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[7] 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 consiste a su vez de 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.
Desde los década de los 50 se comenzaron a formular algoritmos de planificación[13] y el interés

en ellos ha crecido en las décadas siguientes hasta la fecha.

En este trabajo se trata un tipo especial 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 items en m máquinas que tome el mínimo tiempo posible cumpliendo ciertas restricciones de orden para el procesamiento de los n items. Este es un problema de optimización combinatoria en la que se requiere encontrar una secuencia de procesamiento para las operaciones en cada una de las máquinas disponibles y es NP duro cuando el número de máquinas es mayor a dos.

Anteriormente se han propuesto métodos exactos para resolver este problema [7] 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.

Dado el interés que se tiene en este problema y en que su tamaño puede ser suficientemente grande para no poder utlizar los métodos exactos, se han propuesto otros métodos aproximados para encontrar soluciones buenas en tiempos aceptables. Estos métodos pueden agruparse de la siguiente manera[22]:

- **Métodos Constructivos** Estos métodos construyen una planificación mediante el uso de una 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 de 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. 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.

- **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 o bien en procesos naturales como el 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 que mediante el uso de memoria logra escapar de óptimos locales.

Aunque se han obtenido buenos resultados con estos algoritmos se requieren de 24 a 48 horas paralelas de ejecución. 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 [21].

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 distinta 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 (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 limitar el 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 mejores que los métodos simples que se tienen actualmente.

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 cambiar la estructura de vecindad y el conjunto de soluciones representables. 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 se considera solo el caso de minimización sin pérdida de generalidad.

A la definición también pueden agregarse un conjunto de restricciones 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. Estas restricciones se agregan 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.

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.

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 más el espacio de búsqueda, es decir que puede eliminar sesgos. Esta rama de la optimización se conoce como optimización estocástica.

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[15] 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.

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[5]:

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

En muchos problemas de optimización combinatoria el conjunto de soluciones no tiene alguna estructura adicional que ayude a buscar el mínimo de la función objetivo; es raro que exista un ordenamiento de las soluciones o una medida de distancia entre ellas que brinde propiedades a la función objetivo tales como continuidad o suavidad las cuales facilitarían la búsqueda del mínimo. Dicho de otro modo, no tenemos a priori una forma eficiente de explorar el espacio de búsqueda. Ante estas limitaciones surgieron técnicas conocidas como metaheurísticas que buscan facilitar la resolución de estos problemas.

2.2. 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 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 [19] 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.
- 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 existen tres conceptos que determinan el llamado paisaje de búsqueda: las soluciones en sí, la forma en que las soluciones están conectadas y cómo podemos compararlas entre sí. A continuación se describe cada uno de ellos.

2.2.1. 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[8] 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.

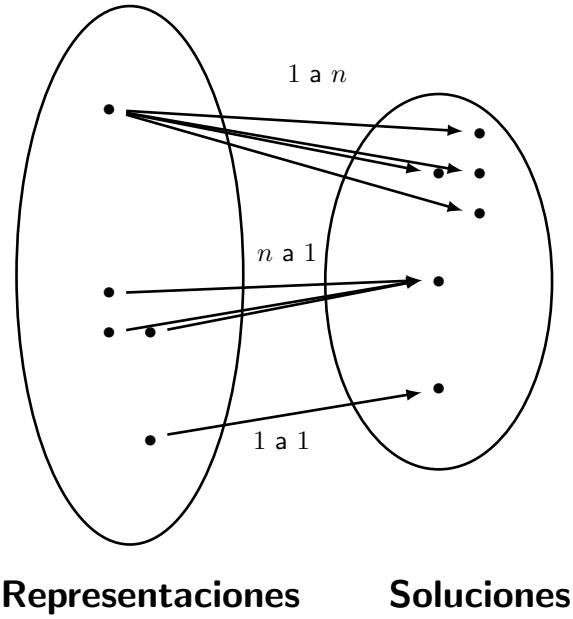


Figura 2.1: Tipos de representaciones

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.

2.2.2. 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.

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

2.2.3. Función de aptitud o fitness

Aunque para un problema de optimización ya se tiene definida una función objetivo que se quiere minimizar, no siempre tenderemos el mejor desempeño de las metaheurísticas con solo esta función por lo que resulta benéfico plantear una nueva función a minimizar con la que tengamos mejor desempeño. 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.

