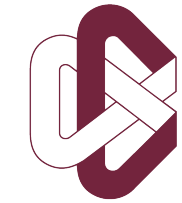


Centro de Investigación en Matemáticas, A.C.

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

Existe una gran cantidad de tipos de problemas de planificación, siendo el problema de planificación de producción tipo taller (JSP por sus siglas en inglés) un tipo específico que ha atraído considerable atención por su dificultad y aplicabilidad. En general estos problemas consisten en encontrar una planificación que cumpla un conjunto de restricciones y optimice alguna función objetivo, siendo la minimización del tiempo requerido para completar un conjunto de trabajos (makespan) una de las funciones objetivo más habituales. En esta tesis se trabaja con la formulación estándar del JSP, en la que precisamente se debe minimizar el makespan, y en particular se analiza el efecto sobre el rendimiento de distintas estrategias de modificación del paisaje de búsqueda.

En la actualidad existen algoritmos exactos para encontrar la solución óptima del JSP [10]. Sin embargo, el tiempo requerido para resolver instancias grandes es demasiado elevado como para que sean prácticos o inclusive factibles. La complejidad de este problema lo hizo un candidato ideal para utilizar métodos aproximados con el fin de generar soluciones aceptables en tiempos razonables. Así, en la actualidad, los mejores resultados obtenidos para las instancias de prueba más difíciles se han encontrado mediante metaheurísticas, y en específico la metaheurística de trayectoria búsqueda tabú es una de las más efectivas. Las metaheurísticas son convenientes por su relativa simplicidad algorítmica frente a otros métodos; sin embargo, el tiempo requerido para obtener soluciones de alta calidad también ha comenzado a crecer y el diseño de las técnicas se ha complicado enormemente, incluyendo así numerosas componentes que no son para nada sencillas de programar e integrar entre sí. A modo de ejemplo, algunos trabajos recientes consideran ejecuciones de varios días en ambientes paralelos, y que integran múltiples componentes como algoritmos evolutivos, búsqueda tabú y modelos subrogados. Estos requerimientos pueden no ser muy prácticos a la hora de resolver problemas de este tipo en empresas, por lo que esta tesis parte de la idea de que es importante dar un paso atrás y replantear un poco la forma de aplicar estos métodos para evitar complicarlos en exceso.

En el campo de las metaheurísticas 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: la representación de las soluciones, la definición de vecindad

de una solución, y la función de aptitud o fitness. En el trabajo se proponen y analizan diferentes alternativas para cada una de estas componentes, y se evalúa el impacto sobre el rendimiento utilizando para ello una de las metaheurísticas de trayectoria más sencillas, la búsqueda local iterada. Esta evaluación experimental considera las instancias dmu, las cuales son uno de los conjuntos de prueba más utilizados actualmente y que aun no ha sido resuelto por completo. A partir de los análisis realizados se pudo concluir que de las tres componentes en las que se plantearon modificaciones, la que más impacto tiene para el rendimiento es la representación de soluciones. Entre otras propuestas, se diseñó una nueva representación basada en algunas propiedades que se estudiaron en los 90s, y junto con definiciones de funciones de fitness y vecindades adecuadas, permitió mejorar de forma significativa la calidad de las soluciones encontradas en ejecuciones a corto plazo. Dado que esta representación difiere de las usadas en algoritmos mucho más complejos como búsqueda tabú, los resultados discutidos en esta tesis abren la puerta a nuevas investigaciones que utilicen este tipo de representación más elaborada, junto a metaheurísticas de trayectoria más complejas. En esta tesis también se proponen nuevas funciones de fitness y definiciones de vecindad, mostrándose que, aunque permiten mejorar aún más las soluciones, el efecto es mucho más limitado. Entre las contribuciones, cabe destacar que se dispone de un optimizador, en el que en ejecuciones de 5 minutos en entornos secuenciales, la mediana del error relativos es menor al 0,21 en todas las instancias tratadas, valor que es significativamente inferior en comparación de los casos en que no se incorporan los cambios propuestos en esta tesis, que alcanzan errores relativos de 0,34.

Agradecimientos

A CONACYT (Consejo Nacional de Ciencia y Tecnología) por el apoyo económico que me fue brindado como parte del Programa Nacional de Posgrados de Calidad (PNPC) en el periodo comprendido entre agosto 2019 y julio 2021.

Al CIMAT (Centro de Investigación en Matemáticas) por abrirme las puertas y proporcionarme el espacio, herramientas y ambiente necesarios para seguir desarrollándome y para la escritura de este documento. Gracias a todo el personal del centro por su gran labor.

A mi asesor el Dr. Carlos Segura González por su tiempo, paciencia y conocimiento los cuales fueron indispensables para la realización de este trabajo.

A mi familia por su apoyo incondicional en esta etapa así como en las anteriores porque sin ellos no habría llegado aquí.

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 dichos objetivos. Al final del mismo se resumen las contribuciones de este trabajo.

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 está compuesta por varios trabajos que a su vez están compuestos por operaciones que pueden o deben repartirse entre distintos recursos, como máquinas o unidades de procesamiento. A grandes rasgos un problema de planificación consiste en asignar tiempos de procesamiento a las operaciones antes mencionadas. A esta asignación se le conoce como **planificación**. 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 la mayor parte de estos contextos se tienen restricciones, por ejemplo que ciertas operaciones respeten algún orden de precedencia o que sean procesadas por algún recurso en específico. Se dice que una planificación es factible si cumple con las restricciones impuestas. En la mayor parte de estos contextos no solo se requiere hallar una planificación factible sino que además se quiere encontrar una que sea óptima en algún

sentido. Una de las funciones objetivo más habitualmente usada consiste en minimizar el tiempo requerido para completar la tarea; sin embargo, existen muchas otras.

Desde los década de los 50 se comenzaron a desarrollar algoritmos para hallar planificaciones óptimas para problemas con dos y tres máquinas [24] y, dado su gran campo de aplicaciones, el interés en ellos ha crecido en las décadas siguientes hasta la fecha. Así, han surgido numerosas formulaciones de problemas de planificación, que incluyen diferentes restricciones y funciones objetivo.

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 (Job Shop Schedulig Problem). En este problema la tarea global está constituida por un conjunto de n trabajos que se deben procesar en m máquinas. Cada uno se esos trabajos están constituido por un conjunto de operaciones (habitualmente m), que se deben procesar en un orden determinado y cada una de ellas en una máquina en específico. El propósito es encontrar una planificación (asignación de las operaciones a las máquinas), que respete el orden establecido y que minimice el tiempo en que finaliza la última operación (makespan). El JSP es un problema de optimización combinatoria que pertenece a la clase de problemas **NP-hard** cuando el número de máquinas es mayor a dos[19].

Anteriormente se han propuesto métodos exactos para resolver el JSP [10]. Sin embargo, la cantidad de recursos computacionales que requieren conforme el tamaño del problema (número de máquinas y trabajos) aumenta los hace inviables excepto para instancias pequeñas de 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 métodos aproximados para encontrar soluciones aceptables en tiempos reducidos. Estos métodos pueden agruparse de la siguiente manera[23, 50]:

- **Métodos Constructivos.** Estos métodos construyen una planificación mediante el uso de alguna regla simple que de forma iterativa va tomando decisiones sobre la solución hasta generar una solución completa. Se caracterizan por ser muy rápidos y conceptualmente sencillos pero habitualmente no alcanzan calidades cercanas a las óptimas. Se pueden clasificar 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 se 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, ya que por sí solas no suele generar soluciones de muy alta calidad. La gran desventaja de estos métodos es que se tienen que construir y entrenar redes a la medida según el tipo de instancia y a menudo se necesita mucha adaptación para aplicarse a nuevas variantes, 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 que sea mejor 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. En general estos métodos se distinguen entre sí por la manera de generar nuevas soluciones y por el criterio de paro. Tienen el inconveniente de quedarse estancados en óptimos locales.

- **Metaheurísticas.** Estos métodos combinan elementos de los previamente mencionados junto con otras ideas para tratar de evitar óptimos locales. Se distingue entre metaheurísticas de trayectoria y metaheurísticas poblaciones, y son los métodos que han dado mejores resultados para las instancias de mayor tamaño. Es muy común que se planteen metaheurísticas inspiradas en la naturaleza como los algoritmos genéticos basados en la evolución, o algoritmos basados en el comportamiento de seres vivos, como el algoritmo de colonia de hormigas o de abejas. Este tipo de optimizadores son estrategias de alto nivel que se pueden adaptar a diferentes problemas, y prácticamente la totalidad de las metaheurísticas más populares han sido adaptadas para tratar el JSP.

Actualmente los algoritmos más exitosos para resolver el JSP son algoritmos meméticos que combinan un método de búsqueda local o una metaheurística de trayectoria, con una metaheurística poblacional que mantiene varias soluciones. En particular, la metaheurística de trayectoria búsqueda tabú es una de las más efectivas en el JSP. Aunque se han obtenido buenos resultados con estos algoritmos, su mayor inconveniente el su alto requerimiento de recursos de cómputo. En particular, los mejores resultados conocidos se han generado haciendo uso de ejecuciones de 24 a 48 horas en entornos paralelos, y los resultados a corto plazo con estos métodos son de baja calidad. La literatura reciente se ha centrado en hacer más eficiente la búsqueda tabú para reducir los tiempos de ejecución, y aunque tener optimizadores muy efectivos a largo plazo es importante, también lo es disponer de métodos que a corto plazo sean capaces de encontrar soluciones de alta calidad.

Como se explicará a detalle más adelante un aspecto muy importante a la hora de diseñar metaheurísticas de trayectoria viene dado por la definición del paisaje de búsqueda del problema, el cual se compone de tres elementos. El primero y más básico es cómo representar computacionalmente una solución en el optimizador, y cómo decodificarla a soluciones del problema a tratar. Actualmente lo más común es que una solución se represente con tantas permutaciones de operaciones como máquinas, y que dichas permutaciones establezcan el orden exacto en el que las operaciones

se ejecutan en la máquina. El segundo es la definición de la vecindad, que permite establecer cómo se generan unas soluciones a partir de otras. En este caso la más exitosa es conocida como N7 y fue propuesta en 2006 [49]. Finalmente, el tercer elemento se conoce como función de fitness o aptitud y nos permite comparar soluciones y decidir si una es mejor que otra. El análisis de la literatura muestra que durante bastantes años no se han realizado innovaciones con respecto al paisaje de búsqueda y que se ha dado énfasis en otras componentes, como operadores de cruce, modelos subrogados o estructuras de datos que permitan reducir los tiempos de los métodos híbridos basados en búsqueda tabú.

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 que los propuestos en la literatura para encontrar soluciones de calidad no muy alejados a los del estado del arte, pero en tiempos reducidos. Las modificaciones propuestas se centran en el paisaje de búsqueda del problema y constituyen en concreto los siguientes objetivos específicos.

Objetivos específicos

- Proponer nuevas funciones de fitness que no solo tomen en cuenta el makespan de la planificación.
- Extender las definiciones de vecindad actuales, para conseguir que los óptimos locales sean de mayor calidad.
- Plantear nuevas representaciones, considerando algunas características que deben cumplir las soluciones óptimas.
- Analizar el impacto que tiene cada una de las propuestas anteriores, tanto en forma aislada como en forma conjunta.

1.3. Hipótesis

La hipótesis de este trabajo es que es posible mejorar de forma significativa los resultados obtenidos con optimizadores de trayectoria simples, si se redefine de forma adecuada el paisaje de búsqueda, y que en particular es posible conseguir resultados no muy lejanos de los reportados por los mejores algoritmos actuales, con algoritmos sencillos y rápidos.

1.4. Propuestas

Para poner a prueba la hipótesis se utiliza una metaheurística de trayectoria sencilla (búsqueda local iterada), que se extiende de las siguientes formas:

1. Se define una nueva vecindad que extiende a una de las más populares, la vecindad N7.
2. Se crean nuevas funciones de fitness que, siguiendo una comparación lexicográfica, no solo toman en cuenta el makespan sino también otras características de la solución.
3. Se integra con una nueva representación que posibilita que sólo se generen un subtipo de soluciones que cumplen un conjunto de características que se sabe que se dan en las soluciones óptimas del JSP.

1.5. Contribuciones

En este trabajo se consiguió plantear modificaciones al paisaje de búsqueda que, combinados con la metaheurística 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 dichas soluciones. Además, estas mejores fueron muy significativas en comparación con las casos en que no se incorporan dichas modificaciones, ya que en estos casos el error relativo se incrementó hasta 0,34 en las instancias más difíciles.

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 con el fin de generar exclusivamente soluciones que cumplen ciertas propiedades que se saben que aparecen en soluciones óptimas. 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.

Las modificaciones propuestas en esta tesis pueden servir como 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. Por su puesto, conseguir esto es un reto importante que requiere refinar aún más la representación y la estructura de vecindad aquí presentadas, así como estudiar la forma en que se debe integrar con muchas de las componentes que han llevado a encontrar las mejores soluciones conocidas hasta la fecha.

Capítulo 2

Marco teórico

Sumario

En este capítulo se presentan diversos conceptos utilizados en este trabajo de tesis, así como definiciones relativas al problema y algunos algoritmos específicos relevantes para el presente trabajo.

2.1. Optimización

La optimización es un área que tiene como propósito 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 al elegir entre diferentes productos con diversas características.

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.

En función de las características del problema y método de optimización, estos se pueden clasificar de diferentes formas. A continuación se presentan algunas de las agrupaciones más importantes.

Optimización con restricciones

A la definición de un problema de optimización también pueden agregarse un conjunto de restricciones, definidas habitualmente como un conjunto de funciones que toman una solución como entrada una solución e indican si esta cumple o no con ellas, e incluso el grado de incumplimiento. Se dice que una solución es factible si cumple con todas 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. En estos casos, el problema se convierte en encontrar una solución factible que optimice la función objetivo.

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 computacionalmente, por lo que es común optar por obtener simplemente 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 continua y discreta

En la definición de problema de optimización dado anteriormente 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 [30] en el sentido de que se puede utilizar el concepto de gradiente para 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. Sin embargo, el uso de gradientes también tiene muchas limitaciones y dificultades, por lo que ambos tipos de optimización son muy complejos y siguen siendo tema de investigación.

Optimización combinatoria

Dentro de los problemas de optimización discreta se distinguen los problemas de optimización combinatoria, en los cuales el conjunto de búsqueda es finito. Los problemas de optimización combinatoria, como el tratado en esta tesis, se pueden modelar con los siguientes elementos [8]:

- Un conjunto de variables $Z = \{z_1, z_2, ..., z_n\}$
- Dominio finito 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 que optimice cierta función objetivo.

Métodos de optimización estocástica

En relación a los métodos de optimización, esta tesis está enfocada en los métodos estocásticos. En contraste con los métodos de optimización deterministas, los métodos estocásticos pueden obtener soluciones diferentes en cada ejecución aunque tenga las mismas condiciones iniciales ya que internamente toman decisiones de forma probabilista. En muchas ocasiones es ventajoso tener esta aleatoriedad porque nos permite explorar el espacio de búsqueda con menos sesgos, lo que es especialmente útil cuando el espacio de búsqueda es poco «predecible» en el sentido en que no podemos a priori establecer reglas que nos permitan generar soluciones que estén en regiones de alta calidad. Uno de los método estocásticos más conocidos es recocido simulado, el cual se inspira en un fenómeno metalúrgico y que encuentra soluciones muy buenas a una gran variedad de problemas de optimización. Posteriormente, se describen otros métodos estocásticos, incluyendo a la búsqueda local iterada, siendo este último el método que se ha utilizado en esta tesis para validar las propuestas realizadas.

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. El área de teoría de la complejidad busca responder, entre otras, a la pregunta: *¿Qué hace a algunos problemas fáciles y a otros difíciles?* [37].

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 describen algunas de estas clases, así como algunas notaciones relacionadas.

Notación gran O

La cantidad de operaciones necesarias para resolver algún problema con un algoritmo específico 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 es de especial relevancia 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 engloban algoritmos que tienen similar costo computacional. La notación gran O es usada ampliamente para describir el comportamiento asintótico de los algoritmos.

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 $n \geq n_0$ [14]. 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.

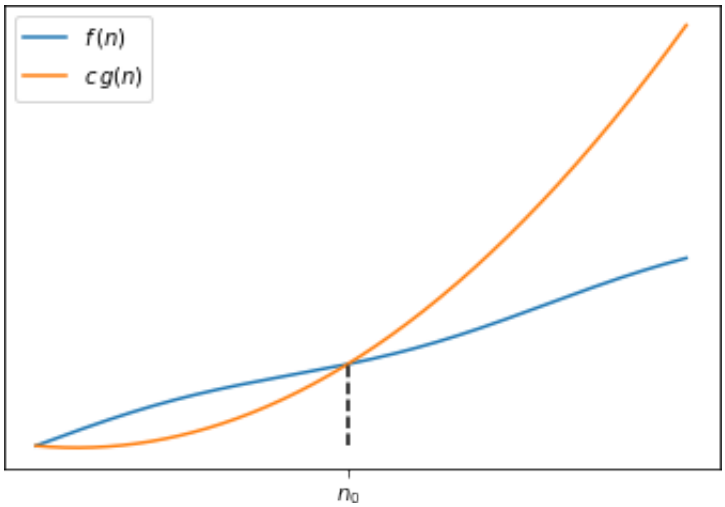


Figura 2.1: Representación de $f(n) = O(g(n))$

[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 $\subseteq$ [22]

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 dígitos toma $O(n)$ operaciones (al considerar que nuestra operación básica es sumar dígitos).

Clases P, NP

Un algoritmo para resolver algún problema puede tener como resultado cualquier cosa; por ejemplo un número si el problema es una suma, un polígono si buscamos una envolvente convexa, una función si resolvemos una ecuación diferencial o hasta una palabra como `sí` o `no` si buscamos saber si un número es primo. Este último tipo de problema es un ejemplo de una clase importante llamada problemas de decisión. Estos problemas son de interés teórico en la teoría de la complejidad por tener una salida muy simple y porque una gran cantidad de otro tipo de problemas pueden formularse como problemas de decisión. Las clases de complejidad más analizadas están definidas para estos problemas de decisión.

Dos de las clases más analizadas son las siguientes:

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

- **NP**. Engloba a 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 ha sido resuelto.

La distinción en algoritmos que toman tiempo polinomial podría parecer arbitraria. Sin embargo, es importante porque los polinomios ejemplifican funciones de crecimiento lento y cumplen con varias propiedades teóricas que simplifican la clasificación de los problemas [46].

