB RAM.

In the next subsections we summarize the results from the experiments. Firstly, we analyze the performance of the schedule builders and the local searchers working on random chromosomes. Then, we consider the MA with different combinations of schedule builders and local searchers. And, finally, we compare the MA with the state-of-the-art methods, to our knowledge the GA proposed in [2] and the BF algorithm proposed in [3]. In the first case we experimented on the benchmark $B1$, and in the second we experimented only with $B2$ due to the fact that BF cannot solve reasonably most of the instances in $B1$.

8.1. Analysis of the schedule builders and the local search algorithms

We started with some experiments aimed at comparing the two considered schedule builders A and A'. To this end, we evaluated 10,000 random chromosomes with both decoders for each one of

Table 1
Average makespan and time taken by A and A' to decode 10,000 random chromosomes, and results from local searchers HC and TS(*Maxiter*) starting from the solutions decoded by A and A', time is given in milliseconds.

	A		A'	
	Makespan	Time (ms)	Makespan	Time (ms)
Initial	2415.33	0.18	1610.32	0.21
HC	2330.53	1.49	1603.32	0.49
TS(10)	2276.43	4.59	1598.33	2.18
TS(50)	2243.27	13.78	1596.36	7.55
TS(100)	2236.98	21.03	1595.60	13.00

the 45 JSO instances derived from the LA36-40 JSP instances, and recorded the average value of the makespan in all cases. These values are reported in Table 1 (row Initial). As we can observe, the makespan obtained with A is much larger than that obtained with A', being the time taken by both schedulers very similar.

However, as decoder A may produce more diverse schedules than A', it could happen that after applying local search to the schedules produced by A the results were better than those from the schedules produced by A'. To assess this, we have applied both local search algorithms, HC and TS, from the schedules obtained by both schedulers, taking different values for the parameter *Maxiter* in TS. The results of these experiments are summarized in the remaining rows of Table 1. As we can observe, all local searchers improve the quality of the initial solutions and the final solutions obtained from those previously decoded with A' are much better than those obtained from the ones decoded by A. Even those obtained by TS(100) from A are much worse than the solutions obtained from A'. Taking into consideration the time required by the algorithms, the best option seems to be the combination A' with TS(10). However, this must be demonstrated in the context of the memetic algorithm and not only in the evaluation of a random initial population.

In these experiments, we have also registered the percentage of neighbors that can be discarded from the makespan estimation and the number of iterations required in each local search. Table 2 shows these values for HC. As we can see, the percentage of neighbors that cannot be discarded after the makespan estimation, and so require evaluation, are about 12% starting from solutions decoded by either A or A'; so the makespan estimation allows the algorithm to discard almost 90% of the neighbors without wasting time evaluating them. However, the number of iterations required from solutions decoded by A is 3 times larger than those required from solutions decoded by A'. This is quite natural as solutions decoded by A are worse and so they have more chance for improving.

8.2. Looking for the best combination

From the results reported in the previous section, the combination A' + TS(10) seems to be the best one. But we have to be aware that these experiments were done over a set of random chromosomes. Now, we consider the evolution of the memetic algorithm with six combinations of decoders and local searchers, namely A, A + HC, A + TS(10), A', A' + HC, A' + TS(10), where A and A' mean that no local search is used.

Table 2
Percentage of neighbors that require evaluation (%eval.), average number of iterations (#iter.) and time taken with HC starting from solutions obtained by each decoder (Dec.).

Dec.	%eval.	#iter.	Time (ms)
A	11.84	6.05	1.49
A'	12.27	2.25	0.49

Table 3
Average rankings of the six versions of the memetic algorithm.

Algorithm	Ranking
A' + TS(10)	1.48
A' + HC	1.64
A'	2.88
A + TS(10)	4.20
A + HC	4.80
A	6.00

In these experiments, the genetic algorithms were parameterized with #*popsize* = 150, $P_c = 0.7$ and $P_m = 0.2$, with an unlimited number of generations but a maximum time of 30 s. When local search was used, the local search probability was set to 0.15 (i.e., local search was issued from 15% of the chromosomes). These values were chosen considering the parameters used in [2] for the GA and also the results from some preliminary experiments. We considered the 45 JSO instances derived from the *LA*36-40 JSP instances. Each algorithm was run 30 times for each instance and the best makespan obtained in each run was registered and averaged for the 30 runs.

In order to infer conclusions for a broader class of problem instances we have made a series of statistical inference tests. Following [34], we have used non-parametric statistical techniques.[4] Concretely, we made Friedman and Iman-Davenport tests to establish a ranking among the algorithms and to deduce whether there are significant differences among the algorithms or not.

Table 3 shows the average ranking computed by the Friedman test. In this case, Friedman and Iman-Davenport tests yielded *p*-values of 1.36E−10 and 3.33E−16 respectively, meaning there are significant differences among the algorithms. As can be seen in the ranking, the three best combinations are those that use A'. Moreover TS(10) is always ahead of *HC* and they are both ahead of the case in which no local search is issued.

So, from these studies, we take A' + TS(10) as the best option and it is then used in MA in the remainder of the experimental study.

8.3. Comparison with state-of-the-art

In this section we report results from MA over the whole set of instances considered and compare these results with the state-of-the-art. As we have mentioned, to our knowledge the best methods proposed so far are the genetic algorithm (GA) proposed in [2] and the best-first search (BF) exact algorithm proposed in [3]. In these experiments the memetic algorithm was parameterized with a #*popsize* = 250, $P_c = 0.7$, $P_m = 0.2$ and a local search probability of 0.15.

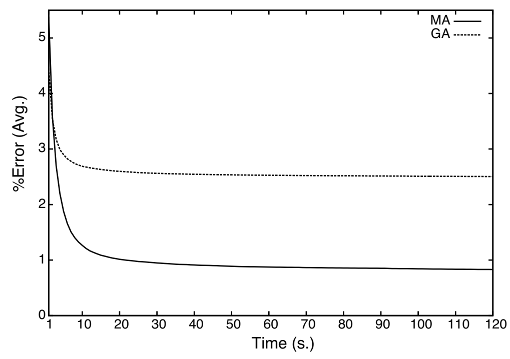
8.3.1. Comparison of MA with GA

Firstly, we compare MA with over the 425 instances of the benchmark set *B*1.

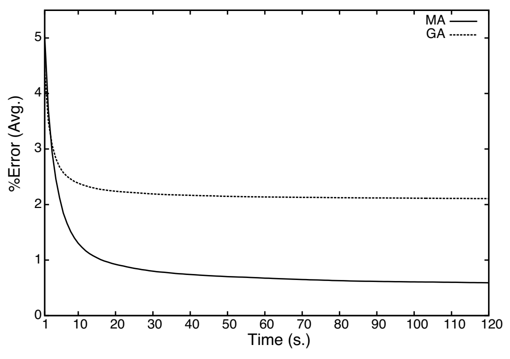
The genetic algorithm was parameterized as in the experimental study reported in [2]; i.e., #*popsize* = 100, $P_c = 0.7$ and $P_m = 0.2$. Both algorithms were run for 120 s (unlimited number of generations), 30 times for each instance and the average of the best solutions reached in the 30 runs was recorded. In order to make a point later, the 30 × 15 instances were run for 300 s as well. Then, the error in percentage w.r.t. the best solution reached in this study is reported for each instance.[5]

[4] To this aim, we have used the software developed by the Research Group on Soft Computing and Intelligent Information Systems at the University of Granada, which is available at http://sci2s.ugr.es/sicidm.

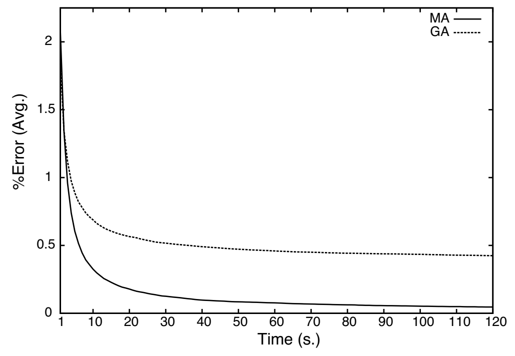
[5] The instances and more details of the experimental study are available at http://www.di.uniovi.es/iscop (Repository).



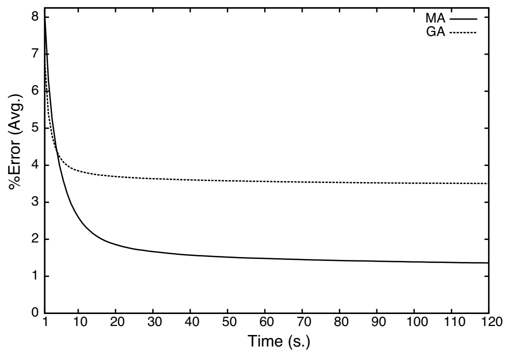
(a) LA: 15× 10



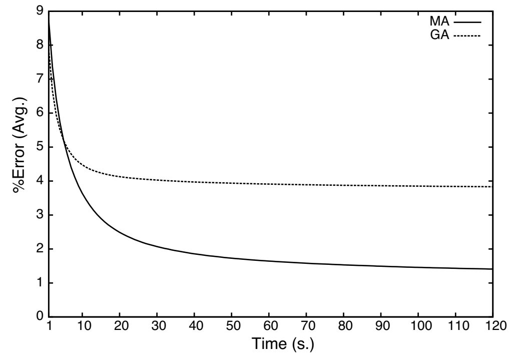
(b) LA: 20× 10



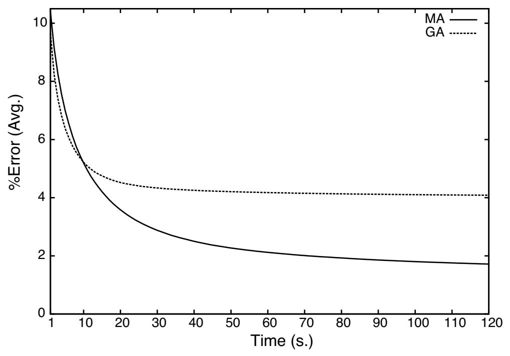
(c) LA: 30× 10



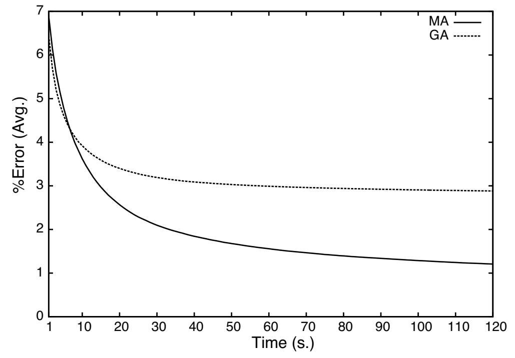
(d) LA: 15× 15



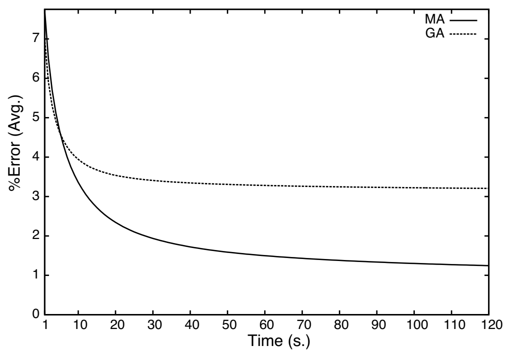
(e) Taillard: 20× 15



(f) Taillard: 20 × 20



(g) Taillard: 30 × 15



(h) All instances

Fig. 3. Evolution of the average error in percent over time (0–120 s) obtained by GA and MA across the Lawrence and Taillard instances. Each plot represents the values averaged for all instances of the same size $n \times m$.

Table 4
Summary of results from BF vs MA on the *B*1 data set. MA was given a time limit of 5, 20 and 120 s. for the subsets SMALL, LARGE and L4J respectively. %Err. Avg. and %Err. Best refer to the error (in percent terms) of the average and best solutions reached in 30 runs of MA, averaged for each subset of instances.

	BF		MA	
Instances	T (s)	%Err.	%Err. Avg.	%Err. Best
SMALL				
1–9	0.00	0.00	0.00	0.00
10–18	0.02	0.00	0.01	0.00
19–27	0.08	0.00	0.03	0.00
LARGE				
L1–L3	2.93	0.00	0.36	0.09
L4–L6	7.33	0.00	0.47	0.14
L7–L9	16.60	0.00	0.68	0.20
L4J				
1op.	0.13	0.00	0.00	0.00
2op.	32.20	0.03	0.07	0.03
3op.	135.00	0.00	1.11	0.31
4op.	134.93	0.00	1.13	0.38

Fig. 3 shows the evolution of the average error over time. At the start, MA and GA return similar results, but in less than 10 s, MA takes the lead and returns much better results from then on. GA takes the lead over the first seconds of the search in some cases because it has time to complete more generations than MA, as it needs to do less computing (no local search). Another note in favor of MA over GA is that for some subsets it did not converge in 120 s, while GA did, which means that this difference would be greater in longer tests.

Fig. 4 reports for each set of instances and number of operators *p* the error in percentage averaged for the instances in each set. We can observe that MA clearly outperforms GA as it always yields better results. Moreover, we observe that in general the error showed by both algorithms and the differences between them grow with the number of operators with some exception for the largest values of *p*. This is natural as large values of *p* mean large search spaces, which results in the solutions reached by MA being more disperse. This may be interpreted as that for large values of *p* there would be still room for improvement if MA was given more time. Indeed, after 300 s it reaches much lower error than in 120 s, as shown in Fig. 4(h); while GA yields almost the same error.

Apart from the average error, we calculated the average Pearson coefficient of variation in percentage terms of the solutions (CV) computed as the ratio of standard deviation and the average of the solutions across all the runs. The values of CV were 0.61 for MA and 0.67 for GA, which tell us that both algorithms are considerably stable as these are low values, being MA slightly more stable than GA.

All things considered, we can conclude that MA clearly outperforms GA, as it achieves better results in average and it is more stable at the same time.

8.3.2. *Comparison of MA with BF*

Now, we compare MA with the best-first search algorithm (BF) proposed in [3]. This algorithm relies on the *OG&T* schedule generation scheme to build up the search space and was evaluated on the benchmark set *B*2.

In these experiments, MA was given the same parameters as in the previous tests, but the time given was in direct relation with the size of the instances. In every case the algorithm was run 30 times for each instance.

Table 4 summarizes the results from BF reported in [3] together with results from MA on the *B*1 data set. Taking into account the time taken by BF, MA was given time limit of 5, 20 and 120 s. for the subsets SMALL, LARGE and L4J respectively. In [3], BF was not given a time limit and so it finished either after certifying the optimal solution or when the memory was exhausted, in this latter case returning a lower bound and the best solution reached. BF is in fact an any-time algorithm which uses a greedy algorithm to reach upper bounds during the search. From the results in Table 4 and the results showed in Fig. 5 on the evolution of MA, we can draw the following conclusions. For the smallest instances, BF finds optimal solutions quickly. However, for these instances, MA is not always able to reach the optimal solution, but it reaches solutions with error in percentage of 0.05 or lower in less than 0.5 s, as it can be observed in Fig. 5(a)–(c).

Table 5
Summary of results from BF vs MA on the 6 LA21 instances. LB and UB are the lower bounds and the makespan (upper bounds) of the solutions reached by BF. Time refers to the time at which BF obtained the returned solution. For MA, we report the Best and Average makespan of the solutions reached in the 30 runs, each run taking a time limit of 120 s. Values in bold correspond to optimal solutions

	BF			MA	
Instance	LB	UB	Time (s)	Best	Average
LA21					
5 op.	**1599**	**1599**	6640	**1599**	**1599.00**
6 op.	**1333**	1377	1346	**1333**	1333.63
7 op.	1145	1255	3702	1149	1161.73
8 op.	1033	1151	51	1073	1086.6
9 op.	1033	1115	1012	1058	1073.00
10 op.	1033	1102	1072	1055	1069.43

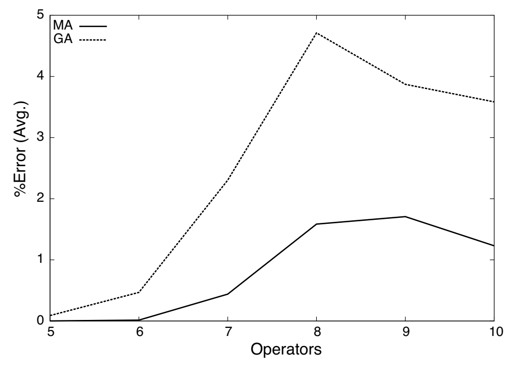
For the instances L1–L3, L4–L6 and L7–L9, BF finds optimal solutions in all cases taking 2.93, 7.33 and 16.60 s in average respectively. Regarding MA, Fig. 5(d), (e) and (f) show that

- for the L1–L3 instances, it yields solutions that have an average error of 1.4% in 1 s, and solutions with an average error lower than 1% in 2 s,
- for the L4–L6 instances, it returns solutions that have an average error close to 3% in 1 s, and solutions with an average error lower than 1% in 4 s,
- and for the L7–L9 instances, it yields solutions that have an average error between 4% and 4.5% in 1 s, and solutions with an average error lower than 1% in 8 s.

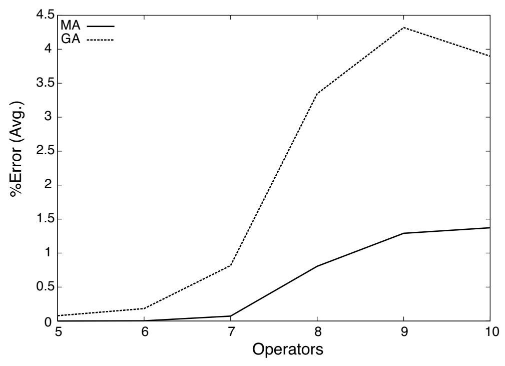
The instances of size 4×30 are studied separately by number of operators. BF reaches optimal solutions in all cases but one (with $p = 2$) in which it shows an average error of 0.03%. This algorithm takes, in average, 0.13, 32.20, 135.00 and 134.93 s for 1, 2, 3 and 4 operators respectively. For MA, Fig. 5(g) shows that

- for the instances with 1 operator, it returns optimal solutions from the start,
- for the instances with 2 operators, it returns solutions having less than 3% average error in 1s and solutions with an average error very close to 0% in 20 s,
- and for the instances with 3 and 4 operators, it yields solutions close to 1% average error in 10 s.

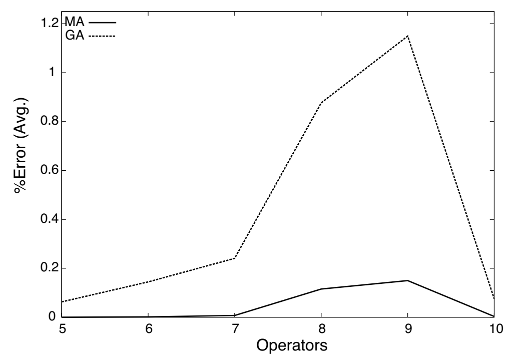
To complete the comparison among BF and MA, we have run both algorithms on the subset of instances in *B*2 derived from the LA21 JSP instance; these are in fact the easiest instances in this set and they can be expected to be hard to solve by BF. MA was given 120 s and BF was not given any time limit. In these experiments the average time taken by BF either to certify an optimal solution or to run out of memory and return lower and upper bounds was 2233 s. Table 5 summarizes the results from these experiments. For BF, we report the lower and upper bounds obtained and the time taken to reach the upper bound returned (which is usually shorter than the time for the memory getting exhausted); while for MA we show the



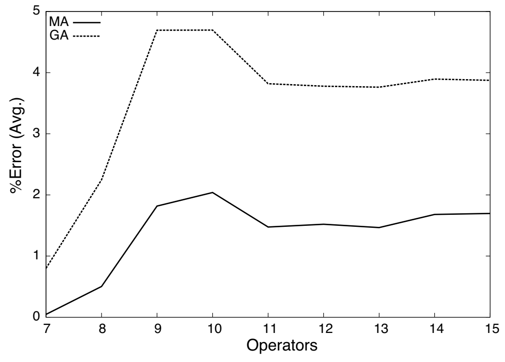
(a) LA: 15 × 10



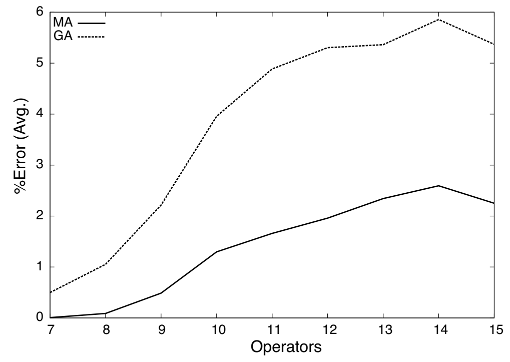
(b) LA: 20 × 10



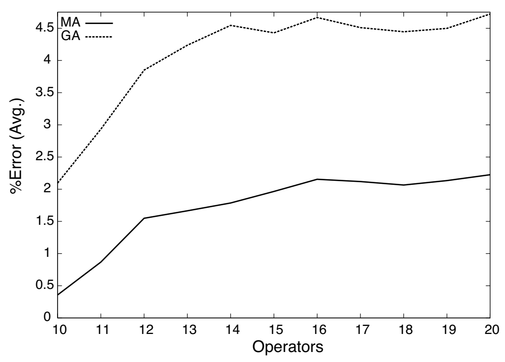
(c) LA: 30 × 10



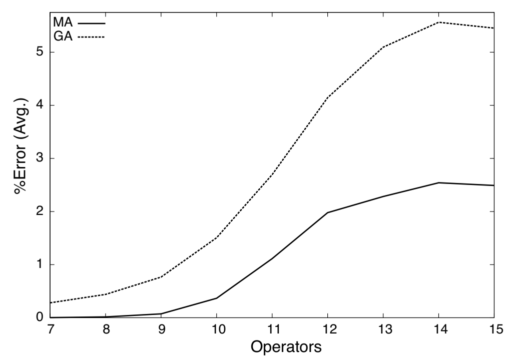
(d) LA: 15 × 15



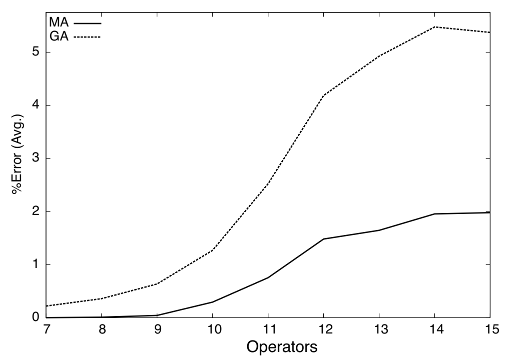
(e) Taillard: 20 × 15



(f) Taillard: 20 × 20

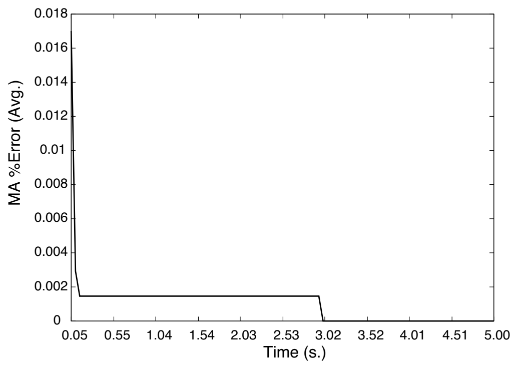


(g) Taillard: 30 × 15

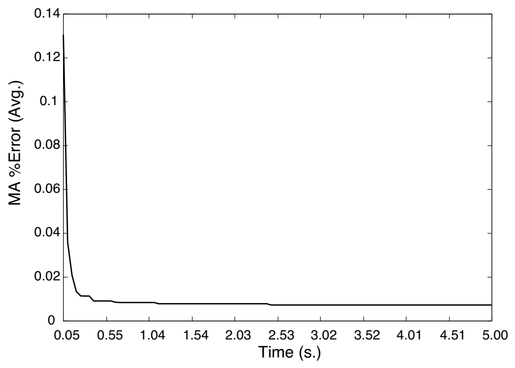


(h) Taillard: 30 × 15 (300s.)

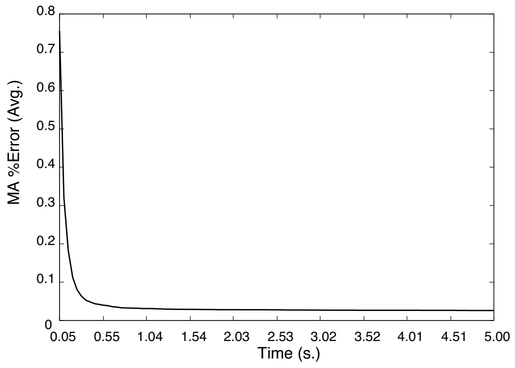
Fig. 4. Average error of GA and MA over the Lawrence and Taillard instances depending on the number of operators. Each plot represents the values averaged for all instances of the same size $n \times m$.



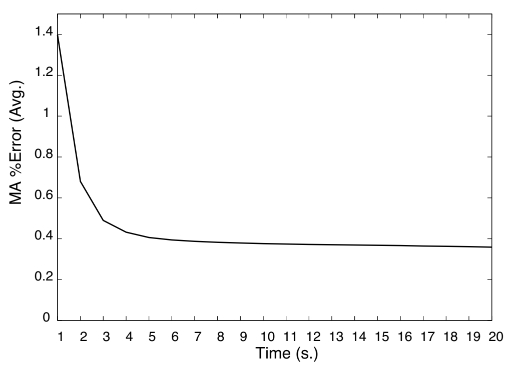
(a) SMALL: 1−9



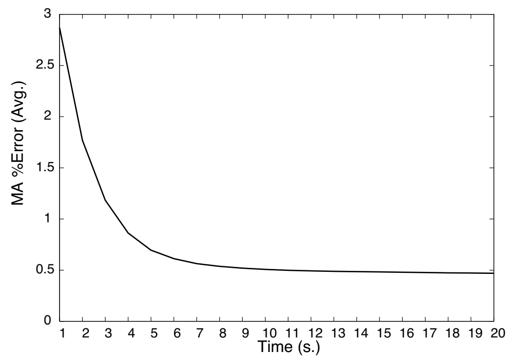
(b) SMALL: 10−18



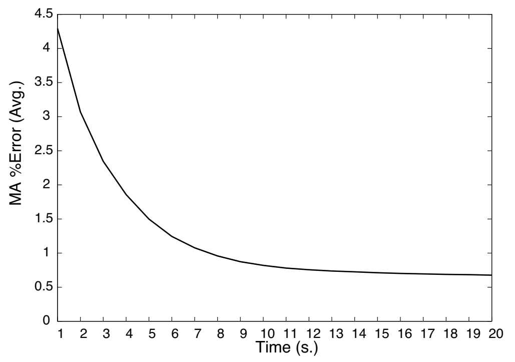
(c) SMALL: 19−27



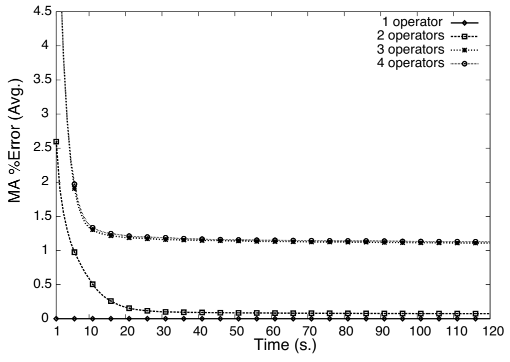
(d) LARGE: L1-L3



(e) LARGE: L4-L6



(f) LARGE: L7-L9



(g) L4J

Fig. 5. Evolution of the average error of MA across Agnetis's ($p = 2$) and L4J ($p = 1, 2, 3, 4$) sets of instances over time.

best and the average makespan reached over the 30 runs for each instance. As expected, the solutions obtained by BF are in general worse than those reached by MA, BF taking much more than 120 s in most cases. However, BF establishes tight lower bounds which in two cases allow for certifying the optimality of the solutions.

