Centro de Investigación en Matemáticas, A.C.

Paisaje de búsqueda en el problema de planificación de producción tipo taller

Tesis que presenta

Juan Germán Caltzontzin Rabell

para obtener el Grado de

Maestro en Ciencias con Especialidad en Computación y Matemáticas Industriales

Director de Tesis

Dr. Carlos Segura González

Guanajuato, Gto. Julio de 2021

Resumen

En este trabajo se analiza el efecto de distintas metodologías de modificación del paisaje de búsqueda del problema de planificación de producción tipo taller (JSP por sus siglas en inglés). Existen una gran cantidad de tipos de problemas de planificación, el JSP es un tipo específico de problema que ha atraido considerable atención por su dificultad. En general estos problemas consisten en encontrar una planificación que minimice el tiempo requerido para completar ciertos procesos.

Aunque existen algoritmos exactos para encontrar la solución óptima JSP[9] el tiempo que requieren problemas grandes es demasiado largo como para que sean prácticos o inclusive factibles. La complejidad de este problema lo hizo un candidato ideal para utlizar métodos aproximados para conseguir una solución aceptable en un tiempo razonable. Actualmente los mejores resultados para las instancias de prueba más difíciles se han encontrado mediante metaheurísticas, en especfico la conocida como búsqueda tabú.

Las metaheurísticas son convenientes por su relativa simplicidad algorítmica frente a otros métodos sin embargo, el tiempo requerido para obtener los resultados del estado del arte ha comenzado a crecer también, por lo que es importante dar un paso atrás y replantear un poco la forma de aplicar estos métodos para evitar complicarlos de más.

En el campo de las metahurísticas hay varios conceptos fundamentales que tienen un gran impacto en su desempeño, en este trabajo se analizan tres de ellos que juntos componen lo que se conoce como paisaje de búsqueda el cual está estrechamente relacionado con la dificultad con el desempeño de las metaheurísticas de trayectoria.

Los conceptos analizados en este trabajo son : representación de las soluciones, definición de vecindad de una solución, función de aptitud o fitness. Para evaluar el impacto de cada uno de ellos se utliza una de las metaheurísticas de trayectoria más sencillas; la búsqueda local iterada.

Se tomaron las instancias dmu que son uno de los conjuntos de prueba más utilizados actualmente y que aun no ha sido resuelto por completo para hacer experimentos computacionales y se encontró que de las tres áreas en las que se plantearon modificaciones, la más importante fue la de la representación de soluciones.

Abstract

En este trabajo se analiza el efecto de distintas metodologías de modificación del paisaje de búsqueda del problema de planificación de producción tipo taller (JSSP por sus siglas en inglés) (En inglés)

Índice general

Capítulo 1

Introducción

Sumario

En este capítulo se presenta el problema de interés así como una revisión de los distintos métodos que se han formulado para resolverlo. Se plantea la hipótesis del trabajo y los objetivos así como las propuestas que servirán para alcanzar los objetivos. Al final del mismo se presentan las contribuciones de este trabajo al problema considerado.

1.1. Antecedentes y Motivación

Los problemas de planificación surgen de manera natural en sistemas de producción o procesamiento en los que la tarea general que se quiere completar a su vez está compuesta por varias subtareas que pueden o deben repartirse entre distintas máquinas o unidades de procesamiento. A grandes rasgos, un problema de planificación consiste en asignar a cada subtarea una máquina que debe procesarla y un tiempo de inicio y fin. Estos problemas pueden surgir de todo tipo de contextos, desde la producción de algo como una bicicleta hasta la forma en que los sistemas computacionales procesan información o cómo se asignan las clases en una escuela.

En muchos de estos contextos no solo se requiere hallar una planificación sino que a demás se quiere encontrar una que sea óptima en algún sentido, habitualmente se quiere la que haga que se complete el trabajo lo más rápido posible aunque puede haber otros criterios como se presenta más adelante.

Desde los década de los 50 se comenzaron a formular algoritmos para hallar planificaciones óptimas para un problema con dos y tres máquinas[21] y el interés en ellos ha crecido en las décadas siguientes hasta la fecha por su gran campo de aplicación.

En este trabajo se trata un tipo específico de problema de planificación conocido como el **Problema de Planificación de Producción tipo Taller** o JSP por sus siglas en inglés. Este problema consiste en hallar una secuencia de procesamiento de n ítems en m máquinas que tome el mínimo tiempo posible cumpliendo ciertas restricciones de orden para el procesamiento de los n ítems. Este es un problema de optimización combinatoria en el que se requiere encontrar una secuencia de procesamiento para las operaciones en cada una de las máquinas disponibles y pertenece a la clase de problemas **NP-completo** cuando el número de máquinas es mayor a dos[16].

Anteriormente se han propuesto métodos exactos para resolver este problema [9] pero la cantidad de recursos computacionales que requieren conforme el tamaño del problema (número de máquinas y trabajos) aumenta los hace imprácticos excepto para instancias pequeñas aproximadamente 10 máquinas y 10 trabajos.

Dado el interés que se tiene en este problema y en que su tamaño puede ser suficientemente grande para no poder utilizar métodos exactos, se han propuesto otros métodos aproximados para encontrar soluciones buenas en tiempos aceptables. Estos métodos pueden agruparse de la siguiente manera[20, 41]:

- **Métodos Constructivos** Estos métodos construyen una planificación mediante el uso de alguna regla simple por lo que son muy rápidos y conceptualmente sencillos. En general pueden clasificarse en tres tipos: los que usan reglas de prioridad, los que usan heurísticas de cuello de botella y los que usan algún método de inserción.

  El primero de estos consiste en establecer una forma de elegir la operación a planificar entre varias disponibles. Esto puede hacerse por ejemplo eligiendo la que tome más tiempo o la que pueda procesarse antes.

  El segundo consiste en replantear el problema como una serie de subproblemas más sencillos que puedan resolverse iterativamente hasta que se tenga una solución completa, por ejemplo planificando una sola máquina manteniendo las otras fijas.

  El tercer tipo construye una solución partiendo del ordenamiento de solo un subconjunto de operaciones y progresivamente agregando más a partir de las que ya se tienen.

- **Métodos de inteligencia artificial** En estos métodos utilizan redes neuronales para encontrar la planificación. Existen muchos tipos de redes que se han diseñado para atacar este problema aunque en general suelen combinarse con otros métodos para obtener resultados competitivos.

  La gran desventaja de estos métodos es que se tienen que construir y entrenar redes a la

medida de cada problema y a menudo se necesita de mucha preparación previa para aplicarse a algún problema lo cual los vuelve poco prácticos.

- **Métodos de búsqueda local** En estos métodos se establece una forma de crear soluciones nuevas a partir de una solución dada para reemplazarla por una nueva y repetir el proceso hasta que se cumpla algún criterio de paro. Para crear nuevas soluciones se hace un cambio pequeño como, por ejemplo, intercambiar el orden de dos operaciones de modo que sea posible evaluar todas las posibles soluciones generadas al aplicar esta modificación. En general estos métodos se distinguen entre sí por la manera en que se escoge la solución con la cual reemplazar la solución inicial y el criterio de paro.

- **Metaheurísticas** Estos métodos combinan elementos de los previamente mencionados para plantear estrategias que obtengan aun mejores resultados. Es muy común que se planteen metaheurísticas inspiradas en la naturaleza como los algoritmos genéticos basados en la evolución, algoritmos basados en el comportamiento de seres vivos como algoritmos de colonia de hormigas o de abejas o bien en procesos naturales como el algoritmo de recocido simulado. Al ser de alto nivel pueden adaptarse a una gran cantidad de problemas.

Actualmente los algoritmos más exitosos para resolver el JSP son algoritmos meméticos que combinan un método de búsqueda local con una metaheurística poblacional que mantiene varias soluciones. El método de búsqueda local más usado se conoce como búsqueda tabú, el cual logra escapar de óptimos locales mediante el uso de memoria.

Aunque se han obtenido buenos resultados con estos algoritmos son necesarias de 24 a 48 horas paralelas de ejecución para obtener los resultados del estado del arte. La literatura reciente se ha centrado en hacer más eficiente la búsqueda tabú para reducir los tiempos de ejecución, dejando de lado los otros elementos que forman parte de las metaheurísticas.

Como se explicará a detalle más adelante el desempeño de las metaheurísticas depende del llamado paisaje de búsqueda del problema, el cual se compone de tres elementos.

El primero y más básico es cómo representar computacionalmente una solución, actualmente lo más común es que una solución se represente como un grafo dirigido cuyos nodos son las operaciones a procesar y las aristas indican el ordenamiento de éstas.

El segundo es la forma de generar una solución a partir de otra, esto se conoce como una estructura de vecindad y la más exitosa es conocida como N7 y fue propuesta en 2006 [40].

El tercer elemento se conoce como función de fitness o aptitud y nos permite comparar soluciones y decidir si una es mejor que otra.

1.2. Objetivo

El objetivo de este trabajo es plantear modificaciones al paisaje de búsqueda del JSP que permitan el uso de mecanismos más simples y rápidos para encontrar soluciones de calidad no muy alejada al estado del arte.

Las modificaciones se centran en el paisaje de búsqueda del problema y constituyen en concreto los siguientes objetivos específicos.

Objetivos específicos

- Proponer una función de fitness que tome en cuenta más que solo el tiempo total de una planificación.
- Plantear una representación con la que se puedan obtener mejores resultados.
- Plantear una nueva estructura de vecindad.

1.3. Hipótesis

Es posible conseguir resultados no muy lejanos de los reportados en el estado del arte con algoritmos sencillos y rápidos si modificamos el paisaje de búsqueda de manera adecuada.
Como se explica mas adelante, el paisaje de búsqueda es la combinación de tres elementos: función de fitness, espacio de búsqueda y estructura de vecindad. Estos tres elementos tienen un efecto muy importante en el éxito que puede llegar a tener una metaheurística

1.4. Propuestas

Para poner a prueba la hipótesis se presentan las siguientes propuestas que serán exploradas mediante experimentos computacionales:

1. Utilizar la búsqueda local iterada 3 (ILS por sus siglas en inglés) por ser un algoritmo simple y rápido.
2. Agregar nuevos movimientos a la vecindad N7.
3. Crear una función de fitness que no solo tome en cuenta el makespan sino también otras características de la solución.
4. Utilizar una nueva representación junto con un esquema de decodificación para reducir el tamaño del espacio de búsqueda.
5. Construir una nueva estructura de vecindad a partir de la nueva representación.

1.5. Contribuciones

En este trabajo se consiguió plantear modificaciones al paisaje de búsqueda que combinados con una metaheurística de búsqueda local iterada obtuvieron resultados con una mediana de error

relativo no mayor al 0,21 para todas las instancias de prueba en un tiempo de 5 minutos no paralelo comparado con las 24 a 48 horas de cómputo paralelo necesarios para hallar los resultados del estado del arte.

De todos los cambios que se implementaron al paisaje de búsqueda, el que tuvo más impacto en el desempeño de la búsqueda local iterada consistió en cambiar la representación y con ella el conjunto de soluciones representables y la estructura de vecindad. No se logró plantear una función de fitness que mostrara una mejora sustancial en los resultados aunque definitivamente tiene un impacto positivo considerar algo más que únicamente el makespan de una planificación para hallar mejores soluciones.

Estas modificaciones pueden servir de punto de partida para proponer estrategias más complejas que puedan acercarse mucho más a los resultados del estado del arte pero en una fracción del tiempo requerido actualmente. También hay trabajo por hacer para refinar la representación y la estructura de vecindad aquí presentadas.

Capítulo 2

Marco teórico

Sumario

En este capítulo se presentan los conceptos sobre los cuales se basa el trabajo así como definiciones relativas al problema y algunos algortimos específicos relevantes para el presente trabajo.

2.1. Optimización

La optimización es una herramienta que nos ayuda a encontrar *la mejor* entre diferentes opciones elegibles. En nuestra vida diaria a menudo nos encontramos en este tipo de situaciones, por ejemplo al elegir entre diferentes rutas para llegar a algún lugar o entre diferentes productos.

En lenguaje matemático un problema de optimización consiste en minimizar o maximizar una función que le asigna un valor a cada una de las opciones que tenemos disponibles. Para la definición formal se adopta la siguiente notación:

- X El conjunto de opciones o soluciones disponibles.
- $f : X \to \mathbb{R}$ La función objetivo a minimizar o maximizar.

El problema consiste en hallar:

$$\min_{x \in X} f(x) \tag{2.1}$$

Es importante mencionar que cualquier problema en el que se requiera encontrar el argumento que hace tomar el valor máximo a f puede transformarse en un problema equivalente de minimización con el reemplazo $f(x) \leftarrow -f(x)$, por lo que puede considerarse solo el caso de minimización sin pérdida de generalidad.

De acuerdo con la definición del problema como tal o con los métodos de solución se distinguen distintas ramas de la optimización. A continuación se presentan algunas de ellas de especial importancia para el presente trabajo.

Optimización con restricciones

A la definición de un problema de optimización también pueden agregarse un conjunto de restricciones las cuales son en la práctica un conjunto de funciones que toman una solución e indican si esta cumple o no con ellas, se dice que una solución es factible si cumple con las restricciones e infactible en caso contrario. Estas restricciones se agregan ya sea porque el problema así lo requiere o bien para asegurarse que la solución tenga sentido, por ejemplo si f tiene como argumento un número real que representa una longitud debe requerirse que esta longitud sea un número positivo.

Optimización global y local

Hallar el mínimo de la función sobre todo el conjunto X recibe el nombre de optimización global, esto puede llegar a ser muy costoso o incluso imposible de obtener por lo que es común optar por obtener un mínimo local, es decir, una solución que sea la mejor de un subconjunto de X. A menudo este subconjunto se define utilizando alguna medida de distancia entre soluciones y considerando a todas las soluciones que estén en algún rango de distancia a otra de referencia o bien añadiendo un operador que permita crear soluciones nuevas a partir de una solución ya conocida.

Optimización estocástica

También puede ser que el algoritmo planteado para resolver el problema de optimización sea no determinista i.e. puede obtener soluciones diferentes aunque tenga las mismas condiciones iniciales. En muchas ocasiones es ventajoso tener algo de aleatoriedad porque nos permite explorar el espacio de búsqueda con pocos sesgos. Estos métodos son especialmente útiles cuando el espacio de búsqueda es poco «predecible» en el sentido en que no podemos a priori distinguir regiones de buena o mala calidad. Tal vez el método más famoso de este tipo es el llamado recocido simulado, el cual se inspira en un fenómeno metalúrgico y que encuentra soluciones muy buenas a una gran variedad de problemas de optimización.

Optimización continua y discreta

En la definición anterior no se requiere que el conjunto de soluciones tenga alguna propiedad o alguna estructura adicional. Por ejemplo el conjunto de soluciones puede ser un conjunto numerable (e.g. los enteros) o no numerable (e.g. los números reales). Estos dos casos dividen a la optimización en dos ramas: optimización continua y optimización discreta. En general los problemas de optimización continua suelen ser más fáciles de abordar[26] porque en muchas ocasiones es posible obtener información del valor de la función objetivo de puntos cercanos a cierto punto conocido mientras que en los problemas discretos esto rara vez puede hacerse.

Optimización combinatoria

Dentro de los problemas de optimización discreta se distinguen los problemas de optimización combinatoria. Formalmente un problema de optimización combinatoria consta de los siguientes elementos[7]:

- Un conjunto de variables $Z = \{z_1, z_2, ..., z_n\}$
- Dominio para cada variable $D_1, D_2, ..., D_n$
- Restricciones entre variables

Estos tipo de problemas aparecen en múltiples escenarios, por ejemplo cuando se requiere hallar una permutación de objetos.

Hasta ahora se han mencionado distintos tipos de problemas pero no se ha dicho nada sobre su dificultad, esta es el área de estudio de la teoría de la complejidad.

2.2. Teoría de la complejidad

De manera empírica sabemos que existen problemas «difíciles» y problemas «fáciles», por ejemplo, es mucho más difícil armar un rompecabezas que comprobar que está bien armado. La teoría

de la complejidad busca responder a la pregunta: *¿Qué hace a algunos problemas fáciles y a otros difíciles?* [33].

Uno de los logros de la teoría de la complejidad ha sido establecer un sistema de clasificación de acuerdo con la dificultad de los problemas. Este sistema consiste en muchas clases a las cuales puede ser asignado un problema en relación a los recursos computacionales necesarios para resolverlo, siendo las clases más estudiadas las que se definen por cantidad de operaciones (tiempo) y por memoria (espacio). A continuación se explica brevemente cómo se procede para describir la cantidad de operaciones necesarias para resolver un problema.

Notación gran O

La cantidad de operaciones necesarias para resolver algún problema puede expresarse como una función del tamaño del mismo, es decir de la forma $f(n)$ donde n es el tamaño del problema. La función f puede ser de infinidad de formas diferentes, pero para fines prácticos solo se considera su comportamiento asintótico, es decir, cuando n tiende a infinito. De esta manera se puede tener un conjunto de clases de equivalencia en las que se considera que los problemas tienen el mismo nivel de dificultad. Para delimitar estas clases se define la notación gran O.

Para dos funciones $f, g : \mathbb{R} \to \mathbb{R}$ se dice que $f(n) = O(g(n))$[1] si existen constantes $c, n_0 \in \mathbb{R}^+$ tal que $0 \leq f(n) \leq cg(n)$ para todo $n \geq n_0$ [12].

De manera intuitiva $f(n) = O(g(n))$ quiere decir que $cg(n)$ es una cota superior a $f(n)$ como puede verse en la figura 2.1.

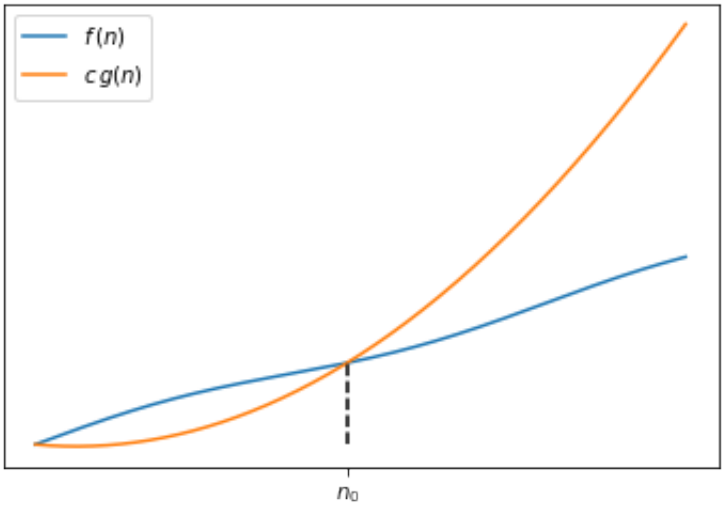


Figura 2.1: Representación de $f(n) = O(g(n))$

Se dice que un algoritmo es $O(g(n))$ si el número de operaciones que requiere para resolver un problema de tamaño n es $O(g(n))$. Por ejemplo, el algoritmo para sumar dos números de n toma

[1] El signo de igual aquí no tiene el sentido usual sino que más bien representa una relación entre conjuntos en la que $=$ quiere decir $\subseteq$ [19]

$O(n)$ operaciones.