Clase NP-hard

Las clases de complejidad antes mencionadas se definieron para problemas de decisión pero son útiles para clasificar otros tipos de problemas. En particular la clase **NP-hard** sirve para clasificar

[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

problemas de optimización como el JSP .
Se dice que un problema de optimización pertenece a **NP-hard** si todos los problemas en **NP** pueden reducirse a él en tiempo polinomial. Para los problemas de optimización que pertenecen a esta clase no existe un algoritmo que encuentre la solución óptima en tiempo polinomial a menos que **P** = **NP**. Es importante mencionar que el problema en cuestión no tiene por qué pertenecer a **NP**.

Muchos problemas de optimización de gran interés pertenecen a la clase **NP-hard**, entre ellos el JSP. Cabe mencionar que al JSP se le puede asociar un problema de decisión que consiste en determinar si una solución dada es óptima o no siendo este problema **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 original. Así, 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 [38, 40] con base en diferentes características:

- **Poblacionales vs de trayectoria**. En las metaheurísticas poblacionales se mantiene un conjunto de soluciones candidatas que se utilizan para generar nuevas soluciones a menudo utilizando nociones de la teoría de la evolución o bien de comportamientos colectivos que se dan en la naturaleza como el comportamiento de una colonia de abejas. Por otro lado las metaheurísticas de trayectoria mantienen una única solución que se va reemplazando conforme el algoritmo avanza de modo que la solución sigue una trayectoria en el espacio de búsqueda.

- **Constructivas vs de mejora**. En las metaheurísticas constructivas se mantienen soluciones parciales a las que se añaden elementos hasta que se consigue una solución completa. En general suelen usar adaptaciones de algoritmos voraces para añadir estos elementos. Por otra parte en las metaheurísticas de mejora se mantienen soluciones completas que van mejorando

a medida que avanza el tiempo ya sea haciendo cambios en ellas o bien reemplazándolas por nuevas soluciones mejores.

- **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 o una serie de características que queramos evitar o bien que sean deseables. De este modo la memoria permite tener algo parecido a un aprendizaje o a un reconocimiento de patrones en el paisaje de búsqueda que puede servir de guía.

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: la representación de 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, el cual es el principal tema de estudio de esta tesis.

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 [47]. 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 código genético un punto en el mapa y un valor de adaptabilidad. En algunas extensiones de este concepto también se agrega información sobre 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 anterior. 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

En ocasiones, el problema de optimización en el que estemos interesados puede surgir directamente de un ámbito de las matemáticas; sin embargo, si estamos interesados en un problema de

nuestro entorno físico es necesario idear una forma de traducirlo a un lenguaje matemático, incluyendo la soluciones al mismo. Por ello, debemos encontrar una forma de representar de manera útil las soluciones posibles. A continuación se considera un ejemplo sencillo donde las soluciones que buscamos son elementos de $\mathbb{R}^2$.

Ejemplo

Supongamos que queremos colocar una lámpara en una habitación de manera que obtengamos la mejor iluminación. En este caso podemos elegir algún punto en la habitación como el origen y representar las soluciones a nuestro problema como puntos en un subconjunto de $\mathbb{R}^2$. Podemos representar estos puntos de varias maneras por ejemplo con sus coordenadas cartesianas o bien con sus coordenadas polares como en la figura 2.2.

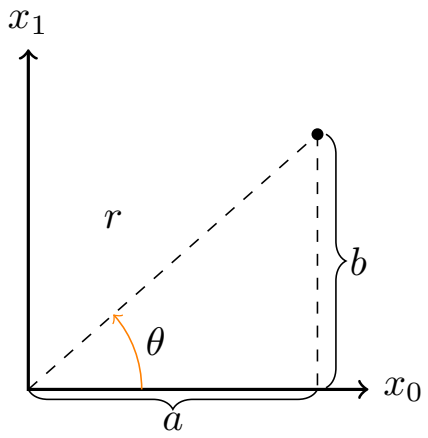


Figura 2.2: Representación en coordenadas cartesianas (a, b) y polares (r, θ) para un punto en $\mathbb{R}^2$

Aunque son equivalentes, estas dos representaciones tienen propiedades diferentes. Por ejemplo, la representación en coordenadas polares no necesariamente es única porque (r, θ) y $(r, \theta + 2\pi)$ representan el mismo punto. Cada una de estas representaciones puede presentar ventajas o desventajas de acuerdo con factores como la forma de la habitación, tal vez sea ventajoso usar la representación cartesiana si la habitación es cuadrada o usar la representación polar si la habitación es redonda.

Con el ejemplo anterior también podemos ver que si queremos establecer algunos operadores que, por ejemplo, perturben la solución tendremos que definirlos de maneras distintas en función de la representación que estemos considerando para asegurarnos que la perturbación tiene el efecto que deseamos. También es importante notar que es posible que el número de soluciones factibles e infactibles cambian en función de la representación, por lo que dependiendo de la misma, se puede facilitar o dificultar la búsqueda.

Formalmente una representación está formada por con conjunto de objetos (representaciones de soluciones) y un mapa que asocia elementos entre el conjunto de representaciones y el conjunto

de las soluciones. 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. En función del tipo de mapeo entre representaciones y soluciones se pueden distinguir las siguientes [12]:

- 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.

Una representación pictórica de los tipo de mapeo se encuentra en la figura 2.3.

Nótese que en una representación se pueden combinar las propiedades anteriores, así una solución podría estar representada por una única representación y otra por muchas. Estas características también son importantes y afectan enormemente al rendimiento de la búsqueda.

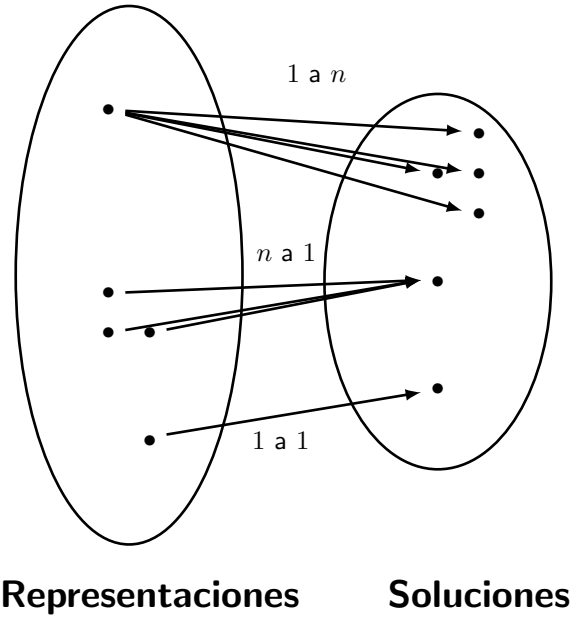


Figura 2.3: Tipos de representaciones

En general, el mapeo 1 a 1 es el más utilizado, sin embargo, todos estos tipos de mapeos han sido útiles. En particular, si conseguimos un mapeo en el que las soluciones de más calidad, estén sobrerrepresentadas, se pueden conseguir búsqueda más efectivas.

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 las soluciones formen una componente conexa, que no tenga un tamaño excesivamente grande y que contenga las vecinos de una solución tenga una calidad no muy distinta a la de la solución inicial; esta última característica le confiere «suavidad» y puede ser de ayuda para tener búsqueda más efectivas, ya que en caso contrario, evaluar una solución prácticamente no ofrece información sobre la calidad de la región en la que se está buscando.

2.4.3. Función de aptitud o fitness

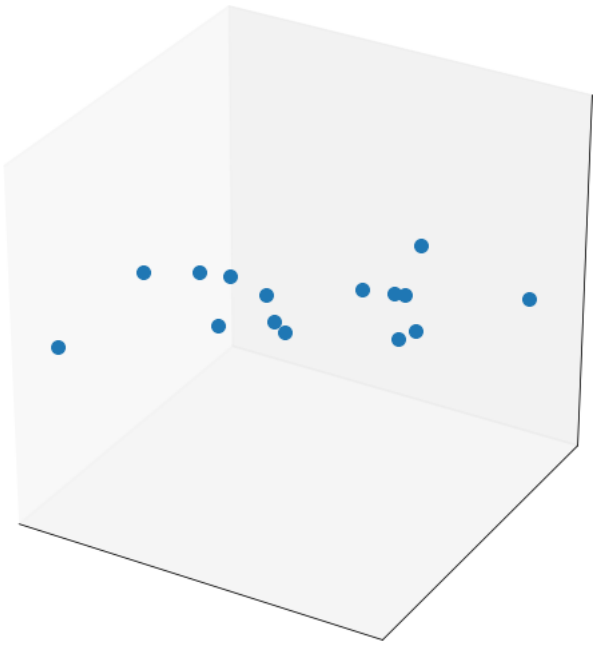
En el momento en el que se define el problema de optimización, se define cuál es la función objetivo; no obstante, puede resultar útil construir otra función a la cual se le conoce como función de aptitud o fitness que toma en cuenta otras características de las soluciones para que el paisaje de búsqueda 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 posea características que la hacen un mejor punto de partida para alguna metaheurística. Por ello, el diseño de funciones de aptitud o fitness apropiadas es muy importante para el buen desempeño de los algoritmos.

La función de aptitud debe asociar a cada solución un elemento de un conjunto en que se tiene definida una noción de ordenamiento. En esencia esta función define un operador de comparación entre soluciones de modo que define qué soluciones se consideran mejor que otras. Una vez que tenemos el conjunto de soluciones representables 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.

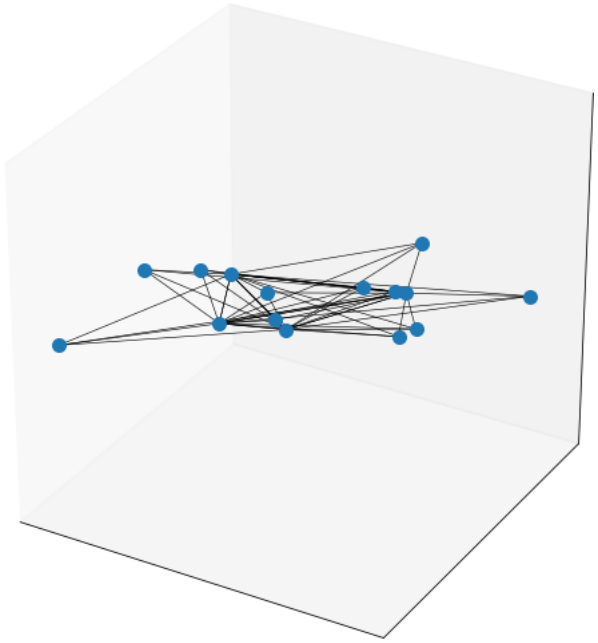
Además, podemos asociar a cada solución en el espacio un valor de aptitud o fitness que mide la calidad de dicha solución. La adición de esta función de aptitud al espacio de búsqueda genera al paisaje de búsqueda. Formalmente el paisaje de búsqueda $\mathcal{L}$ es entonces una tupla conformada por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

En la figura 2.4 se muestra una representación pictórica del paisaje de búsqueda para un problema de optimización.

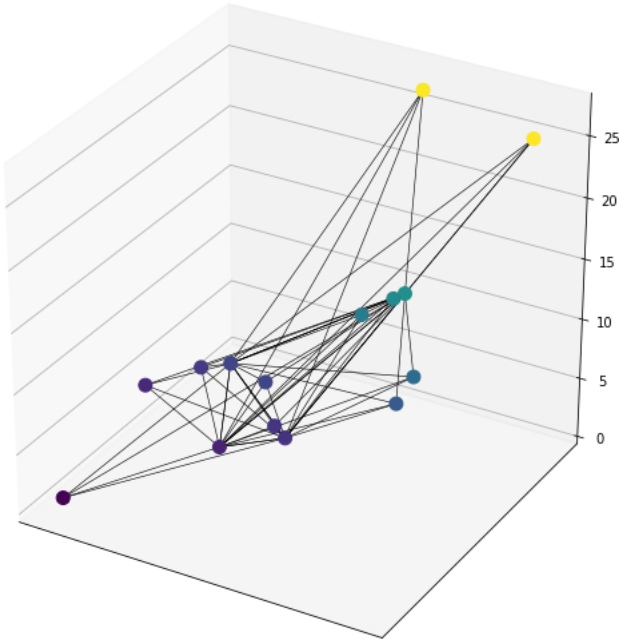
El paisaje de búsqueda es el «terreno» a explorar y puede cambiar si cambiamos cualquiera



(a) Soluciones representables



(b) Relaciones inducidas por los operadores de cambio



(c) Adición de la función de fitness

Figura 2.4: Creación del paisaje de búsqueda

de sus componentes. Así, podría ser que alguna representación nos centre en un subconjunto de soluciones convenientes o que se proponga alguna estructura de vecindad 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 si se usara directamente la función objetivo.

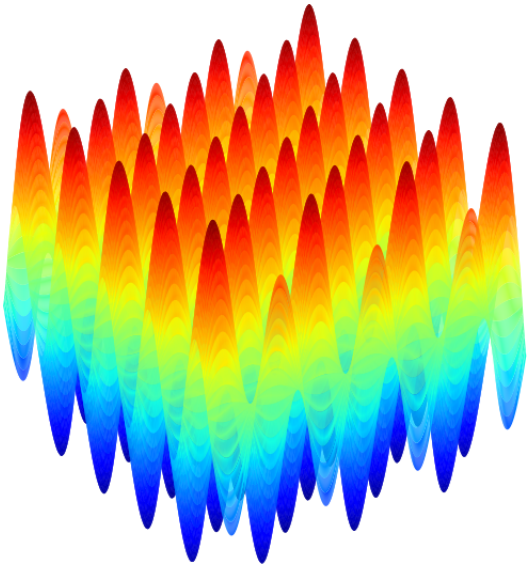
La estructura del paisaje de búsqueda influye de manera determinante en el éxito o fracaso de las metaheurísticas. Dependiendo de la «forma» del paisaje se favorecerá el uso de ciertas metaheurísticas. La «forma» del paisaje hace referencia a cómo cambia el valor de fitness para soluciones conectadas entre sí y para modelarlo de forma adecuada se usan diferentes tipos de mediciones. En la figura 2.5 se muestran los diferentes tipos de paisajes de búsqueda. Una medida ampliamente usada estima cómo cambia el valor de la función de fitness conforme nos acercamos a óptimos locales, mientras que otra se gran popularidad estima qué tanto cambia el fitness entre soluciones vecinas[25]. 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. En esta tesis no nos centramos en cómo cuantificar y analizar el paisaje de búsqueda, por lo que no se entra en más detalle. El objetivo se centra en proponer modificaciones a los paisajes de búsqueda más usado para el JSP, con el fin de mejorar el rendimiento de metaheurísticas sencillas.

Algunas metaheurísticas de relevancia

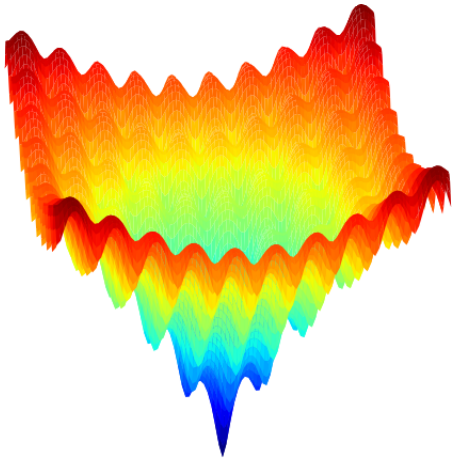
Las metaheurísticas sirven como una estrategia para explorar el paisaje de búsqueda. Una de las más sencillas e intuitivas es conocida como descenso/escalada estocástica y simplemente consiste en reemplazar la solución actual por algún vecino mejor escogido al azar .Esta es una metaheurística de trayectoria y traza un camino entre las soluciones inicial y final.

Algorithm 1: Algoritmo de descenso/escalada estocástica

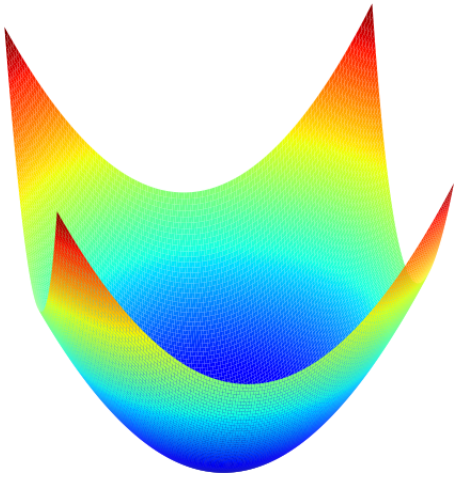
```
Data: Problema de Optimización
Result: Óptimo local x
Generar solución inicial x;
Generar lista de vecinos L de x;
while L no vacía do
    Generar lista de vecinos L de x;
    Escoger al azar un vecino y ∈ L;
    if y < x then
        x ← y;
        Generar lista de vecinos L de x;
    else
    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



(a) Paisaje rugoso con muchos óptimos locales



(b) Paisaje rugoso con con un gran valle



(c) Paisaje suave con un gran valle

Figura 2.5: Diferentes tipos de paisajes de búsqueda. Se muestran paisajes continuos con fines ilustrativos.

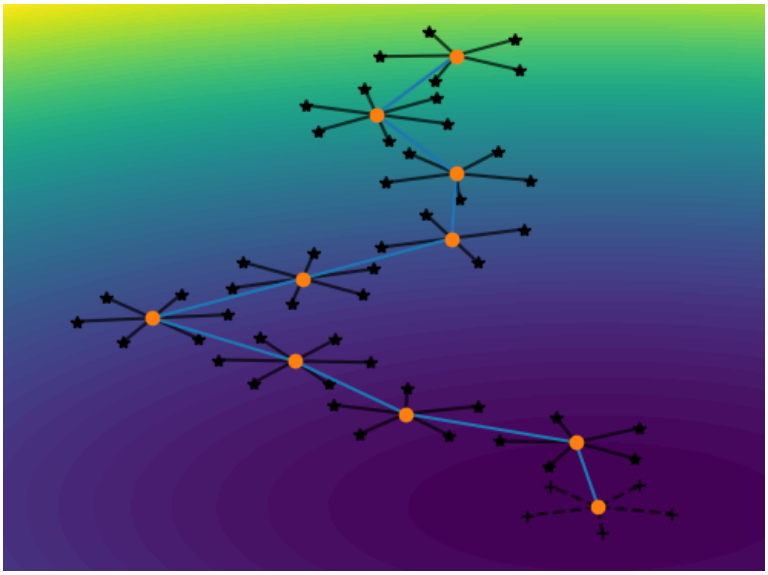


Figura 2.6: Ilustración de un descenso estocástico. Las aristas rayadas representan la vecindad del óptimo local.

Otra metaheurística de trayectoria importante que se ha considerado en la mayor parte de métodos que han alcanzado las mejores soluciones conocidas para el JSP es la llamada búsqueda tabú 2. 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 o características de las mismas para evitar visitarlas de nuevo. De entre las soluciones vecinas no prohibidas por la lista tabú, se mueve a la mejor. Es necesario definir el tamaño de la lista, el tipo de información que contendrá y el criterio de paro.

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 información de x ;
while *no criterio de paro* **do**
    Generar lista de vecinos L;
    Escoger un vecino $y \in L$ que no esté prohibido por TL ;
    $x \leftarrow y$;
    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 búsqueda local

iterada 3. La idea de esta metaheurística es obtener un mínimo local a partir de una solución inicial mediante búsqueda local, y aplicarle una perturbación seguida de búsqueda local para evitar el óptimo local. Este proceso se repite 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. También debe definirse un criterio de aceptación para las soluciones, lo más común es que se pida que la nueva solución tenga un valor de fitness mejor que la anterior, como es el caso en este trabajo.

Algorithm 3: Algoritmo búsqueda local iterada

```
Data: Problema de Optimización
Result: Última solución aceptada 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 es aceptada then
        x ← y;
        x* ← y;
    else
        x ← 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. De este modo una operación solo puede comenzar a procesarse cuando su operación precedente en el trabajo y su operación precedente en la máquina ya han sido procesadas. Es decir que cada operación tiene dos tipos de dependencias, las propias de su trabajo y las de la máquina en donde ha de ser procesada.