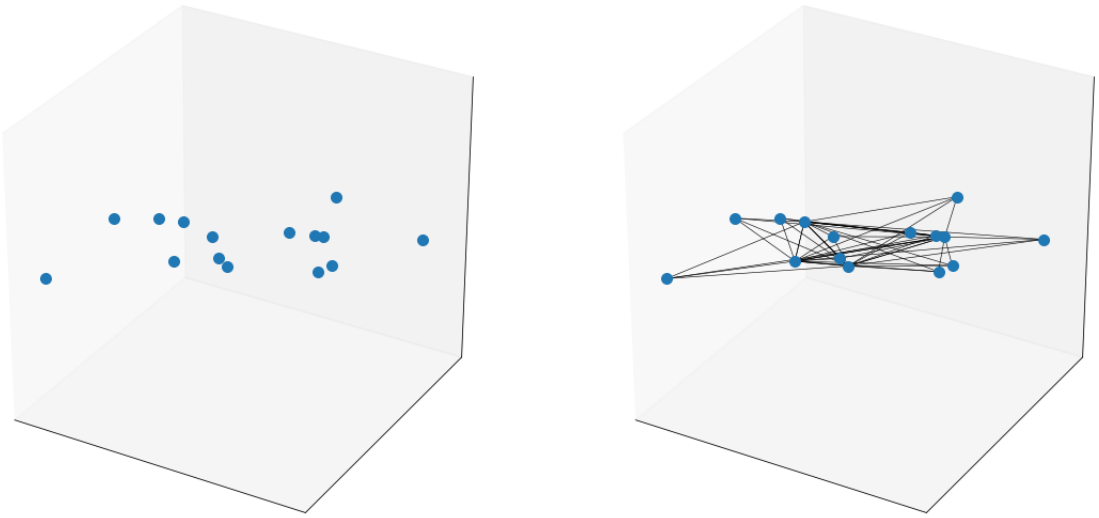
Esta función debe asociar a cada solución un elemento de un conjunto donde esté definido un ordenamiento total. En esencia esta función define un operador de comparación entre soluciones de modo que podemos elegir la mejor de dos soluciones.

2.2.4. Paisaje de búsqueda

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.

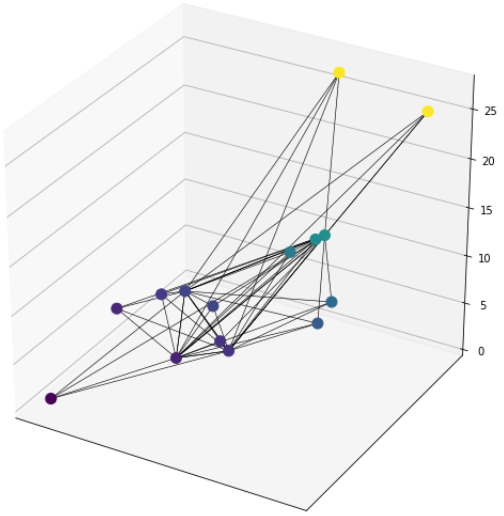
Podemos asociar a cada solución en el espacio un valor de aptitud o fitness que mide la calidad de dicha solución. La adición de esta función de aptitud al espacio de búsqueda genera al paisaje de búsqueda. Formalmente el paisaje de búsqueda $\mathcal{L}$ es entonces una tupla conformada por el espacio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

A continuación se muestra una representación pictórica de cómo se crea el paisaje de búsqueda para un problema de optimización.



(a) Soluciones representables

(b) Relaciones inducidas por los operadores de cambio



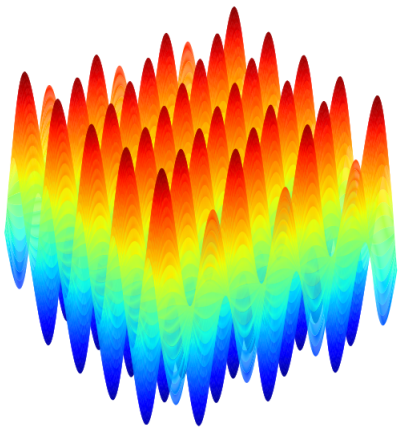
(c) Adición de la función de fitness

Figura 2.2: Creación del paisaje de búsqueda

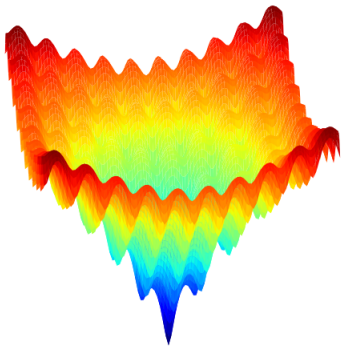
El paisaje de búsqueda es el «terreno» a explorar y puede cambiar si cambiamos cualquiera de sus componentes, podría ser que alguna representación nos centre en un subconjunto de soluciones convenientes o que alguna estructura de vecindad se proponga de modo que las mejores soluciones nunca están muy 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

