CIMAT Centro de Investigación en Matemáticas, A.C.

METAHEURÍSTICAS APLICADAS AL PROBLEMA DE PLANIFICACIÓN PARA PRODUCCIÓN TIPO TALLER

T E S I S

Que para obtener el grado de

Maestro en Ciencias

con Especialización en

Computación y Matemáticas Industriales

Presenta

Oscar Hernández Constantino

Director de Tesis:

Dr. Carlos Segura González

Autorización de la versión final

Guanajuato, Gto., 10 de diciembre de 2019

CENTRO DE INVESTIGACIÓN EN MATEMÁTICAS A.C.

TESIS DE MAESTRÍA

Metaheurísticas Aplicadas al Problema de Planificación para Producción Tipo Taller

Presenta:
Oscar Hernández Constantino

Director de Tesis:
Dr. Carlos Segura González

Para obtener el grado de:

MAESTRO EN CIENCIAS CON ESPECIALIDAD EN COMPUTACIÓN Y MATEMÁTICAS INDUSTRIALES

Área de Ciencias de la Computación

29 de noviembre de 2019

Declaración de Autoría

Yo, Oscar Hernández Constantino, declaro que esta tesis titulada «Metaheurísticas Aplicadas al Problema de Planificación para Producción Tipo Taller» y el trabajo presentado en ella son míos. Confirmo que:

- Este trabajo fue hecho totalmente o principalmente mientras estaba en candidatura para un titulo de investigación en este centro de investigación.
- Si alguna parte de esta tesis ha sido presentada anteriormente para un título o cualquier otra calificación en este Centro de Investigación o cualquier otra institución, ha sido claramente expresado.
- Cuando he consultado el trabajo publicado de otros, ha sido claramente atribuido.
- Donde he citado del trabajo de otros, la fuente siempre se da. Con la excepción de tales citas, esta tesis es completamente mi propio trabajo.
- Reconocí todas las principales fuentes de ayuda.
- Cuando la tesis se basa en el trabajo hecho por mí mismo junto con otros, he dejado en claro exactamente lo que otros hicieron y lo que yo mismo contribuí.

CENTRO DE INVESTIGACIÓN EN MATEMÁTICAS A.C.

Resumen

El problema de planificación para producción tipo taller (JSSP, Job Shop Scheduling Problem) consiste en determinar la forma en que se deben utilizar un conjunto de recursos para realizar un conjunto de trabajos de forma que se minimice el tiempo requerido para procesar los mismos respetando a su vez un conjunto de restricciones, entre las que se incluyen restricciones relativas al orden en que se procesan las operaciones de los trabajos. El JSSP es un problema NP-Difícil que ha sido estudiado desde hace varias décadas y para el cual se han propuesto múltiples estrategias de resolución.

En este trabajo se realizó un análisis del estado del arte y se desarrolló un optimizador que incluyó algunas de las mejores componentes publicadas hasta la fecha, así como nuevas componentes diseñadas en esta tesis. En concreto, se desarrolló un algoritmo memético que incorporó un mecanismo de intensificación basado en Búsqueda Tabú que ha ofrecido muy buenos resultados al ser integrado en otros algoritmos híbridos. Como novedades de la tesis cabe destacar el desarrollo de un nuevo operador de cruce que se diseñó de forma específica para el problema tratado, así como la inclusión de un mecanismo de gestión de diversidad que ha obtenido resultados muy promisorios en otros problemas. Además, se desarrolló un nuevo mecanismo de gestión de diversidad, cuyo propósito es acelerar el proceso de búsqueda para lograr obtener resultados de alta calidad en menores tiempos de ejecución. Se propuso también una nueva medida de similitud entre soluciones y se verificó experimentalmente que al aplicarla en el mecanismo de gestión de diversidad previamente mencionado, se logran mejores resultados que con otras medidas que han sido utilizadas en otros algoritmos del estado del arte.

Debido a la complejidad del problema, en las instancias más grandes se requiere una gran cantidad de recursos computacionales para obtener soluciones de alta calidad, por lo cual se propone la paralelización del método propuesto, así como de dos algoritmos del estado del arte para el JSSP. En particular, con el método propuesto se logró superar el mejor resultado conocido hasta la fecha en 31 instancias, logrando obtener soluciones óptimas al menos en dos de ellas. Este logro se ha verificado utilizando plataformas dedicadas a este problema en específico, y en las cuales el método desarrollado en esta tesis está listado actualmente como el método a batir.