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, de forma ordenada, la máquina en que debe procesarse el trabajo en cada paso y el tiempo que toma procesarlo. Por ejemplo en la tabla 2.2 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 y respete los requisitos impuestos (orden y no concurrencia de diversas operaciones). Así, una planificación consiste en asignar tiempos de inicio y fin a cada operación respetando el orden requerido para cada trabajo.

2.5.1. Tipos de planificaciones

Independientemente de cómo se representen o como se construyan, las planificaciones pueden clasificarse en varios conjuntos. Dentro del conjunto de planificaciones factibles se pueden distinguir tres subconjuntos de interés para el presente trabajo [39]:

- **Planificaciones semi-activas** Son aquellas en las que el tiempo de inicio de cada operación no puede disminuirse sin modificar el orden especificado en la planificación. De modo más intuitivo son aquellas en las que una máquina solo deja de trabajar porque la dependencia de la operación a procesar no ha sido procesada o porque ya terminó todo lo que tenía asignado.

- **Planificaciones activas** Son a su vez un subconjunto de las planificaciones activas y 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.

- **Planificaciones óptimas** Es el conjunto conformado por las planificaciones con el menor makespan posible. Este conjunto contiene planificaciones activas [33] aunque puede o no contener otras planificaciones que solo son factibles.

La relación entre estos conjuntos se ilustra en la figura 2.7.

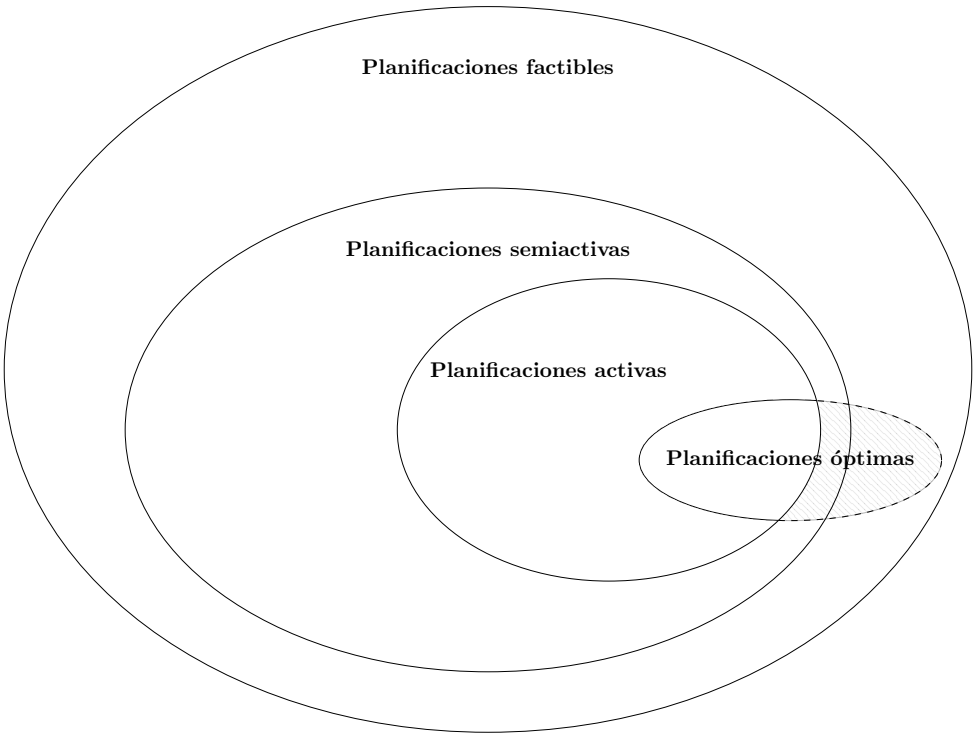


Figura 2.7: Subconjuntos de planificaciones. El área sombreada puede ser una intersección vacía

Es importante mencionar que el conjunto de las planificaciones factibles puede ser de tamaño arbitrario porque para cualquier planificación factible podemos insertar intervalos de tiempo inactivo de duración arbitraria entre operaciones en cualquier máquina sin alterar el cumplimento de las restricciones. Esto también implica que no hay una planificación única que cumpla con algún orden determinado de las operaciones en cada máquina. En la literatura en general, así como en este trabajo, solo se considera el conjunto de planificaciones semi-activas lo cual elimina estos problemas.

Ejemplo

Para mostrar las diferencias entre los tipos de planificaciones se presenta un instancia pequeña en la tabla 2.1 y tres planificaciones para ella en la figura 2.8: una factible pero no semi-activa, una semi-activa pero no activa y una activa. Tanto la planificación factible como la semi-activa siguen el mismo orden de procesamiento de las operaciones, mientras que la activa se construyo de forma que se puede observar que hay operaciones que pueden comenzar a procesarse antes en las planificaciones no activas.

Trabajo	Secuencia de procesamiento (máquina, tiempo)		
0	0,4	1,2	2,1
1	0,5	1,3	2,5
2	0,4	2,3	1,5
3	1,2	0,3	2,5
4	1,5	2,2	0,1
5	2,5	0,4	1,3
6	2,3	0,2	1,1

Tabla 2.1: Instancia simple con 3 máquinas y 7 trabajos

Otra ventaja de centrarnos en las planificaciones semi-activas es que nos permite definir el concepto **ruta crítica**.

Para una planificación semi-activa una **ruta crítica** es una secuencia ordenada de k operaciones $(O_1, O_2, \ldots, O_k)$ que cumple con las siguientes propiedades: el tiempo de inicio de la primera operación es cero y el de finalización de la última es igual al makespan de la planificación, el tiempo de inicio de la operación i es igual al tiempo de finalización de la operación $i-1$ para $i > 1$ y la operación $i-1$ es una dependencia de la operación i para $i > 1$.

La 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 semi-activa siempre tiene al menos una ruta crítica pero puede tener más de una a la vez. 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 la última operación. A este tiempo se le conoce como **makespan**. Sin pérdida de generalidad nos restringimos al caso en el que el tiempo requerido para procesar cada operación es un entero positivo.

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 en la tabla 2.2.

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

Tabla 2.2: Instancia simple con 3 maquinas y 2 trabajos

En la figura 2.9 se muestra 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.

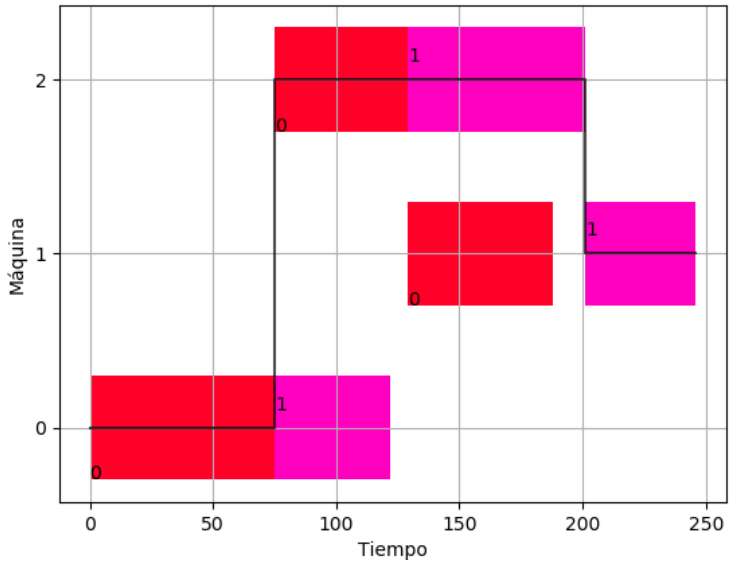


Figura 2.9: Diagrama de Gantt de una planificación posible para la instancia mostrada en la tabla 2.2. Se marcan los trabajos que pertenecen a la ruta crítica con una línea

Si bien es conveniente visualizar una planificación mediante un diagrama de Gantt como el mostrado anteriormente, resulta ventajoso representar computacionalmente estas planificaciones 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 [35]. En la siguiente sección se detallan dos formas de representación de especial importancia.

2.5.2. Representación de planificaciones

De manera general las representaciones para el JSP se pueden clasificar en[12]:

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

En este trabajo se utilizaron dos: una basada en permutaciones (representación directa) y las reglas de prioridad (representación indirecta).

Representación basada en permutaciones

En este modelo se identifica para cada máquina el conjunto de operaciones que deben procesarse en ella. Entonces una planificación se representa como un conjunto de m permutaciones

$(\sigma_0, \sigma_1, \ldots, \sigma_m)$, donde σ_i es la permutación de las operaciones que debe procesar la máquina i. Es decir que se asigna el orden de procesamiento de las operaciones cada máquina. Esta representación no toma en cuenta el orden de procesamiento que deben cumplir las operaciones dentro de sus trabajos correspondientes por lo que no siempre representa planificaciones factibles. Una forma de determinar si la permutación representa una planificación factible se construye 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.

Pueden agregarse dos nodos de control que sirven como el nodo inicial o fuente del que dependen todas las operaciones (S) y final que depende de todos las operaciones $(*)$, las restricciones de precedencia para las operaciones 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 cuya dirección se determina siguiendo la permutación dada. Para que una permutación represente una planificación factible su grafo asociado debe ser acíclico, puesto que la existencia de un ciclo en el grafo indica que hay una operación que está esperando a que se procese alguna otra operación que a su vez depende de que se procese la primera. Una vez que se sabe que el grafo es acíclico se puede proceder a asignar tiempos de inicio y fin a cada operación asignando al nodo fuente (S) tiempo de inicio y finalización igual a cero y marcándolo como visitado. Posteriormente aplicando un algoritmo para recorrer el grafo cerciorándonos de solo visitar un nodo cuando ambas de sus dependencias han sido visitadas y asignarle un tiempo de inicio mayor que el mayor tiempo de finalización de sus dependencias y un tiempo de finalización igual al tiempo inicial más la duración de la misma.

Si se busca generar una planificación semi-activa, entonces el tiempo de inicio de la operación será el máximo de los tiempos de finalización de sus dependencias.

Ejemplo

La planificación mostrada en la figura 2.9 tendría la siguiente representación como permutaciones:

$$\sigma_0 = (O_{0,0}\,,\, O_{1,0})$$
$$\sigma_1 = (O_{0,2}\,,\, O_{0,1})$$
$$\sigma_2 = (O_{0,1}\,,\, O_{1,1})$$

A partir de estas permutaciones se construye el grafo antes descrito y se escogen las direcciones de las aristas disyuntivas como se muestra en la figura 2.10. En este caso se verifica que el grafo es acíclico.

Representación basada en 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.

2.6. 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 [13], evolución diferencial [33], algoritmos de colonia de hormigas [26], algoritmos de enjambre de partículas [51] y búsqueda tabú [49]. Aunque estos son algoritmos muy diferentes entre sí, la mayoría de ellos o extensiones de los mismos, aplican de alguna forma el concepto de búsqueda local para mejorar su desempeño. Otro punto en común es que la mayor parte de algoritmos trabajan con la representación basada en permutaciones con soluciones semi-activas, y que se cuentan con múltiples vecindades para este tipo de representación.

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. A lo largo del tiempo se han propuesto diversas 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. Esto implica que estas vecindades están pensadas exclusivamente para soluciones semi-activas. En las figuras siguientes se presentan figuras ilustrativas para cada una de estas definiciones.

- N1 [7]: 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 crítica, considera

muchos cambios para los que actualmente se tienen demostraciones de que no pueden mejoran el makespan, por lo que en la actualidad prácticamente no se utiliza.

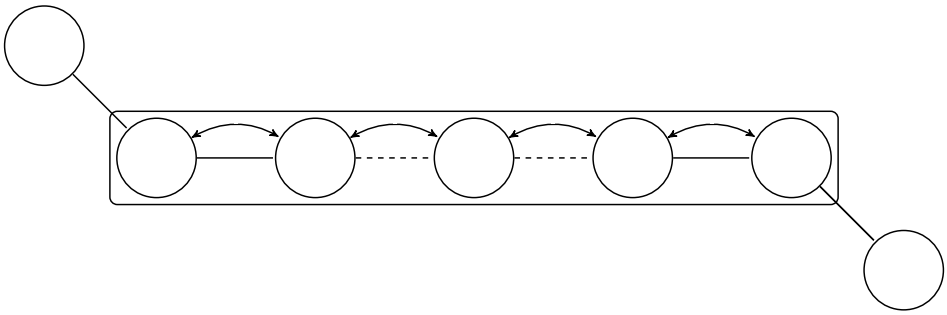


Figura 2.11: Movimientos de la vecindad N1

- N4 [15]: 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.

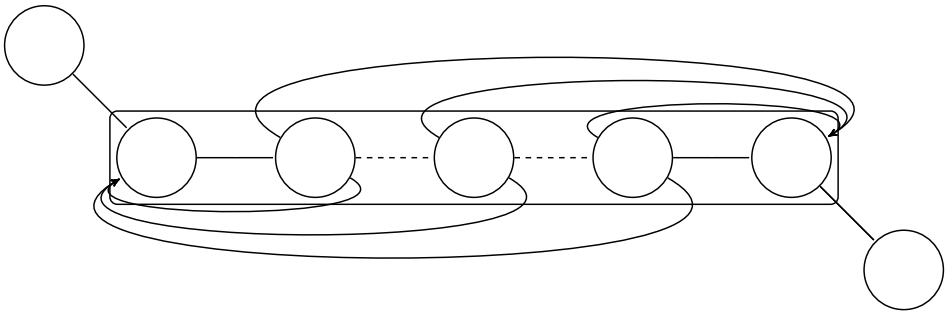


Figura 2.12: Movimientos de la vecindad N4

- N5 [17]: 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, así que se puede utilizar para casos muy grandes, y si se desea reducir el costo computacional.

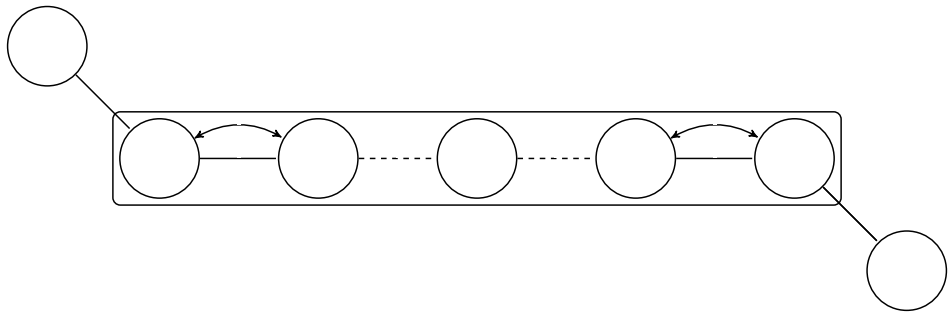


Figura 2.13: Movimientos de la vecindad N5

- N6 [4]: 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 como un refinamiento y extensión de la N4 por lo que tiene un tamaño similar.

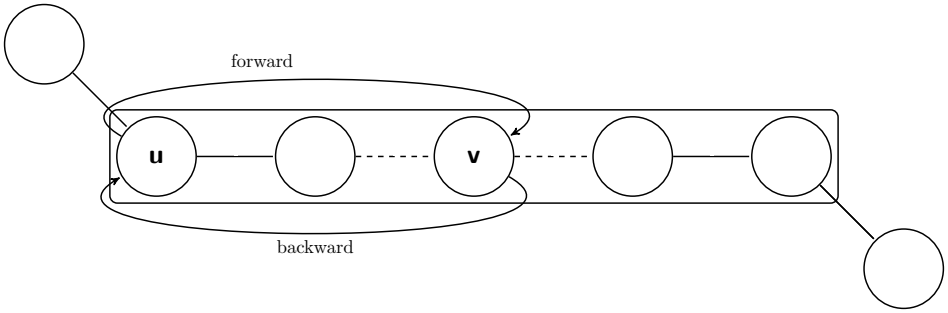


Figura 2.14: Los dos tipos de movimientos para un par (u, v)

- N7 [49]: Esta vecindad considera movimientos entre las operaciones inicial y final del bloque y las operaciones internas de una manera similar que los considerados en la vecindad N4. Los autores enuncian teoremas que permiten solo considerar un subconjunto de movimientos que generan soluciones factibles con lo que se reduce el tamaño de la vecindad.

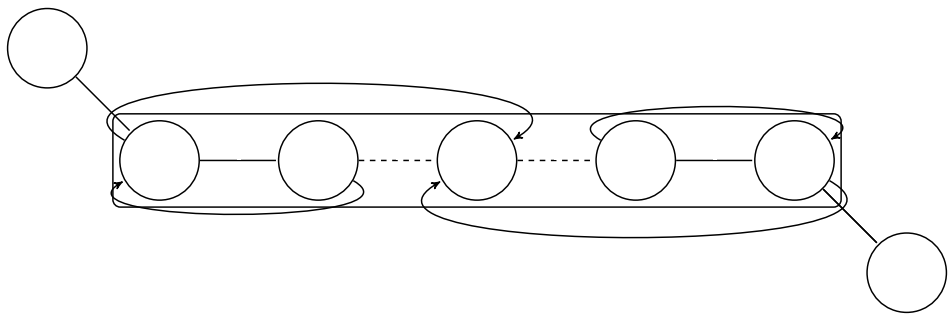


Figura 2.15: Movimientos de la vecindad N7

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 [3].

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 un costo asociado, podemos distinguir dos tipos de funciones de costo en la literatura [9]:

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

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 usado ampliamente como criterio de optimalidad porque está muy relacionado con los costos económicos de la planificación [34].