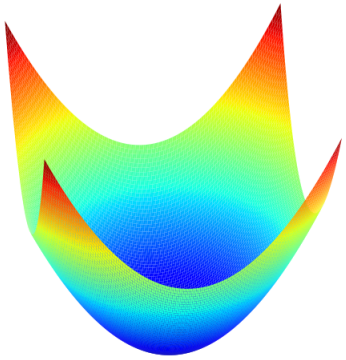
ciertas metaheurísticas. La «forma» del paisaje hace referencia a cómo cambia el valor de fitness para soluciones conectadas entre sí. Dos medidas que son generalmente utilizadas para esto miden cómo cambia el valor de la función de fitness conforme nos acercamos a un óptimo local y la otra mide qué tanto cambia el fitness entre soluciones vecinas[1]. 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) Paisaje rugoso con muchos óptimos locales



(b) Paisaje rugoso con con un gran valle



(c) Paisaje suave con un gran valle

Figura 2.3: Diferentes tipos de paisajes de búsqueda. Se muestran paisajes continuos con fines ilustrativos

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
```

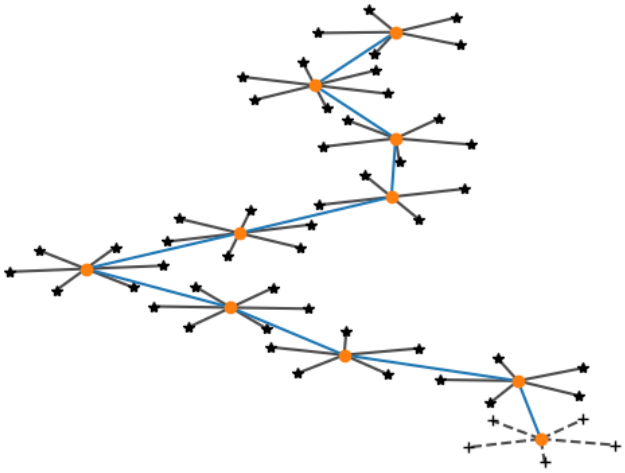


Figura 2.4: Ilustración de una escalada estocástica. Las aristas rayadas representan la vecindad del óptimo local.

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* ← 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 ∩ TL = ∅;
    Escoger un vecino y ∈ L;
    x ← y;
    Actualizar TL;
    if x < x* then
        x* ← 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* ← x while no criterio de paro do
    Obtener una solución y a partir de x mediante búsqueda local 1;
    if y < x* then
        x ← y;
        x* ← y;
    else
        x ← x*;
        Perturbar x;
return x*
```

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

Una instancia del JSP consiste en n trabajos diferentes constituidos cada uno por m operaciones que deben procesarse por un tiempo determinado en m máquinas en una secuencia predeterminada. El objetivo es hallar la planificación que minimiza el tiempo que toma terminar todos los trabajos dado que cada máquina puede procesar solo un trabajo a la vez.

Una planificación consiste en asignar tiempos de inicio y fin a cada operación respetando el orden requerido para cada trabajo. El tiempo que toma terminar todos los trabajos se conoce como makespan y la secuencia de trabajos que toma el mayor tiempo en completarse se conoce como ruta crítica. La ruta crítica puede verse como 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. Una planificación puede tener una o varias rutas críticas.
Sin pérdida de generalidad podemos restringirnos al caso en el que el tiempo requerido para procesar cada operación es un entero positivo.