Even though it is difficult to draw sharp conclusions in a comparison between an exact and an approximate method, we could say that for small instances as those in $B2$, BF may be better than MA due to the fact that BF certifies the optimal solutions for almost every instance taking a reasonable time. However, MA reaches also very good solutions taking very low time. On the other hand, for large instances, as those in $B1$, MA is the best option as it can reach much better solutions in all cases taking much less time than BF. However, the results of BF are also interesting on these instances as they provide lower bounds which, in some cases, allow for establishing the optimality of a solution reached by MA.

9. Conclusions

We have seen that combining hybrid metaheuristics with problem specific neighborhood strategies make it possible to obtain an outstanding procedure to solve the job shop scheduling problem with operators proposed in [1]. We have defined and analyzed a neighborhood structure that was then used as the core of a memetic algorithm that combines a genetic algorithm and tabu search. We have conducted an experimental study over a number of instances of medium and large size. In this study, the proposed memetic algorithm is analyzed and proved to outperform or at least to be quite competitive with the state-of-the-art. The results of the formal and experimental studies carried out on the JSO provide us with useful insights that may help to devise new algorithms to this problem and to some of its variants as well.

Therefore, as future work, we will try to devise new neighborhood structures for the JSO. To this end, we will extend other structures proposed for the JSP, which rely on changing the processing order of more than two tasks in the critical path, and we will also consider changing the operator of critical tasks to devise a new class of neighborhood structures. Moreover, we plan to investigate new versions of this problem, for example the scheduling problem with arbitrary precedence relations proposed in [35] in which the operators have different skills, or the JSO with objective functions other than makespan minimization, for example total flow time minimization which was considered in [36] where the authors proposed a partially informed depth first search algorithm. In addition to memetic algorithms, it would be also interesting to consider other hybrid metaheuristics such as, for example, scatter search with path relinking [37], which has demonstrated to be good in solving other problems, for example the JSP with sequence dependent and non-anticipatory setup times [38].

Acknowledgments

This research has been supported by the Spanish Government and by the Principality of Asturias under research projects TIN2010-20976-C02-02 and FC-13-COF13-035 respectively. Carlos Mencía is supported by SFI grant BEACON (09/IN.1/I2618).

References

[1] A. Agnetis, M. Flamini, G. Nicosia, A. Pacifici, A job-shop problem with one additional resource type, J. Schedul. 14 (3) (2011) 225–237.
[2] R. Mencía, M.R. Sierra, C. Mencía, R. Varela, A genetic algorithm for job-shop scheduling with operators enhanced by weak lamarckian evolution and search space narrowing, Nat. Comp. 13 (2) (2014) 179–192.
[3] M.R. Sierra, C. Mencía, R. Varela, New schedule generation schemes for the job-shop problem with operators, J. Intell. Manuf. (2013) 1–12, http://dx.doi.org/10.1007/s10845-013-0810-6 (ahead-of-print).
[4] P. Moscato, On evolution, search, optimization, gas and martial arts: toward memetic algorithms, California Inst. Technol., Pasadena, CA, Tech. Rep. Caltech Concurrent Comput. Prog. Rep. 826, (1989).
[5] R. Dawkins, The Selfish Gene, Oxford Univ. Press, New York, 1976.
[6] E. Talbi, Metaheuristics: From Design to Implementation, Wiley, 2009.
[7] F. Neri, C. Cotta, Memetic algorithms and memetic computing optimization: a literature review, Swarm Evol. Comp. 2 (0) (2012) 1–14.
[8] N. Krasnogor, J. Smith, A tutorial for competent memetic algorithms: model, taxonomy, and design issues, IEEE Trans. Evol. Comp. 9 (5) (2005) 474–488.
[9] S. Meeran, M. Morshed, A hybrid genetic tabu search algorithm for solving job shop scheduling problems: a case study, J. Intell. Manuf. 23 (4) (2012) 1063–1078.
[10] P. Van Laarhoven, E. Aarts, K. Lenstra, Job shop scheduling by simulated annealing, Oper. Res. 40 (1992) 113–125.
[11] M. Dell' Amico, M. Trubian, Applying tabu search to the job-shop scheduling problem, Ann. Oper. Res. 41 (1993) 231–252.
[12] E. Nowicki, C. Smutnicki, An advanced tabu search algorithm for the job shop problem, J. Schedul. 8 (2005) 145–159.
[13] C.Y. Zhang, P. Li, Y. Rao, Z. Guan, A very fast TS/SA algorithm for the job shop scheduling problem, Comp. Oper. Res. 35 (2008) 282–294.
[14] D.C. Mattfeld, Evolutionary Search and the Job Shop Investigations on Genetic Algorithms for Production Scheduling, Springer-Verlag, 1995.
[15] A. Ponsich, C.A.C. Coello, A hybrid differential evolution tabu search algorithm for the solution of job-shop scheduling problems, Appl. Soft Comp. 13 (1) (2013) 462–474.
[16] C.R. Vela, R. Varela, M.A. González, Local search and genetic algorithm for the job shop scheduling problem with sequence dependent setup times, J. Heurist. 16 (2) (2010) 139–165.
[17] R. Sarker, M. Omar, S.K. Hasan, D. Essam, Hybrid evolutionary algorithm for job scheduling under machine maintenance, Appl. Soft Comp. 13 (3) (2013) 1440–1447.
[18] J. Li, Q. Pan, Chemical-reaction optimization for flexible job-shop scheduling problems with maintenance activity, Appl. Soft Comp. 12 (9) (2012) 2896–2912.
[19] J.J. Palacios, M.A. González, C.R. Vela, I. González-Rodríguez, J. Puente, Genetic tabu search for the fuzzy flexible job shop problem, Comp. Oper. Res. 54 (0) (2015) 74–89.
[20] D. Lei, Co-evolutionary genetic algorithm for fuzzy flexible job shop scheduling, Appl. Soft Comp. 12 (2012) 2237–2245.
[21] R. Mencía, M. Sierra, C. Mencía, R. Varela, Genetic algorithm for job-shop scheduling with operators, Lect. Notes Comp. Sci. 6687 (2) (2011) 305–314.
[22] B. Giffler, G.L. Thompson, Algorithms for solving production scheduling problems, Oper. Res. 8 (1960) 487–503.
[23] P. Brucker, B. Jurisch, B. Sievers, A branch and bound algorithm for the job-shop scheduling problem, Disc. Appl. Math. 49 (1994) 107–127.
[24] C. Mencía, M.R. Sierra, R. Varela, Depth-first heuristic search for the job shop scheduling problem, Ann. Oper. Res. 206 (1) (2013) 265–296.
[25] C. Mencía, M.R. Sierra, R. Varela, Intensified iterative deepening A* with application to job shop scheduling, J. Intell. Manuf. 25 (6) (2014) 1245–1255.
[26] C. Bierwirth, A generalized permutation approach to jobshop scheduling with genetic algorithms, OR Spectrum 17 (1995) 87–92.
[27] M.R. Sierra, R. Varela, Pruning by dominance in best-first search for the job shop scheduling problem with total flow time, J. Intell. Manuf. 21 (1) (2010) 111–119.
[28] C. Artigues, P. Lopez, P. Ayache, Schedule generation schemes for the job shop problem with sequence-dependent setup times: Dominance properties and computational analysis, Ann. Oper. Res. 138 (2005) 21–52.
[29] E. Nowicki, C. Smutnicki, A fast taboo search algorithm for the job shop scheduling problem, Manage. Sci. 42 (1996) 797–813.
[30] H. Matsuo, C. Suh, C. Sullivan, A controlled search simulated annealing method for the general jobshop scheduling problem, Graduate School of Business, University of Texas, 1988, Working paper 03-44-88.
[31] E. Taillard, Parallel taboo search techniques for the job shop scheduling problem, ORSA J. Comp. 6 (1993) 108–117.
[32] J.E. Beasley, Or-library: Distributing test problems by electronic mail, J. Oper. Res. Soc. 41 (11) (1990) 1069–1072.
[33] E. Taillard, Benchmarks for basic scheduling problems, Eur. J. Oper. Res. 64 (2) (1993) 278–285.
[34] S. García, A. Fernández, J. Luengo, F. Herrera, Advanced nonparametric tests for multiple comparisons in the design of experiments in computational intelligence and data mining: Experimental analysis of power, Inform. Sci. 180 (2010) 2044–2064.
[35] A. Agnetis, G. Murgia, S. Sbrilli, A job shop scheduling problem with human operators in handicraft production, Int. J. Prod. Res. 52 (13) (2014) 3820–3831.
[36] C. Mencía, M.R. Sierra, M.A. Salido, J. Escamilla, R. Varela, Solving the job shop scheduling problem with operators by depth-first heuristic search enhanced with global pruning rules, AI Commun. 28 (2015) 365–381.
[37] F. Glover, M. Laguna, R. Marti, Scatter search and path relinking: advances and applications, in: F. Glover, G. Kochenberger (Eds.), Handbook of Metaheuristics, Vol. 57 of International Series in Operations Research and Management Science, Springer, US, 2003, pp. 1–35.
[38] M.A. González, C.R. Vela, R. Varela, I. González-Rodríguez, An advanced scatter search algorithm for solving job shops with sequence dependent and non-anticipatory setups, AI Commun. 28 (2015) 179–193.

5.3. Schedule Generation Schemes and Genetic Algorithm for the Scheduling Problem with Skilled Operators and Arbitrary Precedence Relations

Se presenta la siguiente publicación:

- Raúl Mencía, María R. Sierra, Carlos Mencía, Ramiro Varela. Schedule Generation Schemes and Genetic Algorithm for the Scheduling Problem with Skilled Operators and Arbitrary Precedence Relations. *Twenty-Fifth International Conference on Automated Planning and Scheduling. (ICAPS 2015).* 165-173.
    - Estado: **Publicado.**

Proceedings of the Twenty-Fifth International Conference on Automated Planning and Scheduling

Schedule Generation Schemes and Genetic Algorithm for the Scheduling Problem with Skilled Operators and Arbitrary Precedence Relations

Raúl Mencía[1] and **María R. Sierra**[1] and **Carlos Mencía**[2] and **Ramiro Varela**[1]
[1]Department of Computer Science,
University of Oviedo, 33204 Gijón (Spain)
e-mail: raul89@gmail.com, {sierramaria, ramiro}@uniovi.es
[2]CASL, University College Dublin, Ireland
e-mail: carlos.mencia@ucd.ie

Abstract

In real-life production environments it is often the case that the processing of a task on a given machine requires the assistance of a human operator specially skilled to process that task. In this paper, we tackle a scheduling problem involving operators that are skilled to manage only subsets of the whole set of tasks in a given shop floor. This problem was recently proposed motivated by a handicraft company. In order to solve it, we make some contributions. We first propose a general schedule builder and particularize it to generate several complete solution spaces. This schedule builder is then exploited by a genetic algorithm that incorporates a number of problem-specific components, including a coding schema as well as crossover and mutation genetic operators. An experimental study shows substantial improvements over existing methods in the literature and reveals useful insights of practical interest.

Introduction

The scheduling problem with arbitrary precedence relations and skilled operators, defined in (Agnetis, Murgia, and Sbrilli 2014) and denoted JSSO, is a generalization of the job shop scheduling problem with operators (JSO) defined in (Agnetis et al. 2011), which in turn generalizes the classic job shop scheduling problem (JSP). In the JSP each task belongs to one and only one job, and each job defines a linear ordering for the processing of its tasks. The JSO extends the JSP in such a way that each task must be assisted by a human operator, the number of them being lower than both the number of jobs and the number of machines. In the JSO all the operators are equally skilled to assist any task, what may be unrealistic in many production environments.

The JSSO is motivated by a real-life handicraft company where the operators have different expertise. For example, an apprentice may be able to perform only a limited subset of single assembly tasks while the most difficult operations may require more experienced workers. Therefore, finding good solutions for the JSSO is an issue of major interest for human resources management as it would allow the company to make the best use of its employees and to plan their training for future projects.

To solve the JSSO, two heuristic algorithms were proposed in (Agnetis, Murgia, and Sbrilli 2014), which were evaluated across a benchmark set defined in accordance with the characteristics of the problems of the handicraft company. These algorithms are termed STA-OMSB and MSB-DOS respectively, and both of them exploit the well-known shifting bottleneck heuristic (Adams, Balas, and Zawack 1988). The results of these algorithms were compared with those obtained by Cplex MILP solver from a formulation proposed in the same paper.

In this paper we make several contributions towards understanding and solving the JSSO problem. Firstly, we formalize it using a disjunctive graph model, which makes it easier to reason about the problem. Then, we explore different ways of obtaining reduced dominant solution spaces. Concretely, we propose a general schedule builder, termed $SOG\&T$, and particularize it in three different ways. The hardness of the JSSO motivates the use of approximate algorithms. Accordingly, we propose a genetic algorithm that exploits the $SOG\&T$ and incorporates a novel problem-dependent coding scheme. The results from an experimental study indicate that our approach is very effective and outperforms previous methods in the state of the art for this problem.

The remaining of the paper is organized as follows. In the next two sections, we give a formal definition of the JSSO as a constraint satisfaction problem with optimization and propose a disjunctive graph model to represent problem instances and schedules. Then, we describe and analyze the proposed schedule generation scheme termed $SOG\&T$. After that, we introduce the genetic algorithm devised to solve the JSSO and summarize the results of an experimental study where the genetic algorithm is evaluated and compared with the state of the art. The paper finishes with some conclusions and ideas for future work.

Problem Formulation

In the scheduling problem with skilled operators and arbitrary precedence relations, we are given a set $\mathcal{M}$ of q machines, a set $\mathcal{O}$ of p operators and a set $\mathcal{T}$ of n tasks or operations. The task u requires two resources during its processing time p_u: a particular machine $m_u \in \mathcal{M}$ and one of the operators $o \in \mathcal{O}_u \subseteq \mathcal{O}$, where $\mathcal{O}_u$ denotes the subset of operators skilled to assist the task u. We assume that the processing time p_u is independent of the assisting operator.

There are arbitrary precedence relations among tasks specified by a $task\ graph$ $(\mathcal{T}, E)$ where nodes correspond to tasks and an arc $(u, v) \in E$ means that task u must be performed before task v starts.

The objective is to allocate a starting time st_u and an operator o_u to each task $u \in \mathcal{T}$, such that the makespan is minimized and the following constraints are satisfied:

i. The tasks must be processed following the order expressed by the task graph; i.e., $st_u + p_u \leq st_v$ if $(u, v) \in E$.
ii. Each task u must be assisted by a skilled operator to do it; i.e., $o_u \in \mathcal{O}_u$, and the operator is allocated to the task over its whole processing time.
iii. Two tasks assisted by the same operator or processed on the same machine cannot overlap; i.e., if $(o_u = o_v) \vee (m_u = m_v)$ then $(st_u + p_u \leq st_v) \vee (st_v + p_v \leq st_u)$.
iv. The tasks cannot be preempted; i.e., $C_u = st_u + p_u$, where C_u denotes the completion time of task u.

This problem was firstly defined in (Agnetis, Murgia, and Sbrilli 2014) and denoted JSSO. In some cases, the task graph has a tree structure with the root corresponding to the final assembly task. However, the topology of the task graph may be different and, for example, it may represent a number of independent sequences of operations which are then termed jobs. The JSSO problem is a generalization of the JSP and JSO, and can be seen as a particular case of the Multi-mode Resource Constrained Project Scheduling Problem (MRCPSP).

A Disjunctive Graph Model

Scheduling problems are usually represented by means of a disjunctive model (Roy and Sussman 1964). This kind of modeling allows for solving the problem by deciding about the relative order among operations that share the same resources, instead of considering for every operation all its possible starting times. We propose here to use the following model for the JSSO which is an extension to that used in (Sierra, Mencía, and Varela 2013) for the JSO where all operators can assist any operation and the precedence relations define a number of jobs.

A problem instance is represented by a directed graph $G = (V, E \cup D \cup I \cup O)$ where:

- Each node in V represents either a task in $\mathcal{T}$, or one of the fictitious tasks with null processing time; namely, starting tasks for each operator $o \in \mathcal{O}$, and the the dummy operations *start* and *end*.
- E is the set of arcs of the task graph also called *conjunctive arcs*. $P(v)$ and $S(v)$ will denote the sets of tasks which are predecessors and successors of v respectively in the task graph.
- D is the set of *disjunctive arcs* which represent capacity constraints of the machines. D is partitioned into subsets D_j with $D = \cup_{j=1,\dots,q} D_j$. D_j includes an arc (v, w) for each pair of operations requiring the machine j.
- O is the set of *operator arcs* and includes two types of arcs: one arc (u, v) for each pair of operations of the problem such that $\mathcal{O}_u \cap \mathcal{O}_v \neq \emptyset$, and arcs (o, u) for each operator node o and task u such that $o \in \mathcal{O}_u$.
- The set I includes arcs connecting node *start* to each node $o \in O$ and arcs connecting each task without successors in the task graph to the node *end*.

Arcs are weighted with the processing time of the task at the outgoing node.

From this representation, building a schedule may be viewed as a process of fixing disjunctive and operator arcs. A disjunctive arc between operations u and v gets fixed when either (u, v) or (v, u) is selected and so the other one discarded. If the operator arc (o, u) is fixed, the task u is assisted by o, and consequently all arcs (o', u) for o' other than o are discarded. Also, if the operator arc (u, v) is fixed then u and v are assisted by the same operator, u before v. In this case, the operator arc (v, u) and the remaining operator arcs connecting u or v to operations assisted by operators other than o are discarded. So, discarding both arcs (u, v) and (v, u) means that u and v will be assisted by different operators.

A feasible schedule S is represented by an acyclic subgraph of G, of the form $G_S = (V, E \cup F \cup I \cup Q)$, where $F \subset D$ expresses the processing order of tasks on the machines and $Q \subset O$ expresses the sequences of tasks that are assisted by each operator. In other words:

- $F = \cup_{j=1,\dots,q} F_j$, $F_j \subset D_j$ such that $(u, v) \in F_j$ iff $m_u = m_v$ and u is processed before v in S. So, F_j represents the machine sequence or processing order of tasks in the machine j.
- $Q = \cup_{o=1,\dots,p} Q_o$, where Q_o represents the operator sequence of o, and includes the arcs (o, u), (o, v) and (u, v) for each pair of operations assisted by the operator o such that u is processed before v. Each operation u must be included in one and only one operator sequence.

A *critical path* is a longest cost path in G_S from node *start* to node *end*. The *head* r_v of an operation v is the cost of the longest path from node *start* to node v and defines a lower bound for st_v. For most regular objective functions, in particular for the makespan, taking $st_v = r_v$ produces the optimal value restricted to the processing ordering defined by the solution graph. In this case, the cost of a critical path is the makespan of the schedule S denoted $C_{max}(S)$.

Figure 1 shows a solution graph for a problem instance with 7 tasks, 3 machines and 2 operators.

A partial schedule is given by a subgraph of G where some of the disjunctive and operator arcs have not fixed yet. In such a schedule $PM(v)$ denotes the disjunctive predecessors of v; $w \in PM(v)$ means that $m_w = m_v$ and that the disjunctive arc (w, v) was fixed (analogously, $SM(v)$ denotes the disjunctive successors of v). $PO(v)$ denotes the operator predecessors of v, i.e., $w \in PO(v)$ if the operator arc (w, v) was fixed (analogously, $SO(v)$ are the operator successors of v).

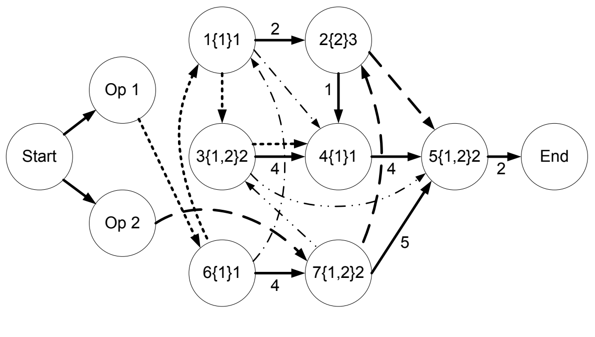


Figure 1: A solution graph for a problem instance with 7 tasks, 3 machines and 2 operators. Each task node includes the task number, the set of skilled operators and the machine using the notation $u\mathcal{O}_u m_u$. Dotted and dashed arcs represent operator sequences starting in an operator node, while dotted-dashed arcs represent machine sequences. For the sake of clarity, only the arcs between consecutive nodes in the relations are represented and only the cost of the arcs in the task graph are displayed. Every arc should be labeled with the processing time of the task at the outgoing node. The makespan is 19 and corresponds to the cost of the critical path (*start* Op1 6 7 3 4 5 *end*).

Schedule Generation Schemes

In this section we propose a new schedule generation scheme for the JSSO which is a generalization of the well-known $G\&T$ algorithm proposed in (Giffler and Thompson 1960) for the classic JSP and also of the $OG\&T$ algorithm proposed in (Sierra, Mencía, and Varela 2013) for the JSO. The new schedule builder is denoted $SOG\&T$ (Skilled Operators $G\&T$).

The $SOG\&T$ Algorithm

This algorithm iterates over n steps and in each iteration one of the tasks is scheduled following an order compatible with the partial ordering defined by the task graph. So, by the time the task u is scheduled, all tasks $v \in P(u)$ have already been scheduled. At this time, u is allocated an operator $o_u = o \in \mathcal{O}_u$ and a starting time st_u such that it is scheduled after all the scheduled tasks that require the same machine m_u or that were allocated the same operator o_u, i.e., it is an *appending* scheme.

Let SC be the set of scheduled operations before the current iteration and let $G_{SC} = (V, E \cup H \cup I \cup R)$ be the partial solution graph built so far. For all $u, v \in SC$ such that $m_u = m_v$, either (u, v) or (v, u) are in H. Also, for all $u \in SC$, (o, u) is in R for some $o \in \mathcal{O}$, and if (o, v) is also in R then either (u, v) or (v, u) are in R as well.

The *set of eligible operations* in the current iteration is defined as:

$$A = \{v; v \notin SC, P(v) \subseteq SC\} \quad (1)$$

A includes the first unscheduled tasks in the task graph. In principle, each task in A is a candidate to be scheduled next and it can be assisted by any of the operators skilled to do it. So, the *set of scheduling options* is defined as:

$$\mathcal{A} = \{(u, o); u \in A, o \in \mathcal{O}_u\} \quad (2)$$

As the set $\mathcal{A}$ may be very large, we will only consider options in a subset $\mathcal{X} \subseteq \mathcal{A}$. If the option $(u, o) \in \mathcal{X}$ is selected for the next scheduling step, then the lowest starting time of u is given by

$$st_u(o) = \max\{st_x + p_x, st_y + p_y, st_z + p_z\} \quad (3)$$

where

- x is the task in $P(u)$ with the largest completion time; i.e.,

$$x = argmax\{st_v + p_v; v \in P(u)\}, \quad (4)$$

- y is the last scheduled task in the machine required by u and
- z is the last scheduled task assisted by the operator o.

The option (u, o) establishes the time interval $[st_u(o), st_u(o) + p_u)$ for processing u under the assistance of the operator o.

Algorithm 1 shows the general structure of the schedule builder $SOG\&T$. It starts from a set A containing the tasks with no predecessors in the task graph and iterates along n steps. In each iteration an option (u, o) is taken non deterministically from a subset $\mathcal{X}$ of the scheduling options. The task u is scheduled at the time $st_u(o)$ given by exp. (3) and assisted by the operator o. Then the partial schedule built so far, G_{SC}, and the set A are updated accordingly for the next iteration. At last, after n iterations, G_{SC} represents a complete schedule.

Algorithm 1: Schedule builder $SOG\&T$. It builds a feasible schedule in n steps.

```
Data: A JSSO problem instance $\mathcal{P}$
Result: A feasible schedule for $\mathcal{P}$
$A = \{u \in \mathcal{T}; \neg\exists(w, u) \in E\}$;
for i=1 to n do
    $\mathcal{A} = \{(u, o); u \in A, o \in \mathcal{O}_u\}$;
    $\mathcal{X}$ = a subset of $\mathcal{A}$;
    choose $(u, o) \in \mathcal{X}$ non deterministically;
    set $st_u = st_u(o)$ and $o_u = o$;
    add $u$ to $SC$ and update $G_{SC}$;
    $A = \{v; v \notin SC, P(v) \subseteq SC\}$;
end
return the built schedule $G_{SC}$;
```

It is clear that any schedule built by Algorithm 1 is feasible. Also, depending on the subset of options considered, the algorithm will generate schedules in different search spaces. We will only consider here dominant search spaces; i.e., spaces containing at least one optimal schedule under the objective function considered, in our case the makespan.

Search spaces

The simplest way to get a dominant search space is taking $\mathcal{X} = \mathcal{A}$ in Algorithm 1. In spite of being dominant, as it is proved below, the search space generated by Algorithm 1 considering all options in $\mathcal{A}$ at each iteration may be very large, even for small instances, as we have mentioned. Fortunately, it is possible to further reduce the search space without loss of dominance.