2.7. Paisaje de búsqueda del JSP

A partir de las definiciones anteriores sobre vecindades, función de aptitud y representaciones de las soluciones podemos construir el paisaje de búsqueda para el 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 difícil de hacer por el tamaño del espacio de soluciones. A la hora de analizar las características del paisaje de búsqueda ante diferentes parámetros de las instancias, a menudo presentan un un patrón *fácil-difícil-fácil* [28], es decir, que para valores muy pequeños o grandes de ciertos parámetros el problema es fácil, y existen ciertos valores intermedios, para los que el problema se hace difícil.

Para el caso particular del 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 o el número de vecinos, entre otros, lo que da pistas de la forma (como las mostradas en la figura 2.5) y dificultad del paisaje de búsqueda. Se ha encontrado que el paisaje de búsqueda para instancias al azar sigue el patrón *fácil-difícil-fácil* dependiendo de la razón entre el número de máquinas y número de trabajos de la misma [42] ($\frac{N}{M}$) conforme esta cantidad varía de 0 a infinito, siendo el caso más *difícil* cuando $\frac{N}{M} \simeq 1$.

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 $\frac{N}{M}$ crece. Para valores pequeños se tiene un paisaje semejante a un gran valle suave, para valores grandes se tiene un pasaje con muchos óptimos locales pero todos ellos 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 [29]. 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 [6]. 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 [45]. 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 «planicies» i. e. zonas en las que todas las soluciones vecinas tienen el mismo makespan, en la práctica estas zonas actúan 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 muy mala calidad, ya sea porque conforman una planicie amplia de la que es difícil salir 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.

Ya que la dificultad de cada instancia puede variar, es importante contar con conjuntos de instancias que permitan evaluar el desempeño de los métodos de solución.

2.8. Conjuntos de instancias de prueba

A lo largo de los años se han propuesto múltiples conjuntos de instancias de prueba para medir el desempeño de los algoritmos para la resolución del JSP. Uno de los más famosos fue propuesto en 1963 [18] y para una de sus instancias de 10×10 la solución óptima no fue encontrada sino hasta finales de los 80s [11].

Debido al aumento de poder de cómputo y al desarrollo de mejores métodos de solución se han

desarrollado conjuntos de prueba cada vez más desafiantes. En la tabla 2.3 se muestran en orden cronológico los conjuntos más usados. La mayor parte de ellos ya han sido resueltos casi en su totalidad.

En general, para generar un conjunto de instancias de prueba se elige un conjunto de valores

Nombre	Rango de tamaños (trabajos×máquinas)	Número de instancias	Instancias sin solución óptima
ft [18]	6×6 - 20×5	3	0
la [27]	10×5 - 15×15	40	0
abz [1]	10×10 - 20×15	5	1
orb [2]	10×10	10	0
swv [41]	20×10 - 50×10	20	5
yn [48]	20×20	4	3
ta [43]	15×15 - 100×20	80	21
dmu [16]	20×15-50×20	80	54

Tabla 2.3: Conjuntos de instancias de prueba populares

para los tamaños de las instancias, una distribución de probabilidad para generar los tiempos de procesamiento de cada operación y se determina una manera de asignar las restricciones de precedencia en cada trabajo. La dificultad de las instancias resultantes depende básicamente de dos factores: el tamaño de la instancia y la forma en la que se eligen las restricciones de precedencia en cada trabajo. Los tiempos de procesamiento de cada operación por lo regular se obtienen de una distribución uniforme.

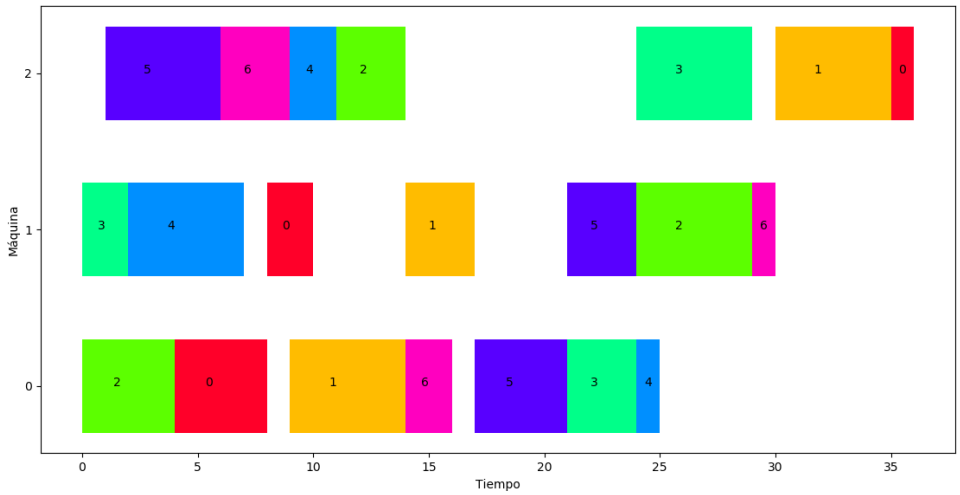
Puede aumentarse arbitrariamente el tamaño de las instancias para hacerlas más difíciles aunque estos aumentos en dificultad no necesariamente se deben a que el problema sea fundamentalmente más complicado de resolver. Existen varias formas de elegir las restricciones de precedencia. Como cada trabajo se procesa solamente una vez en cada una de las m máquinas, escoger estas restricciones de precedencia equivale a escoger una permutación (σ_i) de las m maquinas para cada trabajo. Puede escogerse al azar de entre todas las permutaciones posibles, pero se ha observado que una forma de hacer una instancia mucho más desafiante es dividir a las máquinas en subconjuntos y escoger una permutación para cada uno de ellos. Esto genera un cuello de botella que aumenta mucho el makespan de las soluciones en general y que genera óptimos locales de muy baja calidad.

Por ejemplo si las máquinas se dividen en dos subconjuntos de igual tamaño, las restricciones de precedencia se eligen escogiendo una permutación de las primeras $k = \lfloor \frac{m}{2} \rfloor$ y concatenando una permutación de las restantes de la forma $\sigma_i = (\sigma_{i0}(0, 1, \ldots, k), \sigma_{i1}(k + 1, k + 2, m))$.

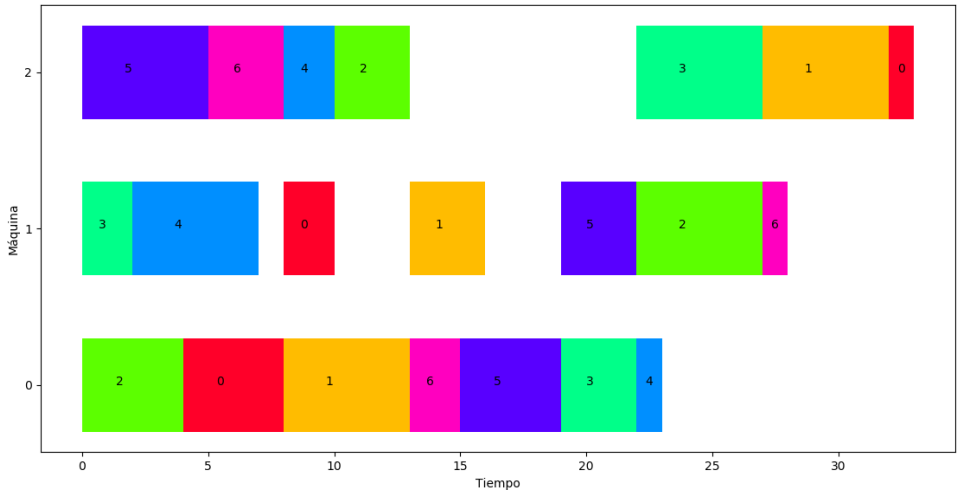
En la figura 2.16 podemos observar un caso simple de esto para 5 trabajos ($n = 5$) y dos máquinas ($m = 2$) en el que la operación inicial de cada trabajo debe procesarse en la misma máquina.

Actualmente el conjunto más popular y reciente es el llamado dmu[16] también conocido como

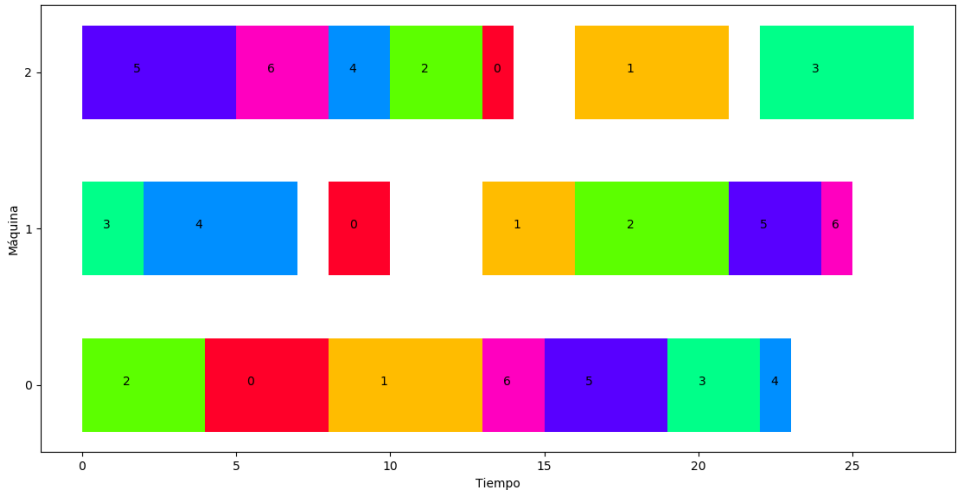
DMU01-80. La segunda mitad de estas instancias **DMU40-80** son consideradas especialmente difíciles porque siguen el esquema antes mencionado en el que todos los trabajos tienen operaciones iniciales en la misma mitad de las máquinas. Este conjunto es el que se eligió para probar las propuestas presentadas en este trabajo.



(a) Planificación factible pero no activa. Pueden observarse tiempos de inactividad que no se deben a ninguna restricción

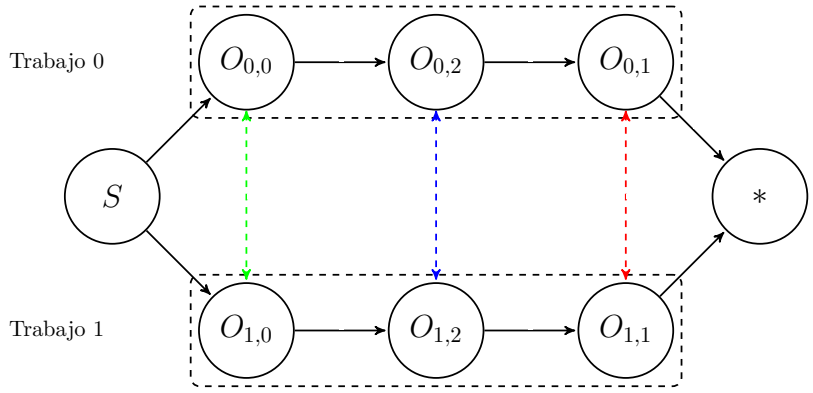


(b) Planificación semi-activa pero no activa. Solo hay tiempo de inactividad si se debe a las restricciones. Es posible reducir el tiempo de inicio de varias operaciones.

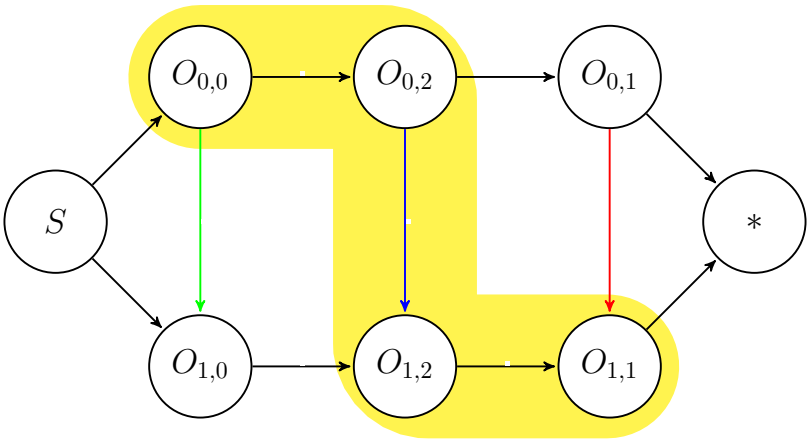


(c) Planificación activa. No es posible reducir el tiempo de inicio de ninguna operación.

Figura 2.8: Tres tipos de planificaciones para la instancia en 2.1

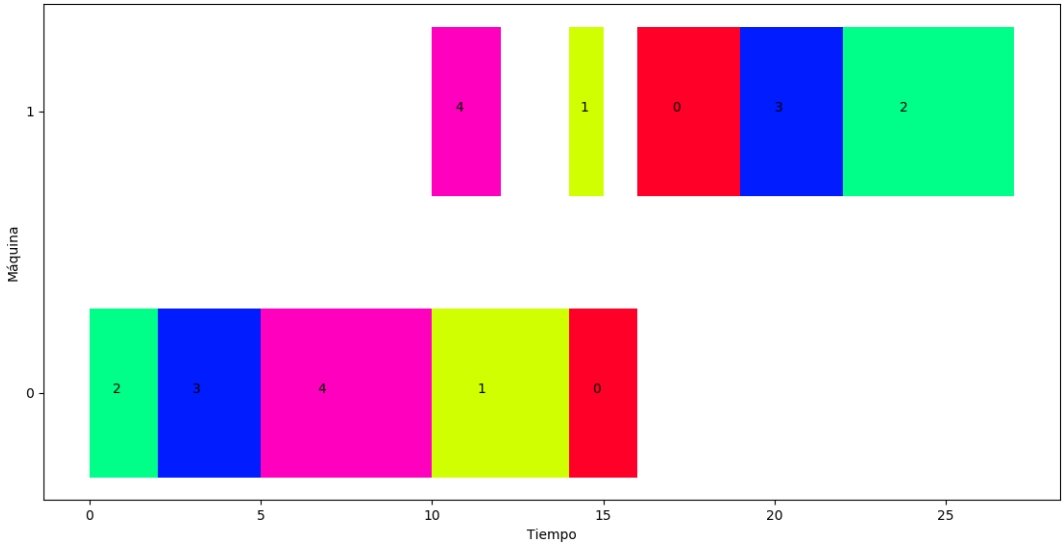


(a) Representación de una instancia, los colores distinguen entre las tres máquinas.

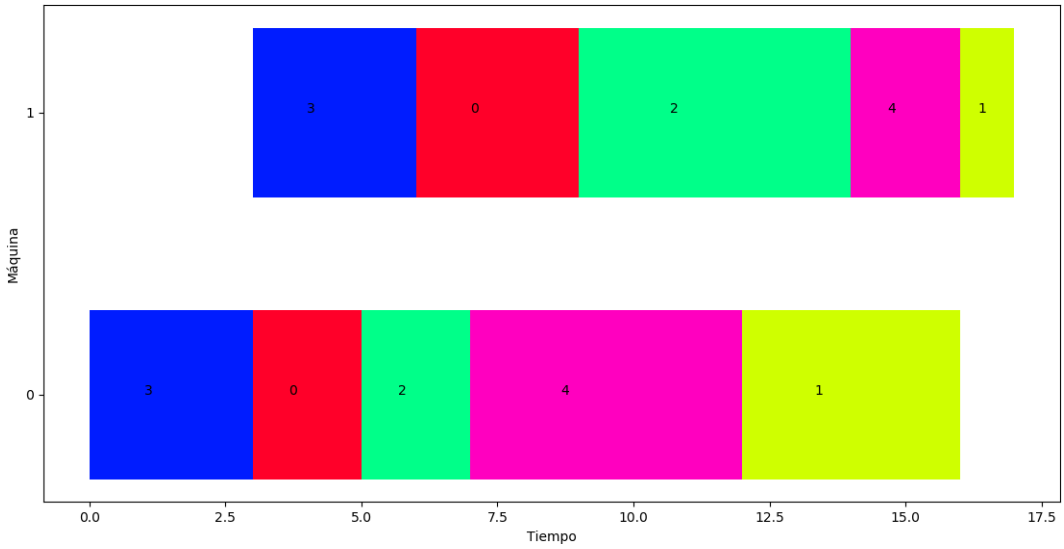


(b) Planificación obtenida al fijar las aristas disyuntivas como en 2.9. La ruta critica se resalta en amarillo

Figura 2.10: Modelo de grafo disyuntivo para la instancia de ejemplo 2.2



(a) Planificación aleatoria



(b) Planificación óptima

Figura 2.16: Ejemplo de planificación para una instancia en la cuál todos los trabajos empiezan en la misma máquina

Capítulo 3

Propuestas

En este capítulo se detalla el conjunto de propuestas algorítmicas ideadas y analizadas en esta tesis. En primer lugar, se propuso una versión de la búsqueda local iterada que usa un paisaje de búsqueda conformado a partir de componentes estándar. A continuación, se propusieron un conjunto de modificaciones para aliviar alguna de las dificultades de ese paisaje. En este capítulo se detallan cada una de las modificaciones realizadas, así como las ideas que hay tras cada una de las modificaciones.

3.1. Búsqueda local iterada

Dado que uno de los objetivos de esta tesis es utilizar técnicas algorítmicas relativamente simples, y sólo introducir cierta complejidad a través de la modificación del paisaje de búsqueda, se decidió utilizar la búsqueda local iterada. Actualmente el paisaje de búsqueda de la mayor parte de los métodos del estado del arte consiste en la representación directa basada en permutaciones, la vecindad N7 y el makespan como función de fitness. Este paisaje de búsqueda se toma entonces como la base para proponer modificaciones a cada una de sus componentes. Como se mostró anteriormente 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. Con el fin de evitar complicar 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). Este vecino es aceptado, independientemente de su nivel de calidad. Así, 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. Nótese que esta perturbación puede parecer muy pequeña, ya que habitualmente se incorporan perturbaciones bastante más disruptivas. Sin embargo, esta tesis está enfocada en obtener soluciones de calidad relativamente alta en tiempo

muy cortos. Algunos estudios iniciales con operadores más disruptivos no ofrecieron ventajas sobre la propuesta de moverse simplemente a un vecino, por lo que por simplicidad y eficiencia se decidió mantener esta decisión de diseño.

Por otro lado también es necesario establecer un criterio de paro. Teniendo en cuenta los objetivos de esta tesis, se fijó el criterio de paro a 5 minutos, lo cual representa una cantidad de tiempo muy pequeña comparada con la requerida por los métodos del estado del arte, que están del orden de varios días de ejecución.