Ejemplo

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

Tabla 2.1: Instancia simple con 3 maquinas y 2 trabajos

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

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

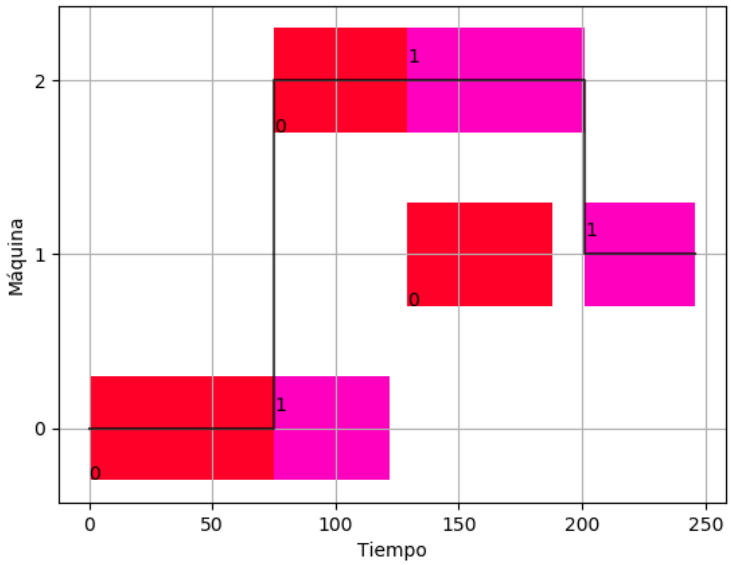


Figura 2.5: Diagrama de gantt de una planificación posible

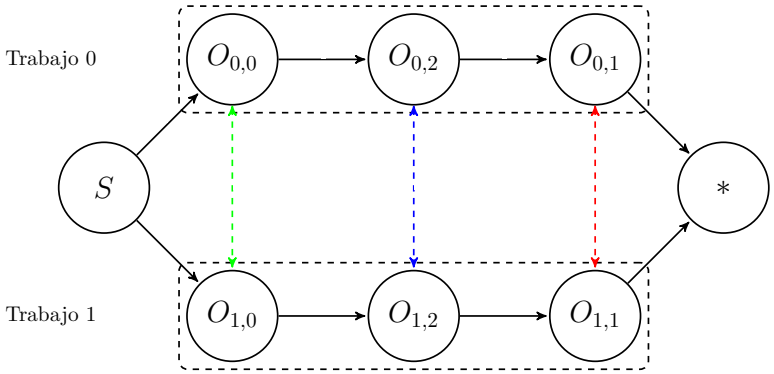
2.4. Representación de planificaciones

Existen varias formas de representar las planificaciones, en este trabajo se utilizaron dos: el grafo disyuntivo y las reglas de prioridad.

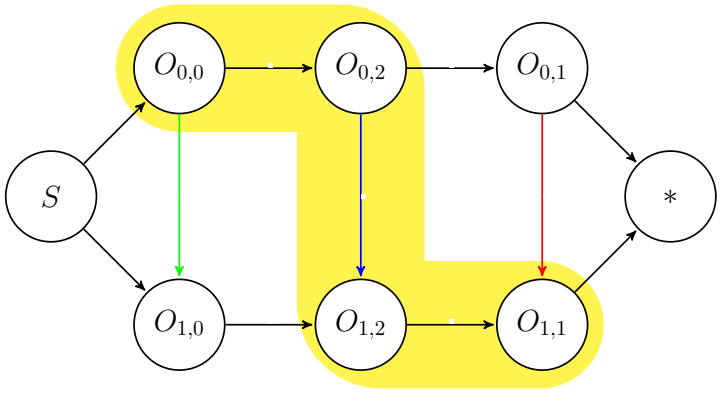
Modelo de grafo disyuntivo

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.



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



(b) Planificación obtenida al fijar las aristas disyuntivas como en 2.5. La ruta critica se resalta en amarillo

Figura 2.6: Modelo de grafo disyuntivo para la instancia de ejemplo 2.1

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. 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 distinguir 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[17].

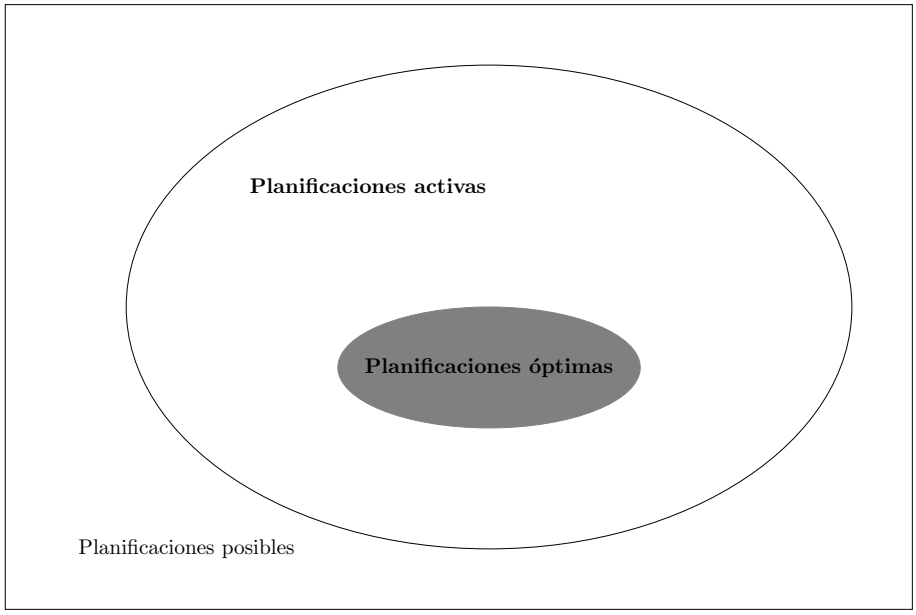


Figura 2.7: Subconjuntos de planificaciones

Estas clasificaciones resultan interesantes porque existen algoritmos para generar tipos de pla-

nificaciones 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. Aunque se han propuesto muchas metaheurísticas, prácticamente todas utilizan el concepto de vecindad. Como ya se explicó, la definición de una estructura de vecindad 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.