Let us now consider the option (v^*, o^*) with the earliest completion time among the options in $\mathcal{A}$; i.e.

$$(v^*, o^*) = \arg\min\{st_u(o) + p_u; (u, o) \in \mathcal{A}\} \tag{5}$$

Let $C^* = st_{v^*}(o^*) + p_{v^*}$. We define the sets of options $\mathcal{A}'$ and $\mathcal{B}$ as follows.

$$\mathcal{A}' = \{(u, o) \in \mathcal{A}; st_u(o) < C^*\} \tag{6}$$

$$\mathcal{B} = \{(u, o) \in \mathcal{A}'; (m_u = m_{v^*} \vee o = o^*)\} \tag{7}$$

The set $\mathcal{A}'$ reduces the scheduling options to the options in $\mathcal{A}$ having a starting time lower than C^*. So, each option in $\mathcal{A}'$ establishes a starting time for a task in the interval $[T_{\mathcal{A}'}, C^*)$ where

$$T_{\mathcal{A}'} = \arg\min\{st_u(o); (u, o) \in \mathcal{A}\}. \tag{8}$$

Then, $\mathcal{B}$ further restricts the number of options by filtering those involving a machine other than m_{v^*} and an operator other than o^* at the same time. So, the set $\mathcal{B}$ contains the options for tasks in A that would require either the machine m_{v^*} or the operator o^* at some time in the interval $[T_{\mathcal{B}}, C^*)$ if they were chosen in the current iteration, where

$$T_{\mathcal{B}} = \arg\min\{st_u(o); (u, o) \in \mathcal{B}\}, \tag{9}$$

being $T_{\mathcal{B}} \geq T_{\mathcal{A}'}$ as the option that establishes the value of $T_{\mathcal{A}'}$ in expression (8) may not be included in $\mathcal{B}$.

The following result establishes that taking $\mathcal{X} = \mathcal{B}$ makes the search space dominant.

Proposition 1. *In at least one of the best schedules that can be eventually reached from the current iteration, one of the operations in A is scheduled in accordance with and option in $\mathcal{B}$.*

Proof. Let S be one schedule that can be eventually built from the current iteration such that none of the tasks in A is scheduled in accordance with an option in $\mathcal{B}$. In S, m_{v^*} and o^* are idle over the interval $[st_{v^*}(o^*), C^*)$. Then v^* could be rescheduled in accordance with the option (v^*, o^*); i.e., starting at $st_{v^*}(o^*)$ and assisted by o^*. As none of the remaining operations has to be delayed and $st_{v^*}(o^*) \leq st_{v^*}$, then $C_{max}(S') \leq C_{max}(S)$. So, if S is one of the best schedules, then S' is also one of the best schedules. □

Corollary 1. *There is a sequence of non deterministic choices of options, each one from the set $\mathcal{B}$ calculated in each iteration, that allows Algorithm 1 to reach an optimal schedule.*

Proof. It follows trivially from Proposition 1 considering the initial iteration in which none of the tasks is scheduled. □

So, due to the fact that $\mathcal{B} \subseteq \mathcal{A}' \subseteq \mathcal{A}$, taking $\mathcal{X} = \mathcal{A}'$ or $\mathcal{X} = \mathcal{A}$ makes the search space dominant as well.

Remark 1. *Regarding the sets $\mathcal{A}'$ and $\mathcal{B}$ is important to make the following observation. As we have mentioned, if in a given iteration the chosen option (u, o) is selected from $\mathcal{A}'$, then $st_u \in [T_{\mathcal{A}'}, C^*)$, while if it is selected from $\mathcal{B}$, then $st_u \in [T_{\mathcal{B}}, C^*)$, with $T_{\mathcal{A}'} \leq T_{\mathcal{B}}$. This fact may have consequences on the idle times of the machines and operators and so on the average makespan of the schedules. So, as a consequence of the potentially larger mean idle times of the schedules generated from $\mathcal{B}$, due to the difference $T_{\mathcal{B}} - T_{\mathcal{A}'}$, the average of these schedules may be larger than the average makespan of the schedules generated from $\mathcal{A}'$.*

Also, if the option (u, o) is selected from $\mathcal{A}$, then $st_u \in [T_{\mathcal{A}'}, C)$, where C is expected to be larger than C^*, so the average makespan of the schedules generated from $\mathcal{A}$ is expected to be larger than that of the schedules generated from $\mathcal{A}'$ or $\mathcal{B}$ as $C - C^*$ is likely greater than $T_{\mathcal{B}} - T_{\mathcal{A}'}$.

Summary of Search Spaces and Schedule Builders

In this section we have introduced the $SOG\&T$ as a generic schedule builder and we have seen how it may be adapted to search in three different dominant search spaces characterized by the sets of options $\mathcal{A}$, $\mathcal{A}'$ and $\mathcal{B}$. Abusing the language we will denote these spaces $\mathcal{A}$, $\mathcal{A}'$ and $\mathcal{B}$ respectively.

Genetic Algorithm for the JSSO

Genetic algorithms have been successfully applied to scheduling problems such as JSP (Mattfeld 1995; Bierwirth 1995) or JSO (Mencía et al. 2014). In these cases, the problems were defined by a set of jobs, what allowed the GAs to use the efficient encoding based on permutations repetition (Bierwirth 1995). In the JSSO, as it was defined here, this encoding cannot be used due to the arbitrary precedence relations. So, in principle, we opted to use a more conventional coding schema as single permutations of tasks.

At the same time, in JSP and JSO no information about operators needs to be included in the chromosome; in the first case no operators exist, while in the second all of them are skilled to assist any operation and so they may be in fact considered as a cumulative resource of capacity p. However, in the JSSO, it is reasonable to express operator preferences for the tasks in the chromosomes in order to get an appropriate coding schema.

In the following subsections, we detail the main components of the proposed GA for the JSSO; namely, the coding schema, the decoding algorithm, the genetic operators and the general structure of the GA used.

Coding Schema

We propose here a coding schema for the JSSO where a chromosome consists of two permutations of symbols. The first one is the *task sequence* which is a conventional permutation of the numbers $1 \dots n$, while the second is the *operator sequence* and is given by a permutation with repetition of the symbols $1 \dots p$ of size n and represents operator preferences. Notice that some operators could appear several times in the chromosome and also some operator may be absent.

For example the following two permutations represent a feasible chromosome for the instance considered in Figure 1.

$$\begin{array}{c}(6,7,5,4,1,2,3)\\(1,2,2,2,1,1,1)\end{array} \tag{10}$$

This encoding should be understood in such a way that if task u appears before task v in the first permutation, then u should be preferably scheduled before v. At the same time, the second permutation represents priorities for the operators to be allocated to tasks. In order to avoid that all tasks see the same order of operators, the first operator for a task will be that in the same position as the task in the first permutation and then the remaining ones are taken following the chromosome as a circular structure. An important observation is that the task ordering and the operators' allocation in the schedule does not only depend on the chromosome, but also on the decoding algorithm, as we will see in the next section.

One of the most interesting properties of this encoding is that any solution can be encoded into a chromosome representing the same machine orderings and operator allocations. At the same time, it is simple and so it allows for designing efficient genetic operators for crossover and mutation. In principle, the only inconvenience comes from the fact that the number of replicas of a symbol in the operator sequence is variable and so an operator could disappear from the chromosome. However, this problem can be easily solved by means of some heuristic repairing as we will see.

Decoding Algorithm

Decoding algorithms map chromosomes to feasible solutions. To this aim, we use the $SOG\&T$ algorithm presented above, exploiting the information encoded in the chromosomes to guide the search. This algorithm is issued and in each iteration the option $(u, o) \in \mathcal{X}$ that is the "leftmost" in the chromosome, i.e., that fulfills the following two conditions, is chosen

- u is the leftmost in the task sequence of the chromosome among all tasks appearing in some option in $\mathcal{X}$, and
- o is the first operator skilled to assist u in the preference list defined by the operator sequence, among the operators appearing in some option in $\mathcal{X}$ for the operation u. If the operator of none of the options for u in $\mathcal{X}$ is present in the operator sequence, then the operator o with $(u, o) \in \mathcal{X}$ that allows u to start earliest is selected.

In the experimental study we will consider all the possibilities above for the set of options $\mathcal{X}$; namely, $\mathcal{A}$, $\mathcal{A}'$ and $\mathcal{B}$. In the three cases, the genetic algorithm searches over dominant search spaces and so it has the chance to reach an optimal schedule.

We illustrate how the decoding algorithm works by means of an example. Let us consider the chromosome in (10). It contains the tentative orderings (6,4,1) and (7,5,3) for machines 1 and 2 respectively, which are inconsistent due to the fact that the task 4 must be processed after the task 1, and the task 5 must be processed after the tasks 7 and 3, in accordance with the precedences expressed by the task graph. In spite of that, the chromosome is feasible as the decoding algorithm will build a feasible schedule from it in which some of the tentative orderings and operator preferences will hold, while others will not. If we consider the set $\mathcal{X} = \mathcal{A}$, in the first iteration the scheduling options would be $\{(1,1),(3,1),(3,2),(6,1)\}$; in all four cases the starting time would be 0. So, according to the chromosome, the task scheduled in this step would be 6, as it is the leftmost one in the chromosome among 1, 3, and 6, and the operator assigned would be 1 (as this is the only option). In the next iteration, the options would be $\{(3,2)\}$ with starting time 0 and $\{(1,1),(3,1),(7,1),(7,2)\}$ with starting time 4. In accordance with the chromosome, the chosen option would be (7,2). In the third iteration the options would be $\{(3,1),(3,2)\}$ with starting time 9 and $\{(1,1)\}$ with starting time 4, and the chosen option would be (1,1). This way, the tentative ordering (4,1) in the chromosome is not kept in the schedule. Finally, we will have the schedule of Fig. 1. It is important to remark that with $\mathcal{X} = \mathcal{A}'$ or $\mathcal{X} = \mathcal{B}$, we could reach different schedules from the same chromosome.

Genetic Operators

We build on conventional two point crossover and single mutation operators, and start with an initial population of chromosomes generated at random following a uniform distribution. Also, in order to better translate characteristics from parents to offsprings, we propose to code back the structure of the schedule into the chromosome. This operation is expected to produce small changes in the chromosome and so we refer to it as weak Lamarckian evolution.

We will use a double-chromosome variant of the classic order crossover (OX) which translates the subsequence of symbols between two cutting points from one parent to the offspring and the relative order of the remaining values from the second. OX may be a good option for the JSSO due to the fact that relevant characteristics, as processing order of tasks on the machines or priorities of operators for assisting the tasks, may be transmitted from parents to offsprings. To avoid a strong disruptive effect of the crossover, the cutting points will be the same in both sequences (tasks and operators).

Also, we will consider single mutation (SM) by swapping two consecutive positions at random. This seems to be appropriate as it may produce small changes. As before, the same positions will be swapped in both sequences. Moreover, to get different distributions of operators in the operator sequences, we will also use an operator mutation (OM) which will change the value in a location of the operator sequence at random. When a chromosome is mutated, one of SM or OM is chosen with probability 0.5.

Genetic Algorithm Structure

We use here a rather conventional genetic algorithm with generational replacement. In order to avoid premature convergence, we opted not to use the classic roulette wheel selection combined with unconditional replacement. Instead, in the selection phase all chromosomes are organized into pairs at random, then each pair undergoes crossover and mutation. After this, the offsprings are evaluated and the schedules coded back into the chromosomes as described in the

Algorithm 2: Genetic Algorithm.

```
Data: A JSSO problem instance $\mathcal{P}$ and a set of
      parameters $(P_c, P_m, \#gen, \#popsize, \mathcal{X})$
Result: A feasible schedule for $\mathcal{P}$
Generate and evaluate the initial population $P(0)$;
for t=1 to #gen-1 do
    Selection: organize the chromosomes in $P(t-1)$
    into pairs at random ;
    Recombination: mate each pair of chromosomes
    and mutate the two offsprings in accordance with
    $P_c$ and $P_m$;
    Evaluation: evaluate the resulting chromosomes
    considering the search space $\mathcal{X}$ and code back the
    schedules into the chromosomes;
    Replacement: make a tournament selection
    among every two parents and their offsprings to
    generate $P(t)$;
end
return the best schedule built so far;
```

section above. Finally, the new population is obtained by means of tournament selection keeping the best two individuals among every two parents and their two offsprings. The algorithm requires 5 parameters: crossover and mutation probabilities (P_c and P_m), number of generations ($\#gen$), population size ($\#popsize$) and the search space considered $\mathcal{X}$. Algorithm 2 shows the main steps of the genetic algorithm.

Experimental Study

To assess the performance of the proposed GA and compare it with the state of the art, we have conducted an experimental study. We considered two sets of instances[1]: firstly a set of instances derived from the well-known FT10 instance for the classic JSP with $n = 100$ tasks and $q = 10$ machines and, on the other hand, a set of instances used in (Agnetis, Murgia, and Sbrilli 2014).

The instances in the first set were generated considering different values for p (5, 7 and 9 operators) and different probabilities, Pr, that an operator can assist one task (0.2 and 0.6). Five instances were generated from each pair (Pr, p) and five more from each p taking Pr as 0.2 or 0.6 at random for each task. So we have 45 instances in all. These instances are denoted 2_5_1, 2_5_2, . . . , 2/6_5_1, etc.

The second set includes 50 instances organized in 5 groups with 10 instances each. Each group is defined by the values of n, p and q ranging in the sets $\{100, 150, 200\}$, $\{10, 15, 20\}$ and $\{15, 30, 50\}$ respectively. The processing times are uniformly distributed in $[1, 100]$. The topology of the task graph was inspired in real-life assembly trees and was distributed in a number of branches, between 3 and 6. The sets of operators skilled for each operation were generated at random and the workload of the machines was roughly balanced. We obtained detailed results from MSB-DOS and Cplex from a personal communication from the authors of (Agnetis, Murgia, and Sbrilli 2014). These results are summarized in Table 2.

[1]The instances and more details of the experimental study are available at http://www.di.uniovi.es/iscop (Repository).

Table 1: Summary of results from GA on the sets of instances derived from the FT10, considering the search spaces $\mathcal{A}'$ and $\mathcal{B}$. The results are averaged for each subset of 5 instances. Times are given in seconds.

Sets			$\mathcal{A}'$			$\mathcal{B}$		
#	Pr	p	Best	Avg.	Time	Best	Avg.	Time
1	2	5	1281.8	1309.3	8.7	1361.4	1397.9	7.2
2	2	7	1109.8	1142.4	8.6	1183.6	1223.5	6.9
3	2	9	1022.0	1051.5	9.5	1123.8	1151.7	7.5
4	6	5	1164.0	1187.6	14.0	1235.6	1265.0	10.9
5	6	7	1009.0	1030.1	16.7	1081.6	1110.7	12.6
6	6	9	972.2	995.6	19.4	1024.2	1046.8	14.5
7	2/6	5	1203.4	1235.7	11.5	1306.4	1349.2	9.1
8	2/6	7	1053.2	1078.0	13.5	1167.4	1196.8	10.0
9	2/6	9	985.6	1016.6	13.7	1111.4	1132.6	10.3
Average			1089.0	1116.3	12.9	1177.3	1208.2	9.9

Evaluation of the Genetic Algorithm

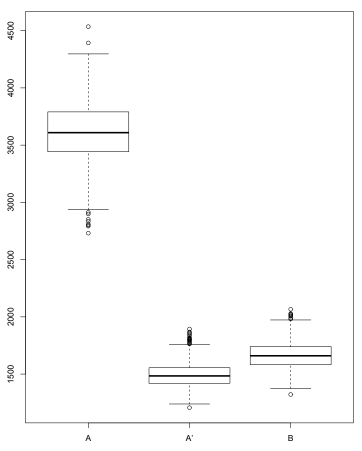
We evaluated different options in the proposed GA, in particular we considered different search spaces $\mathcal{A}$, $\mathcal{A}'$ and $\mathcal{B}$, and coding the schedule back into the chromosome or not. The evaluation was done on the instances in the first set. We have chosen a rather conventional parameter setting: $P_c = 1$[2], $P_m = 0.1$, $\#popsize = 100$, $\#gen = 1000$. The target machine was Intel Core i7-3770K 3.50 GHz. 12 GB RAM and the algorithm was coded in C++.

Firstly, we evaluated the average quality of the schedules in the three search spaces. To do this, we generated 1000 random solutions in each subset. Figure 2 shows the distribution of makespan outcomes for each set of schedules. As we can observe, schedules sampled from the space $\mathcal{A}$ are clearly much worse than the schedules from the other two spaces; and schedules from $\mathcal{A}'$ are better than those from $\mathcal{B}$. So, from these results, the space $\mathcal{A}'$ seems to be good for a successful evolution and convergence of the GA.

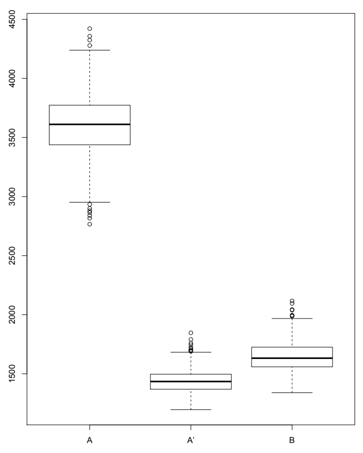
In order to visualize the evolution of the GA, we show in Figure 3 the convergence pattern of the GA for three instances considering the search spaces $\mathcal{A}'$ and $\mathcal{B}$ with and without the coding-back option. In view of these results, the coding-back option is good in all cases and decoding in $\mathcal{A}'$ is better than decoding in $\mathcal{B}$. We conducted the same experiments considering the set $\mathcal{A}$ and the results were much worse as it was expected from the first experiments.

To further assess the differences between decoding in the subsets $\mathcal{A}'$ and $\mathcal{B}$, we solved the 45 instances with the coding-back option. In this case, each instance was solved 10 times and the best and average of the solutions reached

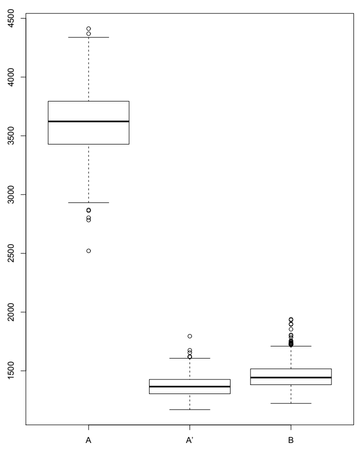
[2]Taking $P_c < 1$ makes that some pairs of chromosomes are not mated and in this case the offsprings are the same as their parents if they are not mutated, so the best parent is chosen twice in the replacement phase, what may contribute to premature convergence.



(a) Instance 2_9_1

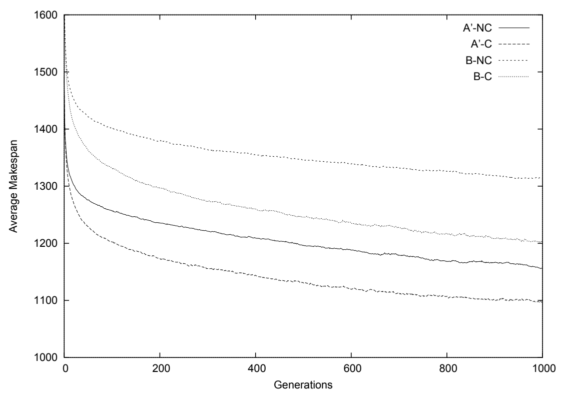


(b) Instance 2/6_9_1

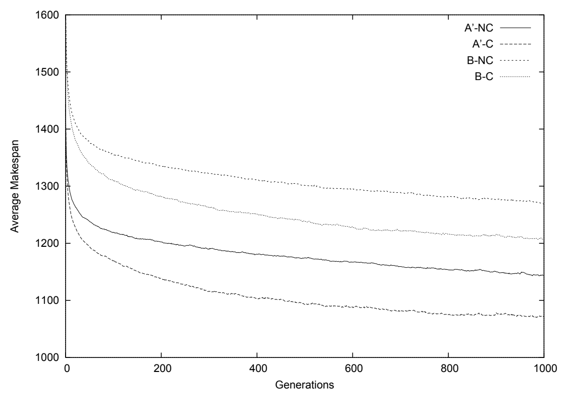


(c) Instance 6_9_1

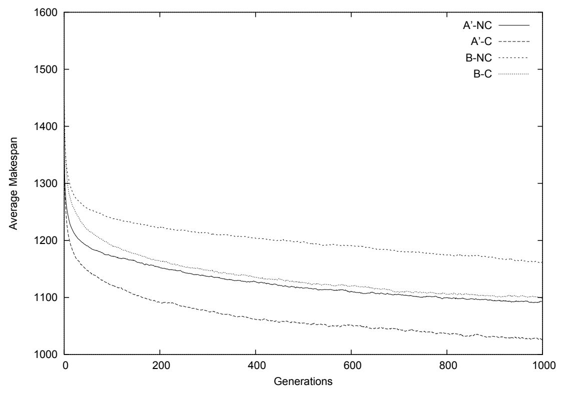
Figure 2: Summary of results from 1000 random schedules for the instances 2_9_1, 2/6_9_1 and 6_9_1 derived from the FT10 in the spaces $\mathcal{A}$, $\mathcal{A}'$ and $\mathcal{B}$.



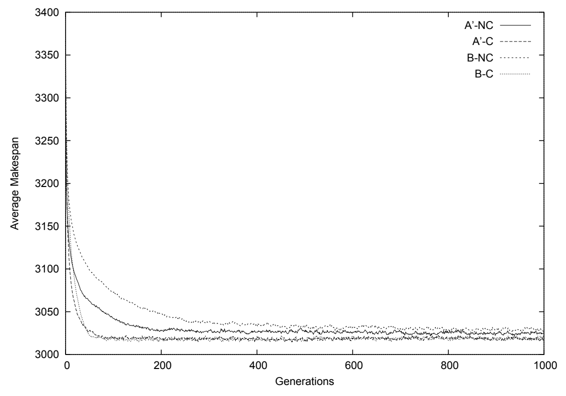
(a) Instance 2_9_1



(b) Instance 2/6_9_1



(c) Instance 6_9_1



(d) Instance $100/10/15\_1$

Figure 3: Convergence of the GA for the instances 2_9_1, 2/6_9_1 and 6_9_1 derived from the FT10 and the instance $100/10/15\_1$ from (Agnetis, Murgia, and Sbrilli 2014), combining the search spaces $\mathcal{A}'$ and $\mathcal{B}$ and the coding (C) and non-coding (NC) back options. Each plot represents the evolution of the mean makespan of the population averaged for 10 runs.

Table 2: Summary of results from MSB-DOS and Cplex averaged for the 10 instances in each of the 10 sets. $\#Opt.$ is the number of instances in each subset which are optimally solved. Time is given in seconds. For GA, the values reported are the average values of the 10 runs for each instance.

Sets				MSB-DOS			CPLEX		GA		
#	Tasks(n)	Operators(p)	Machines(q)	GAP%	Time	#Opt.	Time	#Op.	GAP%	Time	#Opt.
1	100	10	15	0.78	106.44	5.00	106.21	10.00	0.06	1.84	8.00
2	150	15	30	0.20	8.28	6.00	190.54	10.00	0.00	2.87	10.00
3	150	15	50	0.03	9.58	9.00	129.28	10.00	0.00	2.84	10.00
4	200	15	30	0.42	25.62	6.00	755.85	10.00	0.00	3.64	9.70
5	200	20	30	0.41	23.43	6.00	913.32	10.00	0.00	4.24	10.00
Average				0.37	34.67	6.40	419.04	10.00	0.01	3.09	9.54

in all runs were recorded. These results, averaged for each subset of 5 instances, together with the average time taken to solve one instance in each run, are summarized in Table 1. These results confirm that $\mathcal{A}'$ is the best choice as the makespan with $\mathcal{A}'$ is about 8.2% better than it is with $\mathcal{B}$. This is quite reasonable from Remark 1 above. As we can observe, in all cases the values from $\mathcal{A}'$ are better than those from $\mathcal{B}$; even the average values from $\mathcal{A}'$ are better than the best values from $\mathcal{B}$. Also, the time taken with $\mathcal{B}$ is about 23.25% lower than it is with $\mathcal{A}'$, what is natural as $\mathcal{A}'$ offers more options in each step of the schedule builder and so looking for the leftmost one in the chromosome takes more time. We have also registered the results from $\mathcal{A}'$ by the time taken with $\mathcal{B}$, and the difference is still significant, it only drops from 8.2% to 7.7% in favor of $\mathcal{A}'$.

Comparison with Other Methods

In (Agnetis, Murgia, and Sbrilli 2014) the authors proposed two heuristic methods, MSB-DOS and STA-OMSB and they also experimented with Cplex 12.2 on a MILP formulation they proposed. The authors report results from this study averaged for each subset of instances showing clearly that MSB-DOS is the best of the two heuristics proposed. Their target machine was AMD Athlon II 2.70 GHz using Matlab 2009b. Cplex was given a time limit of 3600 s. and was able to find optimal solutions for all the 50 instances.

Table 2 summarizes the results produced by MSB-DOS, Cplex and our GA. GA was parameterized as indicated above, with the only difference that the number of generations given was 250 instead of 1000. The reason for this is that GA needs less generations to converge due to the topology of the precedence graph. This fact can be observed in Figure 3(d), which represents the convergence patterns for the instance $100/10/15\_1$; after generation 250 the GA hardly converges any more with the four combinations. In this example, there are not significant differences between spaces $\mathcal{A}'$ and $\mathcal{B}$ when the coding-back option is considered.

As we can observe, Cplex can reach optimal solutions for all the instances with an average time of 419.04s. MSB-DOS takes much lower time in average, 34.67s, and obtains optimal solutions for 6.4 instances in average for each subset with an average gap of 0.37%. Regarding GA, it is able to obtain optimal solutions in the 10 runs for all instances in three sets, 2,3 and 5; and in average for 8 and 9.7 instances in the remaining two sets. Overall, it was able to obtain the optimal solution in 477 of the 500 runs and in at least one of the 10 runs for 49 of the 50 instances. The only instance not solved optimally at least once was instance 8 of the set 1 where the best makespan reached was 4537, being the optimum 4534 (GA was able to find an optimal solution in 1000 generations). The average gap in percent was 0.01 and the average time taken 3.09s indicating that, despite the differences on the target machines, GA is faster than the methods proposed in the literature, and finds much better solutions than any other approximate approach.

Conclusions and Future Work

We have seen that the well-known $G\&T$ schedule builder proposed in (Giffler and Thompson 1960) for the classic JSP problem may be extended to solve JSSO problem proposed in (Agnetis, Murgia, and Sbrilli 2014), in a similar way as it was previously extended to other problems such as the $EG\&T$ for the JSP with Sequence Dependent Setup times (Artigues, Lopez, and Ayache 2005), the $OG\&T$ for the JSO (Sierra, Mencía, and Varela 2013) or the $fG\&TSGS$ for the JSP with fuzzy processing times (González Rodríguez, Vela, and Puente 2007; Palacios et al. 2014).