3.2. Representación de soluciones activas basada en llaves aleatorias

La representación más usada actualmente basada en permutaciones puede restringirse a representar únicamente soluciones semi-activas. Si bien esto es una ventaja porque se reduce el tamaño del espacio de búsqueda es posible reducirlo aún más si nuestra representación considera únicamente soluciones activas. También es importante mencionar que las soluciones no activas, pueden contener periodos de tiempo en el que alguna máquina ociosa mientras espera que termine de ser procesada la dependencia de la próxima operación, es decir que pueden verse huecos en el diagrama de Gantt que pueden ser aprovechables por otras operaciones. Esto puede generar soluciones de baja calidad que pueden ser difíciles de mejorar.
Otra ventaja importante es que,como se explicó anteriormente, se sabe que entre las soluciones activas, existen soluciones óptimas. Por ello, se han ideado propuestas algorítmicas que permiten generar soluciones activas. En estas propuestas, hay que establecer un modo de elegir entre diferentes operaciones, para lo que se suele usar algún criterio de selección fijo, conocido como regla de prioridad. Esta regla puede ser elegir a la operación más corta o a la que pertenezca al trabajo más largo o incluso puede elegirse una al azar.
Para decodificar la solución a partir de la regla de prioridad normalmente se utiliza el algoritmo de Giffler & Thompson [20] con el cual se generan soluciones activas. A continuación se presenta este algoritmo.

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.

El primer paso es marcar como planificable la primera operación de cada trabajo. Posteriormente se identifica, de entre las operaciones planificables, 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. Nótese que si en esa máquina, se planificara a continuación una operación que empezase después de ese tiempo, no se tendría una solución activa. Por ello, se toma dicho tiempo de finalización $t_f^* = t_f(O_{min})$, y se escoge alguna de las operaciones planificables en M^* cuyo tiempo de inicio sea menor a t_f^* y se planifica. A continuación se actualizan las operaciones planificables y este proceso se repite hasta completar la planificación. Este proceso se muestra en el algoritmo 4.

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 alguna operación $O^* \in C$ mediante algún criterio de selección;
  Asignar tiempo de inicio y fin a O^*;
  Actualizar Ω eliminando a O^* y agregando a su sucesora si existe;

La representación propuesta en este trabajo se basa en asignar a cada operación una llave $k(O_i)$ dada por número real entre 0 y 1 el cual sirve para definir un orden entre operaciones en una misma máquina mediante un proceso de decodificación similar al planteado en [5, 31, 33] el cual consiste en usar el valor de esta llave como criterio de selección en el algoritmo 4. En concreto se elige a la operación que tenga asociada la llave más pequeña. Un punto negativo es que esta representación es n a 1 porque solo importa el valor relativo de las llaves que compiten entre sí, es decir qué diferentes valores de llaves pueden llevarnos a la misma solución. En principio no parece un problema muy grave, pero hay que tenerlo en cuenta a la hora de diseñar los operadores de movimientos, para no quedarse estancado en movimientos que vuelven a generar exactamente la misma solución.

Ejemplo

Como ejemplo ilustrativo se muestran los primeros pasos para construir la planificación mostrada en la figura 3.1 asociada a la instancia de ejemplo mostrada en la tabla 3.1.

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

Tabla 3.1: Instancia simple con 3 maquinas y 2 trabajos

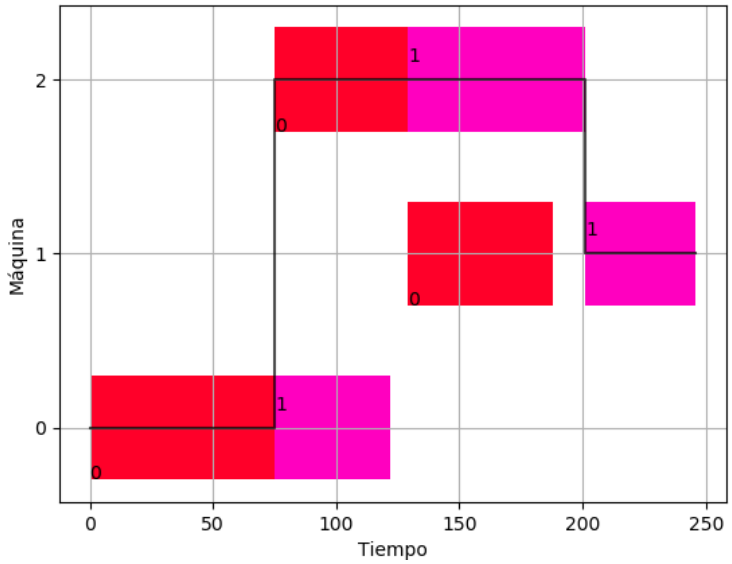


Figura 3.1: Diagrama de Gantt de una planificación posible para la instancia mostrada en la tabla 3.1. Se marcan los trabajos que pertenecen a la ruta crítica con una línea

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 eliminando 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

Realizar una búsqueda a partir de soluciones activas tiene la ventajas de moverse entre soluciones con una calidad media superior. Sin embargo, el proceso de generar las soluciones es costoso, por lo que tiene el inconveniente de analizar una menor cantidad de soluciones candidatas, con respecto al caso en que no se aseguran que las soluciones son activas. Por ello, es posible que utilizar un esquema en que se usen ambos tipos de representaciones de forma simultánea sea ventajoso. Con este fin, de diseñó una forma para movernos movernos entre soluciones de ambas representaciones, en concreto, para generar una solución activa y su representación basada en llaves, a partir de una solución previa representada por una permutación que no tiene que ser activa.

Para hallar la representación de alguna solución representada por una permutación se le asigna a cada operación una prioridad decreciente acuerdo a su posición de forma que se cumpla:

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

donde O_i es la operación número i en la representación basada en permutaciones

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 3.2: Instancia 3 maquinas y 4 trabajos

La figura3.2 muestra un diagrama de Gantt para una posible planificación semi-activa.

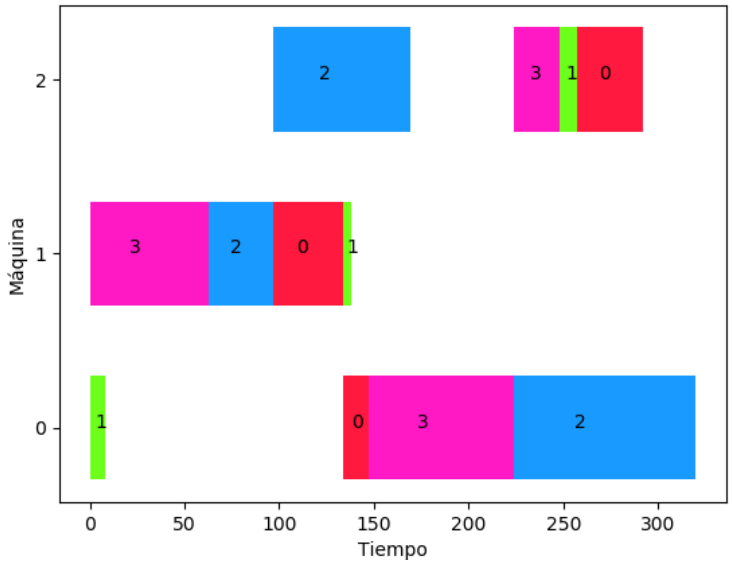


Figura 3.2: Planificación posible para la instancia en 3.2.

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 figura 3.3.

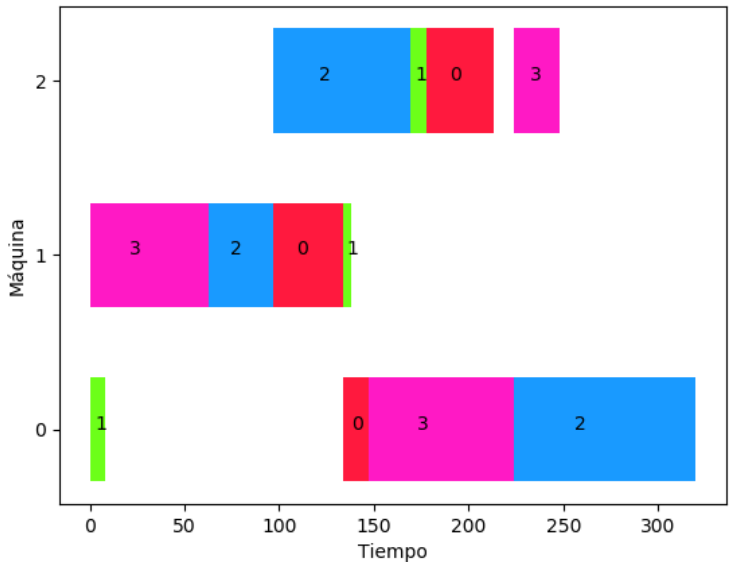


Figura 3.3: Planificación activa reconstruida a partir de la mostrada en la figura 3.2

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

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.

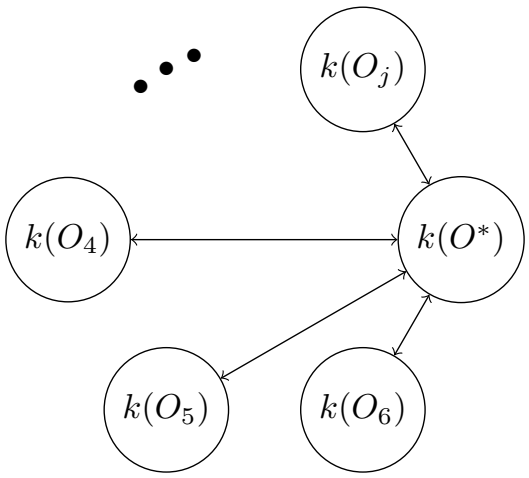


Figura 3.4: Movimientos de la vecindad propuesta. $k(O^*)$ es la llave de la operación elegida, las flechas representan los posibles intercambios entre las j operaciones competidoras.

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. Estas ideas podrían dar lugar a esquemas de reducción de vecindad, tratando de eliminar cambios que sean muy disruptivos; sin embargo, dado que las vecindades que surgen para los casos tratados no son demasiados grandes se manejó la vecindad de forma completa.

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 nunca llevarían a una mejora [7]. Los movimientos planteados se basan en observar que en alguna solución encontrada por una búsqueda local para cada máquina pueden existir periodos de tiempo en la que está inactiva (huecos) 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.

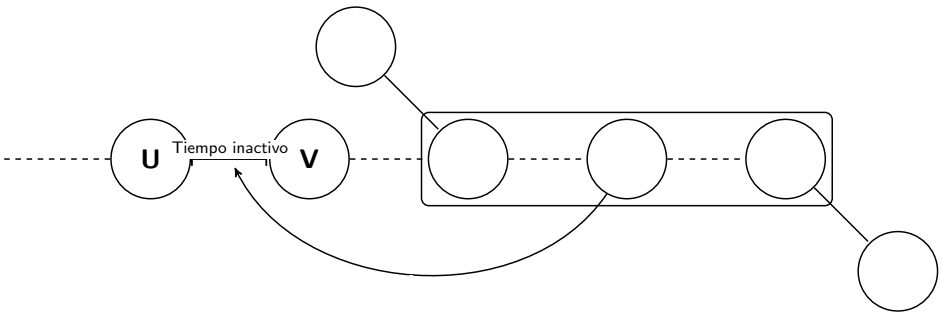


Figura 3.5: Movimientos propuestos. El tiempo de inicio de **v** es mayor al de finalización de **u**

En general esta vecindad no extiende el número de vecinos visitados en una gran cantidad porque existen muchos de estos 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 convierte en uno de optimización multi-objetivo [21, 36, 32]. En [44] se propuso optimizar el makespan pero tomando en en cuenta exclusivamente el tiempo inactivo en la planificación aunque los resultados no fueron muy alentadores.

La propuesta presentada en este trabajo consiste en construir un arreglo de características para cada planificación de modo que puedan compararse lexicográficamente. La idea principal es tener

una forma de distinguir entre soluciones con el mismo makespan, lo que puede ayudar a atravesar regiones que de otro modo serían «planas» y que para una búsqueda local representarían un punto de paro. En el mejor de los casos la adición de estas cantidades no solo nos permitiría atravesar planicies sino que también serviría como guía para mantener una planificación que no sea propensa a estancarse prematuramente en un óptimo local de mala calidad.

Para plantear las características para este arreglo se tomaron en cuenta otros criterios de optimalidad hallados en la literatura así como propuestas propias que se hallaron empíricamente. También se planteó considerar todos los tiempos de finalización de las máquinas ordenados de mayor a menor. 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, especialmente si es una de las máquinas que tardaba más en terminar.

- $\sum \mathbf{J_i}$: También conocido como *Flowtime*, mide los tiempos de finalización de los trabajos y 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. Conforme aumenta el número de máquinas que cumplen esta condición, parece 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, aunque eso no implique reducir el tiempo final de alguna máquina, y así 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 i 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. Se considera el uso de búsqueda local iterada con el makespan como función de fitness, la estructura de vecindad N7 y la representación basada en permutaciones como caso base para comparar contra los cambios propuestos.

4.1. Organización de los resultados

Como se explicó en los capítulos anteriores, las propuestas planteadas en este trabajo son de dos tipos: modificar el paisaje de búsqueda del problema y cambiar la metaheurística con la que se explora este paisaje. En el caso de la metaheurística la única modificación es usar la búsqueda local iterada (ILS). Para el caso del paisaje de búsqueda se presentan varias modificaciones a cada una de sus componentes: representación de soluciones, estructura de vecindad y función de fitness.

Los resultados de aplicar las modificaciones a cada una de las componentes se muestran de modo que las modificaciones se van agregando para mejorar el desempeño del algoritmo. Primero se presenta 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 junto con la nueva vecindad propuesta. El orden de presentación es el siguiente (se asignan acrónimos para facilitar la comparación):

1. **PN7Cmax** Representación basada en permutaciones, vecindad N7 y makespan como función de fitness.

2. **PN7*** Representación basada en permutaciones, vecindad N7 y diversas funciones de fitness.

3. **PN7extTup** Representación basada en permutaciones, vecindad N7 extendida y tupla ordenada de tiempos de finalización como función de fitness.

4. **RpKeTup** Representación basada en llaves aleatorias, vecindad propuesta y tupla ordenada de tiempos de finalización función de fitness.

Cada una de las variaciones propuestas se ejecutó durante 5 minutos por 50 veces para cada instancia del conjunto **DMU01-80** con el fin de obtener un conjunto de resultados para comparar.

Puesto que el objetivo de las modificaciones planteadas en este trabajo es plantear una alternativa a los métodos del estado del arte, se presenta la mediana del error relativo respecto al mejor resultado reportado en la literatura. El error relativo se calcula de la siguiente manera

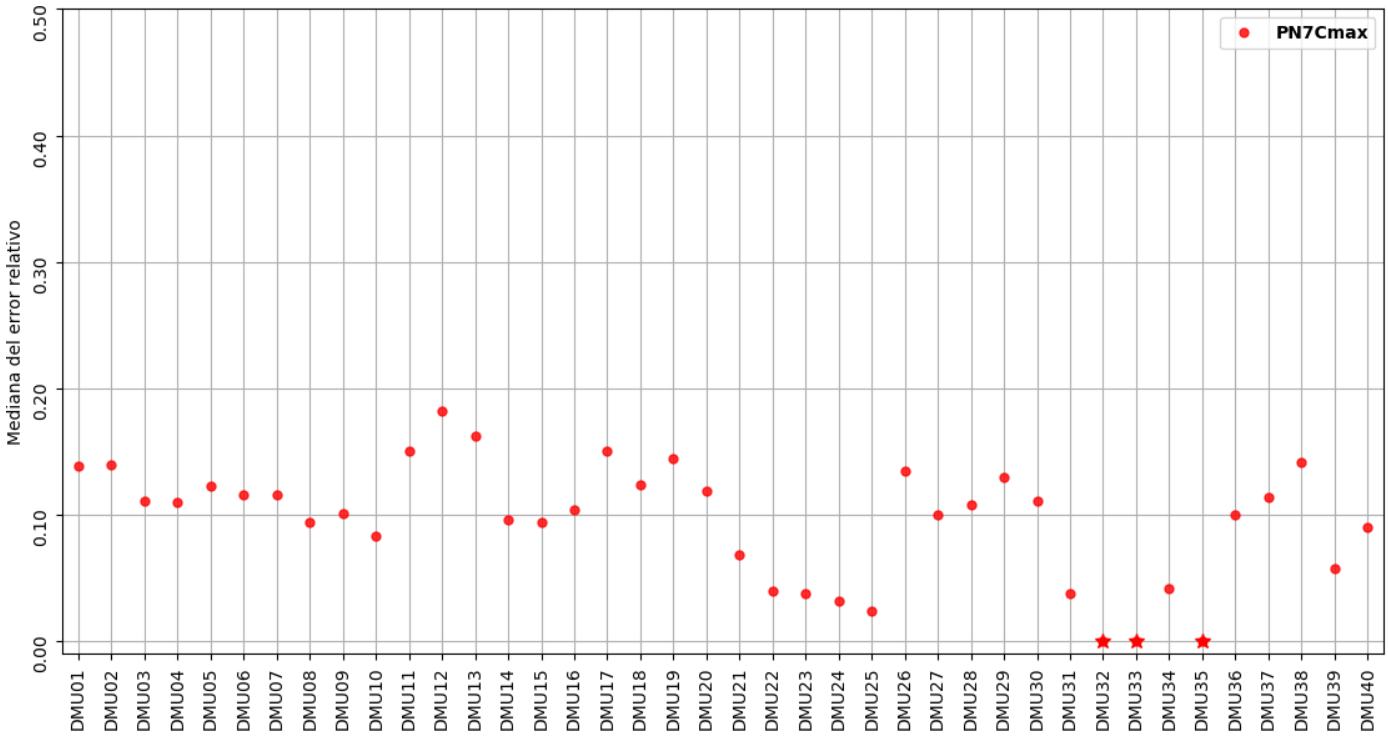
$$\text{error relativo} = \frac{\text{mejor makespan en la literatura} - \text{makespan de la solución encontrada}}{\text{mejor makespan en la literatura}}$$

El mejor resultado reportado en la literatura para cada instancia se muestra en el apéndice A.1.