Vecindades previamente propuestas

Se han propuesto varias estructuras de vecindad al JSP, a continuación se describen las más importantes a la fecha:

- N1 [4] Consiste en considerar todas las soluciones que se crean al intercambiar cualquier par de operaciones adyacentes que pertenecen a un bloque crítico. Esta vecindad es muy grande y considera muchos cambios que no mejoran el makespan.

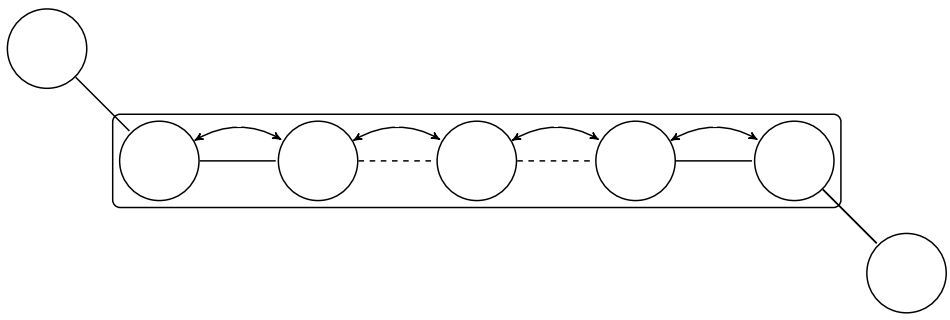


Figura 2.8: Movimientos de la vecindad N1

- N4 [9] 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.

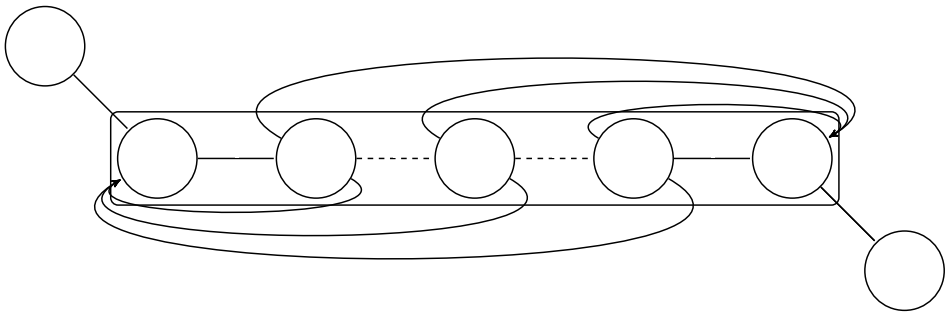


Figura 2.9: Movimientos de la vecindad N4

- N5 [11] Consiste en intercambiar solo las operaciones adyacentes a la final o inicial de un bloque crítico.

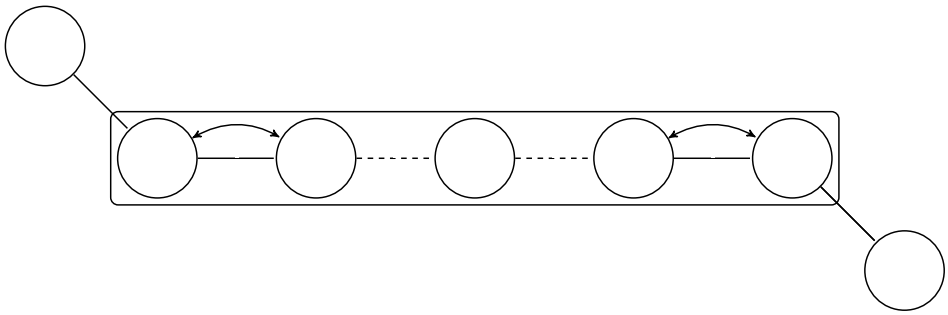


Figura 2.10: Movimientos de la vecindad N5

- 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).