Clases P y NP

La forma de la función g es la base para clasificar los problemas de acuerdo a su complejidad dos de las clases más analizadas son definidas por el número de operaciones y se conocen como:

- **P** Los problemas para los que se conoce un algoritmo para resolverlos que toma a lo más Kn^c operaciones con K, c finitas. El nombre de esta clase hace referencia que pueden resolverse en tiempo polinomial es decir que toman $O(n^c)$ operaciones para alguna c positiva y finita.

- **NP** Los problemas para los que se puede verificar que se encontró una solución en tiempo polinomial. [2]

Una subclase importante de la clase **NP** es la llamada **NP-completo** que incluye a los problemas para los cuales hallar un algoritmo de solución en tiempo polinomial implica hallar uno para todos los problemas en **NP**. De modo intuitivo es el conjunto de los problemas más difíciles en la clase.

Claramente $\mathbf{P} \subseteq \mathbf{NP}$[3] pero determinar si también se cumple esta relación en sentido contrario, es decir determinar si $\mathbf{P} = \mathbf{NP}$, es un problema sumamente difícil con implicaciones importantes que no parece estar cerca de ser resuelto.

La distinción en algoritmos que toman tiempo polinomial parece arbitraria aunque de manera empírica se conocen muchos problemas de interés que pueden ser clasificados con esta definición. De un punto de vista más formal, los polinomios ejemplifican funciones de crecimiento lento y cumplen con varias propiedades teóricas que simplifican la clasificación de los problemas[38].

Muchos problemas de optimización de gran interés pertenecen a la clase **NP** entre ellos el JSP que se aborda que se aborda en el presente trabajo y que en particular es **NP-completo**. Ante la dificultad para resolver estos problemas de manera eficiente surgieron técnicas conocidas como metaheurísticas que buscan facilitar encontrar soluciones aceptables aunque no óptimas.

2.3. Metaheurísticas

Es muy común que en nuestra cotidianidad nos enfrentemos a problemas tan difíciles o para los que tengamos tan poco tiempo de decisión que no podamos hacer un análisis riguroso, en estos casos es muy común que utilicemos algún método (posiblemente basado en la experiencia) que nos permita hallar una solución aceptable, por ejemplo, es común que reemplacemos el problema por uno más simple que sí podemos responder y cuya respuesta está relacionada con nuestro problema

[2] Una definición alternativa pero equivalente se basa en el concepto de máquinas de turing no deterministas.

[3] Si ya tenemos un algoritmo que resuelve un problema en tiempo polinomial, para verificar si una solución es correcta solo hay que resolver el problema

original (no podemos predecir con certeza si lloverá durante el día pero sí podemos responder si el cielo está plagado de nubes oscuras).

En el contexto de la optimización una metaheurística es una metodología de alto nivel que combina diferentes heurísticas y puede aplicarse para resolver de manera aproximada una gran cantidad de problemas. En la práctica existen numerosas metaheurísticas que pueden ser muy diferentes entre sí por lo que no hay un sistema de clasificación universalmente aceptado aunque se han propuesto diferentes criterios de clasificación [34] así como características como:

- De trayectoria vs discontinua. Una metaheurística de trayectoria consiste en, dada una solución inicial, mejorarla de manera iterativa mediante algún operador que «mueve» a la solución a través del espacio de búsqueda
- Basadas en población vs basadas en una sola solución. En las metaheurísticas basadas en población se mantiene un conjunto de soluciones candidatas que a menudo se combinan entre sí en lugar de modificar una sola.
- Basadas en búsqueda local vs constructivas. Como se explicará más adelante, en la búsqueda local, el proceso de mejora implica la evaluación de soluciones muy parecidas a una solución inicial dada mientras que en las constructivas se crean nuevas soluciones de acuerdo a una heurística o algoritmo preestablecido.
- Con uso de memoria vs sin uso de memoria. El uso de memoria consiste en almacenar información que nos ayude a explorar el espacio de búsqueda, por ejemplo una lista de soluciones previamente visitadas.

Los primeros dos de estos criterios están muy relacionados porque casi todas las metaheurísticas discontinuas son poblacionales y muchas de trayectoria son basadas en una sola solución.

Si bien las metaheurísticas son muy diversas existen elementos comunes que tienen un papel determinante en el buen funcionamiento de las mismas. En específico para las metaheurísticas de trayectoria, aunque también aplican para las demás, existen tres conceptos que determinan el llamado paisaje de búsqueda: las soluciones, la forma en que las soluciones están conectadas y cómo podemos compararlas entre sí. En la siguiente sección se introduce el concepto de paisaje de búsqueda y sus componentes.

2.4. Paisaje de búsqueda

El concepto de paisaje de búsqueda surgió en la biología para explicar los mecanismos de evolución de poblaciones de seres vivos[39]. La idea básica es que los genes de algún organismo en particular le confieren ventajas o desventajas en su hábitat, i.e. puede estar más o menos adaptado. De esta manera puede pensarse en clasificar a todos los individuos de acuerdo a su grada de adaptación.

Conforme se introducen nuevas variaciones en el código genético ya sea por la reproducción o mutación, se pueden añadir a los nuevos individuos a la clasificación. Si se continua de esta forma se obtendrá una especie de mapa en el cual podríamos asignar a cada variación del código genético un punto en el mapa y un valor de adaptabilidad, también podemos agregar información de la procedencia de cada genoma.

El proceso previamente descrito tiene como resultado una estructura que puede representarse mediante un grafo. En el ámbito de las metaheurísticas el concepto de paisaje de búsqueda se inspira fuertemente en la descripción biológica. Los conceptos de genoma, adaptabilidad y reproducción y mutación tienen como análogos la representación, la función de fitness o aptitud y la estructura de vecindad.

2.4.1. Representación

Puede ser que el problema de optimización en el que estemos interesados surja directamente de las matemáticas aunque si estamos interesados en un problema de nuestro entorno físico es necesario que tengamos que idear una forma de traducirlo a un lenguaje matemático, incluidas las soluciones al mismo. Debemos encontrar una forma de representar de manera útil las soluciones posibles. Por ejemplo si buscamos un ordenamiento óptimo para algún conjunto de n cosas podemos asignarle a cada elemento del conjunto un número entero del 1 al n, en este modelo el espacio de soluciones está constituido por todas las permutaciones de los números del 1 al n.

Hay varias maneras de representar permutaciones, podemos usar un arreglo de n entradas o bien una matriz de permutación por mencionar algunos. Algo sumamente importante es que estas dos formas de representar las soluciones son muy distintas y se tiene que trabajar con ellas de manera muy diferente. La primera de ellas puede llegar a representar *soluciones no factibles* por ejemplo que aparezca un número repetido, mientras que la segunda no.

Con el ejemplo anterior también podemos ver que si queremos establecer algunos operadores que, por ejemplo, perturben la solución tendremos que definirlos de maneras completamente distintas. También es importante notar que es posible que el número de soluciones no factibles que podamos representar sea mayor que el de las factibles.

Formalmente una representación es un mapa que asocia elementos entre el conjunto de soluciones y el conjunto de las representaciones. Este mapeo no tiene por que ser suryectivo, es decir que puede ser que solo asigne una representación a parte del conjunto de soluciones. También puede ser el caso como se mencionó anteriormente que haya representaciones que no correspondan a soluciones. Pueden distinguirse tres tipos de representaciones[10] de acuerdo a como asocian los elementos de estos dos conjuntos.

- 1 a 1 A cada solución le corresponde una única representación.
- 1 a n La misma representación puede asociarse a varias soluciones.

- n a 1 Una solución puede tener diferentes representaciones.

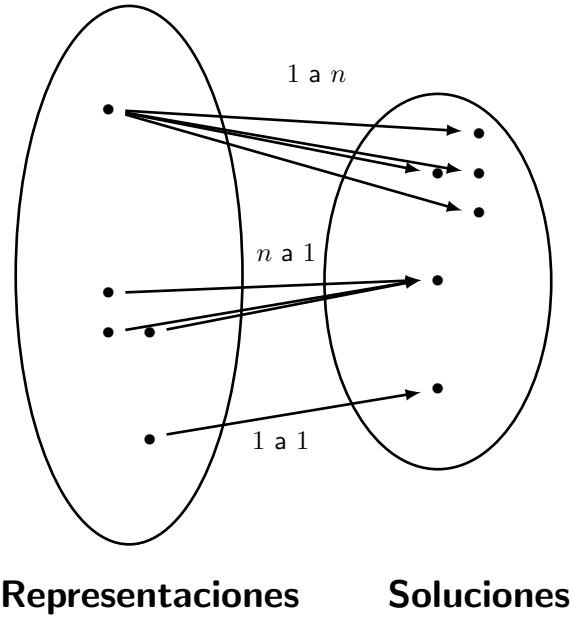


Figura 2.2: Tipos de representaciones

Lo más conveniente suele ser un mapeo 1 a 1 porque resulta mucho más eficiente y lo menos deseable es tener un mapeo n a 1 en el que es posible que la misma representación esté asociada a soluciones de calidad muy diferente.

2.4.2. Vecindad

La definición de vecindad es crucial para las metaheurísticas de trayectoria y las basadas en una sola solución. Formalmente, una vecindad es un mapeo $N : X \to 2^X$ que le asigna a cada solución $x \in X$ un subconjunto de soluciones en X. Intuitivamente podemos pensar que es una forma de definir a las soluciones que «rodean» a otra. Se dice que la solución y es un vecino de x si $y \in N(x)$.

A partir de la definición de vecindad podemos también definir un operador de movimiento $M : X \to X$ cuyo efecto al aplicarlo a una solución sea transformarla en una que pertenezca a su vecindad, i.e. este operador selecciona a un vecino de la solución inicial.

$$M(x) = y \in N(x) \quad x \neq y$$

En general se busca que una vecindad cumpla con ciertas características, entre las más importantes es que esté bien conectada, que no tenga un tamaño excesivamente grande y que contenga soluciones de calidad no muy distinta, esta última característica le confiere «suavidad» y pue-

de ser de ayuda para reducir el número de óptimos locales en los que se pueda atascar alguna metaheurística.

2.4.3. Función de aptitud o fitness

En el momento en el que se define el problema de optimización, se define cuál es la función objetivo, no obstante puede resultar útil construir otra función a la cual se le conoce como función de aptitud o fitness que toma en cuenta otras características de las soluciones para que el paisaje de búsqueda tenga una estructura más favorable para los métodos de solución que se pretendan usar. Por ejemplo puede suceder que aunque dos soluciones tengan asociado el mismo valor de la función objetivo una de ellas posee características que la hacen un mejor punto de partida para alguna metaheurística.

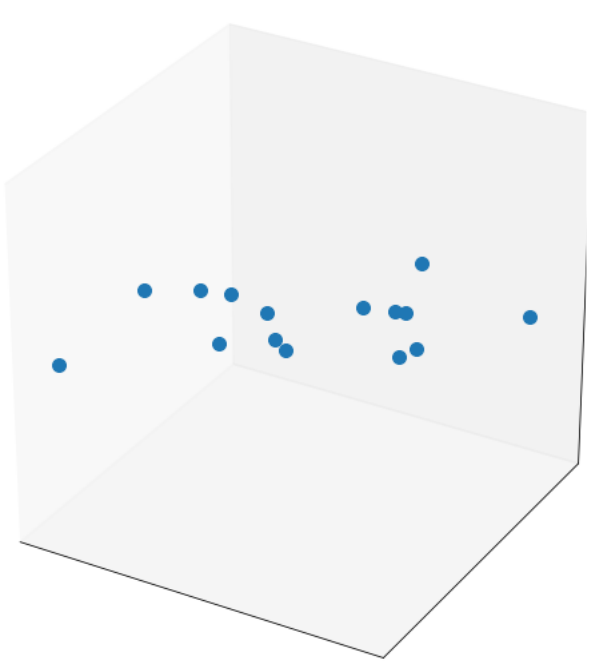
Esta función debe asociar a cada solución un elemento de un conjunto donde esté definida una noción de ordenamiento. En esencia esta función define un operador de comparación entre soluciones de modo que esté definido cuándo una solución es mejor que otra. Una vez que tenemos el espacio de búsqueda y operadores de cambio para generar nuevas soluciones a partir de otras, se define el espacio de búsqueda como un grafo dirigido G en el que los nodos son las soluciones al problema y una solución x está conectada a otra y si podemos generar a y aplicando los operadores de cambio a x.

Podemos asociar a cada solución en el espacio un valor de aptitud o fitness que mide la calidad de dicha solución. La adición de esta función de aptitud al espacio de búsqueda genera al paisaje de búsqueda. Formalmente el paisaje de búsqueda $\mathcal{L}$ es entonces una tupla conformada por el espacio de búsqueda junto con una función objetivo que guía la búsqueda $\mathcal{L} = (G, f)$.

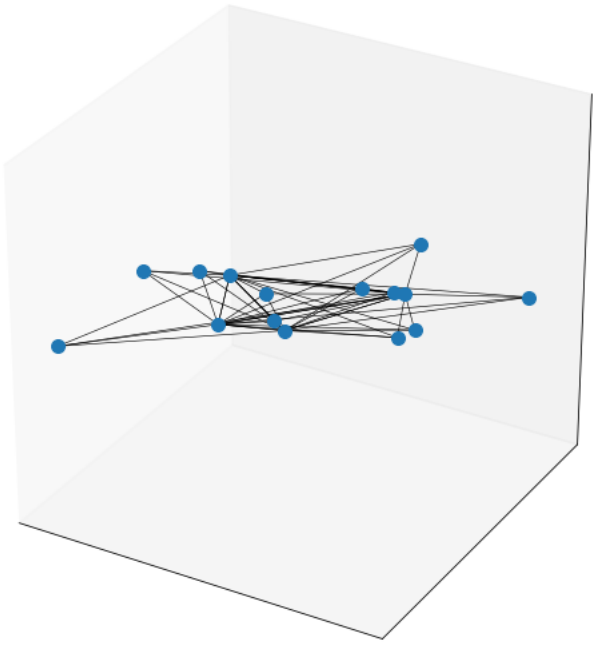
En la figura 2.3 se muestra una representación pictórica del paisaje de búsqueda para un problema de optimización.

El paisaje de búsqueda es el «terreno» a explorar y puede cambiar si cambiamos cualquiera de sus componentes, podría ser que alguna representación nos centre en un subconjunto de soluciones convenientes o que alguna estructura de vecindad se proponga de modo que las mejores soluciones nunca están muy lejos del resto, o que la función de fitness nos ayude a atravesar cúmulos de soluciones que serían iguales sin ella.

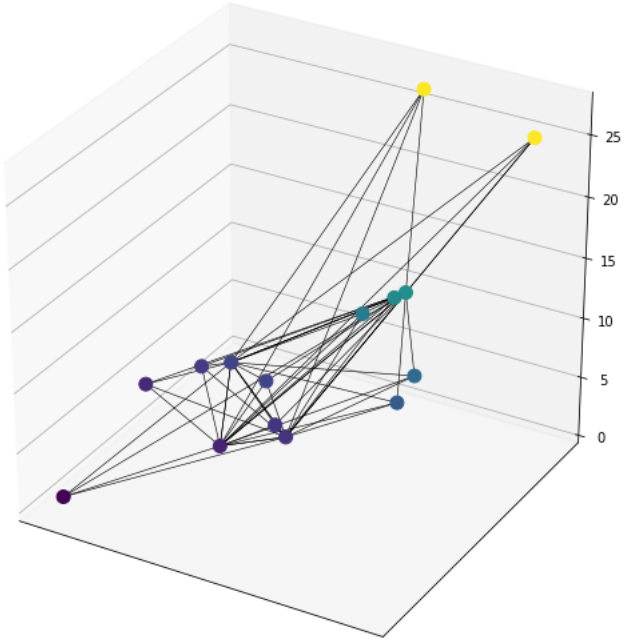
La estructura del paisaje de búsqueda influye de manera determinante en el éxito o fracaso de las metaheurísticas. Dependiendo de la «forma» que tenga el paisaje se favorecerá el uso de ciertas metaheurísticas. La «forma» del paisaje hace referencia a cómo cambia el valor de fitness para soluciones conectadas entre sí. Dos medidas que son generalmente utilizadas para esto miden cómo cambia el valor de la función de fitness conforme nos acercamos a un óptimo local y la otra mide qué tanto cambia el fitness entre soluciones vecinas[22]. La primera de estas medidas nos da una idea de qué tan grandes son los valles que rodean a un mínimo local si es que existen y la segunda nos da una ida de la rugosidad del paisaje.