Much in the same way as $G\&T$, $EG\&T$, $fG\&TSGS$ and $OG\&T$ were exploited in (Mattfeld 1995), (Vela, Varela, and González 2010), (González Rodríguez, Vela, and Puente 2007) and (Mencía et al. 2014) respectively, $SOG\&T$ has been exploited here to devise a decoder which is the core of the GA proposed to solve the JSSO. The success of this algorithm relies not only in the capability of the $SOG\&T$ to generate schedules in different search spaces, but also in the proposed coding schema that encodes candidate solutions by means of a conventional permutation of tasks and a permutation with repetition of operators.

As future work, we plan to design local search algorithms which will be combined with the GA. To this end, we will devise neighborhood structures for the JSSO from the proposed disjunctive model. In principle, we will try to extend the structures proposed in (Dell' Amico and Trubian 1993; Laarhoven, Aarts, and Lenstra 1992) and consider new structures aiming at exploring changes in the assignment of operators. Furthermore, we will tackle the JSSO in the framework of heuristic search using $SOG\&T$ as branching schema, in this case $\mathcal{B}$ will be likely the best option.

In this setting, the disjunctive graph model could help to devise consistent heuristics and dominance rules. We expect it to work well when using the proper search strategy as, for example, in (Mencía, Sierra, and Varela 2013) for the JSO. It will also be interesting to consider and evaluate constraint programming approaches on the JSSO, both general frameworks as CPLEX CP Optimizer (Laborie 2009), or more specific algorithms such as the successful Solution Guided Search (Beck, Feng, and Watson 2011).

Acknowledgments

This research has been supported by the Spanish Government under research projects TIN2010-20976-C02-02 and TIN2013-46511-C2-2-P, and by the Principality of Asturias under project FICYT2013 - COF13-035. Carlos Mencía is supported by SFI grant BEACON (09/IN.1/I2618). We are grateful to the anonymous reviewers for their insightful comments.

References

Adams, J.; Balas, E.; and Zawack, D. 1988. The shifting bottleneck procedure for job shop scheduling. *Management Science* 34(3):391–401.

Agnetis, A.; Flamini, M.; Nicosia, G.; and Pacifici, A. 2011. A job-shop problem with one additional resource type. *J. Scheduling* 14(3):225–237.

Agnetis, A.; Murgia, G.; and Sbrilli, S. 2014. A job shop scheduling problem with human operators in handicraft production. *International Journal of Production Research* 52(13):3820–3831.

Artigues, C.; Lopez, P.; and Ayache, P. 2005. Schedule generation schemes for the job shop problem with sequence-dependent setup times: Dominance properties and computational analysis. *Annals of Operations Research* 138:21–52.

Beck, J. C.; Feng, T. K.; and Watson, J. 2011. Combining constraint programming and local search for job-shop scheduling. *INFORMS Journal on Computing* 23(1):1–14.

Bierwirth, C. 1995. A generalized permutation approach to job shop scheduling with genetic algorithms. *OR Spectrum* 17:87–92.

Dell' Amico, M., and Trubian, M. 1993. Applying tabu search to the job-shop scheduling problem. *Annals of Operational Research* 41:231–252.

Giffler, B., and Thompson, G. L. 1960. Algorithms for solving production scheduling problems. *Operations Research* 8:487–503.

González Rodríguez, I.; Vela, C. R.; and Puente, J. 2007. A memetic approach to fuzzy job shop based on expectation model. In *Proceedings of IEEE International Conference on Fuzzy Systems, FUZZ-IEEE2007*, 692–697. London: IEEE.

Laarhoven, P. J. M. v.; Aarts, E. H. L.; and Lenstra, J. K. 1992. Job shop scheduling by simulated annealing. *Operations Research* 40(1):pp. 113–125.

Laborie, P. 2009. IBM CP Optimizer for detailed scheduling illustrated on three problems. In van Hoeve, W.-J., and Hooker, J., eds., *Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems*, volume 5547 of *Lecture Notes in Computer Science*. Springer Berlin Heidelberg. 148–162.

Mattfeld, D. C. 1995. *Evolutionary Search and the Job Shop Investigations on Genetic Algorithms for Production Scheduling*. Springer-Verlag.

Mencía, R.; Sierra, M. R.; Mencía, C.; and Varela, R. 2014. A genetic algorithm for job-shop scheduling with operators enhanced by weak lamarckian evolution and search space narrowing. *Natural Computing* 13(2):179–192.

Mencía, C.; Sierra, M. R.; and Varela, R. 2013. An efficient hybrid search algorithm for job shop scheduling with operators. *International Journal of Production Research* 51(17):5221–5237.

Palacios, J. J.; Vela, C. R.; Rodríguez, I. G.; and Puente, J. 2014. Schedule generation schemes for job shop problems with fuzziness. In *ECAI 2014 - 21st European Conference on Artificial Intelligence, 18-22 August 2014, Prague, Czech Republic*, 687–692.

Roy, B., and Sussman, B. 1964. Les problemes d'ordonnancements avec contraintes disjonctives. Notes DS no. 9 bis, SEMA, Paris.

Sierra, M.; Mencía, C.; and Varela, R. 2013. New schedule generation schemes for the job-shop problem with operators. *Journal of Intelligent Manufacturing* 1–15.

Vela, C. R.; Varela, R.; and González, M. A. 2010. Local search and genetic algorithm for the job shop scheduling problem with sequence dependent setup times. *Journal of Heuristics* 16(2):139–165.

5.4. Genetic Algorithm for the scheduling problem with arbitrary precedence relations and skilled operators

Se presenta la siguiente publicación:

- Raúl Mencía, María R. Sierra, Carlos Mencía, Ramiro Varela. Genetic Algorithm for the scheduling problem with arbitrary precedence relations and skilled operators. *Integrated Computer-Aided Engineering*. 23(3): 269-285. 2016. DOI: 10.3233/ICA-160519.
    - Estado: **Publicado.**

Integrated Computer-Aided Engineering 23 (2016) 269–285
DOI 10.3233/ICA-160519
IOS Press

Genetic algorithms for the scheduling problem with arbitrary precedence relations and skilled operators

Raúl Mencía[a], María R. Sierra[a], Carlos Mencía[b] and Ramiro Varela[a,*]
[a]*Department of Computer Science, University of Oviedo, Spain*
[b]*CASL, University College Dublin, Ireland*

Abstract. Scheduling problems involving physical machines and human resources are frequent in real production environments. In this paper, we tackle a problem in which a set of tasks must be performed on a set of machines under the assistance of human operators, subject to some constraints such as precedence relations on the tasks, limited capacity of machines and operators, and skills of the operators to assist the processing of tasks. We analyze the problem and propose a generic schedule builder that may be adapted to build schedules in different dominant and non-dominant search spaces. The schedule builder was exploited as a decoder in a genetic algorithm. All the proposals were evaluated on a benchmark set with instances of different characteristics. The experimental study revealed useful insights of practical interest and showed substantial improvements of the genetic algorithm over existing methods in the literature.

Keywords: Scheduling, genetic algorithms, heuristics, schedule generation schemes, constraint satisfaction and optimization

1. Introduction

Scheduling problems arise routinely in a growing number of domains, including engineering, management science or distributed and parallel computing, to mention just a few. The practical significance of scheduling, and the fact that many of the most representative scheduling problems are computationally intractable [18], have attracted a large body of research since the early 60s, especially from areas such as operational research and artificial intelligence. As a result, ever more efficient methods capable of solving ever more challenging scheduling problems have been proposed in the literature [10,11,33].

In the last decade, a considerable effort has been made to extend scheduling models and algorithms in order to capture and deal with complex constraints and features present in real-life environments. Notable examples of this are scheduling problems considering setup times in the use of resources [4], uncertainty in the duration of the tasks [17], flexibility in machines [12] or constraints derived from the presence of human operators [1–3,7,19,31,32].

Arguably, human operators play a fundamental role in numerous production environments, and so their presence imposes a number of constraints. For example, it may be the case that an organization counts on only a few operators to assist the processing of tasks on different machines. This situation may have a significant impact in the production process, as, in principle, at most as many machines as the number of operators can be working in parallel. This assumption was considered in [1,30], where an extension of the classic job-shop scheduling problem (JSP), termed job-shop scheduling with operators (JSO), was proposed.

In [2], the authors formalize the scheduling problem with arbitrary precedence relations and skilled operators (SPSO) motivated by a real-life handicraft com-

*Corresponding author: Ramiro Varela, Department of Computer Science, University of Oviedo, 33204 Gijón, Spain, E-mail: ramiro@uniovi.es.

pany. In this problem, a set of tasks must be performed on a set of physical machines and the processing of a particular task on a machine has to be assisted by an operator skilled to assist that task. Solving this problem is of major interest as it would allow companies to make the best use of their employees and to plan training or recruitment actions for future projects.

To solve the SPSO, two heuristic algorithms were proposed in [2], termed STA-OMSB and MSB-DOS respectively, which were evaluated and compared with an implementation on Cplex MILP solver, across a benchmark set defined in accordance with the characteristics of the problems of the handicraft company. In [31], the authors propose a genetic algorithm, which outperforms the results presented in [2].

The present paper builds on the proposals discussed in [31] and made some new contributions towards understanding and solving the SPSO. Firstly, we enhance the formal analysis of the SPSO. Then, we propose new search spaces, which allowed the genetic algorithm to improve previous solutions. Finally, we present an exhaustive experimental study on existing and new sets of instances, where the genetic algorithm was thoroughly analyzed and compared with other methods.

The remainder of the paper is organized as follows. In the next section, we review some related works. Section 3 is devoted to the formal definition of the SPSO. Section 4 describes and formalizes the general schedule builder termed $SOG\&T$, and show how it can be particularized to build schedules in different search spaces. In Section 5 we describe the proposed genetic algorithm. The results of the experimental study are reported in Section 6. Finally, Section 7 summarizes the main conclusions of the paper and outline some ideas for future work.

2. Related work

Metaheuristics are often used to solve complex problems due to their flexibility and adaptability to the problem characteristics. They may, for example, be predisposed to search over specific and promising regions of the search space [13,24]. We can find many applications in engineering problems. For example, the finite element model (FEM) updating problem has been solved using Fish School Search (FSS) [9]. In [22,23], the authors propose evolutionary algorithms to design free-form steel structures. Also, in [28] a combination of the Finite Element Method, Genetic Algorithms and Regression Trees is used to design and optimize complex welded products. In [21], an integrated methodology based in part on evolutionary computation to design a highly loaded and compact counterrotating compressor is presented.

These techniques have been used in addition with other Artificial Intelligence methods to solve problems exploiting their sinergies [35], e.g. evolutionary computation for machine learning [14,29].

Metaheuristics have, in particular, been applied to scheduling problems. In addition to the genetic and memetic algorithms mentioned in the previous section, other metaheuristics as particle swarm optimization (PSO) or differential evolution (DE) algorithms [36] were applied to them. In [37], a hybrid genetic algorithm that combines local searches with genetic population management techniques is proposed to solve the Skilled Workforce Project Scheduling Problem (SW-PSP), which is of interest in many service centres. In [16], a problem where a set of tasks is to be scheduled on a set of available resources on a fixed but not strict time interval is solved by means of a spectral clustering scheduling scheme. In [27], the authors propose a method that uses honey-bee mating and annealing processes to solve multi-objective planning and scheduling problems. In [20], they use a hybrid genetic algorithm to tackle a production-distribution planning problem.

3. Problem formulation

In the scheduling problem with skilled operators and arbitrary precedence relations, we are given

- $\mathcal{M}$, a set of q machines.
- $\mathcal{O}$, a set of p operators.
- $\mathcal{T}$, a set of n tasks or operations.
- The processing time p_u (integer) for each task $u \in \mathcal{T}$.
- The machine $m_u \in \mathcal{M}$ on which the task u must be processed.
- $\mathcal{O}_u \subseteq \mathcal{O}$, the subset of operators that are skilled to assist the task u.
- $(\mathcal{T}, E)$, the *task graph* that expresses precedence relations on the processing of tasks.

The objective is to allocate a starting time st_u and an operator o_u to assist the processing of each task $u \in \mathcal{T}$, such that the makespan, defined as

$$C_{max} = \max\{C_u; u \in \mathcal{T}\} \tag{1}$$

is minimized, where C_u denotes the completion time of the operation u, and the following constraints are satisfied:

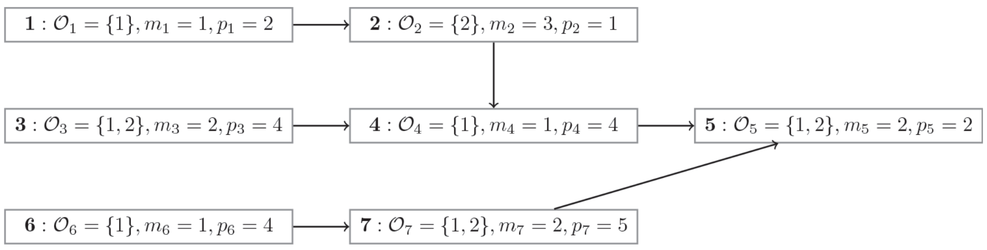

$\mathbf{1}: \mathcal{O}_1 = \{1\}, m_1 = 1, p_1 = 2$
$\mathbf{2}: \mathcal{O}_2 = \{2\}, m_2 = 3, p_2 = 1$
$\mathbf{3}: \mathcal{O}_3 = \{1,2\}, m_3 = 2, p_3 = 4$
$\mathbf{4}: \mathcal{O}_4 = \{1\}, m_4 = 1, p_4 = 4$
$\mathbf{5}: \mathcal{O}_5 = \{1,2\}, m_5 = 2, p_5 = 2$
$\mathbf{6}: \mathcal{O}_6 = \{1\}, m_6 = 1, p_6 = 4$
$\mathbf{7}: \mathcal{O}_7 = \{1,2\}, m_7 = 2, p_7 = 5$

Fig. 1. An SPSO instance with 7 tasks, 3 machines and 2 operators.

- Unary constraints. For every $u \in \mathcal{T}$:
  - $i.$ It must be assisted by an operator skilled to assist it; i.e., $o_u \in \mathcal{O}_u$.
  - $ii.$ It cannot be preempted; i.e., $C_u = st_u + p_u$.
- Binary constraints. For every two tasks $u, v \in \mathcal{T}$:
  - $iii.$ They must be processed following the order expressed by the task graph; i.e., if $(u, v) \in E$ then $st_u + p_u \leqslant st_v$.
  - $iv.$ If they are assisted by the same operator or processed on the same machine they cannot overlap; i.e., if $(o_u = o_v) \vee (m_u = m_v)$ then $(st_u + p_u \leqslant st_v) \vee (st_v + p_v \leqslant st_u)$.

Figure 1 shows the task graph for a problem instance with 7 tasks, 3 machines and 2 operators. Each task node includes the task number, the set of operators that are skilled to assist the task, the machine required by the task and its processing time.

This problem was defined in [2] and denoted JSSO (Job Shop Scheduling with Skilled Operators). We prefer to use the nomenclature SPSO (Scheduling Problem with Skilled Operators) as the term Job Shop is often used in the literature to refer to a particular task graph structure where the tasks are organized into sequences that are called *jobs*. The SPSO is NP-*hard* as it is a generalization of the JSP which is NP-*hard* [18].

Following [34], a feasible schedule S is represented by an acyclic graph of the form $G_S = (V, E \cup F \cup Q \cup I)$, where:

- Each node in V represents either a task in $\mathcal{T}$, or one of the fictitious tasks with null processing time; namely, starting tasks for each operator $o \in \mathcal{O}$, and the dummy operations *start* and *end*.
- E is the set of arcs of the task graph. $P(v)$ and $S(v)$ will denote the sets of tasks which are predecessors and successors of v respectively in the task graph.
- F is a set of arcs that represent the processing orders of tasks on the machines.
- Q is a set of arcs that represent the sequences of tasks assisted by each operator.
- The set I includes arcs connecting node *start* to each node $o \in O$ and arcs connecting each task without successors in the task graph to node *end*.

Figure 2 shows a solution graph for a problem instance with 7 tasks, 3 machines and 2 operators.

With this representation, building a schedule may be viewed as a process of growing up a partial solution graph starting from $F = \emptyset, Q = \emptyset$ and then adding arcs to F and Q until a feasible solution graph is completed.

4. Schedule generation scheme and search spaces

In this section we describe the $SOG\&T$ schedule generation scheme proposed in [31] and show how it may be particularized to search over different search spaces. We will consider and analyze two kinds of them: dominant and non-dominant search spaces; i.e., search spaces that contain at least one optimal solution and those that may not, respectively.

4.1. The $SOG\&T$ schedule builder

As pointed in [31], this algorithm schedules one task in each iteration following an appending schema. Let SC be the set of scheduled operations before the current iteration and let G_{SC} be the partial solution graph built so far. The *set of eligible operations* in the current iteration is defined as:

$$A = \{v; v \notin SC, P(v) \subseteq SC\} \tag{2}$$

A includes the first unscheduled tasks in the task graph. In principle, each task in A is a candidate to be scheduled next and it can be assisted by any of the operators skilled to do it. So, the *set of scheduling options* is defined as:

$$\mathcal{A} = \{(u, o); u \in A, o \in \mathcal{O}_u\} \tag{3}$$

If the option $(u, o) \in \mathcal{A}$ is selected in the current iteration, then u is assisted by the operator o and assigned

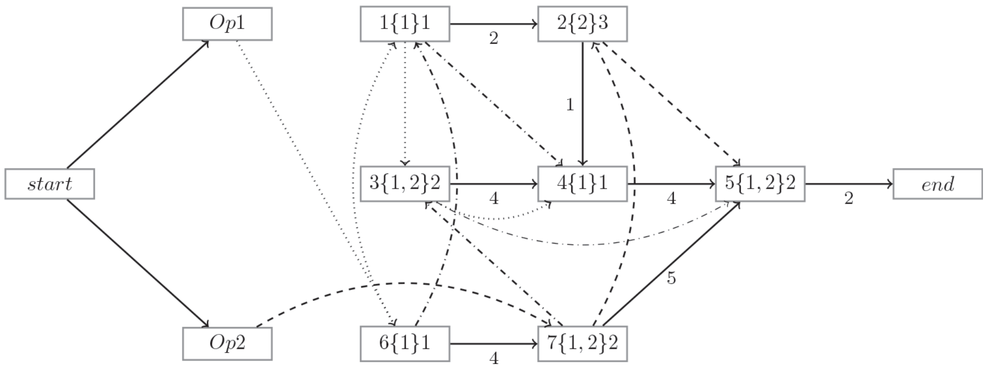


Fig. 2. A solution graph for the problem instance represented in Fig. 1. Each task node includes the task number, the set of skilled operators and the machine using the notation $u\mathcal{O}_u m_u$. Dotted and dashed arcs represent operator sequences starting in an operator node, while dotted-dashed arcs represent machine sequences. For the sake of clarity, only the arcs between consecutive nodes in the relations are represented and only the cost of the arcs in the task graph are displayed. Every arc should be labeled with the processing time of the task at the outgoing node. The makespan is 19 and corresponds to the cost of the critical path ($start$ Op1 6 7 3 4 5 end).

starting time given by

$$st_u(o) = \max\{st_x + p_x, st_y + p_y, st_z + p_z\} \quad (4)$$

where x is the task in $P(u)$ with the largest completion time, y is the last scheduled task on the machine required by u and z is the last scheduled task assisted by the operator o.

Algorithm 1 shows the general structure of the schedule builder $SOG\&T$. It starts from a set A containing the tasks with no predecessors in the task graph and iterates n steps. In each iteration an option (u, o) is taken non deterministically from a subset $\mathcal{X}$ of the set of scheduling options $\mathcal{A}$.

Algorithm 1 Schedule builder $SOG\&T$. It builds a feasible schedule in n steps.

```
Data: A SPSO problem instance $\mathcal{P}$
Result: A feasible schedule for $\mathcal{P}$
$A = \{u \in \mathcal{T}; \neg\exists(w, u) \in E\}$;
for i=1 to n do
    $\mathcal{A} = \{(u, o); u \in A, o \in \mathcal{O}_u\}$;
    $\mathcal{X}$ = a subset of $\mathcal{A}$;
    choose $(u, o) \in \mathcal{X}$ non deterministically;
    set $st_u = st_u(o)$ and $o_u = o$;
    add $u$ to $SC$ and update $G_{SC}$;
    $A = \{v; v \notin SC, P(v) \subseteq SC\}$;
end
return the built schedule $G_{SC}$;
```

4.2. Dominant search spaces

In [31], we have seen that the simplest way to get a dominant search space is taking $\mathcal{X} = \mathcal{A}$ in Algorithm 1. However, the search space can be reduced, without loss of dominance, considering the sets of options $\mathcal{A}'$ and $\mathcal{B}$ defined as follows. Let

$$(v^*, o^*) = \arg\min\{st_u(o) + p_u; (u, o) \in \mathcal{A}\} \quad (5)$$

and let $C^* = st_{v^*}(o^*) + p_{v^*}$. We define the sets $\mathcal{A}'$ and $\mathcal{B}$ as

$$\mathcal{A}' = \{(u, o) \in \mathcal{A}; st_u(o) < C^*\} \quad (6)$$

$$\mathcal{B} = \{(u, o) \in \mathcal{A}'; (m_u = m_{v^*} \lor o = o^*)\} \quad (7)$$

Note that each option in $\mathcal{A}'$ establishes a starting time for a task in the interval $[T_{\mathcal{A}}, C^*)$, and that $\mathcal{B}$ further restricts the options to starting times in $[T_{\mathcal{B}}, C^*)$, where

$$T_{\mathcal{B}} = \min\{st_u(o); (u, o) \in \mathcal{B}\} \geqslant T_{\mathcal{A}}, \quad (8)$$

4.3. Non-dominant search spaces

The dominance of the spaces defined in the previous section relies on the fact that the respective sets of options contain the set $\mathcal{B}$. In this section, we will consider sets of options for which is not guaranteed to contain the set $\mathcal{B}$ and consequently, they may give rise to non-dominant search spaces. These spaces may be interesting if they have small size and at the same time they contain a large fraction of high-quality solutions. Clearly, from the point of view of a genetic algorithm searching over spaces having these characteristics, the fact that they may not contain an optimal schedule should not be considered as a real problem due to the fact that a genetic algorithm cannot guarantee to reach the best solution contained in the search space taking finite time.

To generate non-dominant search spaces, we may start just considering the sets of options $\mathcal{A}, \mathcal{A}'$ and $\mathcal{B}$ and in all three cases discard some of the options that may give rise to the largest idle times of operators and machines. Bearing this in mind, we propose a number of parameterized sets of options, whose rationale is quite similar to that behind the so called hybrid schedules for dynamic job shop scheduling problems used in [8].

The first parameterized set of options is defined as:

$$\mathcal{A}(\delta) = \{(u, o) \in \mathcal{A}; st_u(o) < T_\mathcal{A} + \delta(C - T_\mathcal{A})\} \quad (9)$$

where $\delta \in [0, 1]$ is a parameter. So, $\mathcal{A}(\delta)$ is a reduction of the set $\mathcal{A}$ in which the options that may give rise to idle times larger than $T_\mathcal{A} + \delta(C - T_\mathcal{A})$ are removed. It is clear that for a sufficiently small value of δ, the set $\mathcal{A}(\delta)$ may not contain the set $\mathcal{B}$ and so the generated search space may not be dominant.

Similarly, we define reductions of the sets $\mathcal{A}'$ and $\mathcal{B}$ as

$$\mathcal{A}'(\delta) = \{(u, o) \in \mathcal{A}'; st_u(o) < T_\mathcal{A} + \delta(C^* - T_\mathcal{A})\} \quad (10)$$

and

$$\mathcal{B}(\delta) = \{(u, o) \in \mathcal{B}; st_u(o) < T_\mathcal{B} + \delta(C^* - T_\mathcal{B})\} \quad (11)$$

In the extreme cases of $\delta = 1$ and $\delta = 0$, $\mathcal{A}(1) = \mathcal{A}$, $\mathcal{A}'(1) = \mathcal{A}'$, $\mathcal{B}(1) = \mathcal{B}$ and $\mathcal{A}(0) = \mathcal{A}'(0) = \mathcal{B}(0) = \emptyset$, so the significant cases are those with $\delta \in (0, 1]$. Regarding the relationship among these sets of options, for all δ, $\mathcal{B}(\delta) \subseteq \mathcal{A}'(\delta) \subseteq \mathcal{A}(\delta)$; and for all δ', δ'' such that $\delta' < \delta''$ and $\mathcal{X} \in \{\mathcal{A}, \mathcal{A}', \mathcal{B}\}$, $\mathcal{X}(\delta') \subseteq \mathcal{X}(\delta'')$. Also, $\mathcal{A}(\delta_1) = \mathcal{A}'$ being

$$\delta_1 = (C^* - T_\mathcal{A})/(C - T_\mathcal{A}) \quad (12)$$

In all three cases, δ establishes a bound on the maximum length of the time intervals a machine and an operator remain idle while there is a task that can be processed on the machine and assisted by the operator; the lower δ the lower the bound.

$$\begin{tikzcd}
\mathcal{A} & \mathcal{A}' \arrow[l, "*"'] & \mathcal{B} \arrow[l] \\
\mathcal{A}(\delta) \arrow[u, "*"'] & \mathcal{A}'(\delta) \arrow[l, "*"'] \arrow[u, "*"'] & \mathcal{B}(\delta) \arrow[l] \arrow[u, "*"']
\end{tikzcd}$$

Fig. 3. "Subspace of" relationships among the six search spaces defined. For an arc $(\mathcal{Y}, \mathcal{Z})$, the symbol "*" means that in every iteration, for all options (u, o) in $\mathcal{Z}$ that are not in $\mathcal{Y}$, $st_u(o) \geqslant st_{u'}(o')$ for all (u', o') in $\mathcal{Y}$, and therefore it may be expected that the average makespan of the schedules in the space $\mathcal{Y}$ to be lower than that in the space $\mathcal{Z}$.

4.4. *Summary of search spaces*

In previous sections, we have defined six search spaces for the SPSO, three of them are dominant, while the other three are non-dominant. Each space is characterized by the set of options considered in each iteration of the $SOG\&T$ algorithm (see Algorithm 1), some of these sets are in fact subsets of others and this induces a "subspace of" relation among them. Abusing language each space will be identified by the same symbol as the corresponding set of options, namely $\mathcal{A}, \mathcal{A}', \mathcal{B}, \mathcal{A}(\delta), \mathcal{A}'(\delta)$ and $\mathcal{B}(\delta)$. $\mathcal{A}, \mathcal{A}', \mathcal{B}$ are dominant search spaces, while $\mathcal{A}(\delta), \mathcal{A}'(\delta)$ and $\mathcal{B}(\delta)$ are non-dominant for all $\delta \in (0, 1)$ and become $\mathcal{A}, \mathcal{A}', \mathcal{B}$ respectively for $\delta = 1$.

Figure 3 shows how these search spaces are related through the relation "subspace of". The symbol "*" in some of the arcs has to do with the idle periods allowed in the sets of options at the outgoing and incoming spaces.