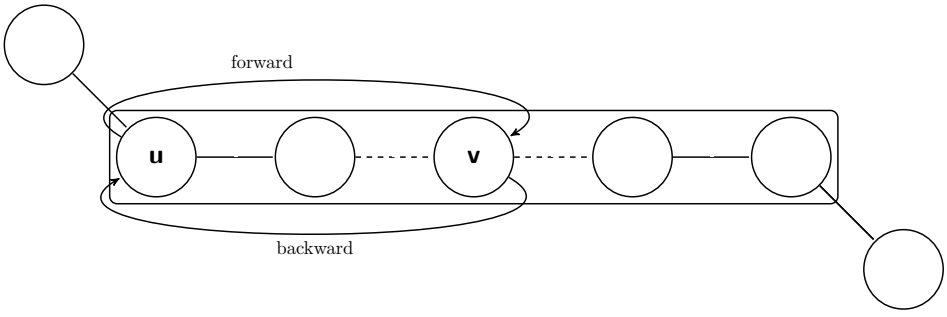


Figura 2.11: Los dos tipos de movimientos para un par (u, v)

- N7 [21] 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.

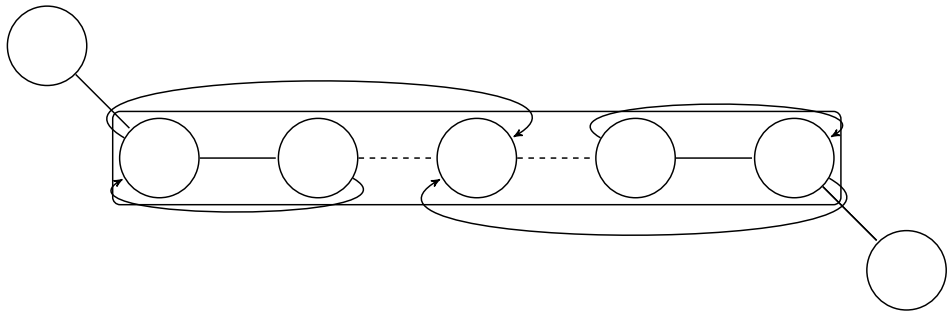


Figura 2.12: Movimientos de la vecindad N7

Las vecindades antes presentadas se han centrado en operaciones que pertenecen a la ruta crítica por varias razones, la vecindad resultante es suficientemente pequeña como para ser explorada en su totalidad, el makespan de una planificación solo puede reducirse haciendo cambios en la ruta crítica y pueden garantizarse que cualquier vecino representa una solución factible.

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 del trabajo J_i y $f_i(C_i)$ a su costo asociado, podemos distinguir dos tipos de funciones de costo en la literatura[6]:

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

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 finali-

zación 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[18].

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

En el caso del JSP se ha encontrado que el paisaje de búsqueda para una instancia al azar tiende a ser muy dependiente de la razón entre el número de máquinas y número de trabajos de la misma [20]. También se ha observado que el paisaje tiende a tener muchos óptimos locales de baja calidad para instancias difíciles [14] y que en general algunas soluciones están mucho mas conectadas que otras [3].

Estas características explican en parte por qué los métodos basados en búsqueda local como la búsqueda tabú combinados con métodos sofisticados de exploración han tenido tan buenos resultados. 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.

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 muy buena.

Capítulo 3

Propuestas

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

3.1. 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 [16] y en [17].

Para decodificar la solución a partir de las llaves se utiliza el algoritmo de Giffler & Thompson [12] 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.

3.1.1. 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 2.5 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.

3.1.2. 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.

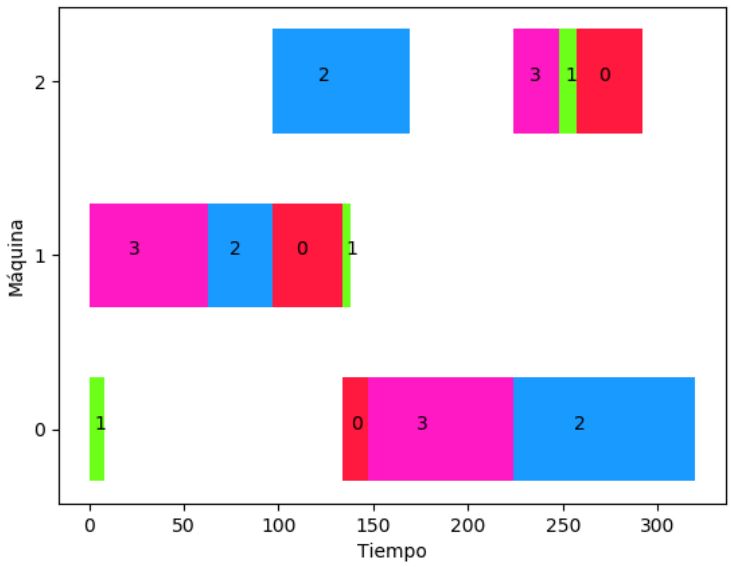


Figura 3.1: Planificación posible para la instancia en 3.1.