(a) Soluciones representables

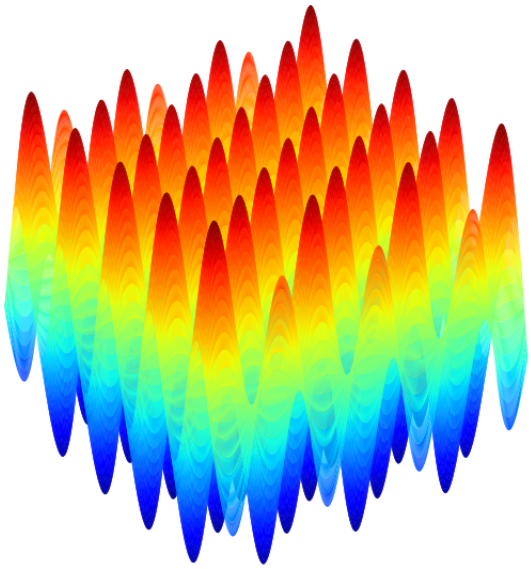


(b) Relaciones inducidas por los operadores de cambio

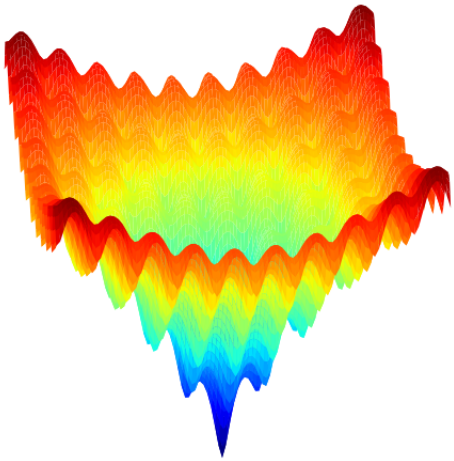


(c) Adición de la función de fitness

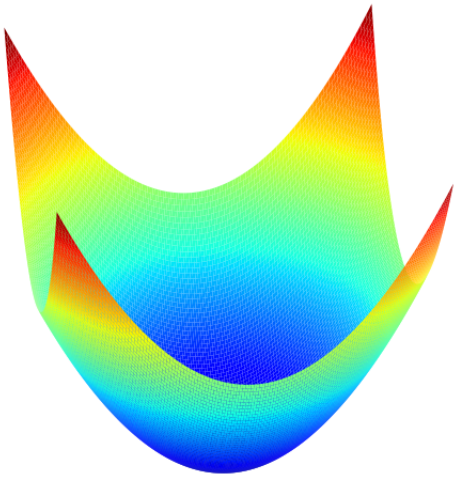
Figura 2.3: Creación del paisaje de búsqueda



(a) Paisaje rugoso con muchos óptimos locales



(b) Paisaje rugoso con con un gran valle



(c) Paisaje suave con un gran valle

Figura 2.4: Diferentes tipos de paisajes de búsqueda. Se muestran paisajes continuos con fines ilustrativos.

Algunas metaheurísticas de relevancia

Las metaheurísticas sirven como una estrategia para explorar el paisaje de búsqueda. Una de las más sencillas e intuitivas es conocida como escalada estocástica y simplemente consiste en reemplazar la solución actual por algún vecino mejor escogido al azar hasta que la solución en la que estemos sea mejor que todos sus vecinos, es decir, un óptimo local. Esta es una metaheurística de trayectoria y traza un camino entre las soluciones inicial y final.

Algorithm 1: Algoritmo de escalada estocástica

```
Data: Problema de Optimización
Result: Óptimo local x
Generar solución inicial x;
while L no vacía do
    Generar lista de vecinos L de x;
    Escoger al azar un vecino y ∈ L;
    if y < x then
        x ← y;
        Generar lista de vecinos L de x;
    else
        Quitar a y de L;
return x
```

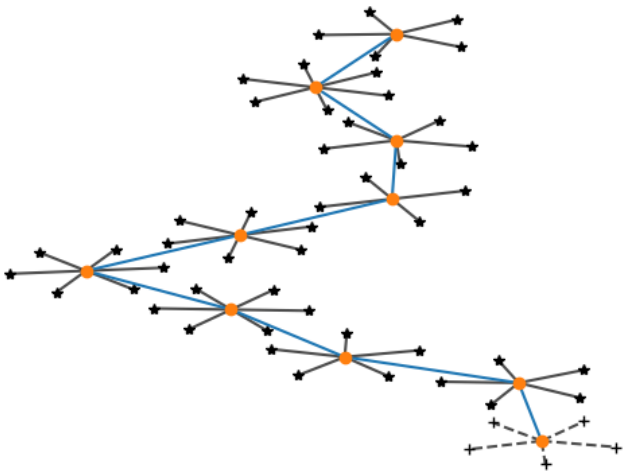


Figura 2.5: Ilustración de una escalada estocástica. Las aristas rayadas representan la vecindad del óptimo local.

Otra metaheurística de trayectoria importante que es parte de muchas de las estrategias que

han obtenido los resultados del estado del arte es la llamada búsqueda tabú. Esta metaheurística se basa en la búsqueda local pero puede aceptar vecinos que no mejoran la función de fitness con la intención de evitar atascarse en un óptimo local de mala calidad. La búsqueda tabú almacena soluciones previamente vistas para evitar visitarlas de nuevo. Es necesario definir el tamaño de la lista, el criterio de paro y cómo escoger una nueva solución.

Algorithm 2: Algoritmo básico de búsqueda tabú

Data: Problema de Optimización
Result: Mejor solución encontrada x^*
Generar solución inicial x;
Inicializar mejor solución como $x^* \leftarrow x$ Inicializar la lista tabú TL con x;
while *no criterio de paro* **do**
    Generar lista de vecinos aceptables L de x tal que $L \cap TL = \varnothing$;
    Escoger un vecino $y \in L$;
    $x \leftarrow y$;
    Actualizar TL;
    if $x < x^*$ **then**
        $x^* \leftarrow x$;
return x^*

Por último se presenta una metaheurística de trayectoria muy sencilla que intenta subsanar el problema de atascarse en óptimos locales de mala calidad de la búsqueda local. La idea es obtener un mínimo local a partir de una solución inicial mediante búsqueda local, aplicarle una perturbación y repetir el proceso hasta que se cumpla algún criterio de paro. Esta estrategia es conceptualmente muy simple aunque se debe definir la perturbación que se hace a la solución y por lo general no es tan sencillo plantear una perturbación adecuada.

Algorithm 3: Algoritmo búsqueda local iterada

Data: Problema de Optimización
Result: Mejor solución encontrada x^*
Generar solución inicial x;
Inicializar mejor solución como $x^* \leftarrow x$ **while** *no criterio de paro* **do**
    Obtener una solución y a partir de x mediante búsqueda local 1;
    if $y < x^*$ **then**
        $x \leftarrow y$;
        $x^* \leftarrow y$;
    else
        $x \leftarrow x^*$;
        Perturbar x;
return x^*

2.5. Problema de planificación de producción tipo taller (JSP)

Dentro de los problemas de planificación, el JSP modela un taller en el que hay m máquinas, cada una de las cuales puede hacer varios tipos de procedimientos pero solo puede hacer uno de ellos a la vez. Se requiere completar n trabajos cada uno de los cuales está compuesto por m operaciones, una por cada máquina en el taller, y que deben procesarse en un orden específico y por un tiempo específico en cada una de ellas.

Por ejemplo una panadería puede modelarse del siguiente modo; las herramientas como batidora, rodillo, horno, etc. pueden representarse como las m máquinas que se utilizan en un orden determinado para hacer n distintos tipos de pan los cuales pueden representarse como los trabajos a completar.

Las secuencias y tiempos de procesamiento para cada trabajo deben de especificarse para definir el problema. Habitualmente se presentan en una tabla de n renglones por m columnas esto se conoce como una instancia del JSP. Para cada renglón, las columnas contienen la máquina en que debe procesarse el trabajo en este paso y el tiempo que toma procesarlo.

Por ejemplo en la tabla 2.1 se presenta una instancia del JSP con 3 máquinas y 2 trabajos. En esta instancia el trabajo 1 debe procesarse primero en la máquina 0 por 75 unidades de tiempo, luego en la máquina 2 por 54 unidades de tiempo y por último en la máquina 1 por 59 unidades de tiempo.

El problema a resolver consiste en hallar una planificación que sea óptima en algún sentido. Una planificación consiste en asignar tiempos de inicio y fin a cada operación respetando el orden requerido para cada trabajo. Existen varias formas de determinar si una planificación es óptima como se explicará más adelante pero la más usada es el tiempo que toma terminar todos los trabajos y se conoce como makespan. Sin pérdida de generalidad podemos restringirnos al caso en el que el tiempo requerido para procesar cada operación es un entero positivo.

Otro concepto que es de gran utilidad en los métodos que se presentarán en las siguientes secciones es la secuencia de trabajos que toma el mayor tiempo en completarse conocida como **ruta crítica**. Esta ruta crítica se compone a su vez de una serie de bloques críticos que consisten en las secuencias de operaciones de la ruta crítica que se ejecutan de forma adyacente en la misma máquina. Es importante mencionar que una planificación siempre tiene al menos una ruta crítica pero puede tener más de una a la vez.

Es muy usual que una planifacación se visualice mediante un diagrama de gantt. El siguiente ejemplo sirve para mostrar los conceptos antes presentados en un caso práctico y simple.

Ejemplo

Se muestra un ejemplo de una instancia con 3 máquinas y 2 trabajos.

Trabajo	Secuencia de procesamiento (máquina, tiempo)		
0	0, 75	2, 54	1, 59
1	0, 47	2, 72	1, 45

Tabla 2.1: Instancia simple con 3 maquinas y 2 trabajos

La siguiente es una posible planificación para la instancia de ejemplo, visualizada mediante un diagrama de gantt. En negro se marcan los trabajos que conforman la ruta crítica.

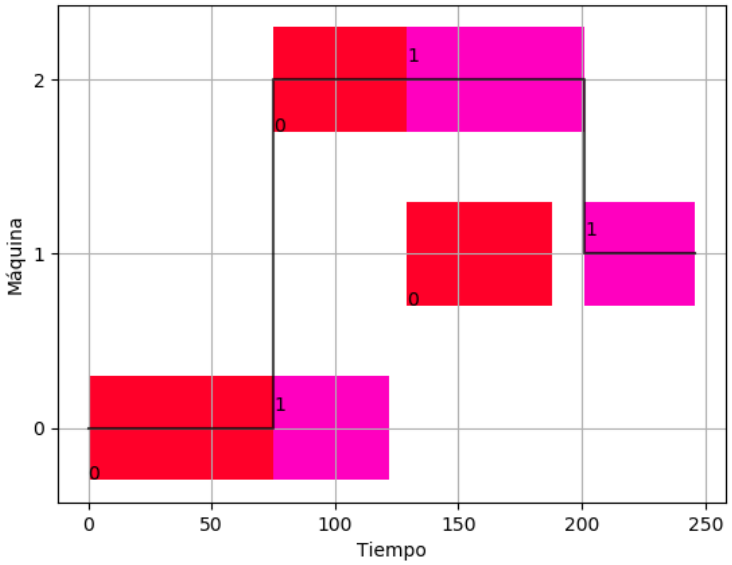


Figura 2.6: Diagrama de gantt de una planificación posible

Si bien es conveniente visualizar una planificación mediante un diagrama de gantt como el mostrado anteriormente, resulta ventajoso representar computacionalmente estas planifaciones de otras formas. La representación de soluciones a un problema resulta ser una elección sumamente importante al momento de desarrollar, implementar y analizar los algoritmos que se propongan para resolver el problema[31]. En la siguiente sección se desarrollan dos formas de representación de especial importancia.

2.5.1. Representación de planificaciones

De manera general existen dos formas de representar las soluciones al JSP [10]:

- **Representación directa** Se almacena el orden de los trabajos en cada máquina o sus tiempos de inicio y final.
- **Representación indirecta** Se almacena información con la que se puede construir una planificación mediante un proceso de decodificación.

En este trabajo se utilizaron dos: el grafo disyuntivo (representación directa) y las reglas de prioridad (representación indirecta).

Modelo de grafo disyuntivo

En este modelo las planificaciones se representan con un grafo dirigido $G = (V, A, E)$ en el que V es un conjunto de nodos que representa las operaciones, las aristas A representan la secuencia que deben seguir las operaciones dentro de un mismo trabajo y E es otro conjunto de aristas que indica el orden de procesamiento en cada una de las máquinas. Es importante mencionar que con este modelo podemos representar planificaciones no factibles, esto se da cuando el grafo G contiene un ciclo.

Formalmente en una instancia del JSP se representa cada operación como un nodo, se agregan dos nodos de control que sirven como el nodo inicial (del que dependen todos los trabajos) y final (que depende de todos los trabajos), las restricciones de precedencia dentro de cada trabajo se representan como aristas dirigidas fijas llamadas aristas conjuntivas y las operaciones que deben procesarse en una misma máquina se unen mediante aristas llamadas disyuntivas, una solución factible o planificación se obtiene al elegir la dirección para cada arista disyuntiva de modo que no se generen ciclos.

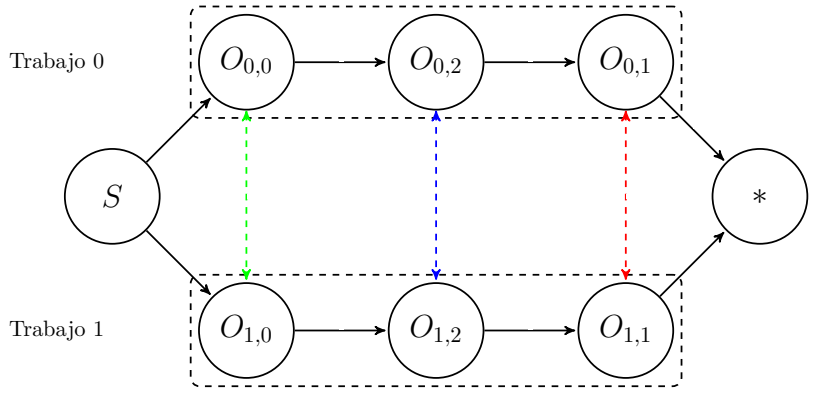
Reglas de prioridad

En esta representación una planificación se construye al aplicar un proceso de simulación en el que para cada maquina se construye una cola con las operaciones cuyas dependencias ya han sido procesadas. Inicialmente se tienen en las colas solo las operaciones iniciales de cada trabajo. Una vez que se tiene esto se utiliza una regla de prioridad para elegir qué operación debe planificarse en qué máquina. Se actualizan las colas para las máquinas que lo requieran y se continua con este proceso hasta completar la planificación (vaciar las colas).

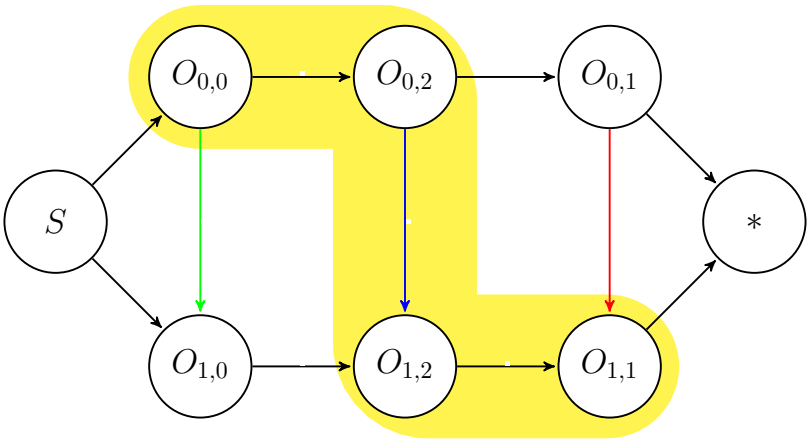
Existen muchas reglas de prioridad que toman en cuenta cosas como la duración de la operación, la cantidad de operaciones restantes, la duración del trabajo al que pertenece una operación, entre muchas otras. La calidad de la planificación construida depende de la regla de prioridad que se utilice y de la estructura de la instancia en sí.

2.5.2. Tipos de planificaciones

Independientemente de cómo se representen o como se construyan las planificaciones pueden clasificarse en varios conjuntos. Dentro de el conjunto de planificaciones factibles se pueden distin-



(a) Representación de una instancia, los colores distinguen entre las tres máquinas.



(b) Planificación obtenida al fijar las aristas disyuntivas como en 2.6. La ruta critica se resalta en amarillo

Figura 2.7: Modelo de grafo disyuntivo para la instancia de ejemplo 2.1

guir dos subconjuntos de interés para el presente trabajo: el conjunto de las planificaciones óptimas que está conformado por las planificaciones con el menor makespan posible y el conjunto de las planificaciones activas. Estas últimas se definen como las planificaciones en las que no es posible disminuir el tiempo de inicio de ninguna operación sin aumentar el tiempo de inicio de otra. Es conocido que las planificaciones óptimas representan un subconjunto de las activas[29].

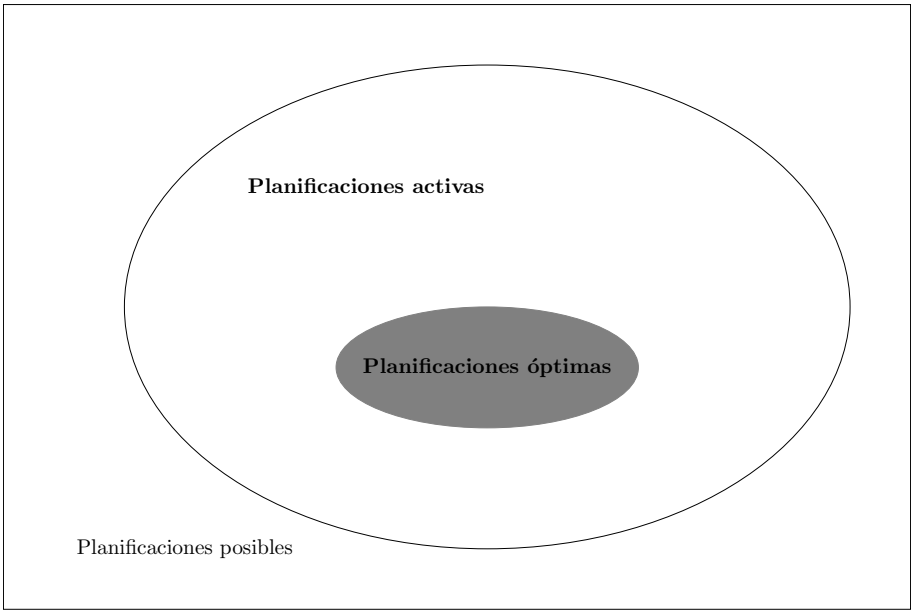


Figura 2.8: Subconjuntos de planificaciones

Estas clasificaciones resultan interesantes porque existen algoritmos para generar tipos de planificaciones de modo que podamos centrarnos en solo un subconjunto del espacio de búsqueda.

2.6. Metaheurísticas aplicadas al JSP

La complejidad del JSP hace que las metaheurísticas sean actualmente los métodos más utilizados para resolver instancias grandes. Estas han conseguido hallar buenas soluciones para los conjuntos de prueba más populares. Se han empleado una gran variedad de metaheurísticas como: algoritmos genéticos[11], evolución diferencial[29], algoritmos de colonia de hormigas[23], algoritmos de enjambre de partículas[42] y búsqueda tabú [40]. Aunque estos son algoritmos muy diferentes entre sí, la mayoría de ellos aplican de alguna forma el concepto de búsqueda local para mejorar su desempeño.

Para aplicar el proceso de búsqueda local es necesario introducir una definición de vecindad, como se ha mencionado anteriormente, esta estructura es parte del paisaje de búsqueda y tiene un gran impacto en los resultados obtenidos. Desde que se propuso la primera vecindad en 1996, se han propuesto vecindades cada vez con mejores resultados. A continuación se presentan las vecindades más importantes hasta la fecha.