5. Genetic algorithm for the SPSO

We use a genetic algorithm with generational replacement. In order to avoid premature convergence, we opted not to use the classic roulette wheel selection combined with unconditional replacement. Instead, in the selection phase all chromosomes are organized into pairs at random, then each pair undergoes crossover and mutation. After this, the offsprings are evaluated. Finally, the new population is obtained by means of tournament selection keeping the best two individuals among every two parents and their two offsprings. In the following we only detail the components that depend on the problem; namely, the coding schema, the decoding algorithm and the genetic operators.

5.1. Coding schema

A chromosome consists of two permutations of symbols. The first one is the *task sequence* which is a conventional permutation of the numbers $1 \ldots n$, while the second is the *operator sequence* and is given by a permutation with repetition of the symbols $1 \ldots p$ of size n and represents operator preferences. Notice that some operators could appear several times in the chromosome and also some operator may be absent. For example the following two permutations represent a feasible chromosome for the instance considered in Fig. 1.

$$\begin{array}{l}(6,7,5,4,1,2,3)\\(1,2,2,2,1,1,1)\end{array} \quad (13)$$

This encoding should be understood in such a way that if task u appears before task v in the first permutation, then u should be preferably scheduled before v. At the same time, the second permutation represents priorities for the operators to be allocated to tasks. Each task has a different priority order for operators. An important observation here is that the task ordering and the operators' allocation in the schedule does not only depend on the chromosome, but also on the decoding algorithm, as we will see in the next section.

One of the most interesting properties of this encoding is that any solution can be encoded into a chromosome representing the same machine orderings and operator allocations. At the same time, it is simple and so it allows for designing efficient genetic operators for crossover and mutation. In principle, the only inconvenience comes from the fact that the number of replicas of a symbol in the operator sequence is variable and so an operator could disappear from the chromosome. However, this problem can be easily solved by means of some heuristic repairing as we will see.

5.2. Decoding algorithm

Decoding algorithms map chromosomes to feasible solutions. To this aim, we use the $SOG\&T$ algorithm presented above, exploiting the information encoded in the chromosomes to guide the search. This algorithm is issued and in each iteration the option $(u, o) \in \mathcal{X}$ that is the "leftmost" in the chromosome, i.e., that fulfills the following two conditions, is chosen

- u is the leftmost in the task sequence of the chromosome among all tasks appearing in some option in $\mathcal{X}$, and
- o is the first operator skilled to assist u in the preference list defined by the operator sequence, among the operators appearing in some option in $\mathcal{X}$ for the operation u. If the operator of none of the options for u in $\mathcal{X}$ is present in the operator sequence, then the operator o with $(u, o) \in \mathcal{X}$ that allows u to start earliest is selected.

In the experimental study we will consider all the possibilities above for the set of options $\mathcal{X}$; namely, the dominant spaces $\mathcal{A}$, $\mathcal{A}'$ and $\mathcal{B}$ and their non-dominant extensions $\mathcal{A}(\delta)$, $\mathcal{A}'(\delta)$ and $\mathcal{B}(\delta)$.

We illustrate how the decoding algorithm works by means of an example. Let us consider the chromosome given in expression (13). It contains the tentative orderings (6,4,1) and (7,5,3) for machines 1 and 2 respectively, which are inconsistent due to the fact that the task 4 must be processed after the task 1, and the task 5 must be processed after the tasks 7 and 3, in accordance with the precedences expressed by the task graph. In spite of that, the chromosome is feasible as the decoding algorithm will build a feasible schedule from it in which some of the tentative orderings and operator preferences will hold, while others will not. If we consider the set $\mathcal{X} = \mathcal{A}$, in the first iteration the scheduling options would be $\{(1,1), (3,1), (3,2), (6,1)\}$; in all four cases the starting time would be 0. So, according to the chromosome, the task scheduled in this step would be 6, as it is the leftmost one in the chromosome among 1, 3, and 6, and the operator assigned would be 1 (as this is the only option). In the next iteration, the options would be $\{(3,2)\}$ with starting time 0 and $\{(1,1), (3,1), (7,1), (7,2)\}$ with starting time 4. In accordance with the chromosome, the chosen option would be (7,2). In the third iteration the options would be $\{(3,1), (3,2)\}$ with starting time 9 and $\{(1,1)\}$ with starting time 4, and the chosen option would be (1,1). This way, the tentative ordering (4,1) in the chromosome is not kept in the schedule. Finally, we will have the schedule of Fig. 2. It is important to remark that with $\mathcal{X} = \mathcal{A}'$ or $\mathcal{X} = \mathcal{B}$, we could reach different schedules from the same chromosome.

5.3. Genetic operators

We build on two point crossover and single mutation operators, and start with an initial population of chromosomes generated at random following an uniform distribution.

We will use a double-chromosome variant of the classic order crossover (OX) which translates the subsequence of symbols between two cutting points from one parent to the offspring and the relative order of the remaining values from the second. OX may be a good option for the SPSO due to the fact that relevant characteristics, as processing order of tasks on the machines or priorities of operators for assisting the tasks, may be transmitted from parents to offsprings. To avoid a strong disruptive effect of the crossover, the cutting points will be the same in both sequences (tasks and operators).

Also, we will consider single mutation (SM) by swapping two consecutive positions at random in both sequences. Also, we will also use an operator mutation (OM), which changes the value in a location of the operator sequence at random. When a chromosome is mutated, one of SM or OM is chosen with probability 0.5.

Additionally, we consider coding back the structure of the schedule into the chromosome. If chosen, it means rebuilding the chromosome structure in such a way that the order of tasks in the task sequence is compatible with the partial ordering on the tasks in the schedule, and that for each task the first operator in the operator list is that assisting the task in the schedule. This way, the machine and operator sequences in the schedule are translated to the chromosome and so these relevant characteristics of the schedules may be better translated from parents to offsprings. This is a sort of Lamarckism as some of the characteristics learned over the breeding of the phenotype (the schedule) from the genotype (the chromosome) are coded back into the chromosome. As the breeding is just produced by decoding the chromosome, this operation is expected to introduce small changes in the chromosome structure and so we refer to the effect it produces in the genetic algorithm as weak Lamarckian evolution.

6. Experimental study

The purpose of this experimental study is to evaluate the characteristics of the proposed search spaces, analyze the performance of the genetic algorithm and compare this algorithm with the state of the art. To this end, we used a benchmark set with instances of different sizes and characteristics. To evaluate the search spaces, we first generated random solutions in each space and analyzed the quality and distribution of the makespan values. Then, we analyzed the evolution of the genetic algorithm when searching solutions in these spaces and in particular the influence that the coding-back option has on the convergence. The objective of these preliminary experiments was to establish reasonable conditions for running the genetic algorithm which, as will see, depend on the size and other characteristics of the instances, and to identify the most appropriate search spaces for each type of instances. In order to avoid a combinatorial explosion on the number of experiments, we opted to fix some of the parameters to rather standard values, in particular $popSize = 100$, $P_c = 1.0$ and $P_m = 0.1$ ($popSize$ is the population size; P_c and P_m are the crossover and mutation probabilities). The values for other parameters as $timeLimit$, $\mathcal{X}$ or $codingBack$ will be established from the results of these preliminary experiments. To demonstrate the performance of the proposed genetic algorithm, we show the results across all instances of the benchmark set under the best conditions we have identified for each type of instances, and when possible we establish a fair comparison with other methods. The target machine was Intel Xeon 2.26 GHz. 24 GB RAM and the algorithms were coded in C++.

6.1. Benchmark sets

We considered three sets of instances:[1] firstly the set proposed in [2] which includes 50 instances organized in 5 groups with 10 instances each. Each group is defined by the values of n, p and q ranging in the sets $\{100, 150, 200\}$, $\{10, 15, 20\}$ and $\{15, 30, 50\}$ respectively. The processing times are uniformly distributed in $[1, 100]$. The topology of the task graph was inspired in real-life assembly trees and was distributed in a number of branches, between 3 and 6. The sets of operators skilled for each task were generated at random and the workload of the machines was roughly balanced. We obtained detailed results from the algorithm MSB-DOS and the Cplex implementation proposed in [2] through personal communication from the authors. These results are summarized in Table 5.

The instances in the second set are derived from job-shop scheduling problem (JSP) instances. A JSP instance is characterized by a number of N jobs and a number of M machines. Each job is a sequence of M operations, each one requiring a different machine and so each job has to visit all the machines, but the order is in general different for every two jobs. Therefore, the task graphs express sequences among operations in

[1]The instances and more details of the experimental study are available at http://www.di.uniovi.es/iscop (Repository).

each job. In these instances $n = N \times M$ and $q = M$. The JSP instances considered are $FT10(10 \times 10)$, $LA21-25(15\times10)$, $LA26-30(20\times10)$, $LA31-35(30\times10)$, $LA36-40(15\times15)$, $ta11-20(20\times15)$, $ta21-30(20\times20)$ and $ta31-40(30\times15)$. For each of these 51 JSP instances, three different numbers of operators p were considered depending on the number of resources q and the skill matrix was created taking different probabilities Pr that an operator is skilled for a given task. Specifically, $p \in \{5,7,9\}$ for $q = 10$, $p \in \{7,11,14\}$ for $q = 15$ and $p \in \{10,15,19\}$ for $q = 20$. Also, we considered 3 different combinations of Pr: 0.2, 0.6 and 0.2 or 0.6 chosen at random for each task. We think these combinations may be significant because they show the cases in which a task requires a high level of specialization, and therefore not many operators can assist the task (0.2), and the cases in which a task requires less specialization and so it may be assisted by more operators (0.6). So, this set contains 459 instances.

For the third set, we considered instances with general precedence relations. To this end, we looked at the precedence relations on tasks for 5 instances of the resource-constrained project scheduling problem (RCPSP): $j1201\_1$, $j1202\_1$, $j1203\_1$, $j1204\_1$ and $j1205\_1$, each one having 120 tasks; then we took $q = 10$ and assigned each task one machine at random. p and Pr taking values as in the instances having $q = 10$ in the second set. So, this set contains 45 instances in all.

The JSP and RCPSP instances considered are available in the OR-library [5].

Notice that the minimum makespan for each instance will strongly depend on both the number of operators available for the tasks and their skills. The number of tasks that may be processed in parallel is bounded by the number of operators available. Furthermore, if the operators have few skills, some of them may even be idle due to being unable to assist none of the available tasks. In contrast to this situation, if there are many operators and they have many skills (or equivalently the tasks may be assisted by many operators), it is more likely that many tasks are processed simultaneously giving rise to a low makespan.

6.2. Analysis of the search spaces

In order to get an initial idea about the quality of solutions in the different search spaces considered, we generated 1000 random schedules in each one of them and observed the diversity and quality of these solutions. We start considering three instances derived from FT10, the smallest instance in the second set, and the dominant search spaces $\mathcal{A}$, $\mathcal{A}'$ and $\mathcal{B}$. Figure 4 shows the makespan outcomes for each set of schedules. It is remarkable that for the three instances the best random solution generated in the set $\mathcal{A}$ is much worse than the worst solutions generated from the sets $\mathcal{A}'$ and $\mathcal{B}$. So, even though the diversity of the solutions from $\mathcal{A}$ is clearly larger than the diversity of the random schedules from $\mathcal{A}'$ and $\mathcal{B}$, the search space $\mathcal{A}$ does not seem to be a good choice. As mentioned earlier, the reason for the low quality of the solutions in this space is that it contains many schedules having large periods of inactivity for machines and operators, and so it may be even difficult to reach a schedule within the spaces $\mathcal{A}'$ and $\mathcal{B}$ if the search is performed on the whole space $\mathcal{A}$. This can be observed, at least, in a random search.

We also analyzed random instances from some large JSP instances, in this case considering both dominant and non-dominant search spaces. Figure 5 shows results from the search spaces $\mathcal{A}(\delta)$, $\mathcal{A}'(\delta)$ and $\mathcal{B}(\delta)$ taking different values of δ. Remember that $\delta = 1$ gives rise to dominant search spaces; $\delta \sim 0$ means that the value of δ is sufficiently close to 0 for considering only the options with starting times $T_{\mathcal{A}}$ or $T_{\mathcal{B}}$ depending on the search space. We can observe that the lower δ the lower both diversity and average makespan. Again, it seems clear that $\mathcal{A}(\delta)$ is not a good option as it shows the worst average makespan for all values of δ. Also, we can see that the average values of the schedules from $\mathcal{A}(0.25)$ and $\mathcal{A}'(1)$ are rather similar, but $\mathcal{A}'(1)$ shows better diversity and it is dominant, while $\mathcal{A}(0.25)$ is not dominant. As before, $\mathcal{A}'(\delta)$ shows better average makespan than $\mathcal{B}(\delta)$ and at the same time similar diversity for all values of δ.

So, from these preliminary results, we discard the search space $\mathcal{A}(\delta)$, for all values of δ, due to the low average quality of the schedules and only consider $\mathcal{A}'(\delta)$ and $\mathcal{B}(\delta)$ in the remaining experiments. Even though $\mathcal{A}'(\delta)$ seems to be the best choice for the sake of successful evolution and convergence of the GA, this hypothesis must be assessed from an analysis of the convergence of the genetic algorithms searching in these spaces. This is the objective of the next section.

6.3. Analysis of the convergence of the genetic algorithm

It is well known that genetic algorithms are sensitive to mechanisms that introduce bias in the search towards regions of the solution spaces with above aver-

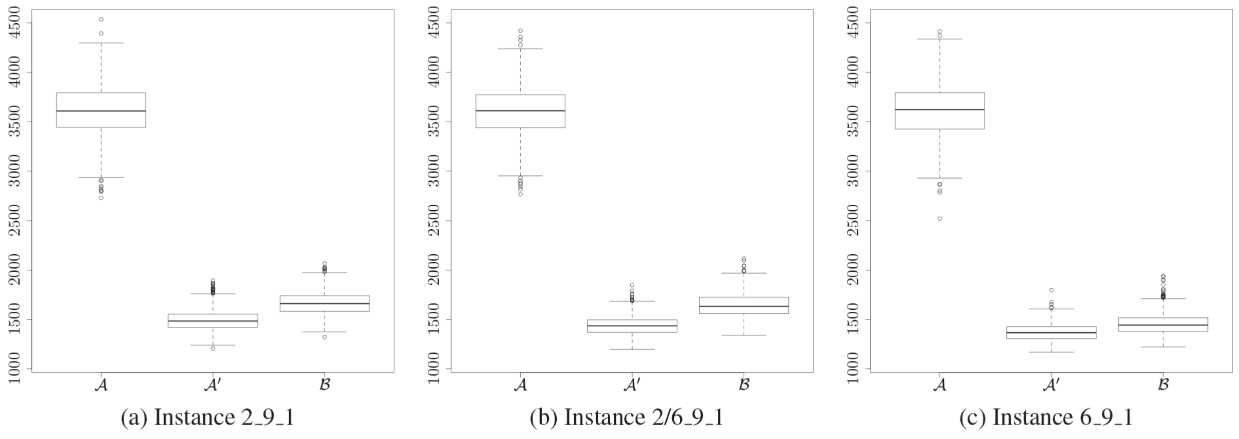


Fig. 4. Distribution of makespan values obtained from 1000 random schedules for the instances 2_9_1, 2/6_9_1 and 6_9_1 derived from the FT10 in the search spaces $\mathcal{A}$, $\mathcal{A}'$ and $\mathcal{B}$.

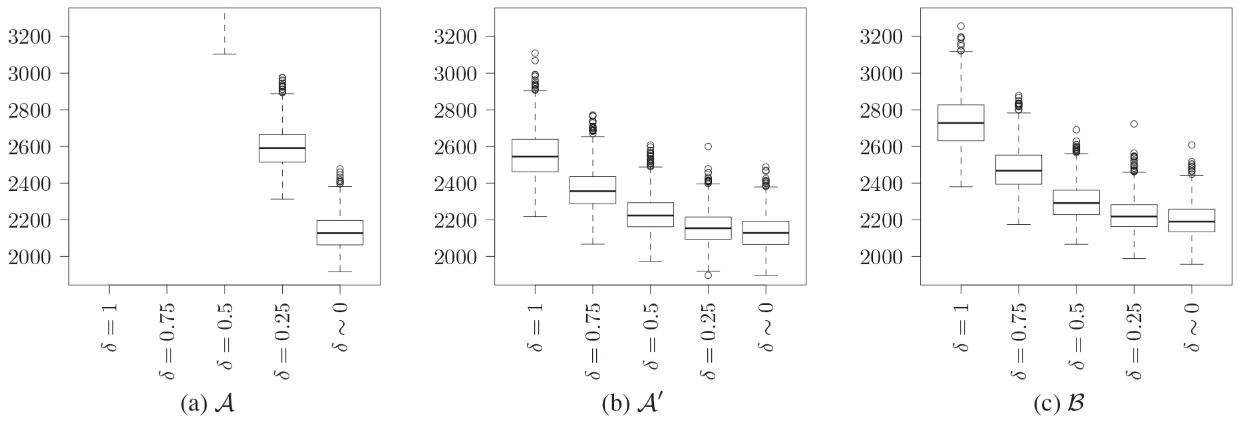


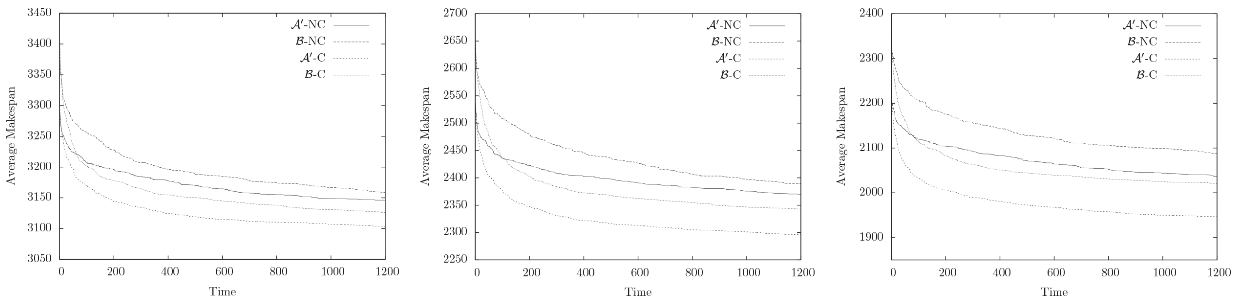
Fig. 5. Distribution of makespan values obtained from 1000 random schedules for the instance LA31_06_9op derived from the LA31 in the search spaces $\mathcal{A}$, $\mathcal{A}'$ and $\mathcal{B}$ with different δ values. Note that in (a) the values obtained with $\delta \in \{1, 0.75, 0.5\}$ are much worse than those obtained in (b) and (c), so they are outside the scale considered.

age solutions. In general these mechanisms may allow the genetic algorithms to reach better solutions but they may also give rise to the well-known phenomenon of premature convergence. In this paper, we introduced a number of elements of this nature as the coding-back option or the definition of different search spaces. It is therefore interesting to analyze how these elements may affect the convergence of the genetic algorithms.

6.3.1. *Coding-back option and search spaces $\mathcal{A}'$ and $\mathcal{B}$*

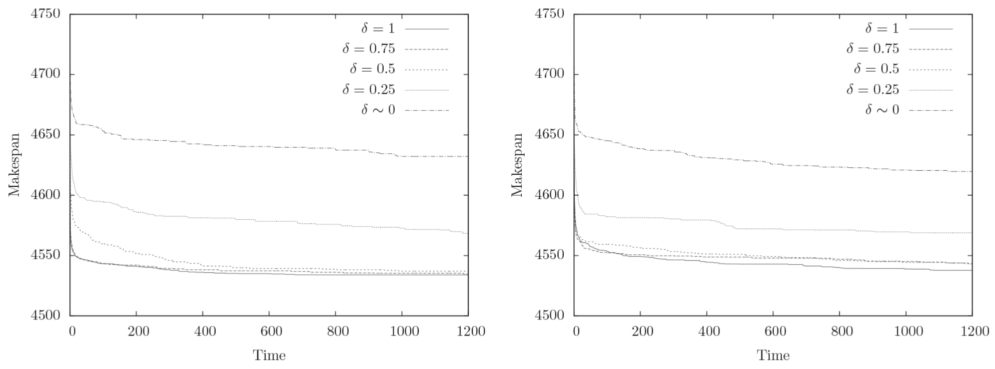
Let us start considering the effect of the coding-back option. Figure 6 shows the convergence patterns of the average solutions, for three instances of the second benchmark set derived from LA31, over the spaces $\mathcal{A}'$ and $\mathcal{B}$ with and without choosing $codingBack$. In all experiments $timelimit = 1200s$ and the remaining parameters were set as indicated before. As we can observe, the best average makespan values in the initial populations are better for $\mathcal{A}'$ than they are for $\mathcal{B}$. Moreover, the convergence is better when the coding-back option is chosen. This is quite clear looking the evolution of the average and best solutions. So, searching on $\mathcal{A}'$ and coding the schedules back into the chromosomes seems to be the best choice to search over a dominant space.

To further assess the differences between decoding in the subsets $\mathcal{A}'$ and B, we solved the 45 instances derived from FT10 with the coding-back option. In these experiments, we set $timeLimit = 120s$. Each instance was solved 30 times and the best and average of the solutions reached in all runs were recorded. We calculated the average error for the best and averages with respect to the best solution found for each in-



(a) Instance LA31_06_5op (b) Instance LA31_06_7op (c) Instance LA31_06_9op

Fig. 6. Convergence of the GA for the instances LA31_06 derived from the LA31, combining the search spaces $\mathcal{A}'$ and $\mathcal{B}$ and $codingBack$ chosen (C) or not chosen (NC). Each plot represents the evolution of the mean makespan of the population averaged for 30 runs.



(a) $\mathcal{A}'(\delta)$ Average Makespan (b) $\mathcal{B}(\delta)$ Average Makespan

Fig. 7. Convergence of the GA for the instance 10_15_100_8 from the Agnetis Benchmark, combining the search spaces $\mathcal{A}'(\delta)$ and $\mathcal{B}(\delta)$ with different δ values and the coding-back (C) option. Each plot represents the evolution of the mean makespan of the population averaged for 30 runs.

stance in these runs. Also, in order to study the stability of the algorithm, we calculated the average Pearson coefficient of variation (CV). These results, averaged for each subset of 5 instances, are summarized in Table 1. As we can observe, searching on $\mathcal{A}'$ the genetic algorithm reaches the best solution found and achieves considerably better results in average in all cases. At the same time, it is more stable in average than it is when it searches on $\mathcal{B}$.

Table 1

Summary of results from GA on the sets of instances derived from FT10, considering the search spaces $\mathcal{A}'$ and $\mathcal{B}$. The results are averaged for each subset of 5 instances with the same p and Pr

Sets		$\mathcal{A}'$			$\mathcal{B}$		
Pr	p	%Err. Best.	%Err. Avg.	%CV	%Err. Best.	%Err. Avg.	%CV
2	5	0.00	3.17	1.57	3.56	7.25	1.71
	7	0.00	2.82	1.45	4.20	8.23	2.05
	9	0.00	2.96	1.79	3.99	8.90	2.15
	Avg.	0.00	2.98	1.60	3.92	8.13	1.97
6	5	0.00	3.09	1.37	5.45	8.81	1.51
	7	0.00	3.44	1.59	5.97	10.09	1.69
	9	0.00	3.41	1.53	3.24	7.44	1.78
	Avg.	0.00	3.32	1.49	4.89	8.78	1.66
2/6	5	0.00	3.42	1.48	6.98	11.09	1.77
	7	0.00	3.26	1.72	8.32	12.90	2.22
	9	0.00	3.05	1.72	7.95	11.66	1.70
	Avg.	0.00	3.24	1.64	7.75	11.89	1.89
	T. Avg.	0.00	3.18	1.58	5.52	9.60	1.84

6.3.2. Search spaces $\mathcal{A}'(\delta)$ and $\mathcal{B}(\delta)$

We now analyze the evolution of the genetic algorithms on the spaces $\mathcal{A}'(\delta)$ and $\mathcal{B}(\delta)$. To this end, we consider 5 different values for the parameter δ: 1, 0.75, 0.5, 0.25 and ~ 0, and instances of different sizes. Remember that $\delta = 1$ means that the search space is dominant while lower values of δ restrict the search to non-dominant spaces. So, one can expect that for small instances the best option could be $\delta = 1$ as for these instances we have the chance of reaching the optimal solution. However, for the largest instances, as the search space grows exponentially with the problem size, the best choice is likely to search over a reduced space.

Let us start with the set of small instances proposed

Table 2
Summary of results for the instances derived from the FT10 ($n = 100$, $q = 10$) considering different values of δ. The results are averaged for instances with the same values of p and Pr. $timeLimit = 120s$

		$\mathcal{A}'(\delta \sim 0)$			$\mathcal{A}'(\delta = 0.25)$			$\mathcal{A}'(\delta = 0.5)$			$\mathcal{A}'(\delta = 0.75)$			$\mathcal{A}'(\delta = 1)$		
Pr	p	%Err. Best	%Err. Avg	%CV	%Err. Best	%Err. Avg	%CV	%Err. Best	%Err. Avg	%CV	%Err. Best	%Err. Avg	%CV	%Err. Best	%Err. Avg	%CV
2	5	1.62	3.71	0.95	1.11	2.53	0.92	0.43	2.53	1.03	0.48	3.01	1.40	1.16	4.37	1.57
	7	2.55	4.46	1.03	1.41	3.27	1.00	0.09	2.61	1.20	0.52	2.99	1.29	1.52	4.39	1.45
	9	1.87	4.26	1.03	1.28	3.66	1.00	0.57	2.88	1.12	0.00	2.72	1.30	0.84	3.83	1.79
	Avg.	2.01	4.14	1.00	1.27	3.15	0.97	0.36	2.67	1.12	0.33	2.91	1.33	1.17	4.19	1.60
6	5	0.96	2.33	0.68	0.15	1.69	0.76	0.30	2.08	0.87	0.98	3.26	1.02	2.19	5.35	1.37
	7	0.67	2.33	0.85	0.25	2.00	0.98	0.46	2.38	1.10	0.55	3.43	1.45	2.13	5.63	1.59
	9	0.92	2.23	0.67	0.21	1.77	0.83	0.21	1.90	1.08	0.21	2.50	1.11	0.64	4.07	1.53
	Avg.	0.85	2.30	0.74	0.20	1.82	0.86	0.32	2.12	1.01	0.58	3.06	1.19	1.65	5.02	1.49
2/6	5	1.39	3.25	0.88	0.14	2.81	1.07	0.75	2.62	0.97	1.10	3.41	1.13	2.29	5.79	1.48
	7	1.51	3.72	0.99	0.89	3.12	1.17	0.08	2.57	1.31	0.53	3.42	1.54	1.50	4.81	1.72
	9	1.09	3.22	1.14	0.86	2.68	1.12	0.21	2.56	1.20	0.10	2.78	1.40	1.32	4.41	1.72
	Avg.	1.33	3.40	1.00	0.63	2.87	1.12	0.35	2.58	1.16	0.58	3.20	1.36	1.71	5.00	1.64
	T. Avg.	1.40	3.28	0.91	0.70	2.61	0.98	0.34	2.46	1.10	0.50	3.06	1.29	1.51	4.74	1.58

in [2]. Figure 7 shows the convergence patterns of the genetic algorithm for one of these instances. As we can see, the worst option is $\delta \sim 0$, followed by $\delta = 0.25$, while there are not too much differences among the remaining values. However, $\delta = 1$ is the best option for both $\mathcal{A}'$ and $\mathcal{B}$. It seems that in both cases the optimal solution is reached, and that for small values of δ the search space contains no optimal schedules.