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.

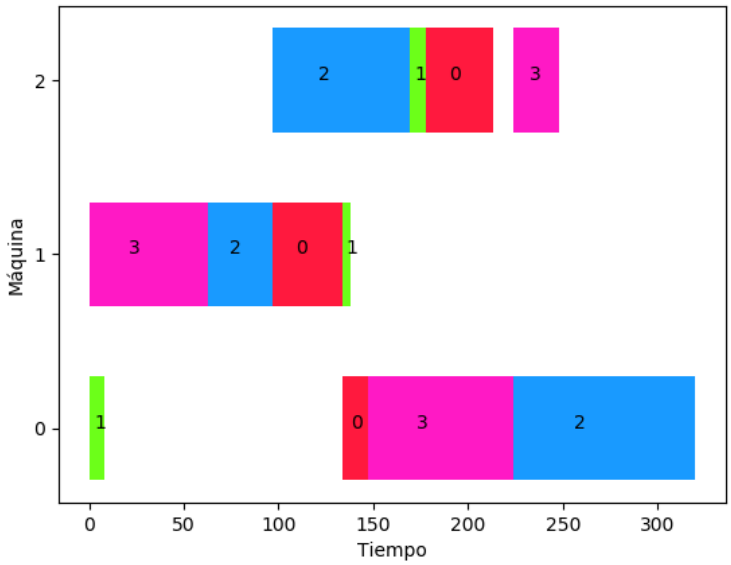


Figura 3.2: Planificación activa reconstruida a partir de la mostrada en 3.1.2

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.

3.2. 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.

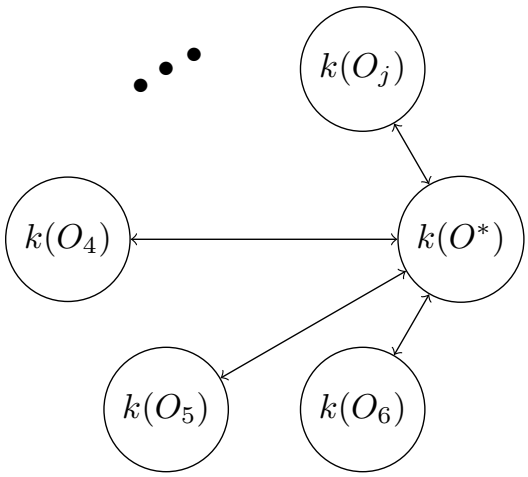


Figura 3.3: Movimientos de la vecindad propuesta. $k(O^*)$ es la llave de la operación elegida, las flechas representan los posibles intercambios entre las j operaciones competidoras.

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 mostará 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.

3.3. Extensión a vecindad N7

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 [4]. 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.

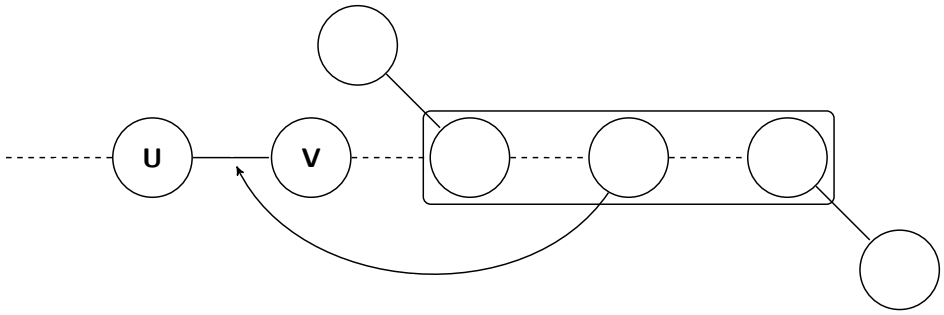


Figura 3.4: Movimientos propuestos. El tiempo de inicio de **v** es mayor al de finalización de **u**

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

3.4. Función de fitness

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

- C_{max} Makespan
- $\sum C_i^2$ Tiempo al cuadrado total
- $\sum J_i$ Flowtime
- $\sum I(C_i = C_{max})$ Número de máquinas cuyo tiempo de finalización es igual al makespan
- Número de rutas críticas
- $\sqrt{Var(C_i)}$ Desviación estándar de los tiempos de finalización

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 [10]. 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.

Búsqueda local iterada

Como se muestra en el algoritmo 3 la búsqueda local iterada requiere que definamas una manera de perturbar una solución dada así como definir un criterio de paro. La perturbación debe ser suficientemente grande como para permitirnos salir 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 definicion es conveniente porque presenta una manera de aceptar cambios que no mejoran la solución pero que a su vez están conectados a soluciones de mejor calidad que la mejor actual.