Agradecimientos

A mi familia, mis padres Daniel y Consuelo, mis hermanos Daniel, Itsia y Angel por ese apoyo incondicional que siempre me brindan; quienes además me han enseñado el valor y trascendencia de pertenecer a una familia unida.

A mi asesor, el Dr. Carlos Segura, por su tiempo, conocimientos y valiosas aportaciones en el proceso de elaboración del presente trabajo.

A mis sinodales, el Dr. Arturo Hernández y el Dr. Jean-Bernard, por sus comentarios y observaciones que enriquecen el resultado de la presente investigación.

Así mismo, quiero agradecer al CONACyT (Consejo Nacional de Ciencia y Tecnología) por el apoyo económico que me brindó durante el período Septiembre 2017-Agosto 2019, gracias al cual, pude concluir los estudios de posgrado concentrándome en los asuntos académicos y administrativos.

Gracias también a la institución educativa que me abrió las puertas y me brindó la oportunidad de seguirme desarrollando académicamente, el CIMAT (Centro de Investigación en Matemáticas), a la que debo el placer de haber conocido grandes profesores y apreciables compañeros de escuela; muchas gracias por todo el apoyo recibido a lo largo de mi trayectoria escolar en esta venerable casa de estudios, particulamente el apoyo económico recibido en Agosto 2017 y en el período Agosto 2019-Noviembre 2019.

A todas las personas que directa o indirectamente han contribuido para que haya alcanzado esta meta personal ¡muchas gracias!

Índice general

Capítulo 1

Introducción

1.1. Introducción

Existe una gran variedad de problemas de planificación, por ejemplo organizar entregas en un servicio de mensajería, determinar las mejores rutas que deben seguir diversos vehículos para satisfacer un servicio (ambulancias o vigilancia en cierta zona), organizar los horarios de las materias en una escuela, planificar el mantenimiento de equipos, entre otros. Es bien conocido que una gran parte de los problemas de planificación son problemas NP-Difíciles (Hart, Ross y Corne, 2005).

De manera general, los problemas de planificación consisten en determinar la forma en que se deben utilizar un conjunto de recursos disponibles para realizar un número de trabajos o tareas. En el problema de planificación para producción tipo taller ($JSSP$, Job Shop Scheduling Problem), cada trabajo se compone de una secuencia de operaciones o tareas que se deben ejecutar en un orden establecido y cada una de ellas en una máquina. Nótese que existen múltiples variaciones del mismo que habitualmente consideran restricciones adicionales o relajan algunas de las existentes, como la inclusión de ventanas de tiempo, inclusión de dependencias adicionales entre operaciones o la flexibilidad para ejecutar operaciones en diferentes máquinas. El objetivo es hacer una asignación de los recursos para completar las tareas, cumpliendo las restricciones y optimizando una función, por ejemplo terminar todas las tareas en el menor tiempo posible.

En este trabajo se revisan los principales métodos del estado del arte para el JSSP, y se proponen nuevas estrategias para solucionarlo y lograr obtener mejores resultados en las principales instancias de prueba que existen para este problema.

1.2. Motivación e Hipótesis

El JSSP ha sido estudiado intensamente desde la década de 1950, debido a que modela muy bien procesos de producción; se han desarrollado una gran cantidad de optimizadores heurísticos así como algoritmos exactos que se han probado en el JSSP (Nagata y Tojo, 2009). Además de su importancia práctica, el JSSP es un problema de optimización desafiante de

un valor académico significativo y es conocido como uno de los problemas combinatorios $NP - Completos$ más difíciles (Lenstra y Kan, 1979). Debido a su dificultad, es usado muchas veces como un problema de prueba para validar nuevos enfoques o paradigmas de resolución de problemas.