Let us now consider the instances derived from FT10 and LA21-25, i.e., the smallest instances of the second set with 100 and 150 tasks respectively, and the instances derived from RCPSP with 120 tasks. Figure 8 show the convergence patterns of the genetic algorithm for one instance derived from FT10, one instance derived from the RCPSP, and one instance derived from LA21, respectively. In all three cases, it seems clear that $\delta = 1$ is the worst choice and that the GA performs better with δ values not be greater than 0.5. Besides, it seems clear that the convergence of the genetic algorithms on $\mathcal{A}'(\delta)$ is better than it is on $\mathcal{B}(\delta)$. Therefore, these results suggest that these instances are harder to solve than those in the first set, the dominant search spaces become so large that it is worth to restrict the search to some subset of schedules with limited idle times for resources even at the risk of leaving all optimal schedules out of the effective search space.

Let us finally consider the largest instances of the benchmark set, i.e., those derived from JSP instances other than FT10 and LA21-25. Figure 9 shows the evolution of the genetic algorithm for one of them. At difference of the smallest and medium size instances, we can observe that the initial values and the evolution of the genetic algorithm are clearly better for the lowest values of δ. In particular $\delta = 1$ is also the worst choice and $\delta \sim 0$ is now the best one. So, for these instances, the search spaces are so huge that restricting the search to the schedules where a machine and an operator are never idle at the same time that there is some task available for them, seems to be the best choice. Also, $\mathcal{A}'(\delta)$ seems to be better than $\mathcal{B}(\delta)$ for all values of δ.

From the above experiments, we consider the space $\mathcal{A}'(\delta)$ as the best one and propose to choose $\delta = 1$ for the small instances, those in the set proposed [2], $\delta \in [0.25, ..., 0.5]$ for medium size instances, those derived from FT10, LA21-25 and RCPSP, and $\delta \sim 0$ for the large ones, the remaining ones in the set.

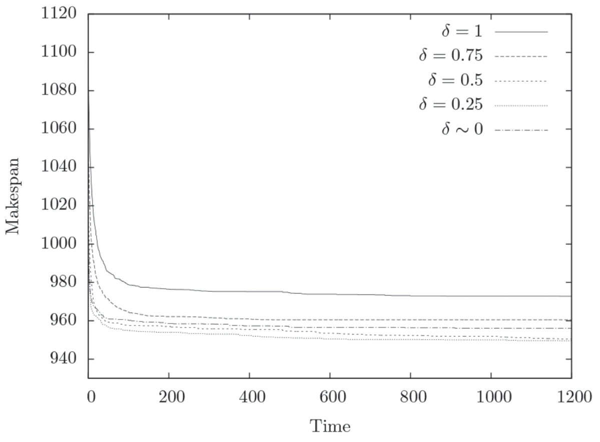
6.4. Evaluation of the genetic algorithm

We evaluated the GA considering the search space $\mathcal{A}'$, the coding-back option and different δ values. The evaluation was done across the second and third sets of instances. For the instances derived from $FT10$ and the medium size instances we considered $\delta \in \{\sim 0, 0.25, 0.5, 0.75, 1\}$ and for the large instances we considered $\delta \sim 0$ and also $\delta = 1$ just for the purpose of comparison.

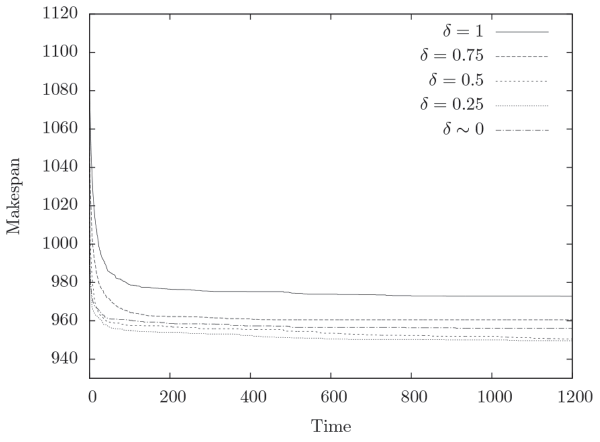
We have chosen a rather conventional parameter setting: $P_c = 1$,[2] $P_m = 0.1$, $popSize = 100$. $timeLimit$ was different for each set of instances: 120 seconds for the instances derived from $FT10$ instance, 150 seconds for the medium size instances and 300 seconds for the large instances. Each instance was solved 30

[2]Taking $P_c < 1$ makes that some pairs of chromosomes are not mated and in this case the offsprings are the same as their parents if they are not mutated, so the best parent is chosen twice in the replacement phase, what may contribute to premature convergence.

FT10_06

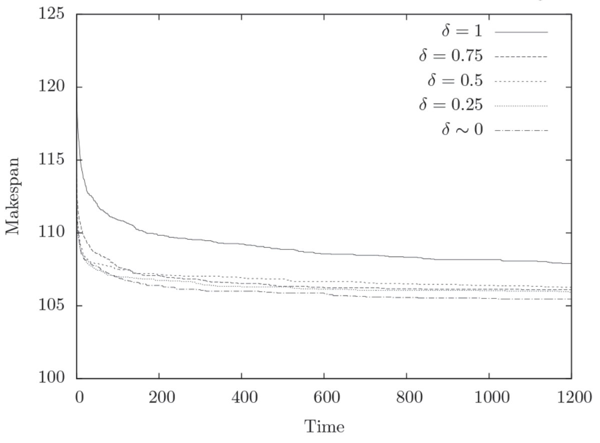


(a) $\mathcal{A}'(\delta)$ Average Makespan

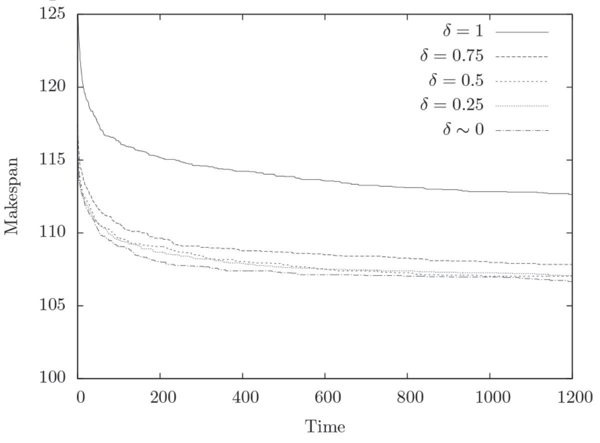


(b) $\mathcal{B}(\delta)$ Average Makespan

j1205_1_06_9op

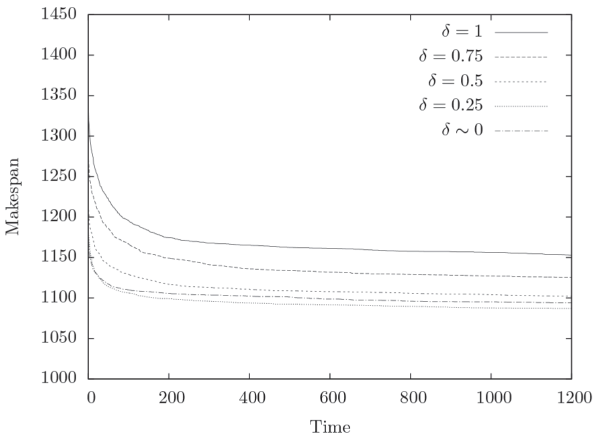


(c) $\mathcal{A}'(\delta)$ Average Makespan

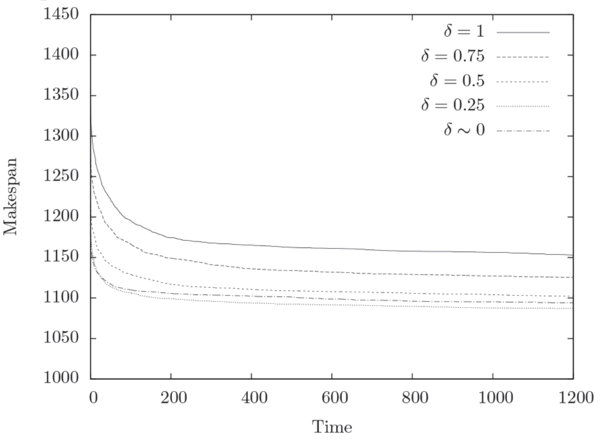


(d) $\mathcal{B}(\delta)$ Average Makespan

LA21_06_9op



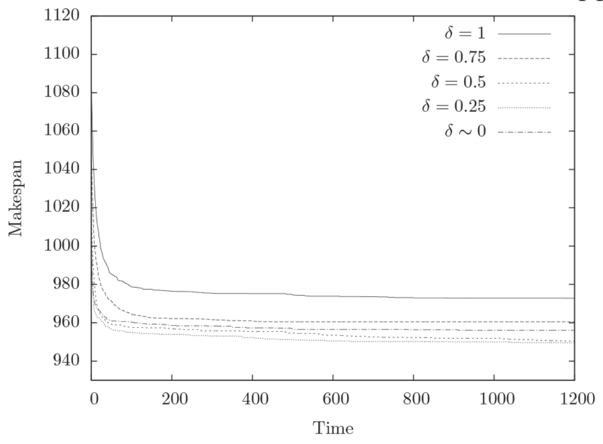
(e) $\mathcal{A}'(\delta)$ Average Makespan



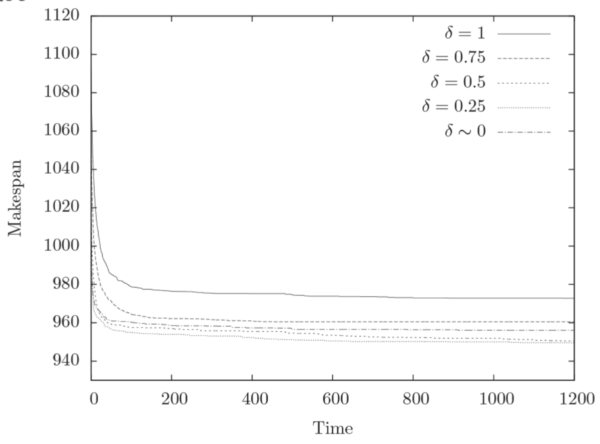
(f) $\mathcal{B}(\delta)$ Average Makespan

Fig. 8. Convergence of the GA for the instances FT10_06, j1205_1_06_9op and LA21_06_9op, derived from the FT10, RCPSP and JSP benchmarks respectively, combining the search spaces $\mathcal{A}'(\delta)$ and $\mathcal{B}(\delta)$ with different δ values and the coding-back (C) option. Each plot represents the evolution of the mean makespan of the population averaged for 30 runs.

FT10_06

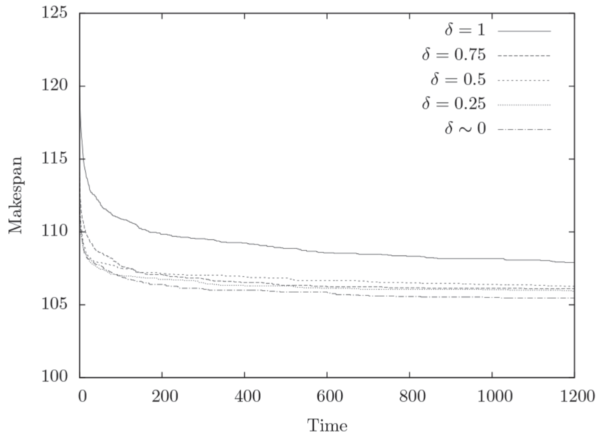


(a) $\mathcal{A}'(\delta)$ Average Makespan

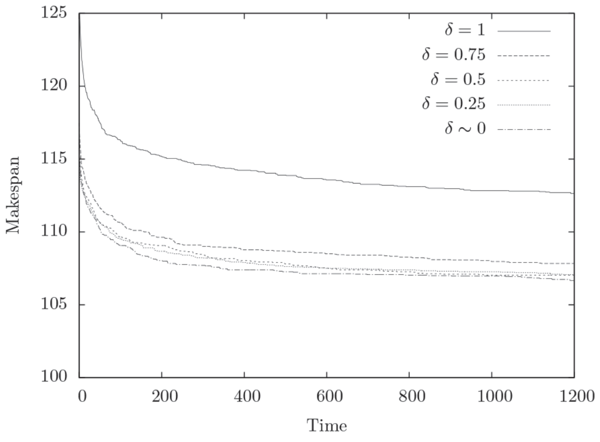


(b) $\mathcal{B}(\delta)$ Average Makespan

j1205_1_06_9op

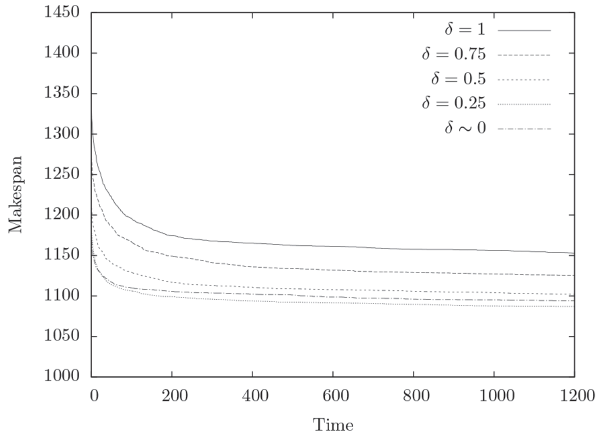


(c) $\mathcal{A}'(\delta)$ Average Makespan

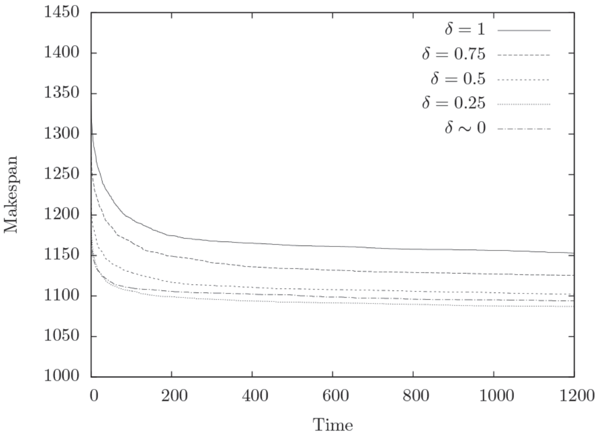


(d) $\mathcal{B}(\delta)$ Average Makespan

LA21_06_9op



(e) $\mathcal{A}'(\delta)$ Average Makespan



(f) $\mathcal{B}(\delta)$ Average Makespan

Fig. 9. Convergence of the GA for the instances LA31_06 derived from the LA31, combining the search spaces $\mathcal{A}'(\delta)$ and $\mathcal{B}(\delta)$ with different δ values and the coding-back (C) option. Each plot represents the evolution of the mean makespan of the population averaged for 30 runs.

Table 3
Summary of results for medium size instances with $\delta \in \{\sim 0, 0.25, 0.5, 0.75, 1\}$ averaged for each subset of instances with the same n. $timeLimit = 150s$

		$\mathcal{A}'(\delta \sim 0)$			$\mathcal{A}'(\delta = 0.25)$			$\mathcal{A}'(\delta = 0.5)$			$\mathcal{A}'(\delta = 0.75)$			$\mathcal{A}'(\delta = 1)$		
Instances	n	%Err. Best	%Err. Avg	%CV	%Err. Best	%Err. Avg	%CV	%Err. Best	%Err. Avg	%CV	%Err. Best	%Err. Avg	%CV	%Err. Best	%Err. Avg	%CV
j120	120	0.75	2.28	0.71	0.85	2.32	0.70	0.90	2.36	0.72	0.75	2.48	0.84	2.31	4.52	1.04
LA21-25	150	0.44	1.91	0.69	0.30	1.86	0.73	0.76	2.50	0.81	1.77	3.79	0.94	3.43	6.15	1.22
	T. Avg.	0.59	2.09	0.70	0.58	2.09	0.71	0.83	2.43	0.76	1.26	3.14	0.89	2.87	5.33	1.13

Table 4
Summary of results for the large instances with $\delta \sim 0$ and $\delta = 1$. The results are averaged for each set of instances with the same n. $timeLimit = 300s$

		$\mathcal{A}'(\delta \sim 0)$			$\mathcal{A}'(\delta = 1)$		
Instances	n	%Err. Best	%Err. Avg	%CV	%Err. Best	%Err. Avg	%CV
LA26-30	200	0.00	1.20	0.56	4.47	6.67	1.02
LA31-35	300	0.00	0.89	0.39	6.41	8.01	0.67
LA36-40	225	0.00	1.68	0.77	3.72	6.33	1.23
Tai_20_15	300	0.00	1.64	0.74	6.76	9.00	0.97
Tai_20_20	400	0.00	1.66	0.75	7.71	9.96	0.92
Tai_30_15	450	0.00	1.22	0.53	8.46	10.22	0.69
	T. Avg.	0.00	1.42	0.64	6.72	8.82	0.90

times and the best and average of the solutions reached in all runs were recorded. Apart from the average error, in order to study the stability of the algorithm with each configuration, we calculated the average Pearson coefficient of variation in percentage terms of the solutions (CV) computed as the ratio of standard deviation and the average of the solutions across all the runs.

Table 2 shows the results obtained for the instances derived from FT10, averaged for each group of 5 instances with the same values of p and P_r. As we can see, in average $\delta = 0.5$ is the best option, $\delta = 0.25$ the second, $\delta = 0.75$ the third and $\delta \sim 0$ and $\delta = 1$ the worst options. We can also see that the algorithm is more stable with lower values of δ, which was predictable since the size of the search space is in inverse ratio with δ. Looking at the results for each Pr, we see a similar behavior with some exceptions. Firstly, for instances with $Pr = 0.2$, the second best option is $\delta = 0.75$ instead of $\delta = 0.25$; which suggest that as these instances are more constrained, the search spaces may be smaller, and so reducing them may lead to lose high-quality solutions. On the other hand when $Pr = 0.6$, the best choice is $\delta = 0.25$, which may be explained by the fact that there are more options in these instances than in the remaining ones and so the GA has to cope with larger search spaces.

Table 3 shows the results the GA obtained with different configurations on the medium size instances. We can see that in average, the best options are those with $\delta \sim 0$ and $\delta = 0.25$, with $\delta \sim 0$ being better than $\delta = 0.25$ in the RCPSP instances, and $\delta = 0.25$ being the best one at the LA21-25 instances. Also, we can see that the larger δ the worse results returned by the GA.

Finally, Table 4 shows the results the GA yielded for the large instances considering two values of δ. We can observe that in every case, the genetic algorithm searching in $\mathcal{A}'(\delta \sim 0)$ layouts much better results and shows more stability than searching in $\mathcal{A}'(\delta = 1)$.

All things considered, we may conclude that the larger the instance is, the lower the value of the parameter δ should be, and that for the largest instances considered, $\delta \sim 0$ is the best option. In fact, we did some more experiments (not reported here) with instances derived from much larger JSP instances (up to 100×20), which support this conclusion. Also, we can say that the algorithm is more stable the lower δ is.

6.5. Comparison with other methods

In [2] the authors proposed two heuristic methods, MSB-DOS and STA-OMSB, and experimented with Cplex 12.2 on a MILP formulation they proposed. They report results from this study averaged for each subset of instances showing clearly that MSB-DOS is the best of the two heuristics proposed. Their target machine was AMD Athlon II 2.70 GHz using Matlab 2009b. Cplex was given a time limit of 3600 seconds. As far as we know, these are the only results on the SPSO published by other authors, for this reason in this study we compare our results with them.

Table 5
Summary of results from MSB-DOS, Cplex and GA averaged for the 10 instances in each of the 5 subsets of instances proposed in [2] . #Opt. is the number of instances in each subset which are optimally solved. Time is given in seconds. GA was run 30 times for each instance with $timeLimit = 5.00$ in each run, %Opt. is the percentage of runs GA reached an optimal solution in each subset

Sets				MSB-DOS			Cplex		GA		
#	n	p	q	GAP%	Time	#Opt.	Time	#Opt.	% Opt	Time	#Opt.
1	100	10	15	0.78	106.44	5.00	106.21	10.00	86.33	5.00	10.00
2	150	15	30	0.20	8.28	6.00	190.54	10.00	100.00	5.00	10.00
3	150	15	50	0.03	9.58	9.00	129.28	10.00	100.00	5.00	10.00
4	200	15	30	0.42	25.62	6.00	755.85	10.00	95.67	5.00	10.00
5	200	20	30	0.41	23.43	6.00	913.32	10.00	98.67	5.00	10.00
Average				0.37	34.67	6.40	419.04	10.00	96.13	5.00	10.00

Table 5 summarizes the results produced by MSB-DOS and Cplex, reported in [2], together with the results of our genetic algorithm (GA). In accordance with previous results, we chosen $\mathcal{X} = \mathcal{A}'(\delta = 1)$, and for the purpose of comparison with the other methods $timeLimit = 5s$. GA was run 30 times for each instance. Considering average values, Cplex can reach optimal solutions for all the instances taking 419.04 s. MSB-DOS takes much lower time, 34.67 s, but obtains optimal solutions for only 6.4 instances in each subset with average gap of 0.37%. Regarding GA, it was able to obtain optimal solutions for every instance in at least one run, and obtained optimal solutions in all 30 runs for all instances of subsets 2 and 3, while it obtained optimal solutions in 86.33%, 95.67% and 98.67% of the runs in subsets 1, 4 and 5 respectively. The hardest instance for GA was the 8th one in set 1 which was optimally solved in only 2 runs, and the second hardest was the 6th instance of set 4 which was optimally solved 18 times. Overall, GA obtained optimal solutions in all 30 runs for 43 instances and in 96.13% of the 1500 runs. Therefore, despite of the difference on the target machines, we can conclude that GA is faster than the methods proposed in the literature, and finds much better solutions than any other approximate method.

7. Conclusions and future work

We have seen that combining some non-trivial schedule builder with a population-based method such a genetic algorithm it is possible to obtain good solutions for a complex problem as the Scheduling Problem with Skilled Operators (SPSO) proposed in [2]. In this vein, the $SOG\&T$ has been exploited to devise a decoder which is the core of the genetic algorithm proposed to solve the SPSO. The success of this algorithm relies not only on the capability of $SOG\&T$ to generate schedules in different search spaces, but also in the proposed coding schema that encodes candidate solutions by means of a conventional permutation of tasks and a permutation with repetition of operators. We have demonstrated that for large instances, the best performance is obtained when the GA is restricted to search on a reduced non-dominant search space, while for small instances searching on a dominant unrestricted search space is preferable. It can be concluded that (i) the search spaces that yield the best balance between intensification and diversification are the most adequate and (ii) this balance strongly depends on the size of the instance.

As a result, we developed a tool that may be helpful for workforce managers to organize the operators' hiring and training processes.

As future work, we plan to enhance the genetic algorithm, to exploit the $SOG\&T$ in other frameworks and also to consider new characteristics of the scheduling problems.

We think that it would be interesting to continue the vein of this work by making the δ parameter a part of the chromosome. This way, the GA would adapt to each instance selecting the most convenient search space for it.

Our next step will be combining local search algorithms with the genetic algorithm. To this end, we will try to devise neighborhood structures for the SPSO from the proposed disjunctive model. In principle, we will try to extend the structures proposed in [15,25], but we will also consider new structures aiming at exploring changes in the assignment of operators.

Furthermore, we will tackle the SPSO in the framework of heuristic search using $SOG\&T$ as branching scheme, in this case the set of options $\mathcal{B}$ will be likely the best option as it gives rise to the smallest dominant search space. In this setting, the disjunctive graph model could help to devise consistent heuristics and dominance rules. It will also be interesting to consider and evaluate constraint programming approaches on the SPSO, both general frameworks as CPLEX CP Optimizer [26,38], or more specific algorithms such as the successful Solution Guided Search [6].

Acknowledgments

This research has been supported by the Spanish Government under research project TIN2013-46511-C2-2-P. Carlos Mencía was supported by SFI grant BEACON (09/IN.1/I2618).