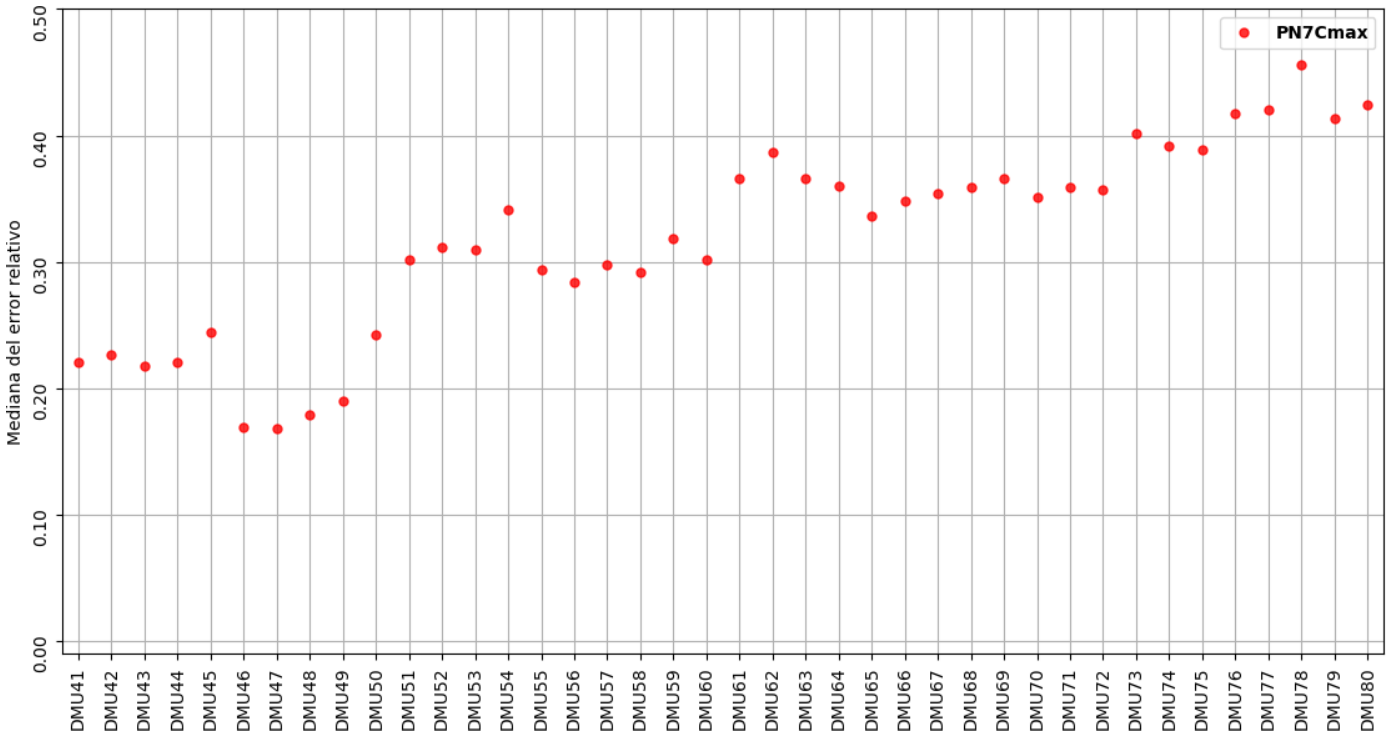
4.2. Resultados para PN7Cmax

Estos resultados sirven como una base para determinar si las modificaciones posteriores resultan en mejoras apreciables puesto que el único cambio con los métodos más extendidos en la literatura es el uso de la búsqueda local iterada.

Se muestran los resultados de manera gráfica para facilitar su visualización. Los resultados detallados se encuentran en el apéndice A.1.



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



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

Figura 4.1: Resultados para **PN7Cmax**. Se marcan los casos en los que se llegó a la mejor solución conocida.

En la figura 4.1 podemos observar que la mediana del error relativo para la segunda mitad de las instancias es considerablemente mayor que para la primera mitad en varios casos incluso llegando a igualar el mejor resultado conocido.

4.3. Resultados para distintas funciones de fitness

La función de fitness se obtiene al construir la dupla formada por el makespan y alguna característica en ese orden. Para determinar cuál es la que obtiene mejores resultados las modificaciones se comparan a pares en cada instancia. 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. 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.

Los acrónimos asignados a cada una de las características propuestas para se muestran en la tabla 4.1.

Los resultados para las propuestas mostradas en 3.5 pueden verse de manera condensada en la figura 4.2. La casilla (i, j) muestra el número de veces que la variante i fue mejor menos el número veces que fue peor que la variante j.

Se muestra también una tupla construida con las características que obtuvieron mejores resultados presentada como **PN7C2/Flow/VarC**. En la tabla 4.2 se presentan ordenadas las variantes por

Característica	Acrónimo
$\mathbf{C_{max}}$ (makespan)	**Cmax**
$\sum \mathbf{C_i^2}$	**C2**
$\sum \mathbf{J_i}$	**Flow**
$\sum \mathbf{I(C_i = C_{max})}$	**Icmax**
Número de rutas críticas	**Rc**
$\mathbf{Var(C_i)}$	**VarC**
$(\{\mathbf{C_i}\})$	**Tup**

Tabla 4.1: Acrónimos asignados a cada característica

el puntaje obtenido. Podemos observar que las primeras tres entradas están relativamente cerca aunque se ven diferencias claras entre ellas.

Podemos observar en la figura 4.2 y en la tabla 4.2 que el makespan fue la peor función de fitness. Para la tupla construida con las mejores características se observan pocas mejoras por lo que podemos concluir que agregar más características no lleva necesariamente a mejores resultados. Los mejores resultados se obtienen al construir la tupla de tiempos ordenados de finalización de las máquinas 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 esta función de fitness se muestran en el apéndice A.2. Con estos resultados podemos concluir que es ventajoso enfocarse en mejorar las máquinas que tardan más tiempo siendo las tres mejores variantes las que toman más en cuenta estas máquinas.

Variante	Puntaje
PN7Tup	264.0
PN7C2/flow/VarC	257.0
PN7C2	252.0
PN7Flow	209.0
PN7VarC	142.0
PN7Rc	-281.0
PN7Icmax	-382.0
PN7Cmax	-461.0

Tabla 4.2: Puntaje para cada una de las variantes de función de fitness propuesta

En la figura 4.3 se muestra una comparación de los resultados hallados para la mejor función de fitness **PN7Tup** contra el caso base que también fue el que obtuvo peores resultados **PN7Cmax**.

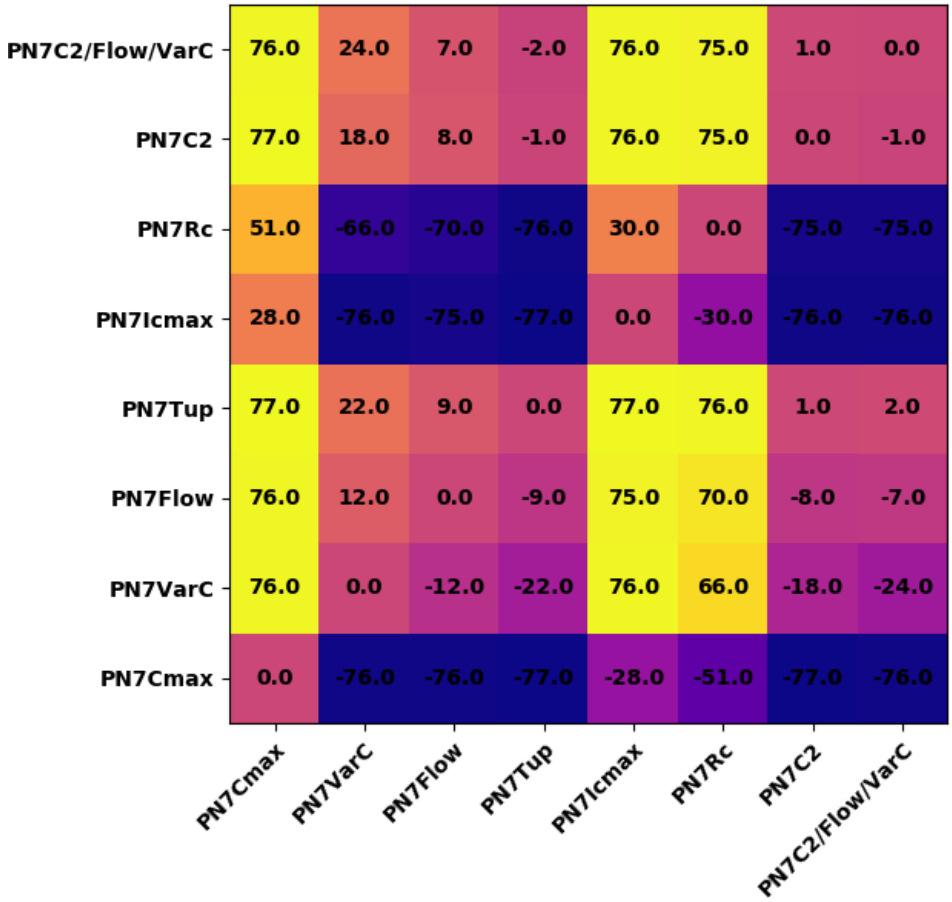
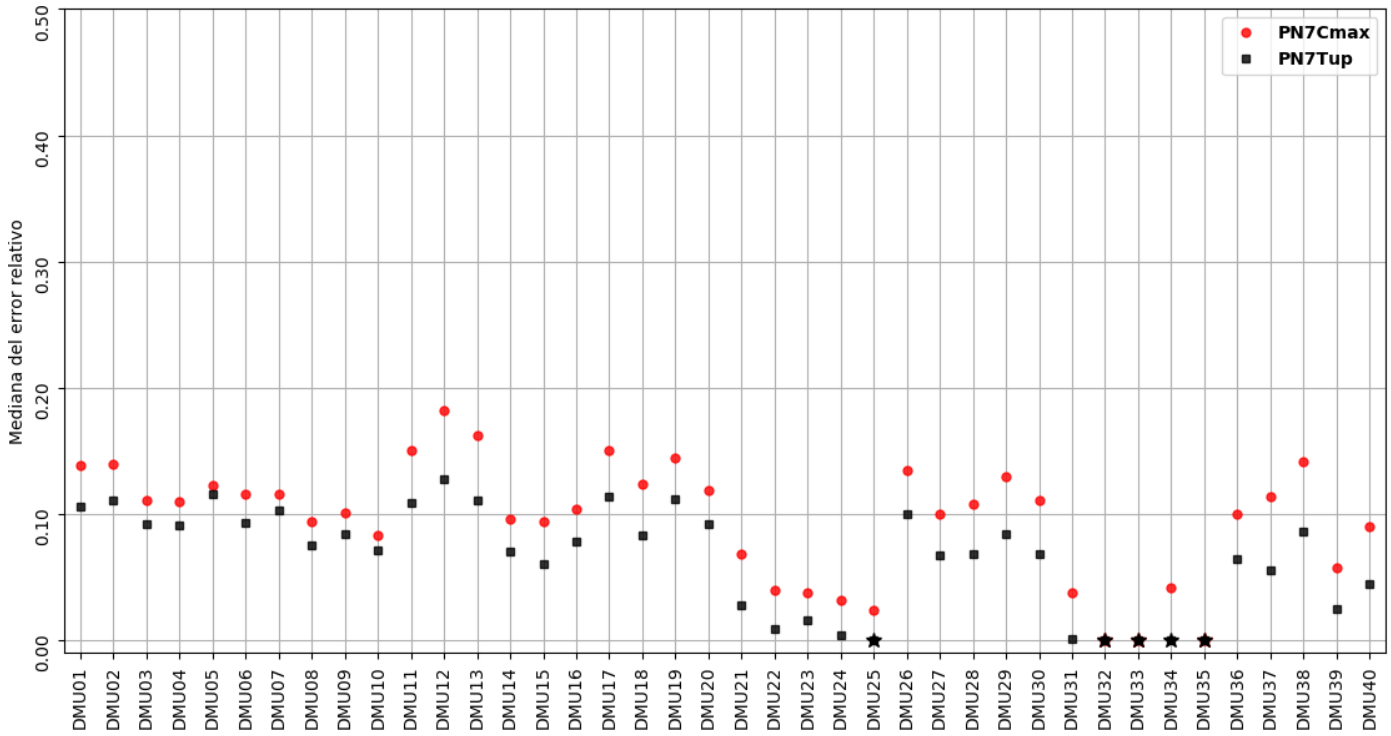
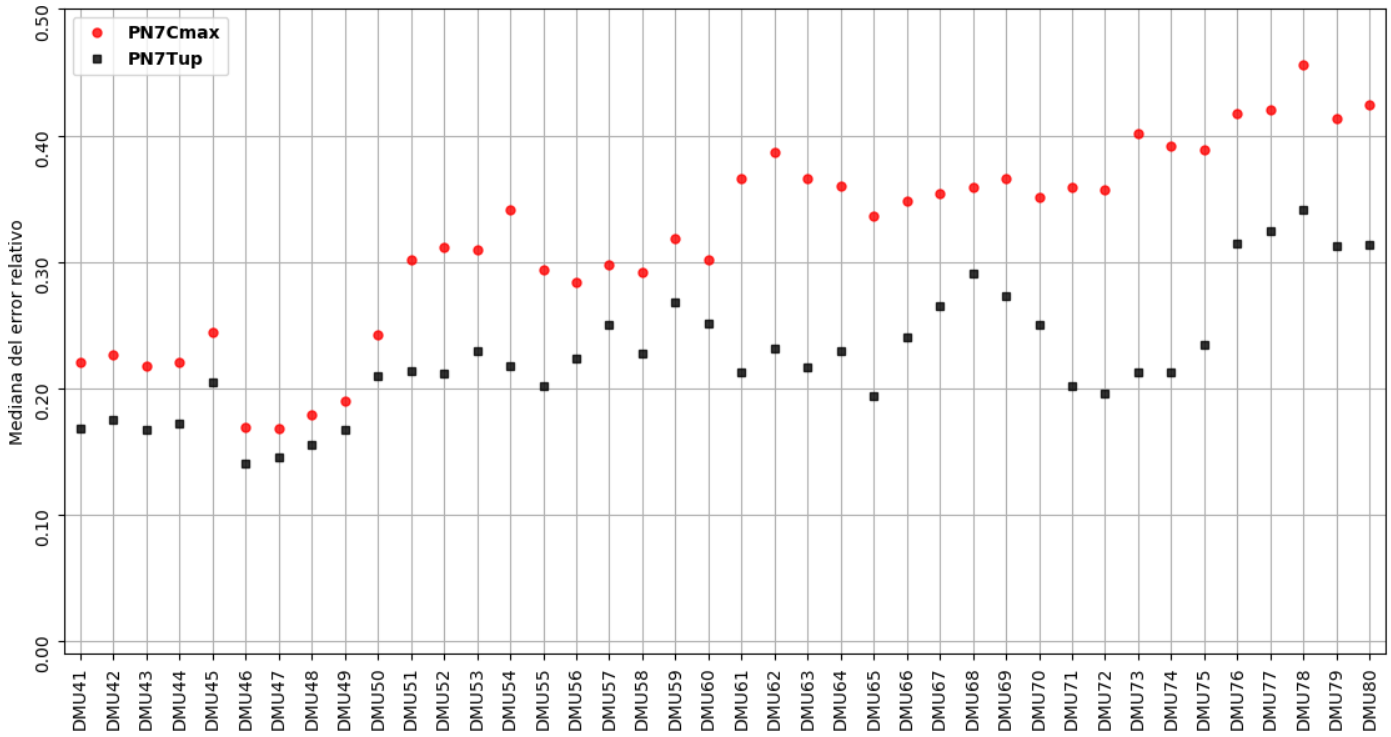


Figura 4.2: Condensado de los resultados para las modificaciones a la función de fitness.

Se pueden apreciar mejoras sustanciales con solo la adición de la función de fitness.



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



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

Figura 4.3: Comparación entre la peor y la mejor función de fitness. Se marcan los casos en los que se llegó a la mejor solución conocida.

4.4. Resultados para PN7extTup

Los resultados para la extensión que considera movimientos con espacios de inactividad de las máquinas se comparan con los mejores mostrados en la sección pasada **PN7Tup**. Se sigue el mismo procedimiento que en la sección anterior para hacer esta comparación. En la figura 4.4 podemos ver que **PN7extTup** no representa una mejora drástica. Aunque el puntaje obtenido por esta modificación fue positivo, en la figura 4.5 podemos observar que la mejora no es consistente por lo que se puede concluir que la extensión de la vecindad planteada no repercute en beneficios importantes.