Vecindades previamente propuestas

Todas las vecindades presentadas aquí se basan en el concepto de ruta crítica y solo consideran cambios dentro de los bloques críticos. En las figuras siguientes se presentan figuras ilustrativas para cada una de estas definiciones.

- N1 [6] Consiste en considerar todas las soluciones que se crean al intercambiar cualquier par de operaciones adyacentes que pertenecen a un bloque crítico. Aunque esta vecindad no es particularmente grande ya que es aproximadamente del tamaño de la ruta crítca, considera muchos cambios que no mejoran el makespan.

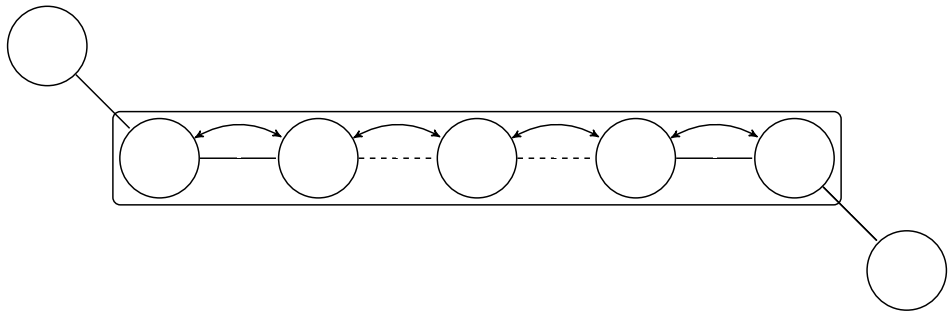


Figura 2.9: Movimientos de la vecindad N1

- N4 [13] Esta vecindad se propuso como un refinamiento y extensión de la vecindad N1 y toma como base el concepto de bloque crítico. Consiste en llevar operaciones internas del bloque crítico al inicio o final. Esta vecindad es de tamaño comparable a la anterior pero presenta movimientos más perturbativos.

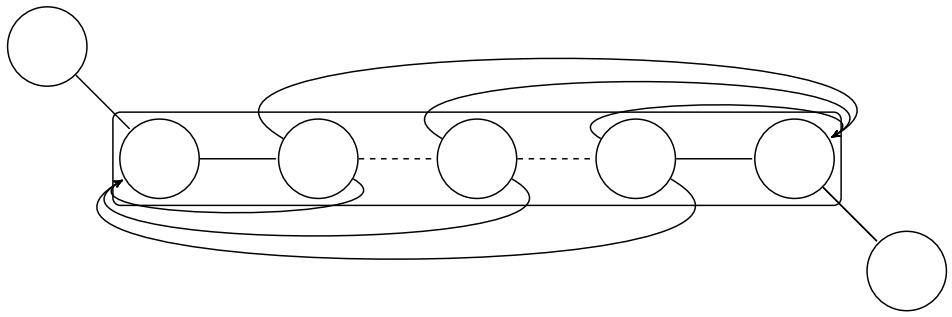


Figura 2.10: Movimientos de la vecindad N4

- N5 [15] Consiste en intercambiar solo las operaciones adyacentes a la final o inicial de un bloque crítico. Es aun más pequeña que la anterior ya que es aproximadamente del tamaño del número de bloques críticos.

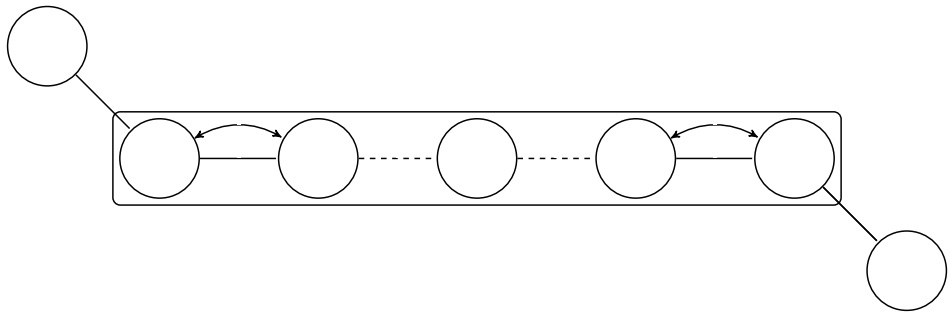


Figura 2.11: Movimientos de la vecindad N5

- N6 [2] Los autores utilizan varios teoremas para identificar pares (u, v) de operaciones dentro de un bloque crítico que puedan llevar a mejorar la solución y a su vez identificar si se tiene que mover a u justo después de v(forward) o bien a v justo antes de u (backward). Esta vecindad puede verse com un refinamiento y extensión de la N4 por lo que tiene un tamaño similar.

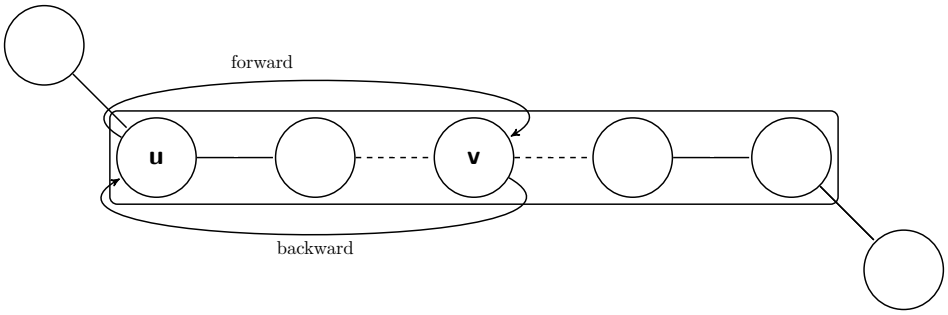


Figura 2.12: Los dos tipos de movimientos para un par (u, v)

- N7 [40] Esta vecindad se plantea como una extensión de la N6 en la cual se toma la idea de los movimientos entre pares de operaciones de un bloque crítico. Los autores toman en cuenta todos los cambios posibles entre el inicio o fin del bloque crítico con todas las operaciones internas.

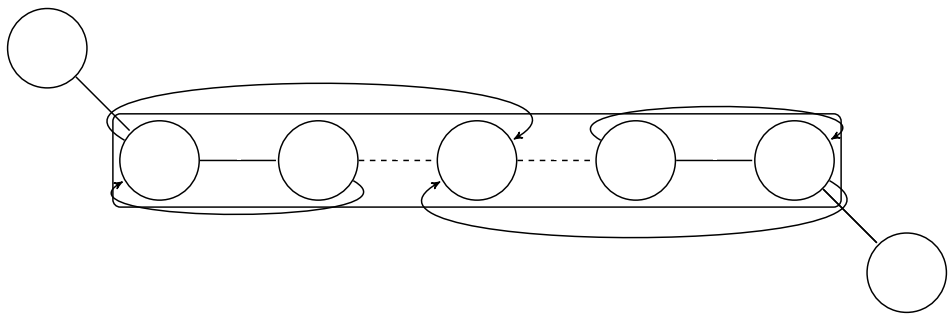


Figura 2.13: Movimientos de la vecindad N7

Las vecindades antes presentadas se han centrado en operaciones que pertenecen a la ruta crítica, esto les confiere las siguientes ventajas: la vecindad resultante es suficientemente pequeña como para ser explorada en su totalidad, para deducir el makespan de una planificación necesariamente deben hacerse cambios en la ruta crítica y puede garantizarse que cualquier vecino representa una solución factible[1].

La literatura reciente se ha centrado en hacer los algoritmos existentes más eficientes dejando de lado la parte de la representación o de la función de fitness. Existen otras ideas que no se han explorado a fondo pero parecen prometedoras como proponer extensiones a alguna de las vecindades, cambiar la representación y proponer una función de fitness que no solo tome en cuenta el makespan de modo que tengamos más formas de diferenciar entre soluciones. En la siguiente sección se presentan algunas otras medidas de calidad para planificaciones del JSP.

Criterios de optimalidad

Como ya se ha mencionado el criterio de optimalidad más usado es el makespan, no obstante existen muchos otros criterios de optimalidad que pueden usarse para asignar un valor de fitness a una planificación.

Si denotamos por C_i al tiempo de finalización de la máquina i y por $f_i(C_i)$ a su costo asociado, podemos distinguir dos tipos de funciones de costo en la literatura[8]:

$$f_{\text{máx}} := \text{máx}\{f_i(C_i)\}$$

y

$$\sum f_i(C) := \sum_{1 \leq i \leq n} f_i(C_i)$$

Los costos asociados a cada uno de los tiempos de finalización de los trabajos pueden tomar muchas formas, por ejemplo pueden introducirse pesos para cada trabajo o fijar tiempos de finalización esperado para cada trabajo y medir la desviación de ellos.

Dependiendo del problema en sí, puede ser que se le de más o menos valor a distintos aspectos de la planificación como el tiempo que están detenidas las máquinas, o el tiempo que tarda un trabajo en particular. El makespan se ha extendido como criterio de optimalidad porque está muy relacionado con los costos económicos de la planificación[30].

Con las definiciones anteriores sobre vecindades, función objetivo y representaciones de las soluciones podemos construir el paisaje de búsqueda para el JSP.

2.7. Paisaje de búsqueda del JSP

Analizar el paisaje de búsqueda de un problema de optimización combinatoria puede ayudar a escoger o diseñar metaheurísticas aunque en la práctica resulta muy díficil de hacer por el tamaño

del espacio de soluciones. Se ha observado que los problemas de optimización combinatoria a menudo presentan un un patrón *fácil-difícil-fácil* [24] de acuerdo con la dificultad para cumplir las restricciones impuestas: el problema es fácil de resolver si tiene constricciones muy débiles o muy fuertes y es difícil si está un punto intermedio. La explicación más extendida para este fenómeno consiste en la competencia de dos factores: por un lado un aumento de las restricciones que implica una reducción de las soluciones factibles a la vez que causa una disminución en el número de conexiones entre soluciones.

Para el JSP se han realizado estudios con instancias pequeñas en los que se puede construir explícitamente todo el paisaje, i. e. se puede evaluar y construir la vecindad correspondiente a todas las soluciones. Con este paisaje construido pueden medirse cantidades de interés como la similitud entre óptimos locales o globales, el número de vecinos entre otros que dan pistas de la forma (como las mostradas en la figura 2.4) que tiene el paisaje de búsqueda.

Se ha encontrado que el paisaje de búsqueda para una instancia al azar sigue el patrón *fácil-difícil-fácil* dependiendo de la razón entre el número de máquinas y número de trabajos de la misma[35] ($\frac{N}{M}$) conforme esta cantidad varía de 0 a infinito, siendo el caso más *difícil* cuando $\frac{N}{M} \simeq 1$. También se ha encontrado este patrón al medir la fuerza de las constricciones la fuerza de las constricciones se mide al comparar el número de soluciones factibles contra el número de bits necesarios para representar una solución[4].

Los experimentos computacionales sugieren que este patrón se da por cambios en la rugosidad del paisaje de búsqueda que se vuelve más y más pronunciada conforme $\frac{N}{M}$ o la fuerza de las constricciones crecen, para valores pequeños se tiene un paisaje semejante a un gran valle suave, para valores grandes se tiene un pasaje con muchos óptimos locales de buena calidad y para valores cercanos a 1 se tienen muchos óptimos locales de baja calidad.

También se ha encontrado que una de las características que hacen difícil a una instancia del JSP es la existencia de muchos óptimos locales de baja calidad que no son parecidos entre sí [25]. Esto implica que sin importar el punto inicial de la metaheurística elegida rara vez estaremos muy lejos de uno de estos puntos. Otro factor que afecta el proceso de búsqueda local es que en general se ha encontrado que algunas soluciones están mucho mas conectadas que otras [5]. Este último hecho puede ser tanto benéfico como perjudicial dependiendo si las soluciones altamente conectadas son de buena o mala calidad.

Estas características explican en parte por qué los métodos basados en búsqueda local como la búsqueda tabú 2 combinados con métodos sofisticados de exploración han tenido tan buenos resultados[37]. Puede ser que las soluciones que estén más conectadas a otras sean de baja calidad y sea necesario evitar ser «atraídos» hacia ellas para llegar a soluciones de mejor calidad. También

se ha observado que el paisaje de búsqueda para el JSP presenta grandes «planices» i. e. zonas en las que todas las soluciones vecinas tienen el mismo makespan, en la práctica estas zonas actuan como un solo óptimo local que está altamente conectado.

En este sentido, podemos pensar que para que una metaheurística basada en búsqueda local tenga mayor probabilidad de encontrar buenas soluciones debemos plantear el paisaje de búsqueda de modo que no nos atasquemos en óptimos locales de mala calidad, ya sea porque están muy conectados y funcionan como atractores o bien porque el paisaje de búsqueda es tan rugoso que hay óptimos locales por doquier y es fácil atascarse en uno de mala calidad. Las soluciones altamente conectadas también pueden servir como un punto de partida hacia una solución de mejor calidad.

Capítulo 3

Propuestas

En este capítulo se detallan las modificaciones hechas a cada una de las componentes del paisaje de búsqueda.

3.1. Búsqueda local iterada

Se propone utilizar la metaheurística de búsqueda local iterada por su simplicidad. Como se muestra en el algoritmo 3 la búsqueda local iterada requiere que definamos una manera de perturbar una solución dada así como definir un criterio de paro. La perturbación debe ser suficientemente grande como para permitirnos escapar de un óptimo local pero no tan grande como para eliminar toda la estructura que se tiene hasta el momento.

Por estas razones y para evitar complicar más el algoritmo con la definición de operadores completamente nuevos que realicen cambios arbitrarios, la perturbación implementada consiste simplemente en reemplazar a la solución actual por un vecino suyo (de acuerdo con la definición de vecindad que se esté usando). Esta definición es conveniente porque presenta una manera de aceptar cambios que no mejoran la solución pero que a su vez pueden estar conectados a soluciones de mejor calidad que la mejor actual.

También es necesario establecer un criterio de paro, en este caso fue el tiempo, se tomaron 5 minutos lo cual representa una cantidad de tiempo muy pequeña comparada con la requerida por los métodos del estado del arte.

3.2. Representación de soluciones activas basada en llaves aleatorias

La representación propuesta se basa en asignar a cada operación una número real entre 0 y 1 el cual sirve para definir un orden entre operaciones en una misma máquina mediante un proceso de decodificación un planteamiento similar puede encontrarse en [3, 27, 29].

Para decodificar la solución a partir de las llaves se utiliza el algoritmo de Giffler & Thompson [17] con el cual se generan soluciones activas.

Una consecuencia importante de este cambio en la representación es que el espacio de búsqueda se reduce a solo las soluciones activas y como se sabe que las planificaciones óptimas son un subconjunto de estas, esta representación puede representar cualquier solución óptima.

Un punto negativo es que esta representación es n a 1 porque solo importa el tamaño relativo de las llaves que compiten entre sí, es decir que diferentes valores de llaves pueden llevarnos a la misma solución. En principio no parece un problema muy grave aunque sería más eficiente tener una representación 1 a 1.

A continuación se presenta el algoritmo para construir planificaciones activas de llaves numéricas.

3.2.1. Algoritmo de Giffler & Thompson

Para explicar el algoritmo de Giffler & Thompson se adoptan las siguientes notaciones:

- O_i la operación i.
- $M(O_i)$ la máquina en la que debe procesarse la operación O_i.
- $t_f(O_i)$ el tiempo en que se terminaría de procesar la operación O_i si se planifica en este paso.
- $t_i(O_i)$ el tiempo en que se comenzaría a procesar la operación O_i si se planifica en este paso.
- $k(O_i) \in [0, 1]$ el valor de la llave asignada a O_i.

El primer paso es marcar como planificable la primera operación de cada trabajo. Posteriormente se identifica la operación O_{min} con el menor tiempo de finalización si se planificase en este paso y la máquina $M^* = M(O_{min})$ en la cuál debe procesarse. Se toma el tiempo de finalización $t_f^* = t_f(O_{min})$ para escoger alguna de las operaciones planificables en m^* cuyo tiempo de inicio sea menor a t_f^*, se actualizan las operaciones planificables y este proceso se repite hasta completar la planificación. El algoritmo es el siguiente:

Algorithm 4: Algoritmo de Giffler & Thompson

Data: Instancia del JSP

Result: Planificación activa

Inicializar el conjunto de operaciones planificables Ω;

while Ω *no vacío* **do**

Determinar la operación con el menor tiempo potencial de finalización $O_{min} = \arg\min_{O \in \Omega} \ t_f(O)$;

Determinar el tiempo de finalización t_f^* y la máquina M^* en que se procesa O_{min};

Identificar el conjunto $C \subset \Omega$ de operaciones que cumplen $t_i(O) < t_f^*$ y $M(O) = M^*$;

Escoger la operación $O^* \in C$ que tenga asignada la llave de mayor valor;

Asignar tiempo de inicio y fin a O^*;

Actualizar Ω eliminando a O^* y agregando a su sucesora si existe;

Ejemplo

Como ejemplo ilustrativo se muestran los primeros pasos para construir la planificación mostrada en la figura 2.6 asociada a la instancia de ejemplo mostrada en la tabla 2.1.

Para recuperar la planificación previamente mostrada se asignan las llaves de la siguiente manera:

$$k(O_{00}) = k(O_{01}) = k(O_{02}) = 1$$

$$k(O_{10}) = k(O_{11}) = k(O_{12}) = 0$$

- Se inicializa Ω agregando las operaciones iniciales de cada trabajo, es decir:

$$\Omega = \{O_{10}, O_{00}\}$$

- Se identifica a la operación que acabaría primero si se planifica ya, en este caso es O_{10} que se debe procesar en la máquina M_0.

- Se identifican las operaciones que podrían comenzar a procesarse antes del tiempo potencial de finalización de O_{10}. En este caso solo O_{00} y O_{10} cumplen las condiciones.

  Como $k(O_{00}) > k(O_{10})$ se planifica O_{00} se le asignan 0 y 54 como tiempos inicial y final.

- Se actualiza Ω removiendo a la operación planificada y agregando a su sucesora O_{02}.

$$\Omega \leftarrow \{O_{10}, O_{02}\}$$

Este proceso continua de acuerdo al algoritmo 4 hasta que todas las operaciones se encuentren planificadas.

3.2.2. Construcción de soluciones activas a partir de no activas

Hasta ahora se ha considerado que la asignación de las llaves a las operaciones se hace de manera aleatoria. Esta asignación también puede hacerse tomando como base una solución previamente dada que no necesariamente tiene que ser activa.