References

[1] A. Agnetis, M. Flamini, G. Nicosia and A. Pacifici, A job-shop problem with one additional resource type, *Journal of Scheduling* **14**(3) (2011), 225–237.
[2] A. Agnetis, G. Murgia and S. Sbrilli, A job shop scheduling problem with human operators in handicraft production, *International Journal of Production Research* **52**(13) (2014), 3820–3831.
[3] C. Artigues, M. Gendreau, L.-M. Rousseau and A. Vergnaud, Solving an integrated employee timetabling and job-shop scheduling problem via hybrid branch-and-bound, *Computers and Operations Research* **36**(8) (2009), 2330–2340.
[4] C. Artigues, P. Lopez and P. Ayache, Schedule generation schemes for the job shop problem with sequence-dependent setup times: Dominance properties and computational analysis, *Annals of Operations Research* **138** (2005), 21–52.
[5] J.E. Beasley, Or-library: Distributing test problems by electronic mail, *Journal of the Operational Research Society* **41**(11) (1990), 1069–1072.
[6] J.C. Beck, T.K. Feng and J. Watson, Combining constraint programming and local search for job-shop scheduling, *INFORMS Journal on Computing* **23**(1) (2011), 1–14.
[7] A.J. Benavides, M. Ritt and C. Miralles, Flow shop scheduling with heterogeneous workers, *European Journal of Operational Research* **237**(2) (2014), 713–720.
[8] C. Bierwirth and D.C. Mattfeld, Production scheduling and rescheduling with genetic algorithms, *Evolutionary Computation* **7**(1) (1999), 1–17.
[9] I. Boulkaibet, L. Mthembu, F.B. de Lima Neto and T. Marwala, Finite element model updating using fish school search and volitive particle swarm optimization, *Integrated Computer-Aided Engineering* **22**(4) (2015), 361–376.
[10] P. Brucker, *Scheduling algorithms*, (4ed.), Springer, 2004.
[11] P. Brucker and S. Knust, *Complex Scheduling*, Springer, 2006.
[12] P. Brucker and R. Schlie, Job-shop scheduling with multi-purpose machines, *Computing* **45**(4) (1990), 369–375.
[13] J.-F. Chen and T.-J. Wu, A computational intelligence optimization algorithm: Cloud drops algorithm, *Integrated Computer-Aided Engineering* **21**(2) (2014), 177–188.
[14] L.F. Coletta, E.R. Hruschka, A. Acharya and J. Ghosh, Using metaheuristics to optimize the combination of classifier and cluster ensembles, *Integrated Computer-Aided Engineering* **22**(3) (2015), 229–242.
[15] M. Dell' Amico and M. Trubian, Applying tabu search to the job-shop scheduling problem, *Annals of Operational Research* **41** (1993), 231–252.
[16] N.D. Doulamis, P. Kokkinos and E. Varvarigos, Resource selection for tasks with time requirements using spectral clustering, *Computers, IEEE Transactions on* **63**(2) (2014), 461–474.
[17] P. Fortemps, Jobshop scheduling with imprecise durations: a fuzzy approach, *Fuzzy Systems, IEEE Transactions on* **5**(4) (1997), 557–569.
[18] M.R. Garey and D.S. Johnson, *Computers and Intractability; A Guide to the Theory of NP-Completeness*, W. H. Freeman & Co., New York, NY, USA, 1979.
[19] O. Guyon, P. Lemaire, E. Pinson and D. Rivreau, Solving an integrated job-shop problem with human resource constraints, *Annals of Operations Research* **213**(1) (2014), 147–171.
[20] L. Jia, Y. Wang and L. Fan, Multiobjective bilevel optimization for production-distribution planning problems using hybrid genetic algorithm, *Integrated Computer-Aided Engineering* **21**(1) (2014), 77–90.
[21] M.M. Joly, T. Verstraete and G. Paniagua, Integrated multi-fidelity, multidisciplinary evolutionary design optimization of counterrotating compressors, *Integrated Computer-Aided Engineering* **21**(3) (2014), 249–261.
[22] M. Kociecki and H. Adeli, Two-phase genetic algorithm for size optimization of free-form steel space-frame roof structures, *Journal of Constructional Steel Research* **90** (2013), 283–296.
[23] M. Kociecki and H. Adeli, Shape optimization of free-form steel space-frame roof structures with complex geometries using evolutionary computing, *Engineering Applications of Artificial Intelligence* **38** (2015), 168–182.
[24] A.V. Kononova, D.W. Corne, P. De Wilde, V. Shneer and F. Caraffini, Structural bias in population-based algorithms, *Information Sciences* **298** (2015), 468–490.
[25] P.J.M.v. Laarhoven, E.H.L. Aarts and J.K. Lenstra, Job shop scheduling by simulated annealing, *Operations Research* **40**(1) (1992), 113–125.
[26] P. Laborie, IBM CP Optimizer for detailed scheduling illustrated on three problems, in: *Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems, W.-J. van Hoeve and J. Hooker, eds, volume 5547 of Lecture Notes in Computer Science*, Springer Berlin Heidelberg, 2009, pages 148–162.
[27] X. Li, W. Li, X. Cai and F. He, A hybrid optimization approach for sustainable process planning and scheduling, *Integrated Computer-Aided Engineering* **22**(4) (2015), 311–326.
[28] R. Lostado, R. Martinez, B. MacDonald and P. Villanueva, Combining soft computing techniques and the finite element method to design and optimize complex welded products, *Integrated Computer-Aided Engineering* **22**(2) (2015), 153–170.
[29] J.M. Luna, J.R. Romero, C. Romero and S. Ventura, Reducing gaps in quantitative association rules: A genetic programming free-parameter algorithm, *Integrated Computer-Aided Engineering* **21**(4) (2014), 321–337.
[30] R. Mencía, M.R. Sierra, C. Mencía and R. Varela, Memetic algorithms for the job shop scheduling problem with operators, *Applied Soft Computing* **34** (2015), 94–105.
[31] R. Mencía, M.R. Sierra, C. Mencía and R. Varela, Schedule generation schemes and genetic algorithm for the scheduling problem with skilled operators and arbitrary precedence relations, In *Proceedings of the Twenty-Fifth International Conference on Automated Planning and Scheduling, ICAPS 2015*, Jerusalem, Israel, June 7–11, 2015, pages 165–173, 2015.
[32] R. Mencía, M.R. Sierra and R. Varela, Genetic algorithm for the job-shop scheduling with skilled operators, In *Bioinspired Computation in Artificial Systems – International Work-Conference on the Interplay Between Natural and Ar-*

tificial Computation, IWINAC 2015, Elche, Spain, June 1–5, 2015, Proceedings, Part II, pages 41–50, 2015.

[33] M.L. Pinedo, *Scheduling: Theory, Algorithms, and Systems*, Springer Publishing Company, Incorporated, 3rd edition, 2008.

[34] B. Roy and B. Sussman, Les problemes d'ordonnancements avec contraintes disjonctives, Notes DS no. 9 bis, SEMA, Paris, 1964.

[35] N. Siddique and H. Adeli, *Computational intelligence: synergies of fuzzy logic, neural networks and evolutionary computing*, John Wiley & Sons, 2013.

[36] M.F. Tasgetiren, M. Sevkli, Y.-C. Liang and M.M. Yenisey, A particle swarm optimization and differential evolution algorithms for job shop scheduling problem, *International Journal of Operations Research* **3**(2) (2006), 120–135.

[37] V. Valls, Á. Pérez and S. Quintanilla, Skilled workforce scheduling in service centres, *European Journal of Operational Research* **193**(3) (2009), 791–804.

[38] P. Vilím, P. Laborie and P. Shaw, Failure-directed search for constraint-based scheduling, In *CPAIOR '15: Proceedings of the 12th International Conference on Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems*, Springer-Verlag, 2015, pp. 437–453.

5.5. Technical Report: A Real-Life Vehicle and Crew Scheduling Problem for Extra-Urban Bus Passenger Routes

Se presenta el siguiente informe técnico.

- Technical Report: A Real-Life Vehicle and Crew Scheduling Problem for Extra-Urban Bus Passenger Routes.

Technical Report: A Real-Life Vehicle and Crew Scheduling Problem for Extra-Urban Bus Passenger Routes

Raúl Mencía, María R. Sierra, Carlos Mencía, Jorge Puente, Ramiro Varela

Departamento de Informatica. Universidad de Oviedo

Abstract

We consider a real-life vehicle and crew scheduling problem for an extra urban passengers company. In this problem there are a number of scheduled trips and the objective is to allocate a vehicle and a driver to each one subject to a number of constraints with the objective of minimize the number of drivers and vehicles. In addition to the commonly considered spatial, temporal and law derived constraints, our problem has some particularities that are not always considered in the literature. For example, the trips are organized in expeditions so that all trips of the same expedition must be allocated the same vehicle, a driver may travel as passenger in a vehicle before attending a trip, some vehicles must undergo programmed revisions over the scheduling horizon, or there are penalties for a driver staying out of home for a nigth, among others. We give a formal definition of the problem and propose a graphical model to represent problem instances and solutions. The purpose of this model is to be helpful for devising schedule builders and neighborhood structures to further develop hybrid metaheuristics to solve the problem.

Keywords: Vehicle and crew scheduling, graphical models

1. Introduction

The Vehicle and Crew Scheduling Problem (VCSP) has been studied over the last decades [5, 8, 15, 6, 12]. It consists in assigning vehicles and drivers to trips such that some constraints are satisfied and some objective functions, as the number of vehicles and crews, are optimized.

In the literature, this problem was firstly divided into two parts and solved following a hierarchical approach: the Vehicle Scheduling Problem (VSP), where vehicles are allocated to the trips [9, 3], and the Crew Scheduling Problem (CSP)[2, 1], where the drivers are assigned to the vehicles. However, as pointed in [6], it often happens that after an optimal allocation of vehicles to trips it is not possible to obtain a good or in some cases even a feasible allocation of drivers to vehicles. For this reason, some authors are of the opinion that it is better to solve the joint problem; i.e., the VCSP, at once. This is the option we consider herein.

Maybe, the most relevant approaches dealing with the joint VCSP are the following. In [6] the authors propose two different models for the problem and experiment with two benchmark sets, one derived from real instances and another one synthetic, termed as Huisman set, which is available (http://people.few.eur.nl/huisman/instances.htm) and that was further used by other authors. In [12] the problem is formulated by a combination of network flow model with a set partitioning/covering model and in [15] a Time-Space Network Approach is proposed. In both cases, results across the Huisman's instaces are reported as well. In all cases, the proposed solutions use column generation and Lagrangian relaxation. Given the complexity of the problem and the size of the real life instances and also the instances of the conventional benchmarks, these implementations often take too much time to reach optimal or near optimal solutions or tight lower bounds. As far as we known, only in a few cases metaheuristics were exploited to solve some simple version the VCSP or just some part of the problem. For example, in [13] the authors combine integer programming with some heuristics to solve an urban bus line problem with single depot. Also, in [8], a genetic algorithm is combined with mathematical programming techniques, and the problem is solved hierarchically.

Therefore, the possibility of exploiting metaheuristics to solve the full problem has not yet been investigated. In this work, we propose to use a similar approach to that used for the JSO in [10] and the JSSO in [11] and so we start proposing a graphical model to represent problem instances and solutions that hopefully will be useful to devise schedule builders and neighborhood structures, which could then be exploited in the framework of hybrid metaheuristics as memetic algorithms, for example.

In this work, we consider a real version of the VCSP from a spanish extra urban bus passengers company. As a consecuence of this, we have to deal with some specific constraints and characteristics derived from the labor rules of the company, the organization of the trips and the spanish and european laws. For example, the trips are organized in expeditions so as all the trips of an expedition have to be served by the same vehicle, all drivers must start and end in a depot and may travel in a vehicle as passengers before attending a trip, there are penalties for a driver staying at night out of home, and some more.

The remainder of this document is structured as follows: Section 2 contains the formulation of our particular version of the VCSP. Section 3 details the graphical model we propose to represent problem instances and solutions. Finally, we present in Secion 4 some conclusions and ideas for future work on developing metaheuristics using the proposed model.

2. Definition of the problem

The Vehicle and Crew Scheduling Problem for Passenger Routes (VCSP_PR) may be formulated as a Constraint Satisfaction Problem with Optimization (CSOP) as follows.

2.1. Problem data

We are given

d1. A set of *depots* or *bases* $\mathcal{B}$, each one with capacity to keep a sufficiently large number of vehicles parked and with a subset of them being repair shops, $\mathcal{BS} \subset \mathcal{B}$ where vehicles may undergo revision.

d2. An ordered set of *classes of vehicles* $\mathcal{C}$. For all $c_i, c_j \in \mathcal{C}$, $c_i < c_j$ iff $1 \leq i < j \leq |\mathcal{C}|$.

d3. A planning horizon $[0, \mathbf{H}]$.

d4. A set of *vehicles* $\mathcal{V}$, where each $V \in \mathcal{V}$ belongs to a base $Base(V) \in \mathcal{B}$ and to a class $Class(V) \in \mathcal{C}$. Each vehicle V is available along a subinterval of the planning horizon $[t^V_{init}, t^V_{end}] \subseteq [0, \mathbf{H}]$.

d5. A set of *drivers* $\mathcal{D}$, each driver D belonging to a base $Base(D) \in \mathcal{B}$ and having skills to drive vehicles of a subset of classes $Skills(D) \subset \mathcal{C}$; i.e., if $C \in Skills(D)$, then D can drive a vehicle of class C, otherwise D cannot.

d6. A set of *relief points* $\mathcal{RP}$, with $\mathcal{B} \subset \mathcal{RP}$, each one being a location (city, village, base, ...) where a vehicle may be parked for a time without being attended and a driver may rest for a time and may swap from a vehicle to another.

d7. A set of *routes* or *expeditions* $\mathcal{E}$ fixed along the planning horizon, where the expedition $E_i, 1 \leq i \leq |\mathcal{E}|$, is defined by a class $Class(E_i)$ and a sequence of M_i trips, $T_{i1}, \ldots, T_{iM_i}$. The set of trips from all expeditions is denoted $\mathcal{T}$. A trip $T \in \mathcal{T}$ is defined by its origin and destination, $Orig(T), Dest(T) \in \mathcal{RP}$, as well as its departure and arrival times, $Dep(T)$ and $Arr(T)$. The duration of the trip is $Dur(T) = Arr(T) - Dep(T)$. The length of the trip is $Dist(T)$.

d8. A set of *revision tasks* $\mathcal{RT}$. A subset of vehicles $\mathcal{V}_{\mathcal{R}} \subseteq \mathcal{V}$ must undergo revision between two consecutive expeditions in a shop of $\mathcal{BS}$. For a vehicle $V \in \mathcal{V}_{\mathcal{R}}$, the revision will take a time t^V_{REV} and must be performed after some traveling time and distance established by the time and distance intervals $[t^V_i, t^V_f] \subseteq [t^V_{init}, t^V_{end}]$ and $[d^V_i, d^V_f]$ respectively. For this purpose, the expedition $E_i = T_{i1}, \ldots, T_{iM_i}$ is associated with a number of candidate revision tasks in $\mathcal{RT}$, each one corresponding to a shop BS that is reachable after the last trip T_{iM_i}, either directly if $Dest(T_{iM_i}) = BS$ or through a deadhead of from $Dest(T_{iM_i})$ to BS (see next data item). In either case, $Orig(RT) = Dest(RT) = BS$, and if the revision task is done on the vehicle V, then $Dep(RT)$, the starting time of the revision task, should be allocated a value in $[t^V_i, t^V_f]$ and $Arr(RT) = Dep(RT) + t^V_{REV}$. After the revision, the vehicle V may be assigned to the first trip of another expedition.

d9. A set of candidate *deadheads* $\mathcal{DH}$. Each $DH \in \mathcal{DH}$ represents a shift of a vehicle without passengers between two relief points $Orig(DH)$ and $Dest(DH)$, which cannot be the same, with duration $Dur(DH) > 0$ and length or distance $Dist(DH) \leq MaxDistDH$. If a deadhead is taken, the shift must be done over a time interval within a time window $[MinDep(DH), MaxArr(DH)]$. There are four types of deadheads:

   i. A *starting deadhead* (T_{DH}, T). T is the first trip of an expedition and T_{DH} is the set of candidate origins of the deadhead. In principle, T_{DH} is a subset of trips such that for each $T' \in T_{DH}$, T' is the last trip of its expedition, $Dest(T') = Orig(DH)$ and $Dep(T) - Arr(T') \geq Dur(DH)$. However, if $Orig(DH)$ is a base B, then B has to be included in T_{DH} to represent the departure of a vehicle from the base to its first trip. Also, if $Orig(DH)$ is a shop BS in which a revision task RT may take place with $Dep(T) - Arr(RT) \geq Dur(DH)$, then RT must be included in T_{DH} as well. In this case, $MaxArr(DH) = Dep(T)$ and $MinDep(DH) = \min\ \{Arr(T'); T' \in T_{DH}\}$. Notice that if a base B is in T_{DH}, as $Arr(B) = 0$, then $MinDep(DH) = 0$.

ii. An *ending deadhead* (T, B). T being the last trip of an expedition and B a base. $Orig(DH) = Dest(T)$, $Dest(DH) = B$, and it must hold that $Dur(DH) \leq \mathbf{H} - Arr(T)$. Also, $MaxArr(DH) = \mathbf{H}$ and $MinDep(DH) = Arr(T)$.

iii. An *input revision deadhead* (T, RT). T being the last trip of some expedition and RT a revision task is a shop BS. In this case $Orig(DH) = Dest(T)$ and $Dest(DH) = BS$. The time window $[MinDep(DH), MaxArr(DH)]$ will depend on the vehicle V undergoing revision: $MinDep(DH) = Arr(T)$, $MaxArr(DH) = t_f^V - t_{REV}^V$.

d10. A set of labor rules that control the daily duties, maximum driving time without rest, rest time between driving periods, rest time between two consecutive daily journeys, holidays, etc. These rules are partly established by the government and partly by the working agreements of the drivers with the company. They are detailed in Appendix A (or in http://www....)

d11. A set of rules that specify time periods for which a driver may incur allowance expenses for meals and overnight hotel stays. These rules are also detailed in Appendix A (or in http://www....).

d12. A number of parameters $P_N; P_{DH} \in [0, 1]$; $MAX_{BS} \in \mathbb{N}^+, BS \in \mathcal{BS}$; $MaxDistDH \geq 0$; $MAX_{LD} \in \mathbb{N}^+$.

2.2. Decision variables

There are five types of decision variables:

- $x_T; T \in \mathcal{T}$
- $y_{DH}, t_{DH}; DH \in \mathcal{DH}$.
- $z_{RT}, t_{RT}; RT \in \mathcal{RT}$.

Variables x_T are assigned tuples of the form (V, D, LD) where $V \in \mathcal{V}$ is the vehicle of the trip T, $D \in \mathcal{D}$ is the driver of the vehicle V in the trip and $LD \subset \mathcal{D}$ is a, possibly empty, set of drivers who travel in the vehicle V as passengers in the trip, with $|LD| \leq MAX_{LD}$.

Variables y_{DH} may be assigned either a null value or a tuple (V, D, LD). If y_{DH} is not null, then t_{DH} must be assigned a starting time for the deadhead.

A variable z_{RT}, with $RT \in \mathcal{RT}$, is assigned 1 if RT is chosen to review a vehicle V and 0 otherwise. If $z_{RT} = 1$ then t_{RT} must be assigned a starting time for the revision task.

The assignment must fulfill a set of constraints and try to optimize some objective functions which are enumerated in the following.

2.3. Constraints

c1. Every trip $T \in \mathcal{T}$ must be assigned a tuple $x_T = (V, D, LD)$ where V and D cannot be null and LD may or may not be null.

c2. For an expedition E_i, all trips must be assigned the same vehicle; i.e., if $x_{T_{ij}} = (V, D, LD)$ and $x_{T_{ik}} = (V', D', LD'), k \neq j$, then $V = V'$.

c3. A trip T of an expedition E must be assigned a vehicle of the appropriate class and a driver with appropriate skills; i.e., if $x_T = (V, D, LD)$ then
   i. $Class(V) \geq Class(E)$, and
   ii. $Class(V) \in Skills(D)$.

c4. If a deadhead DH is assigned a tuple $y_{DH} = (V, D, LD)$, the driver must be skilled to drive the vehicle; i.e., $Class(V) \in Skills(D)$, and the deadhead must be assigned a starting time $t_{DH} = t \in [MinDep(DH), MaxArr(DH) - Dur(DH)]$.

c5. For each $V \in \mathcal{RV}$, one and only one $RT = RT_V^{BS} \in RT_V^{BS_V}$ must be assigned; i.e., $z_{RT} = 1$; and in this case, t_{RT} is assigned a starting time t. If d is the distance covered by V from the beginning of the scheduling horizon, the following conditions must hold:

$$((t \geq t_i^V) \vee (d \geq d_i^V)) \wedge (t \leq t_f^V) \wedge (d \leq d_f^V) \tag{1}$$

c6. For a revision shop $BS \in \mathcal{BS}$ the maximum number of vehicles that can be being repaired at any time is MAX_{BS}.

c7. The percentage of drivers staying overnight out of home must not exceed the $P_N \times 100\%$ of them.

c8. The cost incurred by deadheads must not exceed the $P_{DH} \times 100\%$ of the total cost due to the trips.

c9. The *duty* of a driver D along the scheduling horizon $\mathbf{H}$ is a sequence of tasks $\tau_1^D, \tau_2^D, \ldots, \tau_{n_D}^D$, where τ_i^D starts at time $Dep(\tau_i^D)$ and finishes at $Arr(\tau_i^D)$ and has origin and destinations $Orig(\tau_i^D)$ and $Dest(\tau_i^D)$ respectively. The duty must satisfy the following conditions:
   i. each τ_i^D is either a trip, a deadhead or a resting period in a relief point.
   ii. $Dep(\tau_1^D) = 0$, $Dep(\tau_i^D) \leq Arr(\tau_i^D) = Dep(\tau_{i+1}^D)$, $1 \leq i < n_D$, and $Arr(\tau_{n_D}^D) = \mathbf{H}$.

iii. $Orig(\tau_1^D) = Base(D)$, $Orig(\tau_{i+1}^D) = Dest(\tau_i^D), 1 \leq i < n_D$, and $Dest(\tau_{n_D}^D) = Base(D)$.

iv. If $\tau_i^D = T \in \mathcal{T}$, then either $x_T = (D, V, LD)$, and so D is the driver of the trip, or $x_T = (D', V, LD), D' \neq D \in LD$ and so D goes in the trip as passenger.

v. If τ_i^D is a deadhead DH, then $y_{DH} = (D, V, LD)$ or $y_{DH} = (D', V, LD)$, $D' \neq D \in LD$. Also, $Dep(\tau_i^D) = t_{DH}$ and $Arr(\tau_i^D) = t_{DH} + Dur(DH)$.

vi. If τ_i^D is a resting period in a relief point RP, then $Orig(\tau_i^D) = Dest(\tau_i^D) = RP$ and $\tau_{(i-1)}^D$ and $\tau_{(i+1)}^D$ are trips or deadheads.

vii. This duty must satisfy all the constraints derived from the labor rules (see Appendix 1 or http://www....).

c10. Analogously, the *vehicle block* for V along the planning horizon $\mathbf{H}$ is a sequence of tasks $\tau_1^V, \tau_2^V, \ldots \tau_{n_V}^V$ where τ_i^V starts at time $Dep(\tau_i^V)$ and finishes at $Arr(\tau_i^V)$ and has origin and destinations $Orig(\tau_i^V)$ and $Dest(\tau_i^V)$ respectively. The vehicle block must satisfy the following conditions:

i. each τ_i^V is either a trip, or a deadhead, or a parking period in a relief point, or a revision period in a revision shop.

ii. $Dep(\tau_1^V) = 0$, $Dep(\tau_i^V) \leq Arr(\tau_i^V) = Dep(\tau_{i+1}^V)$, $1 \leq i < n_V$, and $Arr(\tau_{n_V}^V) = \mathbf{H}$.

iii. $Orig(\tau_1^V) = Base(D)$, $Orig(\tau_{i+1}^V) = Dest(\tau_i^V), 1 \leq i < n_V$, and $Dest(\tau_{n_V}^V) = Base(V)$.

iv. If $\tau_i^V = T \in \mathcal{T}$, then $x_T = (D, V, LD)$, and so V is the vehicle of the trip.

v. If τ_i^V is a deadhead DH, then $y_{DH} = (D, V, LD)$ and $Dep(\tau_i^V) = t_{DH}$ and $Arr(\tau_i^V) = t_{DH} + Dur(DH)$.

vi. If τ_i^V is a parking period in a relief point RP, then $Orig(\tau_i^V) = Dest(\tau_i^V) = RP$ and each one of $\tau_{(i-1)}^V$ and $\tau_{(i+1)}^V$ is either a trip or a deadhead or a revision task.

vii. If τ_i^V is a revision task RT in a shop BS, then $Orig(\tau_i^D) = Dest(\tau_i^D) = BS$ and $\tau_{(i-1)}^D$ and $\tau_{(i+1)}^D$ may be resting periods, or trips, or deadheads. Also, $z_{RT} = 1$, $Dep(\tau_i^V) = t_{RT}$ and $Arr(\tau_i^V) = t_{RT} + t_{REV}^V$.

2.4. Objective functions

There are three main of them, all of which must be minimized

o1. The total number of vehicles required N_V calculated as

$$N_V = |\{V \in \mathcal{V}; x_T = (V, D, LD), T \in \mathcal{T}\}| \tag{2}$$

o2. The total number of drivers required N_D calculated as

$$N_D = |\{D \in \mathcal{D}; x_T = (V, D, LD), T \in \mathcal{T}\}| \tag{3}$$

o3. The total cost incurred by deadheads C_{DH} calculated as

$$C_{DH} = \sum_{DH \in \mathcal{DH}, y_{DH} \neq null} Cost(DH). \tag{4}$$

o4. The cost of the allowance expenses of the drivers, denoted C_{ALLEXP}.

o5. The cost due to vehicles and drivers not arriving at their bases at the end of the planning horizon, denoted C_{NOTARR}.

In addition to the previous objectives, some others are usual such as, for example, minimize the cost due to the drivers traveling as passengers, minimize the parking periods of the vehicles out of a base, or optimize the working conditions of the drivers, for example maximizing the resting time between daily duties, among others.

3. Graphical model

In order to represent problem instances and solutions of the VCSP_PR, as well as to devise schedule generation schemes or neighborhood structures, we propose the following graphical model. In this model, a problem graph represents the sets of bases, trips, deadheads, revision tasks and relieves of drivers and vehicles that may take part in a solution. A solution graph is a subgraph of the problem graph with labels in both arcs and nodes that represents an actual solution of the problem.

3.1. Problem graph

A VCSP_PR instance is represented by a directed graph $\mathbf{G} = (\mathbf{V}, \mathbf{A})$ where $\mathbf{V}$ is the set of nodes, and $\mathbf{A}$ is the set of arcs. A node in $\mathbf{V}$ represents either a base or a period of activity for some vehicle, driver or both. In general, an arc in $\mathbf{A}$ represents candidate transfers of at most one vehicle and a number of drivers between two different nodes X and Y such that $Dest(X) = Orig(Y) = RP \in \mathcal{RP}$. So it actually represents a period of inactivity for the vehicle and the drivers at that relief point.

3.1.1. Nodes

The set of nodes is of the form $\mathbf{V} = \mathcal{B}^+ \cup \mathcal{B}^- \cup \mathcal{T} \cup \mathcal{DH} \cup \mathcal{RT}$. Nodes in $\mathcal{T}$ and $\mathcal{DH}$ represent periods of activity for vehicles and drivers, nodes in $\mathcal{RT}$ represent periods of activity for vehicles only, while nodes in $\mathcal{B}^+$ and $\mathcal{B}^-$ represent bases acting as sources and sinks respectively of vehicles and drivers; i.e., they are starting and ending points for drivers and vehicles at times 0 and $\mathbf{H}$ respectively. $B^+ \in \mathcal{B}^+$ and $B^- \in \mathcal{B}^-$ will denote the base B acting as source and sink respectively.