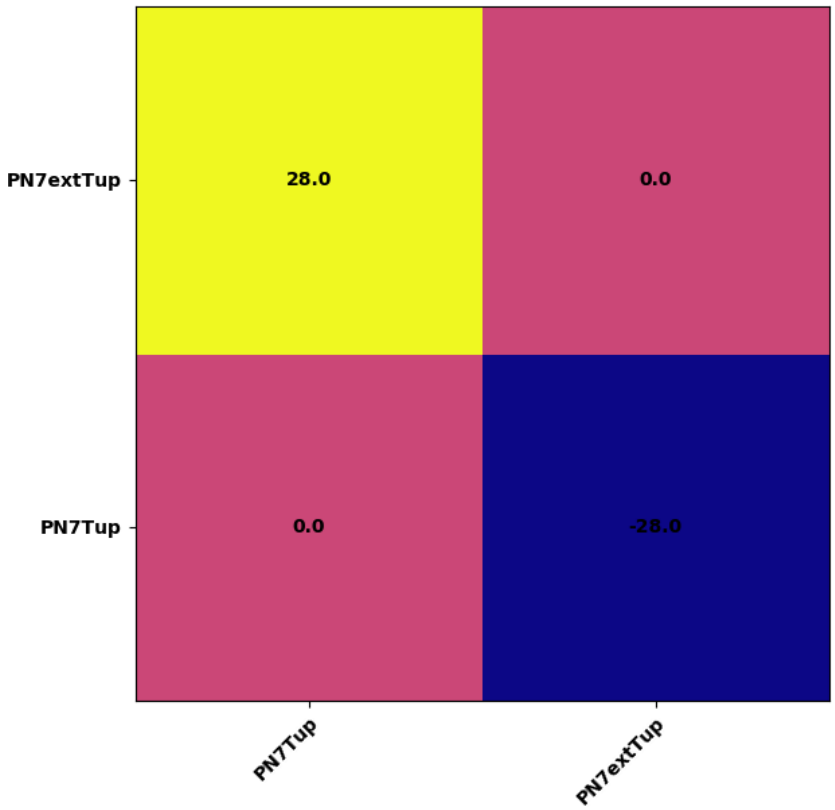
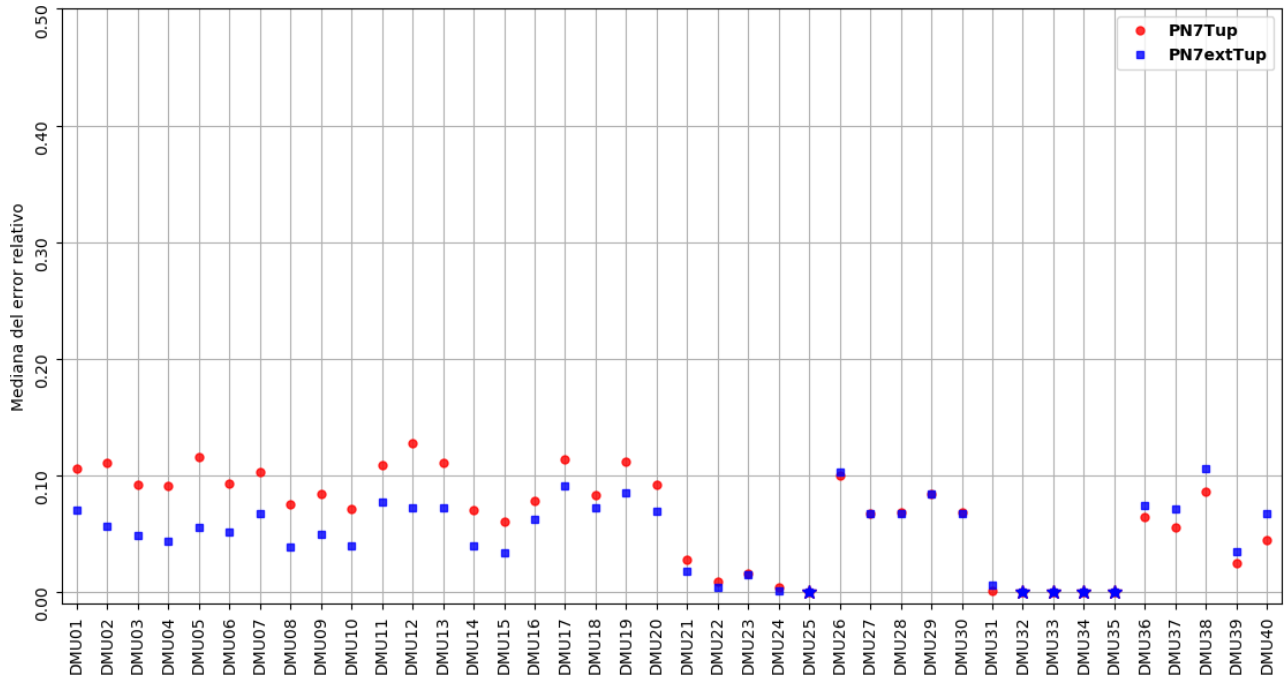
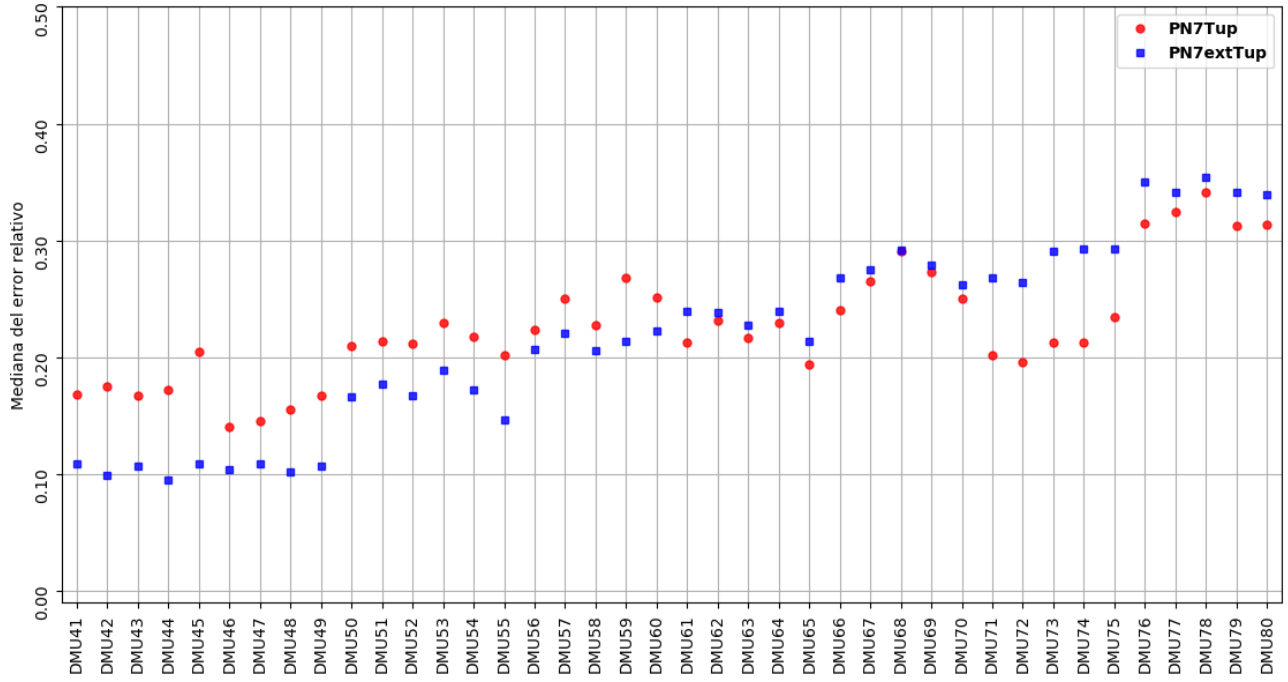


Figura 4.4: Resultados de la comparación

También se muestra de manera gráfica los resultados para todas las instancias junto con los mejores resultados de la sección anterior. Los resultados detallados para la extensión propuesta se encuentran en el apéndice A.3.



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



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

Figura 4.5: Resultados para los métodos **PN7extTup** y **PN7Tup**. 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. Los resultados se muestran en la figura 4.6

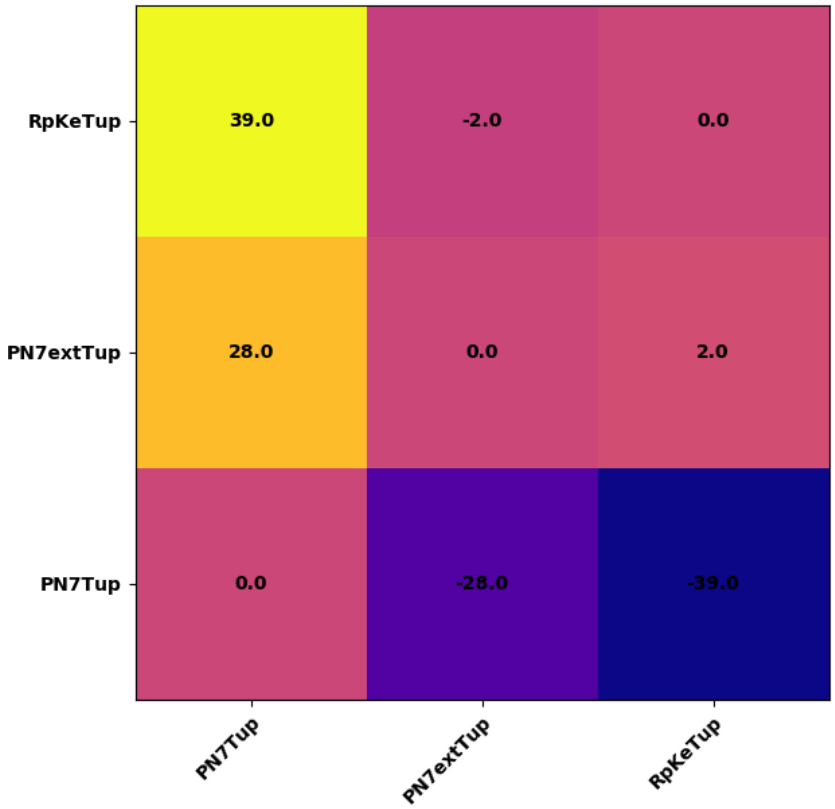
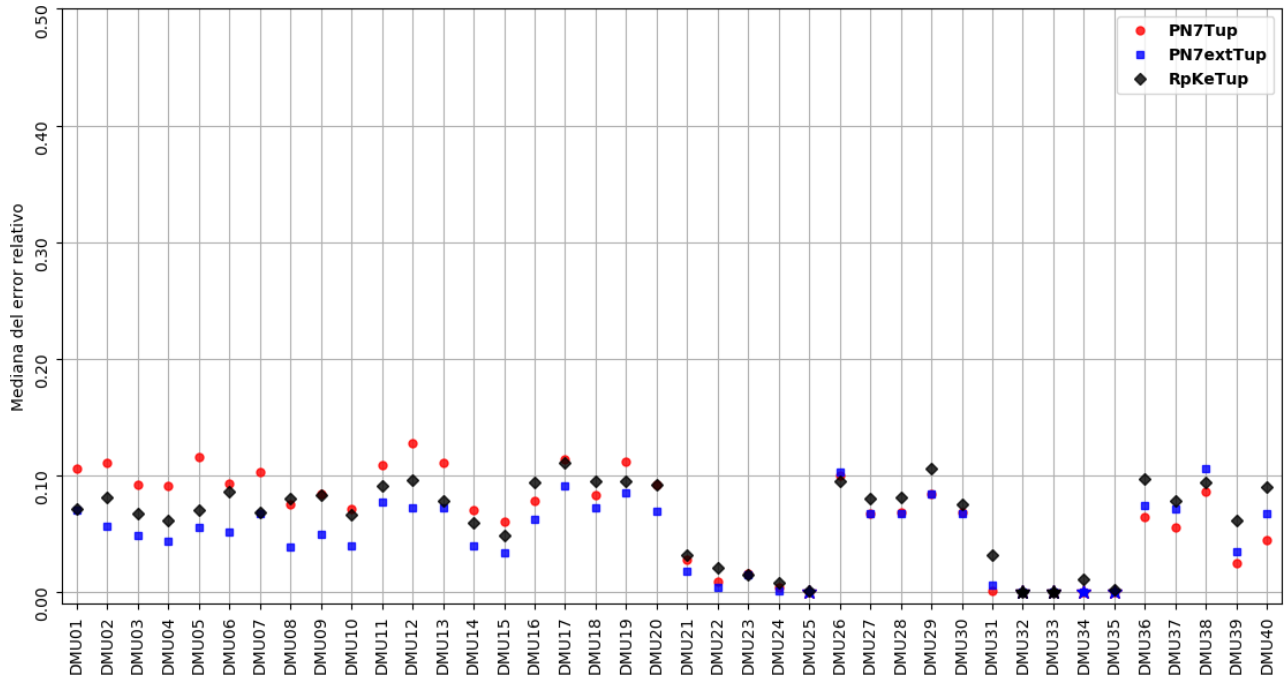


Figura 4.6: Resultados de la comparación entre los tres métodos.

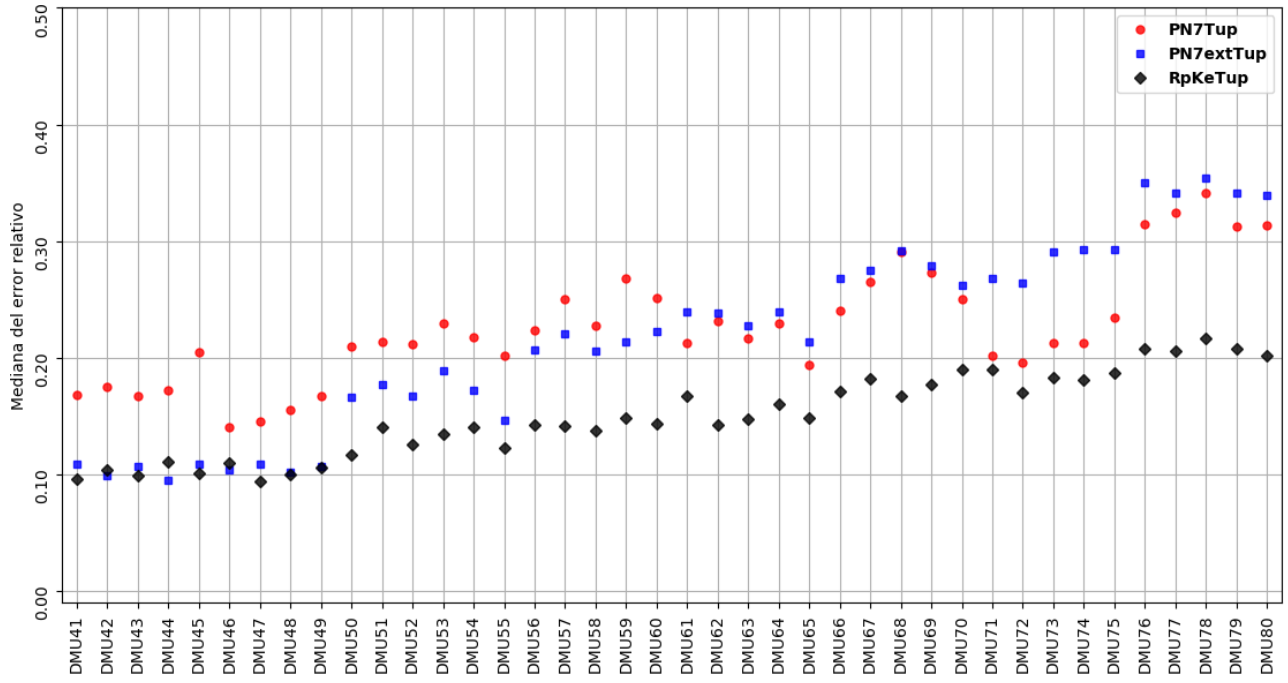
Variante	Puntaje
RpKeTup	37.0
PN7extTup	30.0
PN7Tup	-67.0

Tabla 4.3: Puntaje para cada una de las variantes de función de fitness propuesta

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 4.7: Resultados para **PN7Tup**, **PN7extTup** y **RpKeTup**. Se marcan los casos en los que se llegó a la mejor solución conocida.

Para resaltar las diferencias entre **RpKeTup** y **PN7Tup** 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 como se muestra en la figura 4.8. Este procedimiento nos permite hacernos una idea de cómo se conectan las soluciones. Lo más

deseable es que las soluciones de alta calidad tengan vecindades grandes y las de mala calidad tengan vecindades pequeñas. En la figura 4.8 podemos ver se tiene lo opuesto de lo deseado, es decir que el tamaño de la vecindad de una solución decrece junto con el makespan. Este efecto es más marcado para **RpKeTup** aunque cabe mencionar que en general el tamaño de sus vecindades es mucho más grande que el de las de **PN7Tup**. El patrón bandeado que se observa en la figura 4.8 para **PN7Tup** posiblemente se deba a la dependencia que tiene el tamaño de la vecindad de una solución con su número de rutas críticas ya que en **RpKeTup** esto no se observa.

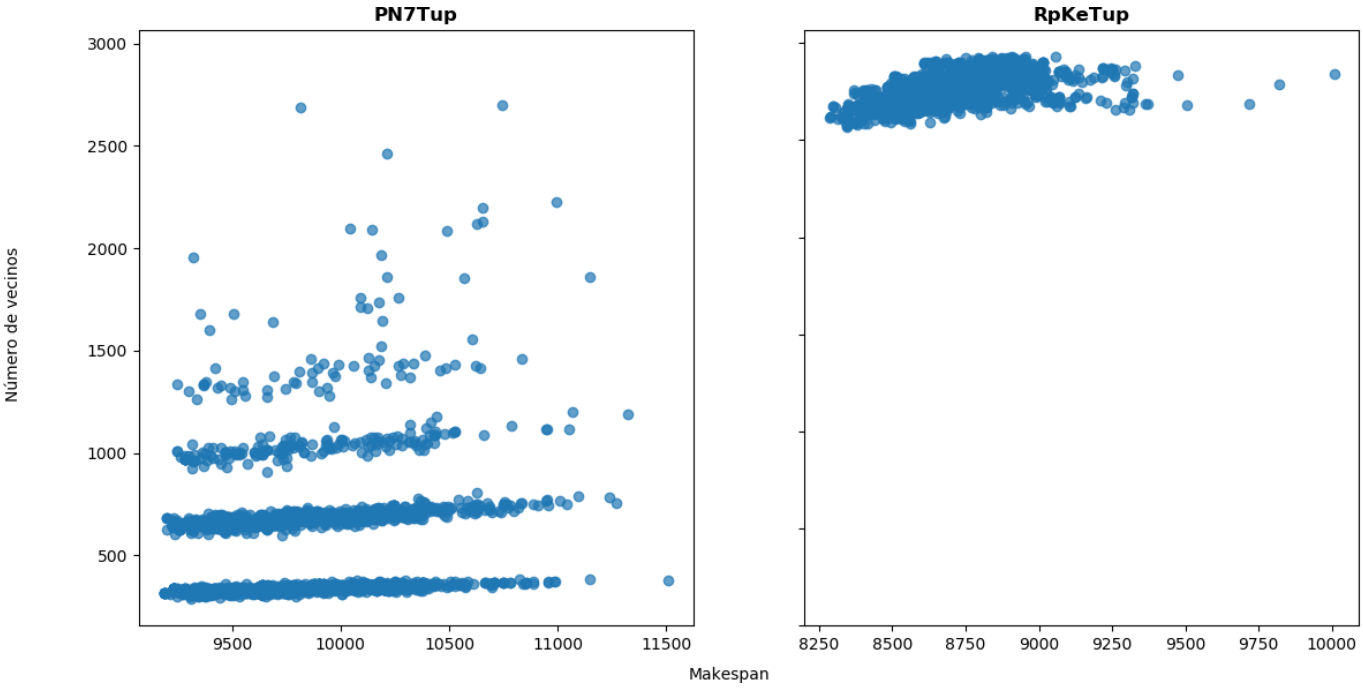
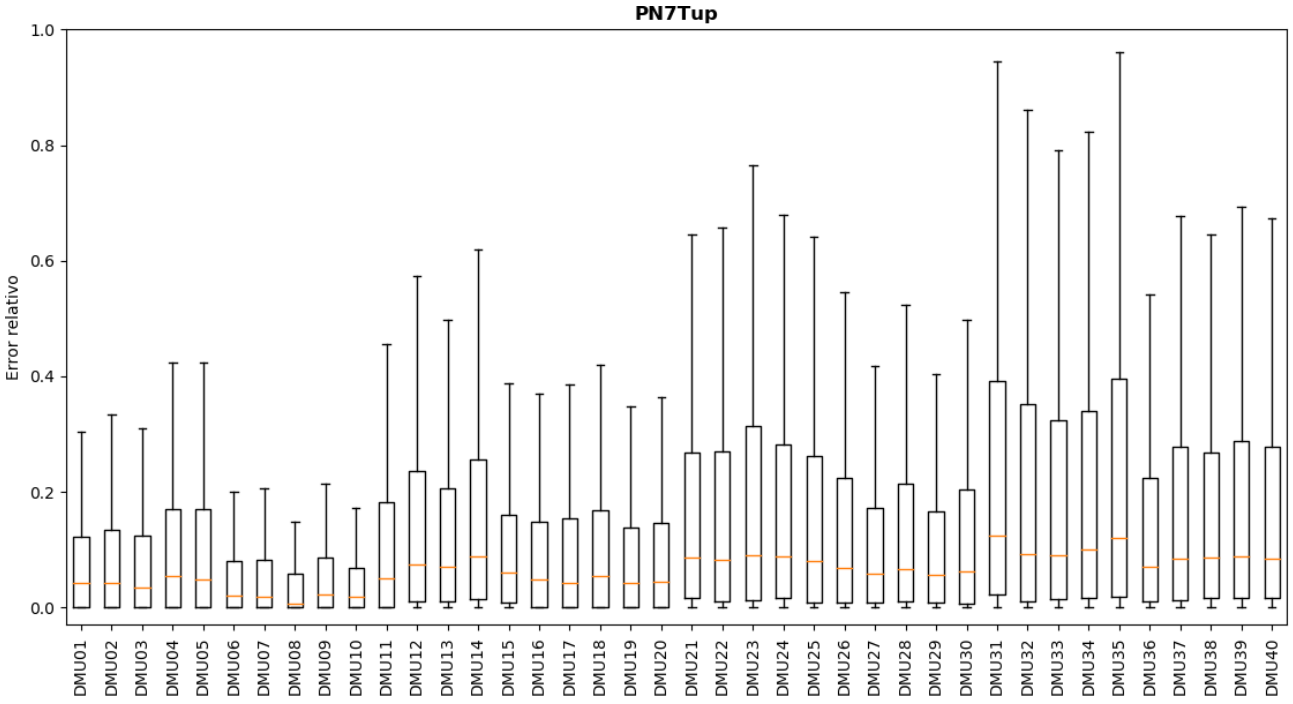