Como se vio en el modelo de grafo disyuntivo construir una planificación consiste en elegir direcciones de aristas disyuntivas. Esto al final genera un orden para las operaciones en cada máquina i.e. una permutación. Para hallar la representación de alguna de estas soluciones en el esquema propuesto se toma esta permutación de las operaciones para cada máquina y se le asigna a cada una una prioridad decreciente acuerdo a su posición en la permutación.

$$k(O_i) > k(O_{i+1})$$

donde O_i es la operación número i en procesarse

Esta asignación nos asegura que se mantenga el orden de muchas operaciones pero convierte a la solución en activa. A continuación se muestra un ejemplo ilustrativo

Ejemplo

Consideremos la instancia del JSP mostrada en la siguiente tabla:

Trabajo	Secuencia de procesamiento (máquina, tiempo)		
0	1, 37	0, 13	2, 35
1	0, 8	1, 4	2, 9
2	1, 34	2, 72	0, 96
3	1, 63	0, 77	2, 24

Tabla 3.1: Instancia 3 maquinas y 4 trabajos

La siguiente figura muestra un diagrama de gantt para una posible planificación.

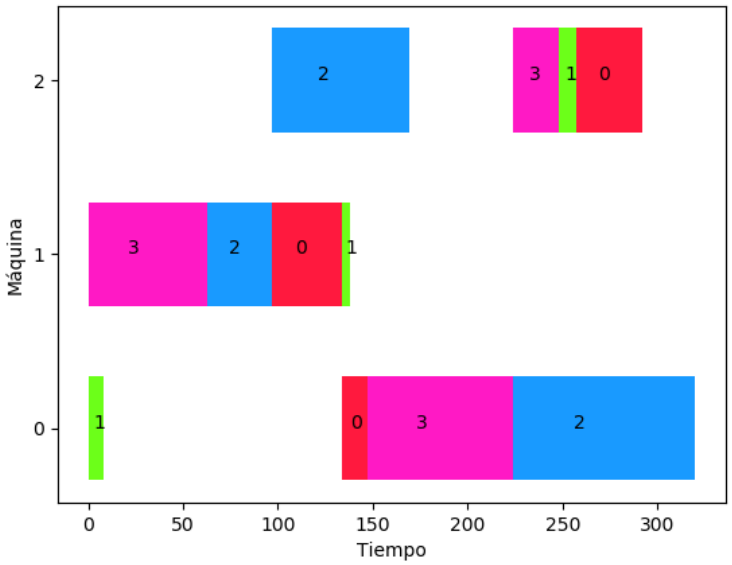


Figura 3.1: Planificación posible para la instancia en 3.1.

A partir de la planificación mostrada podemos asignar las prioridades del modo previamente dicho con lo que después de construir la planificación a partir de estas prioridades obtenemos la planificación que se muestra en la siguiente figura.

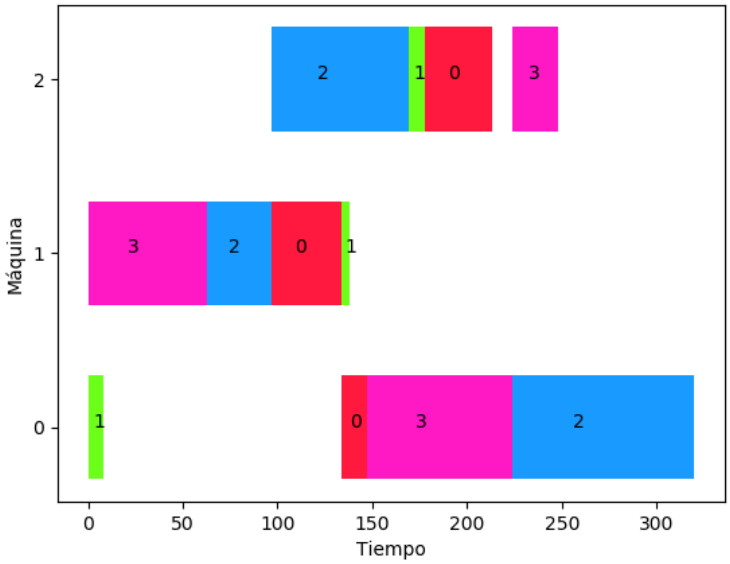


Figura 3.2: Planificación activa reconstruida a partir de la mostrada en 3.1

Podemos notar como dos operaciones cambian de lugar de modo que se planifican antes sin aumentar el tiempo de inicio de otras.

Una característica importante de esta representación es que el algoritmo usado para decodificar una solución activa a partir de las llaves numéricas considera en cada paso varias operaciones candidatas a planificar que cumplen con los criterios antes mencionados. Estas operaciones son la pieza en la que se basa la siguiente propuesta para una estructura de vecindad.

3.3. Vecindad basada en soluciones activas

Esta vecindad surge del cambio de representación propuesto. En cada paso del algoritmo para construir una solución activa se consideran varias operaciones que «compiten» para ser planificadas (i.e. son planificables en ese momento) de las cuales se elige la que tiene la llave de mayor valor numérico.

La idea es construir la vecindad a partir de estas operaciones que compiten. En un principio puede pensarse en considerar todos los posibles ordenamientos posibles para dichas operaciones pero esto da lugar a una vecindad demasiado grande por lo que se considera únicamente hacer cambios por pares de llaves entre la operación elegida y todas sus competidoras.

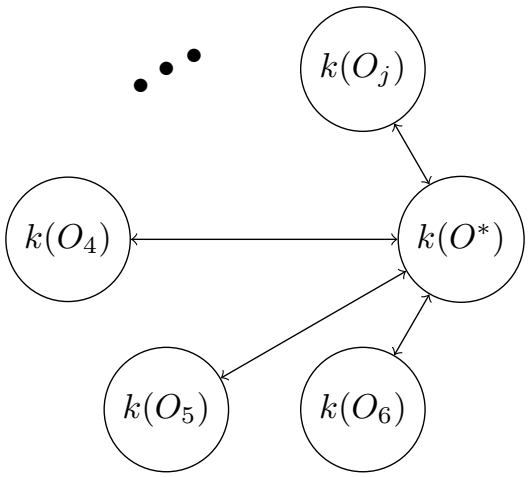


Figura 3.3: Movimientos de la vecindad propuesta. $k(O^*)$ es la llave de la operación elegida, las flechas representan los posibles intercambios entre las j operaciones competidoras.

El tamaño de esta vecindad puede ser bastante grande ya que para cada paso del algoritmo 4 se tienen tantos movimientos como operaciones competidoras, las cuales pueden ir desde 1 hasta el número de trabajos n siendo en el peor caso de tamaño $O(nm*n)$. Como se mostrará mas adelante en la validación experimental, en realidad el número de operaciones que compiten por lo general es solo una fracción pequeña de n con lo que la vecindad puede manejarse sin problemas.

Los movimientos de esta vecindad pueden generar soluciones muy diferentes dependiendo de dónde se encuentren en la planificación las operaciones consideradas. Si se cambia la llave de una operación que fue planificada al principio puede ser que la solución cambie en muchos otros lugares porque el conjunto de operaciones disponibles puede llegar a ser completamente distinto al llegar a un paso más avanzado en la construcción de la solución.

3.4. Extensión a vecindad N7

Puesto que cada vez tenemos a nuestra disposición más recursos computacionales como punto de partida se planteó agregar movimientos a la vecindad N7 con la cual se han obtenido los resultados del estado del arte. Los movimientos que plantea esta vecindad solo tienen que ver con pares de operaciones en la ruta crítica por lo que una extensión sencilla consiste en considerar movimientos de operaciones que pueden no pertenecer a la ruta crítica.

Es importante resaltar que si no se planteara también una función de fitness que no tome solo en cuenta el makespan estos movimientos podrían nunca llevarían a una mejora [6]. Los movimientos planteados se basan en observar que en alguna solución encontrada por una búsqueda local para cada maquina pueden existir periodos de tiempo en la que está inactiva pero existe alguna operación en la ruta critica que podría comenzar a procesarse en este periodo y que se procesa después. Ninguna de las vecindades previamente propuestas considera movimientos de las operaciones de la ruta crítica más allá del bloque crítico por lo que estos movimientos representan un conjunto previamente no explorado de soluciones.

Intuitivamente lo que se pretende es llenar un «hueco» en la planificación con una operación de un bloque crítico.

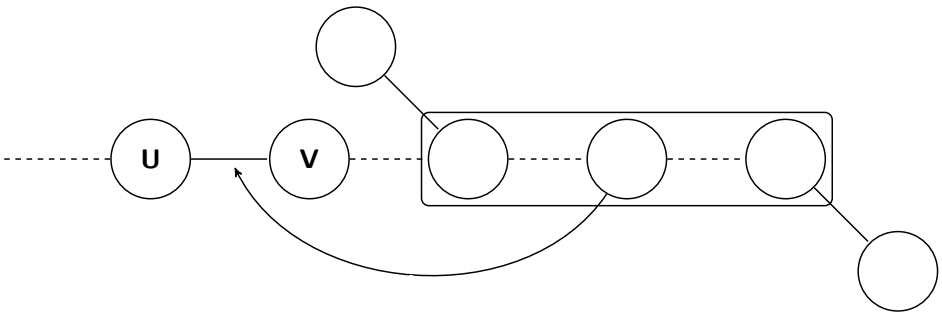


Figura 3.4: Movimientos propuestos. El tiempo de inicio de **v** es mayor al de finalización de **u**

En general esta vecindad resulta de un tamaño muy reducido porque existen muchos movimientos que hacen que la solución resultante sea infactible.

3.5. Función de fitness

La función de fitness es el elemento del paisaje de búsqueda al que se han reportado menos modificaciones en la literatura. En general cuando se utilizan otros criterios de optimalidad el problema se vuelve uno de optimización multiobjetivo[18, 32, 28] en lugar de construir una función objetivo nueva se ha propuesto tomar en cuenta el tiempo inactivo en la planificación[36] aunque con resultados no muy alentadores.

La idea principal de la propuesta de este trabajo es la de tener un arreglo de características de la planificación y compararlos lexicográficamente para distinguir entre las soluciones. La idea

principal es tener una forma de distinguir entre soluciones con el mismo makespan, esto nos ayuda a atravesar regiones que de otro modo serían «planas» que están plagadas de óptimos locales. En el mejor de los casos la adición de estas cantidades no solo nos permite atravesar planicies sino que también sirve como guía para mantener una planificación que se estanque prematuramente en óptimo local de mala calidad.

Para plantear qué características son las que se iban a tomar en cuenta para este arreglo se tomaron en cuenta otros criterios de optimalidad hallados en la literatura así como propuestas propias. También se planteó considerar todos los tiempos de finalización de las máquinas. Las características tomadas en cuenta fueron las siguientes(J_i es el tiempo de finalización del trabajo i y C_i el tiempo de finalización de la máquina i):

- $\mathbf{C_{max}} = \max \mathbf{C_i}$: Este es el criterio de optimalidad más usado y el más ampliamente reportado en la literatura.
- $\sum \mathbf{C_i^2}$: La ventaja de esta medida es que toma en cuenta todos los tiempos de finalización de las máquinas y al elevar al cuadrado las que toman más tiempo en acabar contribuyen mucho más que las que toman poco tiempo. Con esta medida podemos distinguir entre dos soluciones con el mismo makespan pero en la que se ha mejorado el tiempo de alguna máquina.
- $\sum \mathbf{J_i}$: También conocido como *Flowtime*, como mide los tiempos de finalización de los trabajos puede llevar a encontrar mejoras en distintas máquinas a la vez.
- $\sum \mathbf{I}(\mathbf{C_i} = \mathbf{C_{max}})$: Número de máquinas cuyo tiempo de finalización es igual al makespan. Entre más máquinas cumplan esta condición es menos probable que un solo cambio contribuya a mejorar el makespan ya que se tendría que reducir a la vez el tiempo de todas ellas.
- **Número de rutas críticas**. Esta medida está directamente relacionada con la anterior pero puede tener la ventaja de poder hacer cambios progresivos para eliminar las rutas críticas y lograr una mejora en el makespan.
- $\mathbf{Var}(\mathbf{C_i})$ Varianza de los tiempos de finalización. Esta medida es parecida a la suma de tiempos de finalización al cuadrado pero en este caso se mide la distancia a la duración promedio por lo que no solo se centra en las máquinas que tardan más sino también en las que se tardan menos. Esto puede ayudar a que no se planifiquen operaciones en una solo subconjunto de máquinas mientras que las otras permanecen en espera.
- $(\{\mathbf{C_i}\})$ Tupla de tiempos de finalización ordenados. Esta tupla engloba de cierto modo varias de las características previamente mencionadas pero tiene la ventaja de requerir menos trabajo computacional para construir al solo ser necesario ordenar cantidades ya conocidas.

Estas características se combinaron de varias formas como se mostrará en la validación experimental.

Capítulo 4

Validación experimental

En este capítulo se muestran los resultados obtenidos para cada una de las propuestas. Se muestran ordenados de forma que los cambios propuestos se van implementado progresivamente. Inicialmente se considera solo el uso de búsqueda local iterada con el makespan como fución de fitness y la estructura de vecindad N7 como punto de partida.

4.1. Conjunto de instancias de prueba

A lo largo de los años se han propuesto múltiples instancias de prueba para medir el desempeño de los algoritmos para la solución del JSP. Estos conjuntos de prueba han aumentado progresivamente en tamaño y dificultad.

Actualmente el conjunto más popular se debe a E. Demirkol, S. Mehta y R. Uzsoy [14]. Consiste en 80 instancias que van desde 20 trabajos y 15 máquinas (20×15) hasta 50 trabajos y 20 máquinas (50×20) y se conoce como **DMU01-80**. La segunda mitad de estas instancias **DMU40-80** son consideradas especialmente difíciles porque las operaciones iniciales de todos los trabajos tienen que ser procesadas en solo una fracción de las máquinas lo cual genera un cuello de botella al inicio de la planificación.

En este trabajo se ponen a prueba las modificaciones planteadas al comparar los resultados obtenidos con los mejores reportados en la literatura hasta la fecha.

Las modificaciones se presentan en el siguiente orden: en primer lugar se presentan los resultados de búsqueda local con la vecindad N7 y con función de fitness igual al makespan, es decir que el único cambio es en la metaheurística utilizada para tener una linea base. En segundo lugar se presentan las modificaciones hechas a la función de fitness. Posteriormente se presentan los resultados de la extensión a la vecindad N7. Por último se presentan los resultados del cambio de representación junto con la nueva estructura de vecindad basada en soluciones activas.

Cada una de las variaciones propuestas se ejecutó 50 veces para cada instancia para obtener un conjunto de resultados que podemos comparar.

Los resultados se muestran de modo que se va aumentando el grado de las modifcaciones.

Comenzamos con solo el cambio en metaheurística, seguido del cambio en la función de fitness, posteriormente se considera la extensión de vecindad planteada y por último el cambio de representación. El orden es el siguiente:

1. ILS sin modificaciones.

2. ILS con diferentes funciones de fitness.

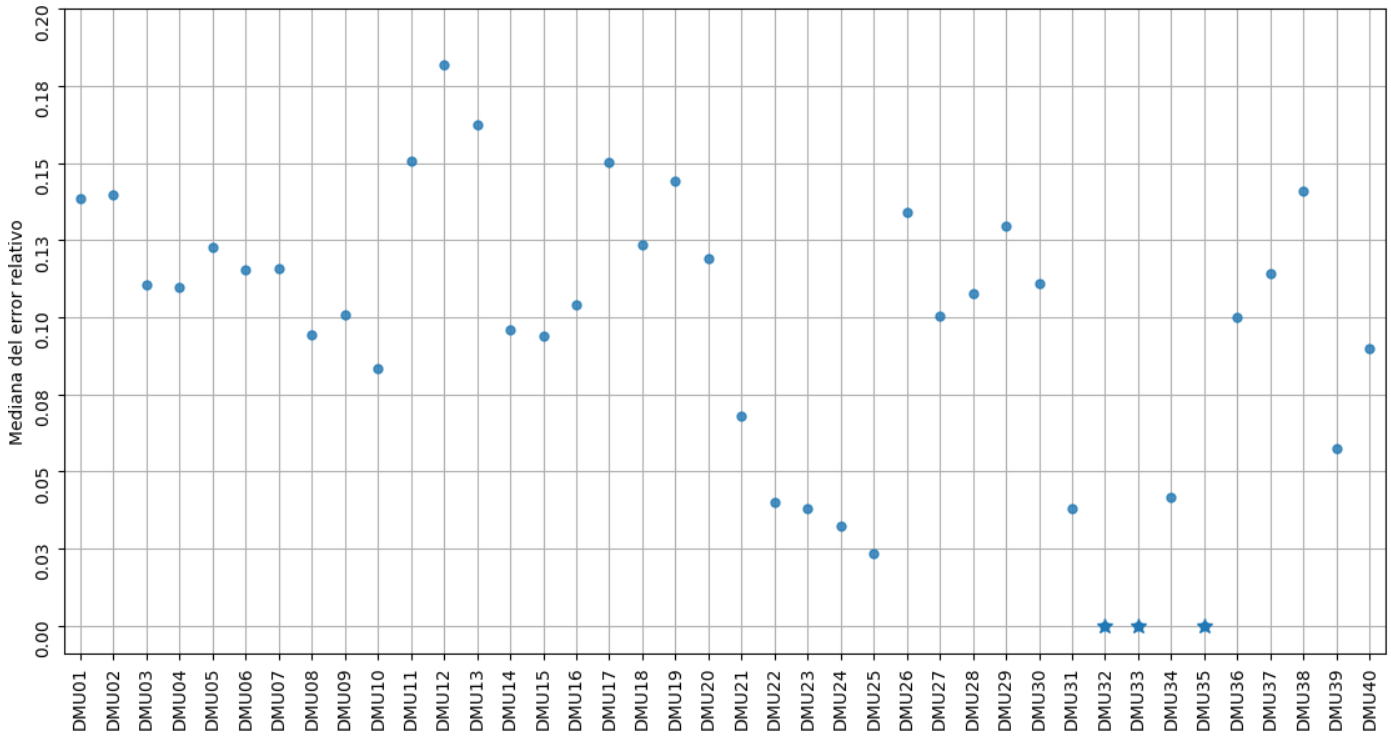
3. ILS con extensión de vecindad.