El criterio de paro en este caso fue el tiempo, se tomaron 5 minutos para todos los experimentos lo cual representa una cantidad de tiempo muy pequeña comparada con la requerida por los métodos del estado del arte.

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.

Instancia	Tamaño	ILS con N7		Estado del arte
		Mejor	Mediana	
DMU01	20×15	2793	2867	2563
DMU02	20×15	2853	3003	2706
DMU03	20×15	2880	2968	2731
DMU04	20×15	2831	2912	2669
DMU05	20×15	2934	3056	2749
DMU06	20×20	3395	3565	3244
DMU07	20×20	3239	3359	3046
DMU08	20×20	3332	3423	3188
DMU09	20×20	3247	3363	3092
DMU10	20×20	3100	3215	2984
DMU11	30×15	3681	3819	3430
DMU12	30×15	3737	3955	3492
DMU13	30×15	3893	4082	3681
DMU14	30×15	3526	3650	3394
DMU15	30×15	3452	3548	3343
DMU16	30×20	3985	4075	3751
DMU17	30×20	4142	4262	3814
DMU18	30×20	4074	4174	3844
DMU19	30×20	4033	4170	3764
DMU20	30×20	3939	4038	3703
DMU21	40×15	4408	4503	4380
DMU22	40×15	4738	4819	4725
DMU23	40×15	**4668**	4743	4668
DMU24	40×15	**4648**	4676	4648
DMU25	40×15	**4164**	4164	4164
DMU26	40×20	5027	5131	4647
DMU27	40×20	5038	5171	4848
DMU28	40×20	4897	5035	4692
DMU29	40×20	4924	5067	4691
DMU30	40×20	4932	5093	4732
DMU31	50×15	**5640**	5649	5640
DMU32	50×15	**5927**	5927	5927
DMU33	50×15	**5728**	5728	5728
DMU34	50×15	**5385**	5385	5385
DMU35	50×15	**5635**	5635	5635
DMU36	50×20	5819	5981	5621
DMU37	50×20	5991	6218	5851
DMU38	50×20	6054	6199	5713
DMU39	50×20	5773	5875	5747
DMU40	50×20	5700	5815	5577

Instancia	Tamaño	ILS con N7		Estado del arte
		Mejor	Mediana	
DMU41	20×15	3613	3791	3248
DMU42	20×15	3750	3978	3390
DMU43	20×15	3782	4058	3441
DMU44	20×15	3891	4109	3475
DMU45	20×15	3647	3962	3266
DMU46	20×20	4462	4664	4035
DMU47	20×20	4372	4545	3942
DMU48	20×20	4200	4360	3763
DMU49	20×20	4117	4311	3710
DMU50	20×20	4352	4560	3729
DMU51	30×15	4695	5063	4156
DMU52	30×15	4895	5195	4303
DMU53	30×15	5098	5399	4378
DMU54	30×15	4933	5247	4361
DMU55	30×15	4831	5142	4263
DMU56	30×20	5910	6115	4941
DMU57	30×20	5587	5842	4653
DMU58	30×20	5619	5825	4701
DMU59	30×20	5562	5881	4616
DMU60	30×20	5759	5936	4721
DMU61	40×15	5893	6366	5171
DMU62	40×15	5943	6492	5248
DMU63	40×15	6095	6447	5313
DMU64	40×15	6044	6372	5226
DMU65	40×15	5915	6181	5184
DMU66	40×20	6870	7229	5701
DMU67	40×20	7066	7379	5779
DMU68	40×20	7175	7501	5763
DMU69	40×20	6914	7309	5688
DMU70	40×20	7148	7385	5868
DMU71	50×15	7186	7449	6207
DMU72	50×15	7381	7758	6463
DMU73	50×15	7175	7437	6136
DMU74	50×15	7175	7552	6196
DMU75	50×15	7388	7644	6189
DMU76	50×20	8620	8905	6718
DMU77	50×20	8569	9032	6747
DMU78	50×20	8796	9286	6755
DMU79	50×20	8817	9149	6910
DMU80	50×20	8287	8845	6634

Capítulo 5

Conclusiones y Trabajos a Futuro

Conclusiones

Bibliografía

[1] The origins of order: self-organization and selection in evolution, 1993.

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

[3] 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.

[4] 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.

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

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

[7] 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.

[8] 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.

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

[10] 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.

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

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

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

[14] 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.

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

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

[17] 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.

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

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

[20] 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.

[21] 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.

[22] 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.