Figura 4.8: Comparación de tamaño de la vecindad contra makespan de los óptimos locales para la instancia **DMU78**

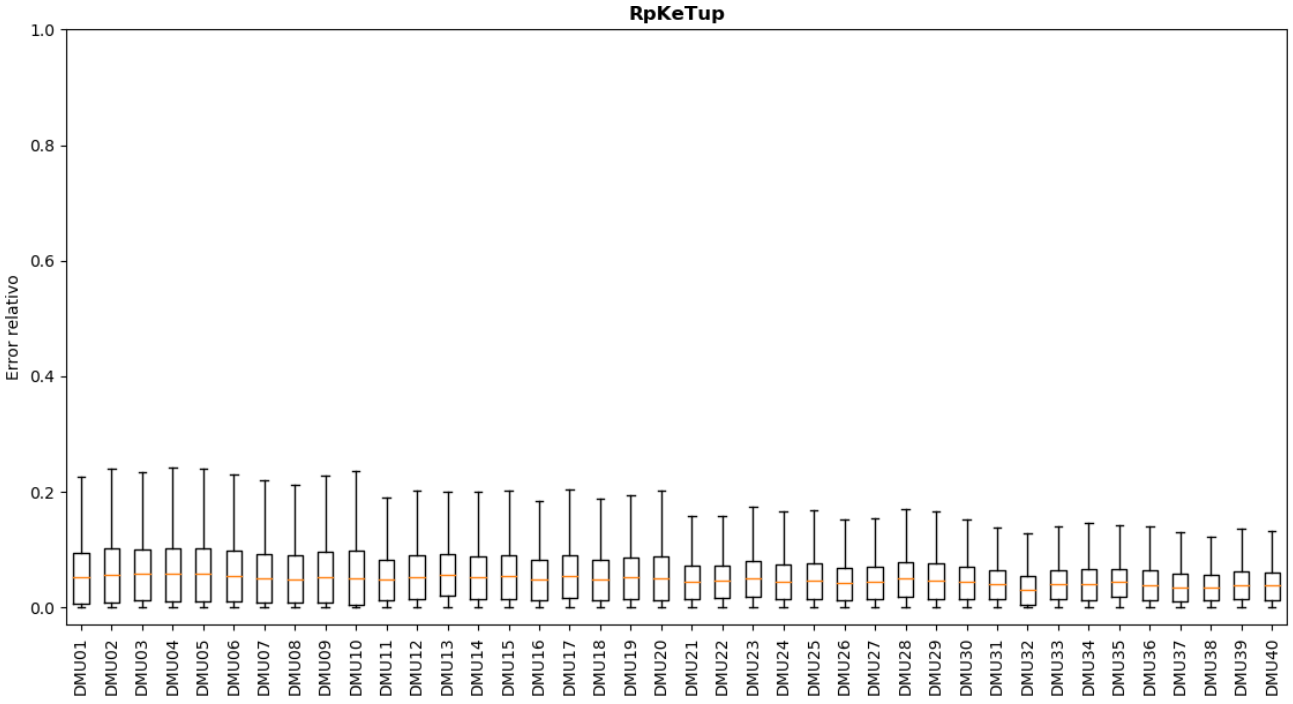
Otro modo de visualizar las diferencias entre ambas representaciones también se presentan diagramas de caja para la diferencia relativa relativa del makespan de los óptimos locales con sus vecinos. Esto nos da una idea de cómo se relacionan las soluciones de acuerdo con su makespan. Si estos valores son muy cercanos entre sí esto nos da un indicio de la suavidad del paisaje de búsqueda.

En las figuras 4.9 y 4.10 observamos que los vecinos de los óptimos locales para **RpKeTup** son más parecidos que los de **PN7Tup**, esto nos dice que el paisaje de búsqueda es más suave lo cual como se ha mencionado antes es ventajoso para las metaheurísticas de trayectoria.

Este efecto se debe a que la representación basada en permutaciones considera movimientos que pueden ser bastante disruptivos como atrasar toda una cadena de operaciones dejando huecos grandes en la planificación, mientras que en la nueva representación basada en llaves aleatorias el proceso de decodificación mediante el algoritmo de Giffler Thompson 4 evita que se generen estos huecos lo que tiene como resultado un paisaje de búsqueda mucho más suave.

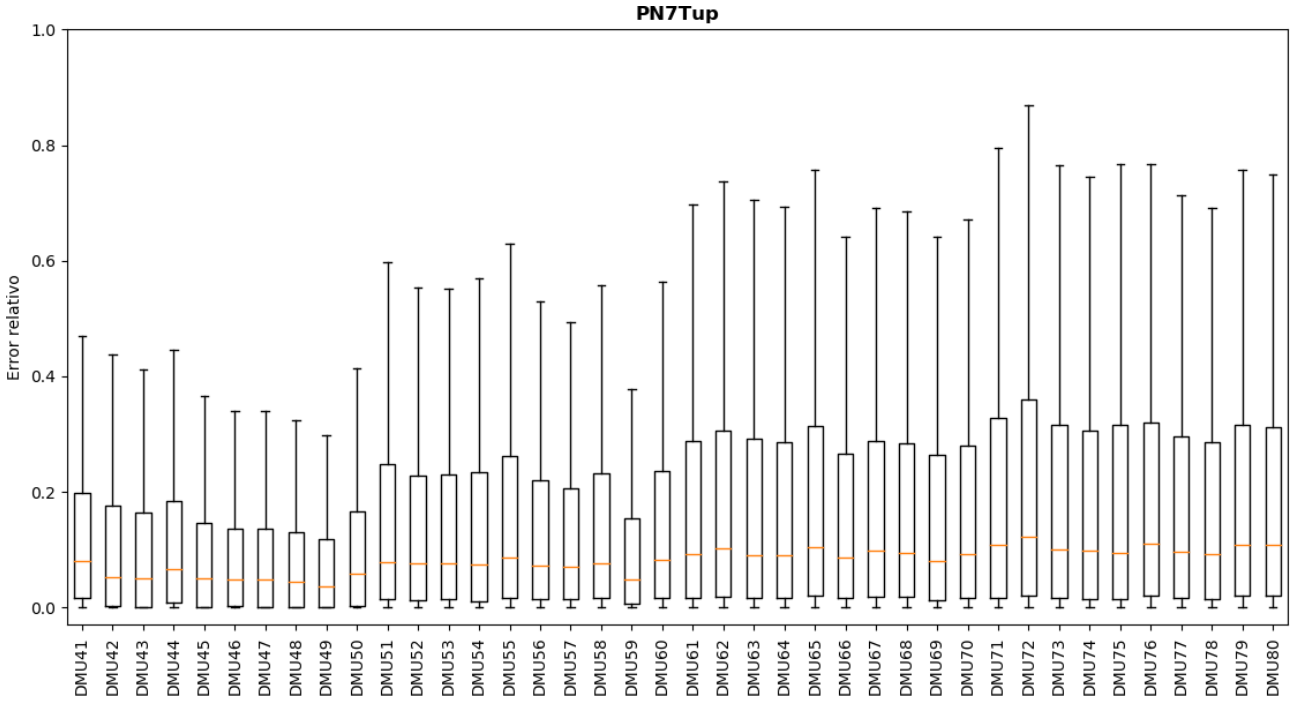


(a) Representación original con vecindad n7 instancias **DMU01-40**

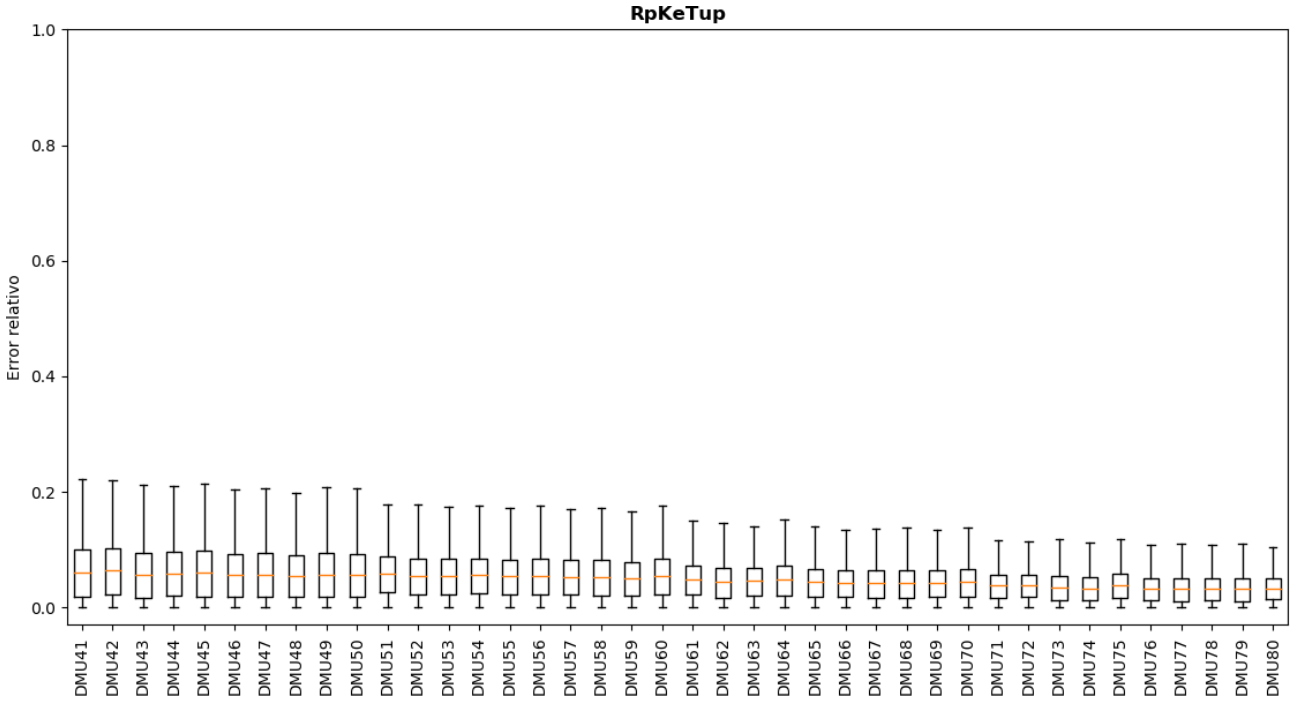


(b) Representación propuesta instancias **DMU01-40**

Figura 4.9: Diagramas de caja de la diferencia relativa en makespan de un óptimo local con sus vecinos



(a) Representación original con vecindad n7 instancias **DMU41-80**



(b) Representación propuesta instancias **DMU41-80**

Figura 4.10: Diagramas de caja de la diferencia relativa en makespan de un óptimo local con sus vecinos

Capítulo 5

Conclusiones y Trabajo a Futuro

El problema de planificación tipo taller (JSP) es un problema de optimización combinatoria **NP-hard** el cual ha recibido mucha atención desde hace varias décadas por su aplicación en ambientes de manufactura. Debido a la dificultad de este problema, no se conocen algoritmos eficientes que puedan resolverlo de manera general, es por ello que durante los últimos años, se han desarrollado múltiples optimizadores para el JSP basados en métodos aproximados que han incrementando cada vez su complejidad, tanto en implementación como en mantenimiento.

En la actualidad, los optimizadores que ofrecen los mejores resultados son técnicas híbridas que combinan múltiples ideas de muy diferente índole y requieren de grandes cantidades de recursos computacionales. Los mayoría de los optimizadores actuales se basan en la idea de plantear un modo de explorar un paisaje de búsqueda que se construye con tres componentes: la representación que se le da a las soluciones, una estructura de vecindad que permite generar un conjunto de soluciones nuevas a partir de una conocida y una función de fitness que permite comparar las soluciones. Aunque se han usado diferentes tipos de representaciones, vecindades y funciones de fitness, gran parte de los esquemas actuales consideran aspectos comunes, y en particular, suelen usar la vecindad conocida como N7, una representación basada en permutaciones y el makespan como función de fitness.

En esta tesis se propone un conjunto de modificaciones a cada una de las componentes del paisaje de búsqueda con el fin de analizar qué efecto tienen los mismos sobre el rendimiento. Para explorar los paisajes resultantes de las modificaciones se utiliza una metaheurística sencilla, la búsqueda local iterada (ILS), con el fin de obtener soluciones de alta calidad en tiempos reducidos. Se realizan cambios en términos de XX, YY y ZZ. Los resultados muestran que lo que más efecto tiene es la representación. En particular, ... Por otro lado, la función de fitness también es bastante importante. En particular, debido a la existencia de múltiples zonas planas cuando solo se usa la función de fitness, el optimizador se queda bloqueado en óptimos de baja calidad. Sin embargo, si se extiende para ofrecer direcciones de mejorar cuando el makespan no permite dirigir la búsqueda, se obtienen soluciones de mejor calidad. Se probaron varias formas de extender la función de fitness y en particular, la que obtuvo mejor rendimiento consiste en... En lo que se refiere a la extensión

de la vecindad, ...

Luego puedes hablar de cómo esto genera nuevas opciones.

Nótese que en esta tesis, el paisaje de búsqueda sólo se integró con la metaheurística búsqueda local iterada, lo que permitió generar buenas soluciones en tiempos reducidos. Dadas las importantes mejoras obtenidas, se abre la puerta a incluir estos cambios con otras metaheurísticas de mayor complejidad y con metaheurísticas híbridas. Esto es un reto importante porque en su mayor parte, estas metaheurísticas incluyes pasos dependientes de la representación como cálculo aproximado de fitness, prohibiciones de ciertos tipos de movimientos, etc. por lo que no es para nada trivial realizar la integración, aunque parece muy prometedor.

Los resultados obtenidos muestran una clara diferencia en cuestión de dificultad entre las instancias **DMU01-40** y las **DMU41-80**. La búsqueda local iterada con la vecindad N7 y la función de fitness propuesta muestra un desempeño muy diferente en estas 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 crece con el tamaño de la instancia considerada.

La propuesta de extensión para la vecindad N7 no mostró ventajas importantes frente a la vecindad original y muestra el mismo comportamiento a pesar de plantearse como una posible manera de escapar de óptimos locales y de mantener soluciones más prometedoras.

La función de fitness construida con las características presentadas muestra que es benéfico considerar otros factores de la planificación a demás del makespan para obtener mejores resultados. Aunque sí se tienen mejoras con la adición de estas característica es posible que exista una mejor manera de construir la función de fitness por ejemplo con una suma ponderada aunque eso implica introducir y determinar los pesos a usar para cada una de ellas lo cual puede complicar de más el algoritmo.

El cambio de representación tuvo en gran efecto positivo en el desempeño de la búsqueda local. Este cambio logra una disminución sustancial del error relativo en la segunda mitad de las instancias. En esta nueva estructura de vecindad el tamaño de la misma está mucho más relacionado con el makespan de la solución, esto sugiere que las mejores soluciones están poco conectadas haciéndolas más difíciles de obtener. Un aspecto positivo de la nueva estructura de vecindad es que a pesar de presentar esta clara correlación entre makespan y conectividad, el tamaño de la misma no fluctúa tanto como sucede con la vecindad N7 a demás de que es más grande por lo que los problemas derivados de esta correlación no son muy apreciables. Por último, la nueva vecindad muestra una mayor suavidad en el paisaje de búsqueda esto también implica que las soluciones tienden más a agruparse con soluciones de calidad similar, esta característica del paisaje de búsqueda puede aprovecharse en el futuro.

La modificación que tuvo el mayor impacto positivo fue la introducción de la nueva representación junto con la nueva estructura de vecindad. Las propuestas aquí presentadas indican posibles modificaciones a los métodos actuales o un punto de partida para proponer algunos nuevos.

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 para vecindad N7 con tupla como fitness

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

[2] Applegate, D., and Cook, W. A computational study of the job-shop scheduling problem. *ORSA Journal on computing 3*, 2 (1991), 149–156.

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

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

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

[6] 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.

[7] 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.

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

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

[10] 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.

[11] Carlier, J., and Pinson, É. An algorithm for solving the job-shop problem. *Management science 35*, 2 (1989), 164–176.

[12] 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.

[13] 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.

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

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

[16] Demirkol, E., Mehta, S., and Uzsoy, R. Benchmarks for shop scheduling problems. *European Journal of Operational Research 109*, 1 (1998), 137–141.

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

[18] Fisher, H. Probabilistic learning combinations of local job-shop scheduling rules. *Industrial scheduling* (1963), 225–251.

[19] 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.

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

[21] 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.

[22] 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.

[23] 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.

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

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

[26] 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.

[27] Lawrence, S. Resource constrained project scheduling. *an experimental investigation of heuristic scheduling techniques* (1984).

[28] 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.

[29] 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.

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

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

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

[33] 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.

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

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

[36] 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.

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

[38] Sörensen, K., and Glover, F. Metaheuristics. *Encyclopedia of operations research and management science 62* (2013), 960–970.

[39] Sprecher, A., Kolisch, R., and Drexl, A. Semi-active, active, and non-delay schedules for the resource-constrained project scheduling problem. *European Journal of Operational Research 80*, 1 (1995), 94–102.

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

[41] Storer, R. H., Wu, S. D., and Vaccari, R. New search spaces for sequencing problems with application to job shop scheduling. *Management science 38*, 10 (1992), 1495–1509.

[42] 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.

[43] Taillard, E. Benchmarks for basic scheduling problems. *european journal of operational research 64*, 2 (1993), 278–285.

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

[45] 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.

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

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

[48] Yamada, T., and Nakano, R. A genetic algorithm applicable to large-scale job-shop problems. In *PPSN* (1992), vol. 2, pp. 281–290.

[49] 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.

[50] 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.

[51] 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.