4. ILS con la representación y vecindades propuestas.

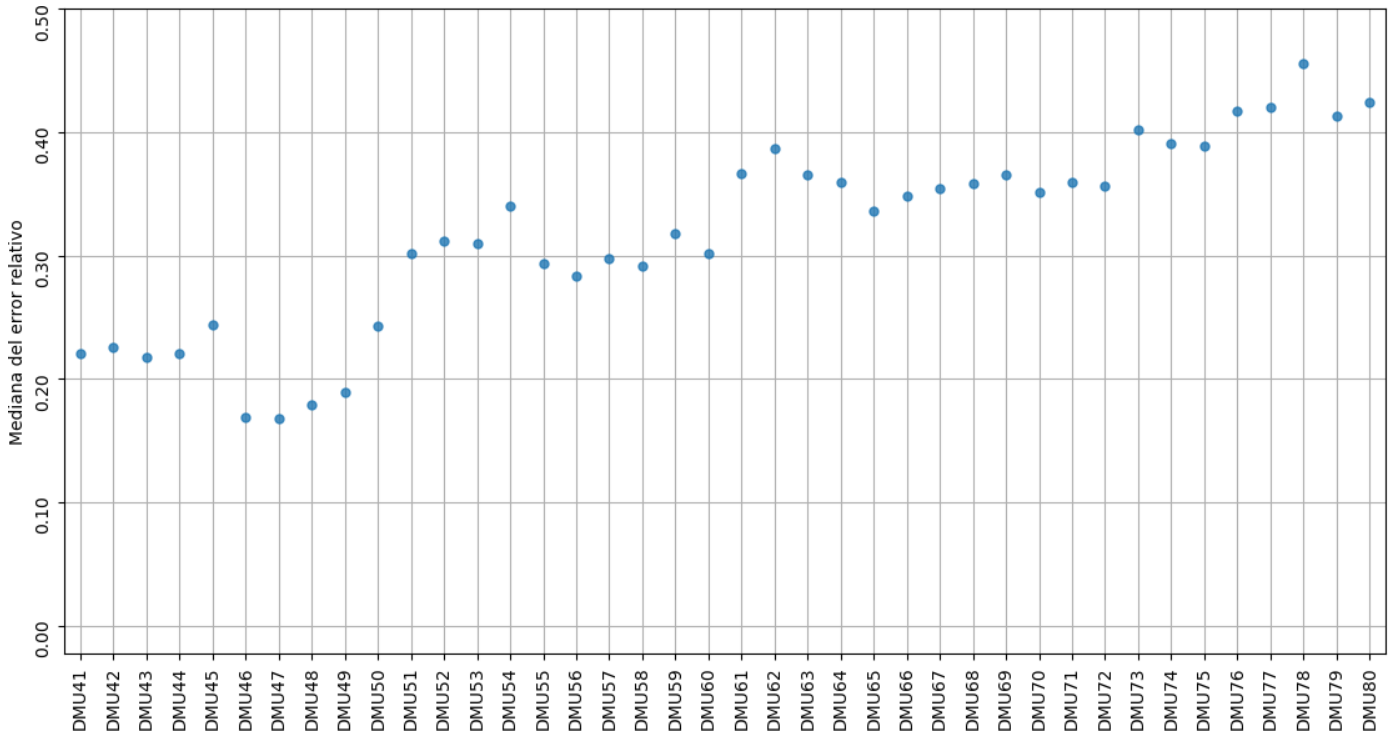
4.2. ILS con vecindad N7

A continuación se muestran los resultados de únicamente cambiar la metaheurística. Estos resultados sirven como una base para determinar si las modificaciones posteriores resultan en mejoras apreciables.

A continuación se muestran los resultados de manera gráfica para facilitar su visualización. Los resultados detallados se encuentran en el apéndice A.1. Para cada instancia se muestra la mediana del error relativo de los resultados con respecto a los mejores resultados reportados en la literatura mostrados en el ápendice A.1.



(a) Resultados para las instancias **DMU01-40**



(b) Resultados para las instancias **DMU41-80**

Figura 4.1: Resultados para la propuesta más simple. Se marcan los casos en los que se llegó a la mejor solución conocida.

4.3. Función de fitness

Para comparar los resultados obtenidos de las características propuestas para agregar al makespan en la función objetivo, se comparan por pares los conjuntos de resultados obtenidos para cada instancia. Para determinar si los conjuntos de resultados muestran diferencias estadísticamente significativas se utiliza la prueba de Wilcoxon con un nivel de significancia de 0,01.

Para determinar cuál es la función de fitness que obtiene mejores resultados las modificaciones a la función de fitness se comparan a pares. Si se encuentra que la diferencia entre dos modificaciones es estadísticamente significativa, se le suma un punto a la ganadora y se le resta uno a la perdedora. La función de fitness se obtiene al construir la dupla formada por el makespan y la característica en ese orden.

Los resultados para las propuestas mostradas en 3.5 pueden verse de manera condensada en la siguiente figura. El cuadro (i, j) muestra el número de veces que i fue mejor que j.
Todas las pruebas se realizaron con la vencindad N7. Se muestra también una tupla construida con las características que obtuvieron mejores resultados. Estas características se ordenan de acuerdo a su número de comparaciones ganadas totales .

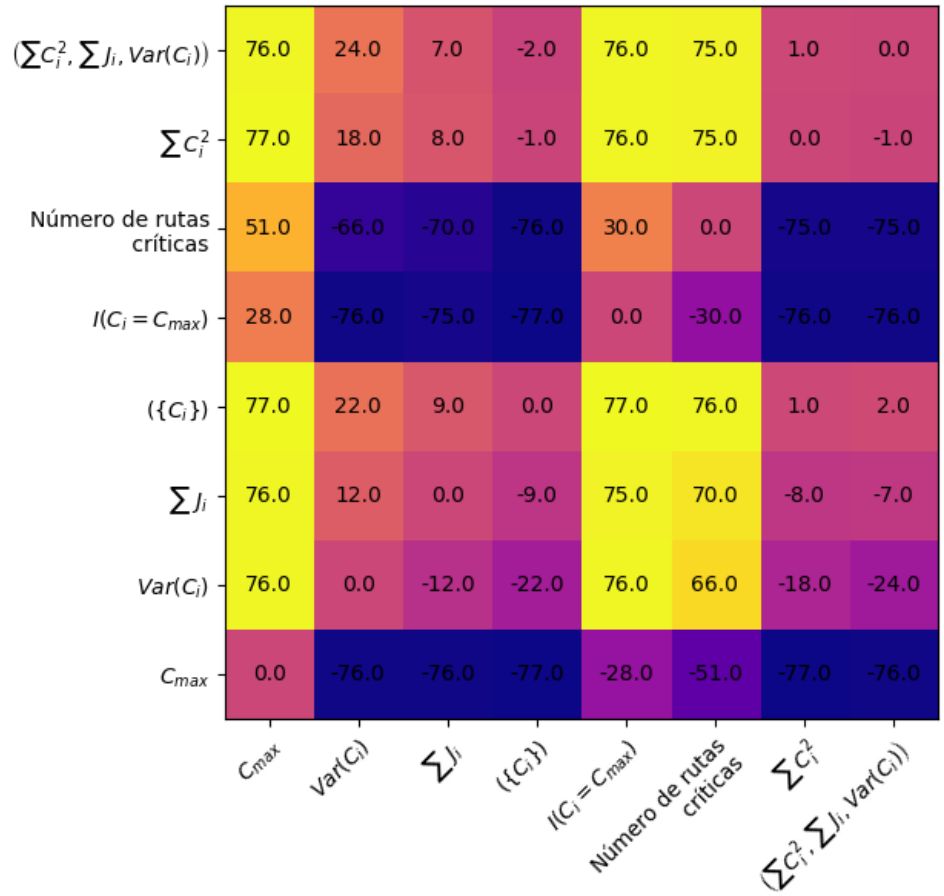


Figura 4.2: Condensado de los resultados para las modificaciones a la función de fitness.

Los mejores resultados se obtienen al construir la tupla mencionada previamente por lo que de ahora en adelante la función de fitness queda fija de esta manera y en los resultados subsecuentes solo se considera este caso. Los resultados detallados para este caso se muestran en el ápendice A.2

4.4. Extensión a vecindad N7

Los resultados para la extensión que considera movimientos con espacios de inactividad de las máquinas se comparan con el mejor mostrado en la sección pasada. A continuación se muestra una figura en donde se comparan estos métodos de la forma que se planteó en la sección pasada.

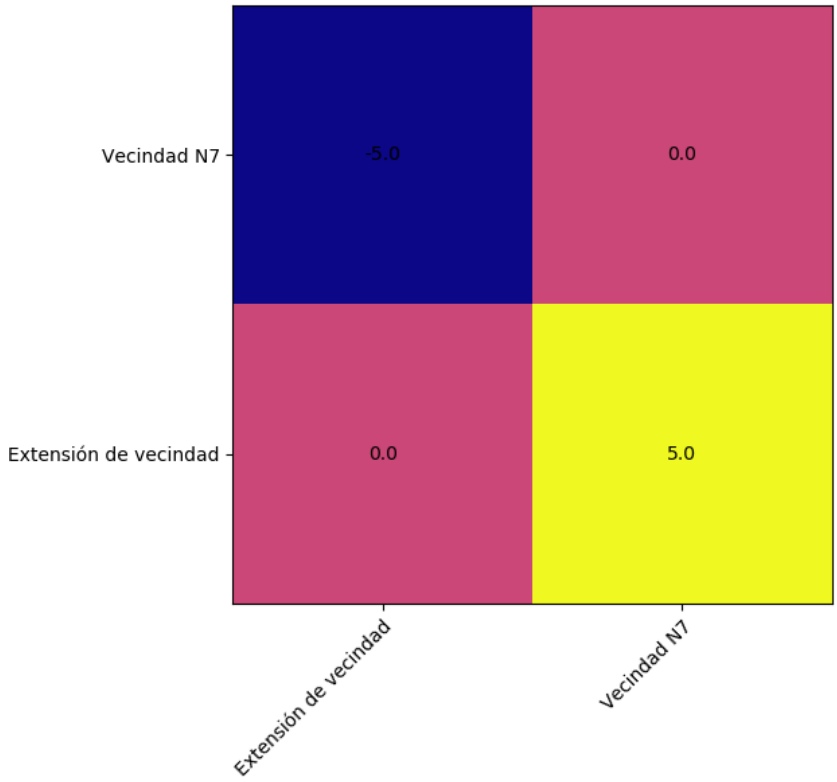
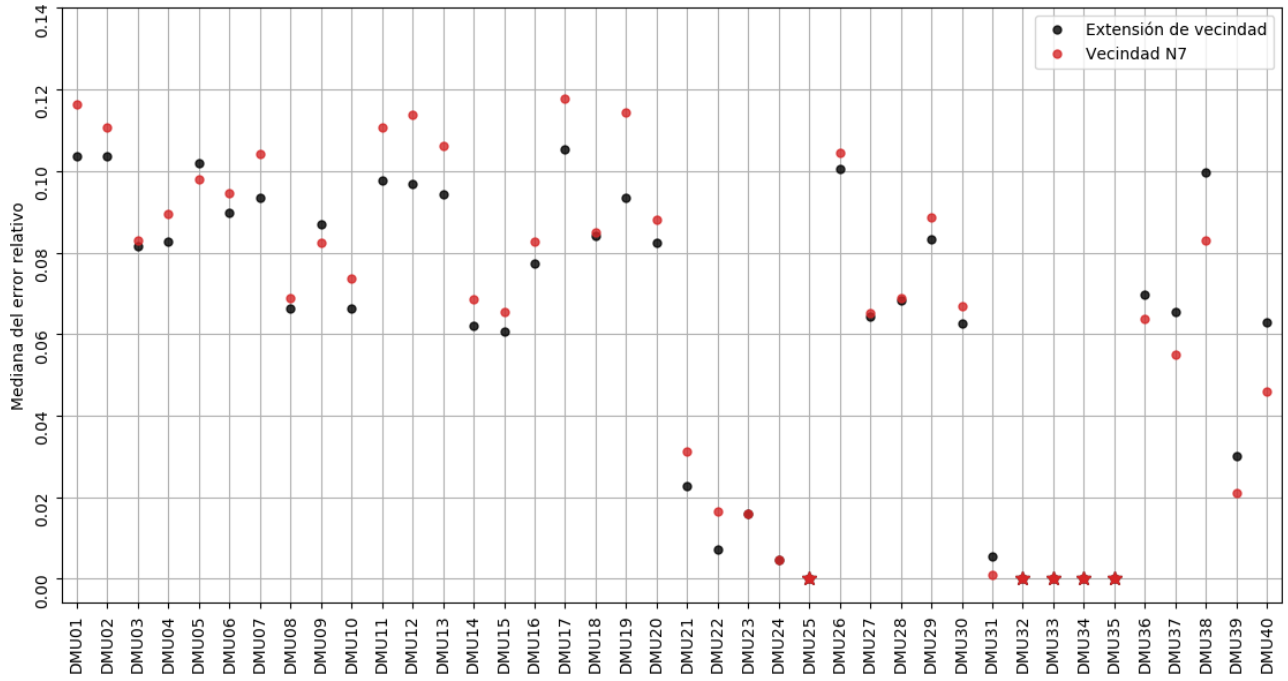
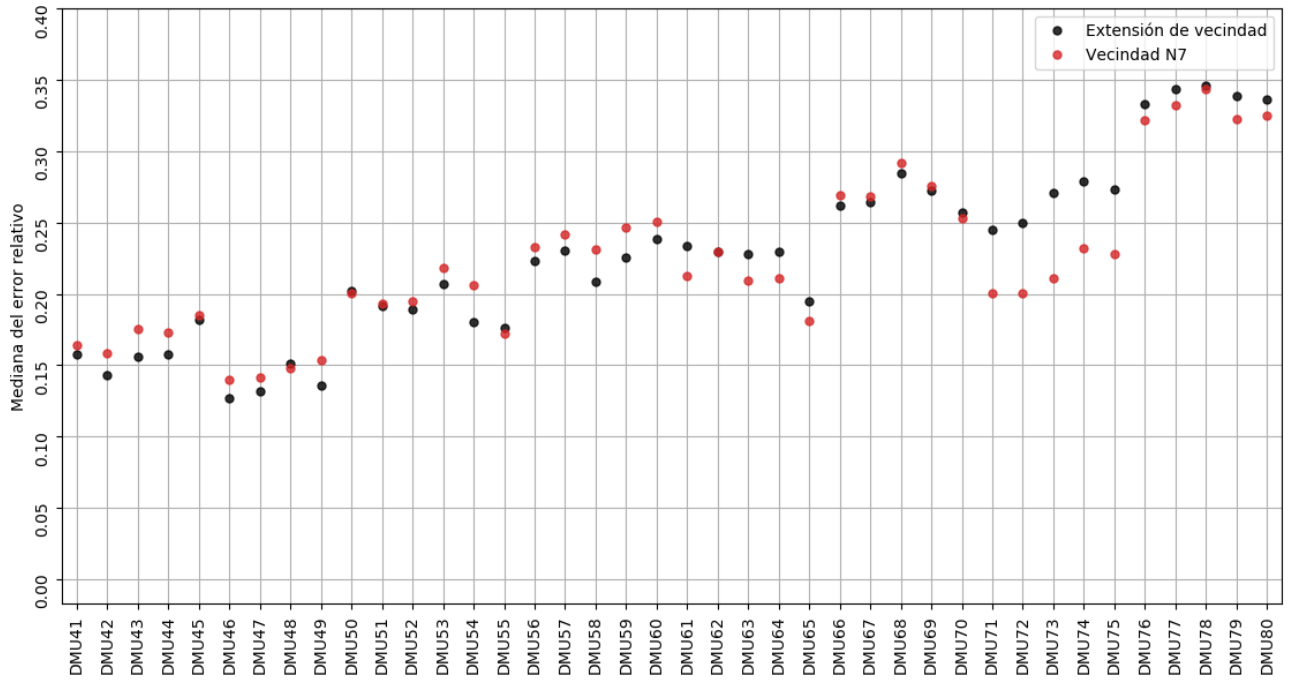


Figura 4.3: Resultados de la comparación

También se muestra de manera gráfica los resultados para todas las instancias. En la siguiente figura se muestra se muestra la mediana del error relativo para cada método e instancia. Los resultados detallados para la extensión propuesta se encuentran en el ápendice A.3.



(a) Resultados para las instancias **DMU01-40**



(b) Resultados para las instancias **DMU41-80**

Figura 4.4: Resultados para ambos métodos. Se marcan los casos en los que se llegó a la mejor solución conocida.

4.5. Cambio de representación y vecindad

El cambio de representación y vecindad se compara con las dos propuestas previas. Se sigue el mismo procedimiento mencionado para determinar si hay una diferencia significativa entre los resultados obtenidos.