Each node $X \in \mathbf{V}$ has origin and destination in the relief points $Orig(X)$ and $Dest(X)$ respectively. For nodes $B^+ \in \mathcal{B}^+$ and $B^- \in \mathcal{B}^-$, $Orig(B^+) = Dest(B^+) = Orig(B^-) = Dest(B^-) = B$ and for nodes in $\mathcal{RT}$, $Orig(X) = Dest(X) = BS$ if $X = RT_V^{BS}$. However, for nodes in $\mathcal{T}$ and $\mathcal{DH}$, $Orig(X) \neq Dest(X)$. Each of these last nodes represents a potential transfer of a vehicle, a driver and a set of drivers who travel as passengers between the locations $Orig(X)$ and $Dest(X)$ during a time $[Dep(X), Arr(X)]$. Moreover, each node has a given duration $Dur(X) = Arr(X) - Dep(X)$, and so it is greater than 0 for nodes in $\mathcal{T}$, $\mathcal{DH}$ and $\mathcal{RT}$, while it is taken to be 0 for nodes in $\mathcal{B}^+$ and $\mathcal{B}^-$. Also, each node X has a distance $Dist(X)$, which is 0 for nodes in $\mathcal{B}^+$, $\mathcal{B}^-$ and $\mathcal{RT}$, while it is greater than 0 for nodes in $\mathcal{T}$ and $\mathcal{DH}$.

In general, when a node X is selected to take part in a schedule, it is labeled with a triplet (V, D, LD). We will denote $Vechicle(X) = V$ and $Crew(X) = \{D\} \cup LD$. In accordance with all the aforementioned, the labels will have the following restrictions depending on the type of the node:

- Nodes in $\mathcal{B}^+$ and $\mathcal{B}^-$ are not labeled.
- For nodes $T \in \mathcal{T}$, LD may be null (i.e., empty), but V and D cannot be null. Furthermore, V must be of an appropriate class for T and D must be of an appropriate class for V.
- For nodes in $\mathcal{DH}$, LD may be null, but V and D cannot be null. D must be of an appropriate class for V.
- For nodes in $\mathcal{RT}$, V is not null, but D and LD are null.

3.1.2. Arcs

For each pair of nodes $X, Y \in \mathbf{V}$ such that

$$(Dest(X) = Orig(Y) = RP \in \mathcal{RP}) \wedge (Dep(Y) \geq Arr(X)) \qquad (5)$$

it is possible transferring the vehicle or some driver from the activity in node X to the activity in node Y, the vehicle and the drivers being idle over the interval $[Arr(X), Dep(Y)]$. So, there is an arc $\langle X, Y\rangle \in \mathbf{A}$ to express this option[1]. If an arc is selected to build a schedule, it will be labeled with a pair (V, LD) which is subject to some restrictions that depend on the type of the source and destination nodes of the arc, in particular if the source or destination is a base, all vehicle and drivers must belong to that base.

There are 18 different classes of arcs organized in 5 disjoint subsets, $\mathbf{A} = A_0 \cup A_1 \cup A_2 \cup A_3 \cup A_4$, where:

- Arcs in A_0 are called *starting-ending arcs*. Either the origin or the destination is a base and V or LD but not both may be null. The may be of the form

  1. $\langle B^+, T\rangle$, a starting arc, T being the first trip of an expedition with origin in the base B.
  2. $\langle T, B^-\rangle$, an ending arc, T being the last trip of an expedition with destination in the base B.

- Arcs in A_1, called *trip arcs*, are of the form

  3. $\langle T, T'\rangle$, T and T' being two consecutive trips of the same expedition. V is the same as in T and T' and so it cannot be null, while LD may or may not be null.

- Arcs in A_2 are called *deadhead arcs*. Either the origin or the destination, but not both, are deadheads. If one of these arcs is selected for a schedule, V and LD, but not both, may be null. There are the following types of deadhead arcs.

  For each starting deadhead $DH = (T_{DH}, T)$ and for each $X \in T_{DH}$ there is an incoming arc $\langle X, DH\rangle$ which may be of one of the three forms:

  4. $\langle B^+, DH\rangle$ if $Orig(DH)$ is the base B.

[1]To avoid confusion with the notation (X, Y) used to represent trips or deadheads between relief points X and Y, we use the notation $\langle X, Y\rangle$ for the arc from node X to node Y in the problem and solution graphs.

5. $\langle BS, DH\rangle$ if $Orig(DH)$ is the shop BS and there is a revision task RT in BS such that $Arr(RT) \leq Dep(DH)$. This is an output-revision-deadhead arc.
6. $\langle T', DH\rangle$ if T' is the first trip of some expedition.

If DH is selected for a schedule, then at least one of these incoming arcs must be selected as well. In one and only one of the selected arcs V must be the same as in DH and in the remaining ones the vehicle must be null. On the other hand, there is only one outgoing arc, which must be selected if DH is selected, V being the same as in DH:

7. $\langle DH, T\rangle$.

For each ending deadhead $DH = (T, B)$, there are two arcs, which must also be selected if DH is selected, V being the same as in DH in both arcs, while LD may be null in any of them:

8. $\langle T, DH\rangle$.
9. $\langle DH, B^-\rangle$.

For each input revision deadhead $DH = (T, RT)$ there are two input-revision-deadhead arcs, which must be selected if RT is selected to repair the vehicle V after attending the trip T, V is the repaired vehicle and LD may be null in both arcs :

10. $\langle T, DH\rangle$.
11. $\langle DH, RT\rangle$.

- Arcs in A_3 are called *relief arcs*. They represent transfers of drivers between trips or deadheads and so if a relief arc $\langle X, Y\rangle$ is selected then both X and Y must be selected as well. V is null and $LD = Crew(X) \cap Crew(Y) \neq \oslash$. A relief arc may be of the following forms:

  12. $\langle T, T'\rangle$, being T and T' trips from different expeditions such that T is not the last or T' is not the first in their respective expeditions.
  13. $\langle T, DH\rangle$, if T is not the last trip of its expedition.
  14. $\langle DH, T\rangle$, if $DH = (T_{DH}, T')$ and $T \neq T'$.
  15. $\langle DH_1, DH_2\rangle$.

- Arcs in A_4 are called *transfer arcs*. These arcs represent the transfer of the vehicle and/or some drivers from one expedition to another so they are of the form

16. $\langle T, T' \rangle$, being T the last trip of an expedition and T' the first trip of another expedition. If chosen, LD or V are not null and if V is not null then it is the same as in T and T'.

- Arcs in A_5 are called *revision arcs*. In addition to the input-revision-deadhead and output-revision-deadhead arcs, there are the following two types of revision arcs. In both cases LD is null and obviously V is not null, if the arc is selected for a schedule.

17. $\langle RT, T \rangle$, if T is the first trip of some expedition, RT is done in shop BS, $Orig(T) = BS$ and $Dep(T) \geq MarArr(RT)$.
18. $\langle T, RT \rangle$, if T is the last trip of some expedition, RT is done in shop BS and $Dest(T) = BS$. In this case, $MinDep(RT) = Arr(T)$.

3.2. Solution graph

From the graph $\mathbf{G} = (\mathbf{V}, \mathbf{A})$ that represents a problem instance, allocating values to the decision variables x_T, y_{DH} and z_{RT} may be viewed as a process in which some nodes and arcs get labeled with tuples of the form (V, D, LD) and (V, LD) respectively, such that the following constraints are satisfied:

1. All nodes $T \in \mathcal{T}$ must be assigned a label.
2. For each driver D and for each vehicle V involved in a schedule, the labels must define a closed tour; i.e., a tour starting and finishing in its base. For each driver D its tour defining a feasible duty and for each vehicle V its tour defining a feasible vehicle block.
3. For each vehicle $V \in \mathcal{V}_R$, one and only one revision node $RT \in \mathcal{RT}$ must be labeled and included in the vehicle block in a position such that the revision is done after a time t and a traveling distance d satisfying condition (1).

So, a solution graph is a subgraph of $\mathbf{G}$ denoted $\mathbf{G_S} = (\mathbf{V_S}, \mathbf{A_S})$, where $\mathbf{V_S} \subset \mathbf{V}$ and $\mathbf{A_S} \subset \mathbf{A}$, in which arcs and nodes are labeled in such a way that the above conditions are satisfied. Each deadhead DH and each revision task RT included in $\mathbf{V_S}$ get an interval of possible starting times. From these intervals, the values of the remaining decision variables, namely t_{DH} and t_{RT} for all selected deadheads and revision tasks, may be easily established in a postprocessing procedure.

4. Conclusions and future work

In previous sections we have given a formal definition for the particular VCSP of an extra urban bus passengers company. It is a quite general definition that in some practical situations may be simplified. For example, in some cases there is not necessary to consider revision tasks of the vehicles over the scheduling horizon, or even this scheduling horizon may be restricted to only one journey, what notably simplifies the problem. Also, it could be simplified to the version considered in [6], where no drivers as passengers are considered, for example. To represent problem instances and solutions we propose a graphical model inspired in the disjuntive models commonly used for the family of job shop scheduling problems. With this representation a solving procedure may be viewed as a process of building a solution graph from the problem graph in which the selected arcs get labelled with vehicles and crews and the starting and ending times of the selected deadheads fixed. Therefore, the graphical model is expected to be helpful to devise schedule builders similar to the *G&T* algorithm [4] and some of its variants [14, 11] and also neighborhood structures in a similar way as done for job shop scheduling problems in [7, 10]. With these elements our purpose is to develop hybrid metaheuristics as memetic algorithms to obtain efective solutions to the considered version of the VCSP in an efficient way.

References

[1] J. Beasley and B. Cao. A dynamic programming based algorithm for the crew scheduling problem. *Computers & Operations Research*, 25(7):567–582, 1998.

[2] G. Edwards. An approach to the crew scheduling problem. *New Zealand Operational Research*, 8:153–171, 1980.

[3] R. Freling, A. P. Wagelmans, and J. M. P. Paixão. Models and algorithms for single-depot vehicle scheduling. *Transportation Science*, 35(2):165–180, 2001.

[4] B. Giffler and G. L. Thompson. Algorithms for solving production scheduling problems. *Operations Research*, 8:487–503, 1960.

[5] D. Huisman. *Integrated and dynamic vehicle and crew scheduling.* 2004.

[6] D. Huisman, R. Freling, and A. P. Wagelmans. Multiple-depot integrated vehicle and crew scheduling. *Transportation Science*, 39(4):491–502, 2005.

[7] P. J. M. v. Laarhoven, E. H. L. Aarts, and J. K. Lenstra. Job shop scheduling by simulated annealing. *Operations Research*, 40(1):pp. 113–125, 1992.

[8] B. Laurent and J.-K. Hao. Simultaneous vehicle and crew scheduling for extra urban transports. In *International Conference on Industrial, Engineering and Other Applications of Applied Intelligent Systems*, pages 466–475. Springer, 2008.

[9] B. Laurent and J.-K. Hao. Iterated local search for the multiple depot vehicle scheduling problem. *Computers & Industrial Engineering*, 57(1):277–286, 2009.

[10] R. Mencía, M. R. Sierra, C. Mencía, and R. Varela. Memetic algorithms for the job shop scheduling problem with operators. *Applied Soft Computing*, 34:94–105, 2015.

[11] R. Mencía, M. R. Sierra, C. Mencía, and R. Varela. Genetic algorithms for the scheduling problem with arbitrary precedence relations and skilled operators. *Integrated Computer-Aided Engineering*, 23(3):269–285, 2016.

[12] M. Mesquita and A. Paias. Set partitioning/covering-based approaches for the integrated vehicle and crew scheduling problem. *Computers & Operations Research*, 35(5):1562–1575, 2008.

[13] M. M. Rodrigues, C. C. de Souza, and A. V. Moura. Vehicle and crew scheduling for urban bus lines. *European Journal of Operational Research*, 170(3):844 – 862, 2006.

[14] M. R. Sierra, C. Mencía, and R. Varela. New schedule generation schemes for the job-shop problem with operators. *Journal of Intelligent Manufacturing*, 26(3):511–525, 2015.

[15] I. Steinzen, V. Gintner, L. Suhl, and N. Kliewer. A time-space network approach for the integrated vehicle-and crew-scheduling problem with multiple depots. *Transportation Science*, 44(3):367–382, 2010.

Capítulo 6

INFORME SOBRE LA CALIDAD DE LAS PUBLICACIONES

En este capítulo se incluye información sobre la calidad y el factor de impacto de las publicaciones presentadas en el capítulo anterior.

Artículo 1

- Raúl Mencía, María R. Sierra, Carlos Mencía, Ramiro Varela. A genetic algorithm for job-shop scheduling with operators enhanced by weak Lamarckian evolution and search space narrowing. *Natural Computing*. 13(2): 179-192. 2014. DOI: 10.1007/s11047-013-9373-x.
    - Estado: **Publicado.**
    - Factor de impacto (JCR 2014): 0.757
    - Factor de impacto (5 años): Not Available
    - Categorías:
        - COMPUTER SCIENCE, THEORY & METHODS: 65/102(Q3), **T2**

Artículo 2

- Raúl Mencía, María R. Sierra, Carlos Mencía, Ramiro Varela. Memetic Algorithms for the Job Shop Scheduling Problem with Operators. *Applied Soft Computing*, 2015, 34: 94-105. 2015. DOI: 10.1016/j.asoc.2015.05.004.
    - Estado: **Publicado.**
    - Factor de impacto (JCR 2015): 2.857
    - Factor de impacto (5 años): 3.288
    - Categorías:
        - COMPUTER SCIENCE, ARTIFICIAL INTELLIGENCE: 21/130(Q1), **T1**

    - COMPUTER SCIENCE, INTERDISCIPLINARY APPLICATIONS: 16/104(Q1), **T1**

Artículo 3

- Raúl Mencía, María R. Sierra, Carlos Mencía, Ramiro Varela. Schedule Generation Schemes and Genetic Algorithm for the Scheduling Problem with Skilled Operators and Arbitrary Precedence Relations. *Twenty-Fifth International Conference on Automated Planning and Scheduling. (ICAPS 2015).* 165-173.

  - Estado: **Publicado.**
  - CORE: A*

Artículo 4

- Raúl Mencía, María R. Sierra, Carlos Mencía, Ramiro Varela. Genetic Algorithm for the scheduling problem with arbitrary precedence relations and skilled operators. *Integrated Computer-Aided Engineering*. 23(3): 269-285. 2016. DOI: 10.3233/ICA-160519.

  - Estado: **Publicado.**
  - Factor de impacto (JCR 2015): 4.981
  - Factor de impacto (5 años): 2.904
  - Categorías:
    - COMPUTER SCIENCE, ARTIFICIAL INTELLIGENCE: 5/130(Q1), **T1**
    - COMPUTER SCIENCE, INTERDISCIPLINARY APPLICATIONS: 2/104(Q1), **T1**
    - ENGINEERING, MULTIDISCIPLINARY: 1/85(Q1), **T1**

Bibliografía

[1] A. Agnetis, M. Flamini, G. Nicosia, and A. Pacifici. A job-shop problem with one additional resource type. *Journal of Scheduling*, 14(3):225–237, 2011.

[2] A. Agnetis, G. Murgia, and S. Sbrilli. A job shop scheduling problem with human operators in handicraft production. *International Journal of Production Research*, 52(13):3820–3831, 2014.

[3] C. Artigues, P. Lopez, and P. Ayache. Schedule generation schemes for the job shop problem with sequence-dependent setup times: Dominance properties and computational analysis. *Annals of Operations Research*, 138:21–52, 2005.

[4] E. Balas, N. Simonetti, and A. Vazacopoulos. Job shop scheduling with setup times, deadlines and precedence constraints. *Journal of Scheduling*, 11:253–262, 2008.

[5] P. Baptiste, C. Le Pape, and W. Nuijten. *Constraint-Based Scheduling*. Kluwer Academic Publishers, Norwell, MA, USA, 2001.

[6] P. Baptiste, C. Le Pape, and W. Nuijten. *Constraint-based scheduling: applying constraint programming to scheduling problems*, volume 39. Springer Science & Business Media, 2012.

[7] J. E. Beasley. Or-library: Distributing test problems by electronic mail. *Journal of the Operational Research Society*, 41(11):1069–1072, 1990.

[8] C. Bierwirth. A generalized permutation approach to job shop scheduling with genetic algorithms. *OR Spectrum*, 17:87–92, 1995.

[9] C. Bierwirth and D. C. Mattfeld. Production scheduling and rescheduling with genetic algorithms. *Evolutionary Computation*, 7(1):1–17, Mar. 1999.

[10] P. Brucker. *Scheduling algorithms (4. ed.)*. Springer, 2004.

[11] P. Brucker, B. Jurisch, and B. Sievers. A branch and bound algorithm for the job-shop scheduling problem. *Discrete Applied Mathematics*, 49:107–127, 1994.

[12] B. Çaliş and S. Bulkan. A research survey: review of ai solution strategies of job shop scheduling problem. *Journal of Intelligent Manufacturing*, 26(5):961–973, 2015.

[13] J. Carlier and E. Pinson. An algorithm for solving the job-shop problem. *Management Science*, 35(2):164–176, 1989.

[14] L. Davis. Applying adaptive algorithms to epistatic domains. In *IJCAI*, volume 85, pages 162–164, 1985.

[15] M. Dell' Amico and M. Trubian. Applying tabu search to the job-shop scheduling problem. *Annals of Operational Research*, 41:231–252, 1993.

[16] M. Dorigo, V. Maniezzo, and A. Colorni. Ant system: optimization by a colony of cooperating agents. *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, 26(1):29–41, 1996.

[17] M. Dorigo and T. Stützle. Ant colony optimization: overview and recent advances. *Handbook of metaheuristics*, 2010.

[18] M. Garey and D. Johnson. *Computers and Intractability*. Freeman, 1979.

[19] B. Giffler and G. L. Thompson. Algorithms for solving production scheduling problems. *Operations Research*, 8:487–503, 1960.

[20] F. Glover. Heuristics for integer programming using surrogate constraints. *Decision Sciences*, 8(1):156–166, 1977.

[21] F. Glover. Tabu search part i. *ORSA Journal on computing*, 1(3):190–206, 1989.

[22] F. Glover. Tabu search part ii. *ORSA Journal on computing*, 2(1):4–32, 1990.

[23] F. Glover and M. Laguna. Tabu search. In *Handbook of Combinatorial Optimization*, pages 3261–3362. Springer New York, 2013.

[24] F. Glover, M. Laguna, and R. Martí. Fundamentals of scatter search and path relinking. *Control and cybernetics*, 29(3):653–684, 2000.

[25] M. A. González, C. R. Vela, and R. Varela. A new hybrid genetic algorithm for the job shop scheduling problem with setup times. In *Proceedings of the Eighteenth International Conference on Automated Planning and Scheduling (ICAPS-2008)*, Sidney, 2008. AAAI Press.

[26] M. A. González, C. R. Vela, and R. Varela. Scatter search with path relinking for the flexible job shop scheduling problem. *European Journal of Operational Research*, 245(1):35 – 45, 2015.

[27] I. González Rodríguez, C. R. Vela, and J. Puente. A memetic approach to fuzzy job shop based on expectation model. In *Proceedings of IEEE International Conference on Fuzzy Systems, FUZZ-IEEE2007*, pages 692–697, London, 2007. IEEE.

[28] R. Graham, E. Lawler, J. Lenstra, and A. Rinnooy Kan. Optimization and approximation in deterministic sequencing and scheduling: a survey. *Annals of Discrete Mathematics*, 4:287–326, 1979.

[29] J. H. Holland. *Adaptation in Natural and Artificial Systems: An Introductory Analysis with Applications to Biology, Control and Artificial Intelligence*. MIT Press, Cambridge, MA, USA, 1992.

[30] D. Huisman, R. Freling, and A. P. Wagelmans. Multiple-depot integrated vehicle and crew scheduling. *Transportation Science*, 39(4):491–502, 2005.

[31] J. Kennedy and R. C. Eberhart. Particle swarm optimization. In *Proceedings of the IEEE International Conference on Neural Networks*, pages 1942–1948, 1995.

[32] S. Kirkpatrick, D. G. Jr., and M. P. Vecchi. Optimization by simmulated annealing. *Science*, 220(4598):671–680, 1983.

[33] W. Ku and J. C. Beck. Mixed integer programming models for job shop scheduling: A computational analysis. *Computers & OR*, 73:165–173, 2016.

[34] P. J. M. v. Laarhoven, E. H. L. Aarts, and J. K. Lenstra. Job shop scheduling by simulated annealing. *Operations Research*, 40(1):pp. 113–125, 1992.

[35] P. Laborie. IBM ILOG CP optimizer for detailed scheduling illustrated on three problems. In van Hoeve and Hooker [59], pages 148–162.

[36] J. Y. Leung, editor. *Handbook of Scheduling - Algorithms, Models, and Performance Analysis*. Chapman and Hall/CRC, 2004.

[37] D. C. Mattfeld. *Evolutionary Search and the Job Shop Investigations on Genetic Algorithms for Production Scheduling*. Springer-Verlag, 1995.

[38] C. Mencía, M. R. Sierra, and R. Varela. Depth-first heuristic search for the job shop scheduling problem. *Annals OR*, 206(1):265–296, 2013.

[39] C. Mencía, M. R. Sierra, and R. Varela. An efficient hybrid search algorithm for job shop scheduling with operators. *International Journal of Production Research*, 51(17):5221–5237, 2013.

[40] C. Mencía, M. R. Sierra, and R. Varela. Intensified iterative deepening A* with application to job shop scheduling. *Journal of Intelligent Manufacturing*, 2013.

[41] R. Mencía, M. R. Sierra, C. Mencía, and R. Varela. Genetic algorithm for job-shop scheduling with operators. In J. M. Ferrández, J. R. Á. Sánchez, F. de la Paz, and F. J. Toledo, editors, *New Challenges on Bioinspired Applications - 4th International Work-conference on the Interplay Between Natural and Artificial Computation, IWINAC 2011, La Palma, Canary Islands, Spain, May 30 - June 3, 2011. Proceedings, Part II*, volume 6687 of *Lecture Notes in Computer Science*, pages 305–314. Springer, 2011.

[42] R. Mencía, M. R. Sierra, C. Mencía, and R. Varela. A genetic algorithm for job-shop scheduling with operators enhanced by weak lamarckian evolution and search space narrowing. *Natural Computing*, 13(2):179–192, 2014.

[43] R. Mencía, M. R. Sierra, C. Mencía, and R. Varela. Memetic algorithms for the job shop scheduling problem with operators. *Appl. Soft Comput.*, 34:94–105, 2015.

[44] R. Mencía, M. R. Sierra, C. Mencía, and R. Varela. Schedule generation schemes and genetic algorithm for the scheduling problem with skilled operators and arbitrary precedence relations. In R. I. Brafman, C. Domshlak, P. Haslum, and S. Zilberstein, editors, *Proceedings of the Twenty-Fifth International*

Conference on Automated Planning and Scheduling, ICAPS 2015, Jerusalem, Israel, June 7-11, 2015., pages 165–173. AAAI Press, 2015.

[45] R. Mencía, M. R. Sierra, C. Mencía, and R. Varela. Genetic algorithms for the scheduling problem with arbitrary precedence relations and skilled operators. *Integrated Computer-Aided Engineering*, 23(3):269–285, 2016.

[46] R. Mencía, M. R. Sierra, and R. Varela. Genetic algorithm for the job-shop scheduling with skilled operators. In J. M. F. de Vicente, J. R. Á. Sánchez, F. de la Paz López, F. J. Toledo-Moreo, and H. Adeli, editors, *Bioinspired Computation in Artificial Systems - International Work-Conference on the Interplay Between Natural and Artificial Computation, IWINAC 2015, Elche, Spain, June 1-5, 2015, Proceedings, Part II*, volume 9108 of *Lecture Notes in Computer Science*, pages 41–50. Springer, 2015.

[47] M. Mesquita and A. Paias. Set partitioning/covering-based approaches for the integrated vehicle and crew scheduling problem. *Computers & Operations Research*, 35(5):1562–1575, 2008.

[48] L. Michel, editor. *Integration of AI and OR Techniques in Constraint Programming - 12th International Conference, CPAIOR 2015, Barcelona, Spain, May 18-22, 2015, Proceedings*, volume 9075 of *Lecture Notes in Computer Science*. Springer, 2015.

[49] E. Nowicki and C. Smutnicki. An advanced tabu search algorithm for the job shop problem. *Journal of Scheduling*, 8:145–159, 2005.

[50] I. Oliver, D. Smith, and J. R. Holland. Study of permutation crossover operators on the traveling salesman problem. In *Genetic algorithms and their applications: proceedings of the second International Conference on Genetic Algorithms: July 28-31, 1987 at the Massachusetts Institute of Technology, Cambridge, MA*. Hillsdale, NJ: L. Erlhaum Associates, 1987., 1987.

[51] I. Ono, M. Yamamura, and S. Kobayashi. A genetic algorithm for job-shop scheduling problems using job-based order crossover. In *Proceedings of IEEE International Conference on Evolutionary Computation*, pages 547–552, May 1996.

[52] J. J. Palacios, C. R. Vela, I. González-Rodríguez, and J. Puente. Schedule generation schemes for job shop problems with fuzziness. In *ECAI 2014 - 21st European Conference on Artificial Intelligence, 18-22 August 2014, Prague, Czech Republic*, pages 687–692, 2014.

[53] M. L. Pinedo. *Scheduling: Theory, Algorithms, and Systems*. Springer Publishing Company, Incorporated, 3rd edition, 2008.

[54] B. Roy and B. Sussman. Les problemes d'ordonnancements avec contraintes disjonctives. Notes DS no. 9 bis, SEMA, Paris, 1964.

[55] M. R. Sierra, C. Mencía, and R. Varela. New schedule generation schemes for the job-shop problem with operators. *Journal of Intelligent Manufacturing*, 26(3):511–525, 2015.

[56] I. Steinzen, M. Becker, and L. Suhl. A hybrid evolutionary algorithm for the vehicle and crew scheduling problem in public transit. In *IEEE Congress on Evolutionary Computation*, pages 3784–3789, 2007.

[57] I. Steinzen, V. Gintner, L. Suhl, and N. Kliewer. A time-space network approach for the integrated vehicle-and crew-scheduling problem with multiple depots. *Transportation Science*, 44(3):367–382, 2010.

[58] E.-G. Talbi. *Metaheuristics: from design to implementation*, volume 74. John Wiley & Sons, 2009.

[59] W. J. van Hoeve and J. N. Hooker, editors. *Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems, 6th International Conference, CPAIOR 2009, Pittsburgh, PA, USA, May 27-31, 2009, Proceedings*, volume 5547 of *Lecture Notes in Computer Science*. Springer, 2009.

[60] C. R. Vela, R. Varela, and M. A. González. Local search and genetic algorithm for the job shop scheduling problem with sequence dependent setup times. *Journal of Heuristics*, 16(2):139–165, 2010.

[61] P. Vilím, P. Laborie, and P. Shaw. Failure-directed search for constraint-based scheduling. In Michel [48], pages 437–453.

[62] C. Y. Zhang, P. Li, Y. Rao, and Z. Guan. A very fast TS/SA algorithm for the job shop scheduling problem. *Computers and Operations Research*, 35:282–294, 2008.