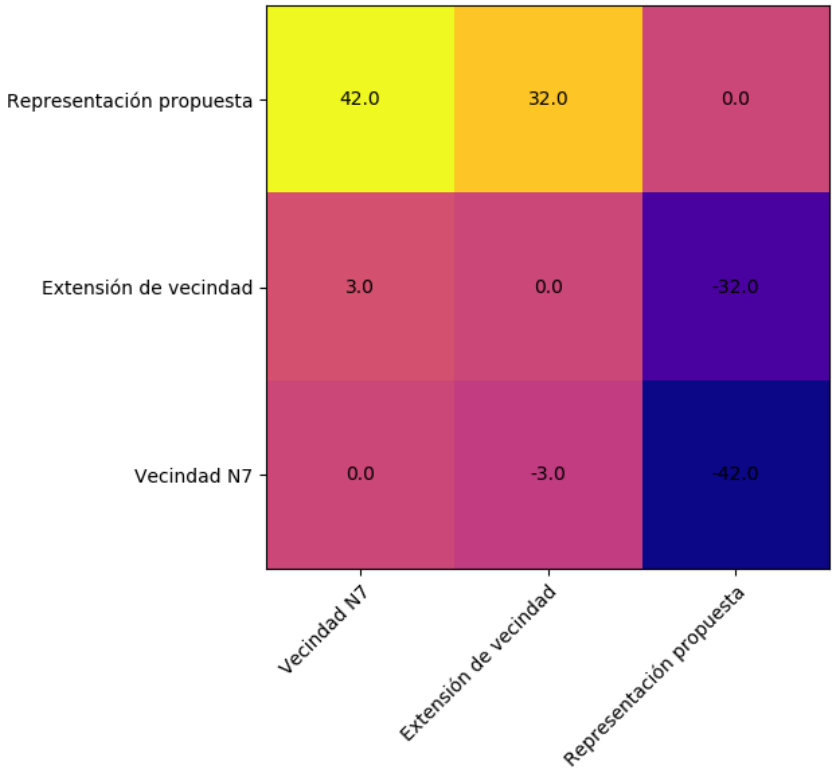
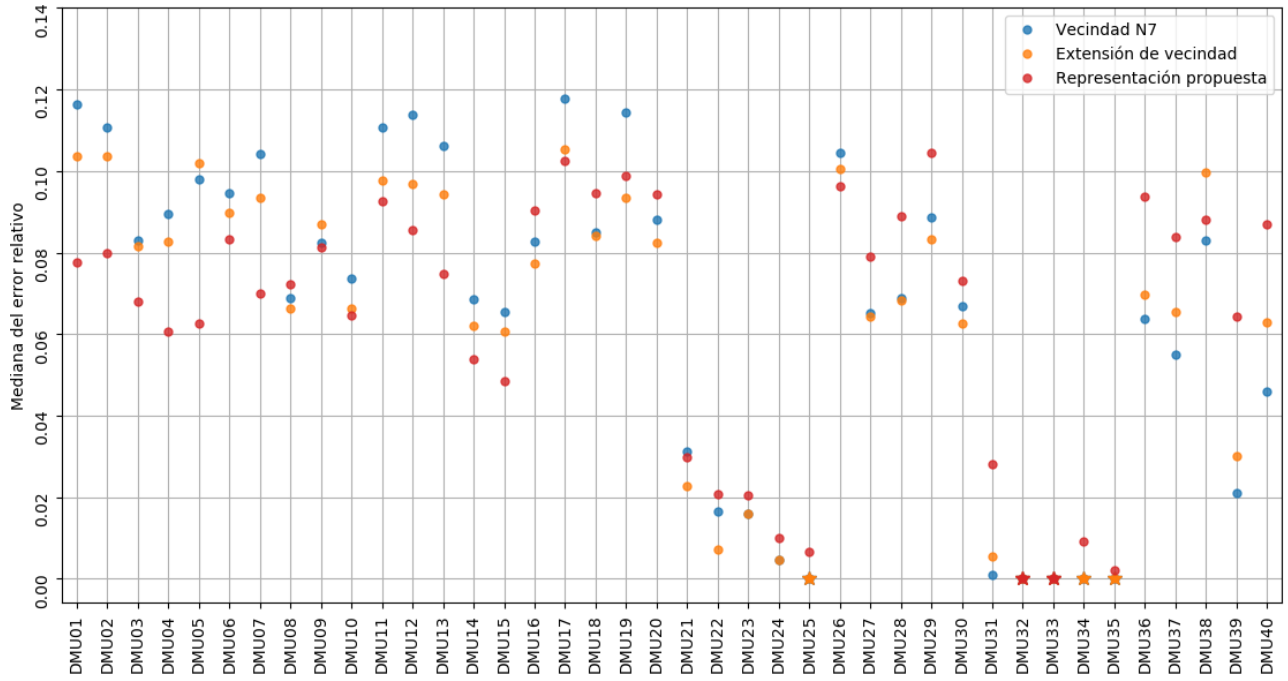
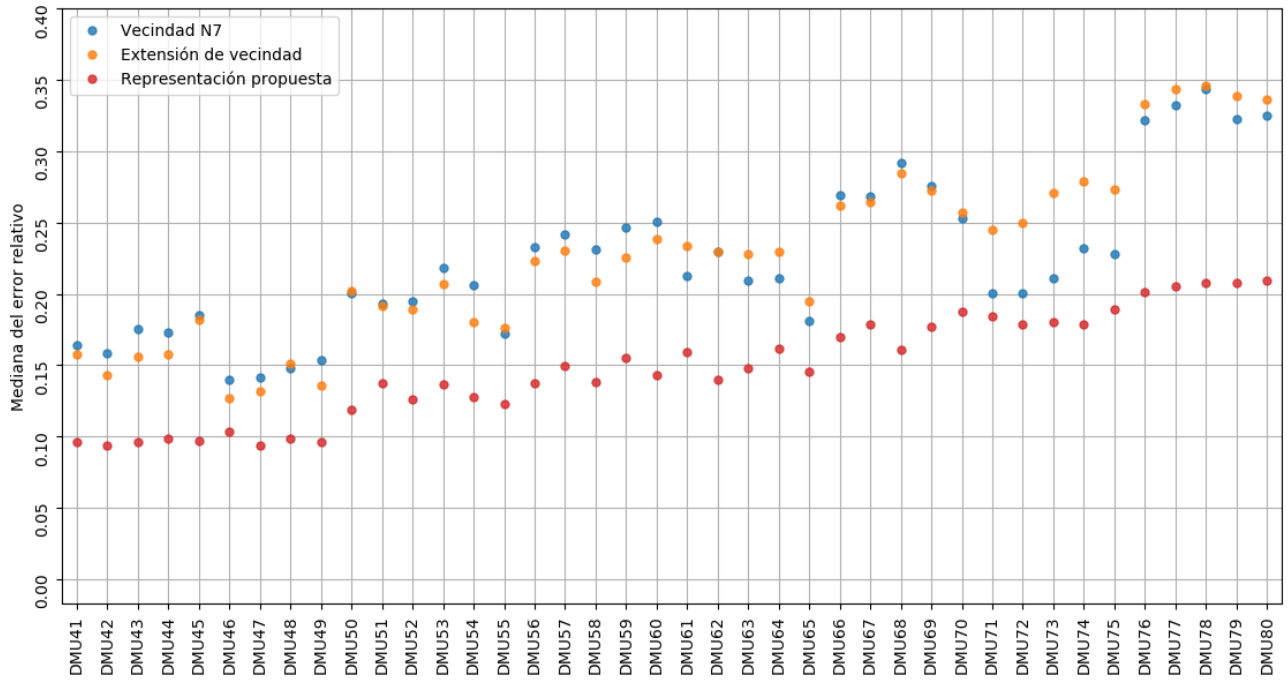


Figura 4.5: Resultados de la comparación entre los tres métodos.

También se muestra la mediana del error relativo alcanzado por los distintos métodos. Los resultados detallados para la extensión de vecindad se encuentran en el apéndice A.4.



(a) Resultados para las instancias **DMU01-40**



(b) Resultados para las instancias **DMU41-80**

Figura 4.6: Resultados para ambos métodos. Se marcan los casos en los que se llegó a la mejor solución conocida.

Para resaltar las diferencias entre los dos métodos se tomó la instancia en la que se obtuvieron los resultados más dispares, en este caso fue la **DMU78** y se registró para cada óptimo local visitado su makespan así como el tamaño de su vecindad.

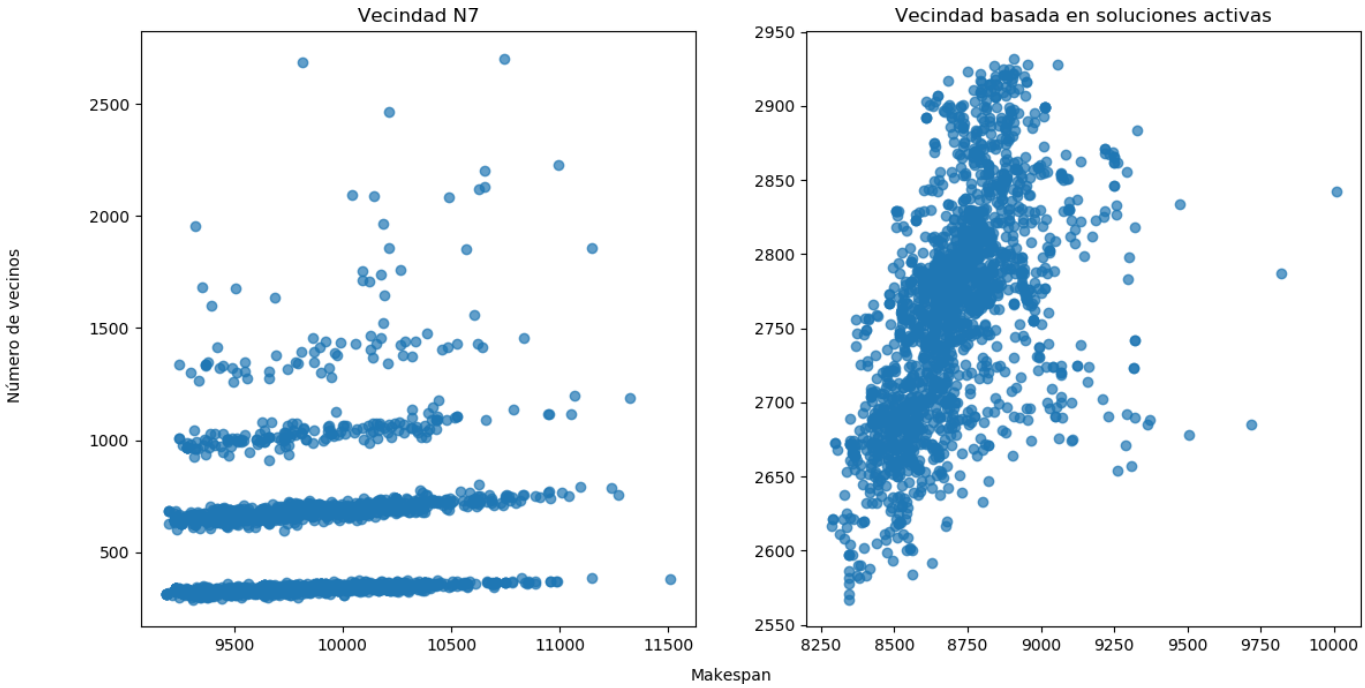


Figura 4.7: Comparación de tamaño de la vecindad contra makespan de los óptimos locales para la instancia **DMU78**

Capítulo 5

Conclusiones y Trabajos a Futuro

Sumario

5.1. Conclusiones

Los resultados obtenidos muestran una clara diferencia entre las instancias **DMU01-40** y las **DMU41-80** en cuestión de dificultad.

La búsqueda local iterada con la vecindad N7 y la función de fitness propuesta muestra un desempeño muy diferente en las dos mitades. En la primera mitad se obtienen resultados con bajo error relativo e inclusive en varias ocasiones se alcanzan los resultados del estado del arte mientras que en la segunda mitad el error relativo es bastante más alto y se crece con el tamaño de la instancia considerada. La extensión propuesta no mostró ventajas importantes frente a la vecindad N7 y muestra el mismo comportamiento a pesar de plantearse como una posible manera de escapar de óptimos locales.

La función de fitness planteada en general muestra que es benéfico considerar características de la planificación a demás del makespan para obtener mejores resultados. Aunque sí se tienen mejoras con la adición de estas características es posible que exista una mejor manera de construir la función de fitness para lograr una mejor exploración del pasaje de búsqueda.

El cambio de representación tuvo en gran efecto en la mejora del desempeño de la búsqueda local. Este cambio logra una disminución sustancial del error relativo en la segunda mitad de las instancias. Puede observarse en la figura 4.7 que en esta nueva estructura de vecindad el tamaño de la misma está mucho más relacionado con el makespan de la solución. También puede verse que el cambio de representación desde un inicio restringe a soluciones de mucha más calidad y que están más conectadas que las que se encuentran con la vecindad N7.

Apéndice A

Apéndice

Instancia	Tamaño	Estado del arte	Instancia	Tamaño	Estado del arte
DMU01	20×15	2563	DMU41	20×15	3248
DMU02	20×15	2706	DMU42	20×15	3390
DMU03	20×15	2731	DMU43	20×15	3441
DMU04	20×15	2669	DMU44	20×15	3475
DMU05	20×15	2749	DMU45	20×15	3266
DMU06	20×20	3244	DMU46	20×20	4035
DMU07	20×20	3046	DMU47	20×20	3942
DMU08	20×20	3188	DMU48	20×20	3763
DMU09	20×20	3092	DMU49	20×20	3710
DMU10	20×20	2984	DMU50	20×20	3729
DMU11	30×15	3430	DMU51	30×15	4156
DMU12	30×15	3492	DMU52	30×15	4303
DMU13	30×15	3681	DMU53	30×15	4378
DMU14	30×15	3394	DMU54	30×15	4361
DMU15	30×15	3343	DMU55	30×15	4263
DMU16	30×20	3751	DMU56	30×20	4941
DMU17	30×20	3814	DMU57	30×20	4653
DMU18	30×20	3844	DMU58	30×20	4701
DMU19	30×20	3764	DMU59	30×20	4616
DMU20	30×20	3703	DMU60	30×20	4721
DMU21	40×15	4380	DMU61	40×15	5171
DMU22	40×15	4725	DMU62	40×15	5248
DMU23	40×15	4668	DMU63	40×15	5313
DMU24	40×15	4648	DMU64	40×15	5226
DMU25	40×15	4164	DMU65	40×15	5184
DMU26	40×20	4647	DMU66	40×20	5701
DMU27	40×20	4848	DMU67	40×20	5779
DMU28	40×20	4692	DMU68	40×20	5763
DMU29	40×20	4691	DMU69	40×20	5688
DMU30	40×20	4732	DMU70	40×20	5868
DMU31	50×15	5640	DMU71	50×15	6207
DMU32	50×15	5927	DMU72	50×15	6463
DMU33	50×15	5728	DMU73	50×15	6136
DMU34	50×15	5385	DMU74	50×15	6196
DMU35	50×15	5635	DMU75	50×15	6189
DMU36	50×20	5621	DMU76	50×20	6718
DMU37	50×20	5851	DMU77	50×20	6747
DMU38	50×20	5713	DMU78	50×20	6755
DMU39	50×20	5747	DMU79	50×20	6910
DMU40	50×20	5577	DMU80	50×20	6634

Tabla A.1: Mejores resultados reportados en la literatura a la fecha.

A.1. Resultados paraN7 con makespan

Instancia	Tamaño	N7 con makespan		
		Mediana	Error relativo	Mejor
DMU01	20×15	2918	0.14	2793
DMU02	20×15	3084	0.14	2944
DMU03	20×15	3033	0.11	2922
DMU04	20×15	2961	0.11	2873
DMU05	20×15	3086	0.12	2912
DMU06	20×20	3618	0.12	3428
DMU07	20×20	3398	0.12	3309
DMU08	20×20	3488	0.09	3350
DMU09	20×20	3403	0.10	3270
DMU10	20×20	3232	0.08	3166
DMU11	30×15	3947	0.15	3784
DMU12	30×15	4127	0.18	3910
DMU13	30×15	4279	0.16	4049
DMU14	30×15	3720	0.10	3621
DMU15	30×15	3657	0.09	3516
DMU16	30×20	4141	0.10	4061
DMU17	30×20	4387	0.15	4148
DMU18	30×20	4319	0.12	4128
DMU19	30×20	4307	0.14	4145
DMU20	30×20	4143	0.12	3974
DMU21	40×15	4677	0.07	4565
DMU22	40×15	4915	0.04	4760
DMU23	40×15	4846	0.04	4743
DMU24	40×15	4798	0.03	4695
DMU25	40×15	4262	0.02	**4164**
DMU26	40×20	5270	0.13	5102
DMU27	40×20	5335	0.10	5213
DMU28	40×20	5198	0.11	5061
DMU29	40×20	5298	0.13	5054
DMU30	40×20	5258	0.11	5128
DMU31	50×15	5855	0.04	5703
DMU32	50×15	5927	0.00	**5927**
DMU33	50×15	5728	0.00	**5728**
DMU34	50×15	5609	0.04	5415
DMU35	50×15	5635	0.00	**5635**
DMU36	50×20	6183	0.10	6038
DMU37	50×20	6519	0.11	6260
DMU38	50×20	6519	0.14	6272
DMU39	50×20	6078	0.06	5910
DMU40	50×20	6078	0.09	5853

Instancia	Tamaño	N7 con makespan		
		Mediana	Error relativo	Mejor
DMU41	20×15	3964	0.22	3690
DMU42	20×15	4157	0.23	3886
DMU43	20×15	4191	0.22	3940
DMU44	20×15	4243	0.22	4015
DMU45	20×15	4062	0.24	3822
DMU46	20×20	4716	0.17	4539
DMU47	20×20	4605	0.17	4462
DMU48	20×20	4438	0.18	4251
DMU49	20×20	4414	0.19	4247
DMU50	20×20	4634	0.24	4472
DMU51	30×15	5409	0.30	5159
DMU52	30×15	5645	0.31	5327
DMU53	30×15	5735	0.31	5473
DMU54	30×15	5847	0.34	5386
DMU55	30×15	5517	0.29	5192
DMU56	30×20	6343	0.28	6079
DMU57	30×20	6037	0.30	5761
DMU58	30×20	6071	0.29	5782
DMU59	30×20	6084	0.32	5760
DMU60	30×20	6143	0.30	5979
DMU61	40×15	7065	0.37	6588
DMU62	40×15	7276	0.39	6702
DMU63	40×15	7256	0.37	6868
DMU64	40×15	7105	0.36	6711
DMU65	40×15	6928	0.34	6397
DMU66	40×20	7688	0.35	7355
DMU67	40×20	7827	0.35	7508
DMU68	40×20	7829	0.36	7534
DMU69	40×20	7769	0.37	7483
DMU70	40×20	7928	0.35	7629
DMU71	50×15	8438	0.36	7980
DMU72	50×15	8769	0.36	8226
DMU73	50×15	8601	0.40	8019
DMU74	50×15	8620	0.39	8177
DMU75	50×15	8596	0.39	8109
DMU76	50×20	9519	0.42	9198
DMU77	50×20	9584	0.42	9233
DMU78	50×20	9832	0.46	9431
DMU79	50×20	9768	0.41	9401
DMU80	50×20	9447	0.42	9193

A.2. Resultados paraN7 con tupla

Instancia	Tamaño	N7 con tupla		
		Mediana	Error relativo	Mejor
DMU01	20×15	2861	0.12	2751
DMU02	20×15	3005	0.11	2875
DMU03	20×15	2957	0.08	2878
DMU04	20×15	2907	0.09	2777
DMU05	20×15	3018	0.10	2913
DMU06	20×20	3551	0.09	3438
DMU07	20×20	3363	0.10	3257
DMU08	20×20	3407	0.07	3312
DMU09	20×20	3347	0.08	3224
DMU10	20×20	3204	0.07	3121
DMU11	30×15	3810	0.11	3680
DMU12	30×15	3889	0.11	3740
DMU13	30×15	4072	0.11	3944
DMU14	30×15	3627	0.07	3482
DMU15	30×15	3562	0.07	3449
DMU16	30×20	4061	0.08	3955
DMU17	30×20	4263	0.12	4135
DMU18	30×20	4170	0.08	4041
DMU19	30×20	4194	0.11	4062
DMU20	30×20	4029	0.09	3929
DMU21	40×15	4517	0.03	4391
DMU22	40×15	4802	0.02	4738
DMU23	40×15	4743	0.02	4684
DMU24	40×15	4669	0.00	**4648**
DMU25	40×15	4164	0.00	**4164**
DMU26	40×20	5132	0.10	4988
DMU27	40×20	5164	0.07	5011
DMU28	40×20	5014	0.07	4922
DMU29	40×20	5107	0.09	4893
DMU30	40×20	5049	0.07	4919
DMU31	50×15	5645	0.00	**5640**
DMU32	50×15	5927	0.00	**5927**
DMU33	50×15	5728	0.00	**5728**
DMU34	50×15	5385	0.00	**5385**
DMU35	50×15	5635	0.00	**5635**
DMU36	50×20	5979	0.06	5868
DMU37	50×20	6172	0.05	6036
DMU38	50×20	6187	0.08	5989
DMU39	50×20	5867	0.02	5790
DMU40	50×20	5833	0.05	5704

Instancia	Tamaño	N7 con tupla		
		Mediana	Error relativo	Mejor
DMU41	20×15	3780	0.16	3656
DMU42	20×15	3928	0.16	3734
DMU43	20×15	4043	0.18	3730
DMU44	20×15	4075	0.17	3857
DMU45	20×15	3871	0.19	3616
DMU46	20×20	4601	0.14	4374
DMU47	20×20	4499	0.14	4333
DMU48	20×20	4320	0.15	4132
DMU49	20×20	4280	0.15	4112
DMU50	20×20	4476	0.20	4214
DMU51	30×15	4958	0.19	4766
DMU52	30×15	5140	0.19	4888
DMU53	30×15	5334	0.22	5081
DMU54	30×15	5261	0.21	4826
DMU55	30×15	4996	0.17	4764
DMU56	30×20	6090	0.23	5764
DMU57	30×20	5776	0.24	5527
DMU58	30×20	5789	0.23	5433
DMU59	30×20	5756	0.25	5420
DMU60	30×20	5906	0.25	5597
DMU61	40×15	6269	0.21	5952
DMU62	40×15	6453	0.23	6033
DMU63	40×15	6426	0.21	6039
DMU64	40×15	6328	0.21	6006
DMU65	40×15	6123	0.18	5760
DMU66	40×20	7234	0.27	6925
DMU67	40×20	7331	0.27	7069
DMU68	40×20	7447	0.29	7088
DMU69	40×20	7254	0.28	6995
DMU70	40×20	7351	0.25	6849
DMU71	50×15	7452	0.20	7236
DMU72	50×15	7761	0.20	7519
DMU73	50×15	7429	0.21	7100
DMU74	50×15	7632	0.23	7329
DMU75	50×15	7600	0.23	7221
DMU76	50×20	8883	0.32	8449
DMU77	50×20	8991	0.33	8559
DMU78	50×20	9078	0.34	8575
DMU79	50×20	9137	0.32	8625
DMU80	50×20	8793	0.33	8440

A.3. Resultados para Extensión de vecindad con tupla

Instancia	Tamaño	Extensión de vecindad con tupla		
		Mediana	Error relativo	Mejor
DMU01	20×15	2828	0.10	2728
DMU02	20×15	2986	0.10	2847
DMU03	20×15	2953	0.08	2854
DMU04	20×15	2889	0.08	2814
DMU05	20×15	3029	0.10	2902
DMU06	20×20	3535	0.09	3415
DMU07	20×20	3331	0.09	3227
DMU08	20×20	3399	0.07	3289
DMU09	20×20	3360	0.09	3208
DMU10	20×20	3181	0.07	3080
DMU11	30×15	3765	0.10	3635
DMU12	30×15	3830	0.10	3650
DMU13	30×15	4028	0.09	3901
DMU14	30×15	3604	0.06	3495
DMU15	30×15	3545	0.06	3443
DMU16	30×20	4041	0.08	3910
DMU17	30×20	4216	0.11	4074
DMU18	30×20	4167	0.08	4066
DMU19	30×20	4116	0.09	3985
DMU20	30×20	4008	0.08	3911
DMU21	40×15	4479	0.02	4396
DMU22	40×15	4759	0.01	4738
DMU23	40×15	4743	0.02	4684
DMU24	40×15	4669	0.00	**4648**
DMU25	40×15	4164	0.00	**4164**
DMU26	40×20	5114	0.10	5004
DMU27	40×20	5160	0.06	4993
DMU28	40×20	5012	0.07	4901
DMU29	40×20	5081	0.08	4976
DMU30	40×20	5028	0.06	4921
DMU31	50×15	5671	0.01	**5640**
DMU32	50×15	5927	0.00	**5927**
DMU33	50×15	5728	0.00	**5728**
DMU34	50×15	5385	0.00	**5385**
DMU35	50×15	5635	0.00	**5635**
DMU36	50×20	6013	0.07	5872
DMU37	50×20	6233	0.07	6071
DMU38	50×20	6282	0.10	6152
DMU39	50×20	5920	0.03	5856
DMU40	50×20	5928	0.06	5758

Instancia	Tamaño	Extensión de vecindad con tupla		
		Mediana	Error relativo	Mejor
DMU41	20×15	3759	0.16	3598
DMU42	20×15	3875	0.14	3748
DMU43	20×15	3977	0.16	3792
DMU44	20×15	4021	0.16	3811
DMU45	20×15	3860	0.18	3606
DMU46	20×20	4548	0.13	4469
DMU47	20×20	4461	0.13	4310
DMU48	20×20	4330	0.15	4192
DMU49	20×20	4214	0.14	4015
DMU50	20×20	4482	0.20	4279
DMU51	30×15	4953	0.19	4746
DMU52	30×15	5116	0.19	4864
DMU53	30×15	5285	0.21	4881
DMU54	30×15	5148	0.18	4797
DMU55	30×15	5014	0.18	4704
DMU56	30×20	6045	0.22	5845
DMU57	30×20	5726	0.23	5452
DMU58	30×20	5683	0.21	5394
DMU59	30×20	5658	0.23	5283
DMU60	30×20	5845	0.24	5552
DMU61	40×15	6381	0.23	6018
DMU62	40×15	6454	0.23	6150
DMU63	40×15	6526	0.23	6218
DMU64	40×15	6428	0.23	6212
DMU65	40×15	6196	0.20	5930
DMU66	40×20	7194	0.26	6934
DMU67	40×20	7307	0.26	6997
DMU68	40×20	7405	0.29	7232
DMU69	40×20	7236	0.27	7004
DMU70	40×20	7378	0.26	7137
DMU71	50×15	7727	0.24	7474
DMU72	50×15	8075	0.25	7691
DMU73	50×15	7798	0.27	7448
DMU74	50×15	7923	0.28	7555
DMU75	50×15	7879	0.27	7620
DMU76	50×20	8957	0.33	8641
DMU77	50×20	9067	0.34	8735
DMU78	50×20	9091	0.35	8795
DMU79	50×20	9254	0.34	8940
DMU80	50×20	8867	0.34	8574

A.4. Resultados para Nueva vecindad con tupla

Instancia	Tamaño	Nueva vecindad con tupla		
		Mediana	Error relativo	Mejor
DMU01	20×15	2762	0.08	2631
DMU02	20×15	2922	0.08	2806
DMU03	20×15	2917	0.07	2794
DMU04	20×15	2831	0.06	2724
DMU05	20×15	2921	0.06	2847
DMU06	20×20	3514	0.08	3383
DMU07	20×20	3259	0.07	3167
DMU08	20×20	3418	0.07	3306
DMU09	20×20	3343	0.08	3209
DMU10	20×20	3176	0.06	3063
DMU11	30×15	3747	0.09	3601
DMU12	30×15	3790	0.09	3672
DMU13	30×15	3956	0.07	3832
DMU14	30×15	3576	0.05	3482
DMU15	30×15	3505	0.05	3429
DMU16	30×20	4090	0.09	3945
DMU17	30×20	4205	0.10	4112
DMU18	30×20	4207	0.09	4036
DMU19	30×20	4136	0.10	4013
DMU20	30×20	4052	0.09	3925
DMU21	40×15	4511	0.03	4388
DMU22	40×15	4822	0.02	4743
DMU23	40×15	4763	0.02	4684
DMU24	40×15	4694	0.01	4653
DMU25	40×15	4192	0.01	**4164**
DMU26	40×20	5094	0.10	4939
DMU27	40×20	5231	0.08	5088
DMU28	40×20	5109	0.09	4963
DMU29	40×20	5181	0.10	4955
DMU30	40×20	5078	0.07	4931
DMU31	50×15	5799	0.03	5694
DMU32	50×15	5927	0.00	**5927**
DMU33	50×15	5728	0.00	**5728**
DMU34	50×15	5435	0.01	**5385**
DMU35	50×15	5647	0.00	**5635**
DMU36	50×20	6148	0.09	5954
DMU37	50×20	6341	0.08	6132
DMU38	50×20	6215	0.09	6060
DMU39	50×20	6116	0.06	5962
DMU40	50×20	6061	0.09	5886

Instancia	Tamaño	Nueva vecindad con tupla		
		Mediana	Error relativo	Mejor
DMU41	20×15	3560	0.10	3422
DMU42	20×15	3707	0.09	3599
DMU43	20×15	3772	0.10	3654
DMU44	20×15	3818	0.10	3687
DMU45	20×15	3581	0.10	3384
DMU46	20×20	4452	0.10	4312
DMU47	20×20	4312	0.09	4196
DMU48	20×20	4134	0.10	3938
DMU49	20×20	4066	0.10	3912
DMU50	20×20	4171	0.12	4034
DMU51	30×15	4728	0.14	4538
DMU52	30×15	4846	0.13	4692
DMU53	30×15	4975	0.14	4796
DMU54	30×15	4919	0.13	4796
DMU55	30×15	4785	0.12	4650
DMU56	30×20	5622	0.14	5462
DMU57	30×20	5349	0.15	5223
DMU58	30×20	5352	0.14	5170
DMU59	30×20	5333	0.16	5140
DMU60	30×20	5397	0.14	5249
DMU61	40×15	5994	0.16	5844
DMU62	40×15	5982	0.14	5755
DMU63	40×15	6099	0.15	5955
DMU64	40×15	6072	0.16	5888
DMU65	40×15	5940	0.15	5712
DMU66	40×20	6669	0.17	6484
DMU67	40×20	6810	0.18	6592
DMU68	40×20	6692	0.16	6459
DMU69	40×20	6697	0.18	6524
DMU70	40×20	6968	0.19	6721
DMU71	50×15	7349	0.18	7064
DMU72	50×15	7618	0.18	7374
DMU73	50×15	7240	0.18	7057
DMU74	50×15	7303	0.18	7108
DMU75	50×15	7362	0.19	7174
DMU76	50×20	8073	0.20	7844
DMU77	50×20	8131	0.21	7893
DMU78	50×20	8161	0.21	7827
DMU79	50×20	8345	0.21	8139
DMU80	50×20	8021	0.21	7787

Bibliografía

[1] Balas, E. Machine sequencing via disjunctive graphs: an implicit enumeration algorithm. *Operations research 17*, 6 (1969), 941–957.

[2] Balas, E., and Vazacopoulos, A. Guided local search with shifting bottleneck for job shop scheduling. *Management Science 44*, 2 (1998), 262–275.

[3] Bean, J. C. Genetic algorithms and random keys for sequencing and optimization. *ORSA journal on computing 6*, 2 (1994), 154–160.

[4] Beck, J. C., and Jackson, W. K. Constrainedness and the phase transition in job shop scheduling. *Technical Report, School of Computing Science* (1997).

[5] Bierwirth, C., Mattfeld, D. C., and Watson, J.-P. Landscape regularity and random walks for the job-shop scheduling problem. In *European Conference on Evolutionary Computation in Combinatorial Optimization* (2004), Springer, pp. 21–30.

[6] Błażewicz, J., Domschke, W., and Pesch, E. The job shop scheduling problem: Conventional and new solution techniques. *European journal of operational research 93*, 1 (1996), 1–33.

[7] Blum, C., and Roli, A. Metaheuristics in Combinatorial Optimization: Overview and Conceptual Comparison. *ACM Computing Surveys 35*, 3 (2003), 268–308.

[8] Brucker, P. *Due-Date Scheduling*. 2001.

[9] Brucker, P., Jurisch, B., and Sievers, B. A branch and bound algorithm for the job-shop scheduling problem. *Discrete Applied Mathematics 49*, 1-3 (1994), 107–127.

[10] Cheng, R., Gen, M., and Tsujimura, Y. A tutorial survey of job-shop scheduling problems using genetic algorithms - I. Representation. *Computers and Industrial Engineering 30*, 4 (1996), 983–997.

[11] Cheng, R., Gen, M., and Tsujimura, Y. Tutorial survey of job-shop scheduling problems using genetic algorithms: Part II. Hybrid genetic search strategies. *Computers and Industrial Engineering 37*, 1 (1999), 51–55.

[12] Cormen, T. H., Leiserson, C. E., Rivest, R. L., and Stein, C. *Introduction to algorithms*. MIT press, 2009.

[13] Dell'Amico, M., and Trubian, M. Applying tabu search to the job-shop scheduling problem. *Annals of Operations research 41*, 3 (1993), 231–252.

[14] Demirkol, E., Mehta, S., and Uzsoy, R. A computational study of shifting bottleneck procedures for shop scheduling problems. *Journal of Heuristics 3*, 2 (1997), 111–137.

[15] Eugeniusz Nowicki, C. S., and To. A Fast Taboo Search Algorithm for the Job Shop Problem. *Manage. Sci. 42*, 6 (2003), 797–813.

[16] Garey, M. R., Johnson, D. S., and Sethi, R. The complexity of flowshop and jobshop scheduling. *Mathematics of operations research 1*, 2 (1976), 117–129.

[17] Giffler, B., and Thompson, G. L. Algorithms for Solving Production-Scheduling Problems, 1960.

[18] Gong, G., Deng, Q., Chiong, R., Gong, X., and Huang, H. An effective memetic algorithm for multi-objective job-shop scheduling. *Knowledge-Based Systems 182* (2019), 104840.

[19] Graham, R. L., Knuth, D. E., Patashnik, O., and Liu, S. Concrete mathematics: a foundation for computer science. *Computers in Physics 3*, 5 (1989), 106–107.

[20] Jain, A. S., Meeran, S., et al. A state-of-the-art review of job-shop scheduling techniques. Tech. rep., Technical report, Department of Applied Physics, Electronic and Mechanical ..., 1998.

[21] Johnson, S. M. Optimal two-and three-stage production schedules with setup times included. *Naval research logistics quarterly 1*, 1 (1954), 61–68.

[22] Kauffman, S. A., et al. *The origins of order: Self-organization and selection in evolution*. Oxford University Press, USA, 1993.

[23] Kılıç, S., and Kahraman, C. Metaheuristic Techniques for Job Shop Scheduling Problem and a Fuzzy Ant Colony Optimization Algorithm. *Fuzzy Applications in Industrial Engineering 425*, 2006 (2007), 401–425.

[24] Mammen, D. L., and Hogg, T. A new look at the easy-hard-easy pattern of combinatorial search difficulty. *Journal of Artificial Intelligence Research 7* (1997), 47–66.

[25] Mattfeld, D. C., Bierwirth, C., and Kopfer, H. A search space analysis of the job shop scheduling problem. *Annals of Operations Research 86* (1999), 441–453.

[26] Nocedal, J., and Wright, S. *Numerical optimization*. Springer Science & Business Media, 2006.

[27] Norman, B. A., and Bean, J. C. A random keys genetic algorithm for job shop scheduling. Tech. rep., 1996.

[28] Ponnambalam, S., Ramkumar, V., and Jawahar, N. A multiobjective genetic algorithm for job shop scheduling. *Production planning & control 12*, 8 (2001), 764–774.

[29] Ponsich, A., and Coello Coello, C. A. A hybrid Differential Evolution - Tabu Search algorithm for the solution of Job-Shop Scheduling Problems. *Applied Soft Computing Journal 13*, 1 (2013), 462–474.

[30] Rand, G. K. *Machine scheduling problems: Classification, complexity and computations*, vol. 1. 1977.

[31] Rothlauf, F. Representations for genetic and evolutionary algorithms. In *Representations for Genetic and Evolutionary Algorithms*. Springer, 2002, pp. 9–30.

[32] Sakawa, M., and Kubota, R. Fuzzy programming for multiobjective job shop scheduling with fuzzy processing time and fuzzy duedate through genetic algorithms. *European Journal of operational research 120*, 2 (2000), 393–407.

[33] Sipser, M. Introduction to the theory of computation. *ACM Sigact News 27*, 1 (1996), 27–29.

[34] Stegherr, H., Heider, M., and Hähner, J. Classifying Metaheuristics: Towards a unified multi-level classification system. *Natural Computing 0* (2020).

[35] Streeter, M. J., and Smith, S. F. How the landscape of random job shop scheduling instances depends on the ratio of jobs to machines. *Journal of Artificial Intelligence Research 26* (2006), 247–287.

[36] Uckun, S., Bagchi, S., Kawamura, K., and Miyabe, Y. Managing genetic search in job shop scheduling. *IEEE expert 8*, 5 (1993), 15–24.

[37] Watson, J.-P., Beck, J. C., Howe, A. E., and Whitley, L. D. Problem difficulty for tabu search in job-shop scheduling. *Artificial intelligence 143*, 2 (2003), 189–217.

[38] Wigderson, A. P, np and mathematics–a computational complexity perspective. In *Proceedings of the ICM* (2006), vol. 1, pp. 665–712.

[39] Wright, S., et al. The roles of mutation, inbreeding, crossbreeding, and selection in evolution.

[40] Zhang, C. Y., Li, P. G., Guan, Z. L., and Rao, Y. Q. A tabu search algorithm with a new neighborhood structure for the job shop scheduling problem. *Computers and Operations Research 34*, 11 (2007), 3229–3242.

[41] Zhang, J., Ding, G., Zou, Y., Qin, S., and Fu, J. Review of job shop scheduling research and its new perspectives under Industry 4.0. *Journal of Intelligent Manufacturing 30*, 4 (2019), 1809–1830.

[42] Zhang, Z., Guan, Z., Zhang, J., and Xie, X. A novel job-shop scheduling strategy based on particle swarm optimization and neural network. *Int. J. Simul. Model 18*, 4 (2019), 699–707.