Una de las razones de la dificultad de resolver el JSSP tiene que ver con el tamaño del espacio de búsqueda, aunque esa no parece ser la única razón. Así, en casos como el TSP (Traveling Salesman Problem), el tamaño del espacio de búsqueda es $O(N!)$, donde N es el número de nodos (ciudades) para las cuales se debe determinar el orden en que se deben visitar. Sin embargo, con métodos exactos se han podido resolver casos de gran tamaño, por ejemplo, con casi $100,000$ nodos. En el caso del JSSP, cuyo espacio crece en la forma $O((n!)^m)$, donde n es el número de trabajos y m el número de máquinas, incluso en casos relativamente pequeños, los optimizadores exactos no son aplicables. Una dificultad añadida de este problema, es que no todas las soluciones son factibles, lo que dificulta la utilización de técnicas exactas. Por ejemplo, instancias con 300 operaciones ($n = 20$ trabajos y $m = 15$ máquinas), tales como $abz08$, $ta18$, $dmu01$ no han podido ser tratadas con técnicas exactas, y por ello aún no se sabe si las soluciones encontradas con los métodos aproximados son óptimas o no. Es por ello, que en este problema en particular, es especialmente útil disponer de buenos optimizadores aproximados. Nótese también que en los principales conjuntos de prueba existen instancias para las cuales la diferencia entre la cota inferior y el mejor valor conocido aún es grande, por lo que se cree que en esas instancias se tiene un gran margen de mejora.

Los algoritmos híbridos son métodos que combinan diferentes heurísticas para lograr mejorar el proceso de búsqueda, generalmente combinando técnicas poblacionales, para mejorar la exploración del espacio de búsqueda, con heurísticas de trayectoria, para intensificar la búsqueda en zonas promisorias. Este tipo de métodos han logrado obtener buenos resultados cuando se aplican en diferentes problemas, y el $JSSP$ no es la excepción. Sin embargo, estos algoritmos tienden a tener problemas de convergencia prematura, por lo que después de un número de iteraciones, ya no se logra encontrar mejores soluciones. Para evitar este problema, se han propuesto diferentes estrategias que retrasan esta condición.

En el TSP, el algoritmo híbrido con gestión explícita de diversidad propuesto en (Segura y col., 2015), logra obtener resultados competitivos en ejecuciones a largo plazo, llegando a obtener una solución óptima en una instancia con más de 30 mil nodos. En el caso del $JSSP$, algoritmos híbridos como los propuestos en (Peng, Lü y Cheng, 2015), y (Wang, 2012) logran obtener las mejores soluciones en aproximadamente el $90\,\%$ de las instancias en los principales conjuntos de prueba de la literatura.

En particular, las estrategias de control explícito de diversidad junto a los mejores componentes que se han desarrollado hasta la fecha han funcionado para mejorar el estado del arte en diferentes problemas, por ejemplo en el problema de asignación de frecuencias (Segura y col., 2016b), el problema de ruteo de vehículos (González y col., 2017), el problema de particionado de gráficas (Romero Ruiz y Segura, 2018). Por lo anterior, se plantea la siguiente

hipótesis de estudio para este trabajo: es posible mejorar los resultados del estado del arte para el JSSP si se consideran estrategias de control explícito de diversidad.

1.3. Descripción General del Problema

El problema de Planificación para Producción tipo Taller ($JSSP$) consiste en, dado un conjunto de trabajos y un conjunto de máquinas, determinar una forma adecuada de llevar a cabo los trabajos establecidos. En concreto, cada trabajo está compuesto de una secuencia ordenada de operaciones y cada una de ellas debe ser procesada por una máquina predeterminada durante un tiempo de procesamiento específico.

Existen diversas formas de medir la bondad de una solución. Una de las más usadas es el tiempo transcurrido desde que empieza la primera operación hasta que termina la última, lo que se conoce como makespan. Para resolver el problema se debe determinar el orden en que los trabajos deben ser procesados en cada una de las máquinas, de manera que se minimice el tiempo necesario para completar todos los trabajos, sujeto a las restricciones impuestas por la precedencia de operaciones que determinan las secuencias de los trabajos y las restricciones de procesamiento en las máquinas que pueden procesar solo una tarea a la vez, sin interrupciones.

1.4. Objetivos

El objetivo general de este trabajo es lograr resultados que superen a los del estado del arte para el JSSP.

Los objetivos específicos son los siguientes:

- Implementación de algoritmos del estado del arte para el JSSP.
- Mejorar un algoritmo híbrido, para el JSSP, considerando esquemas de manejo explícito de diversidad.
- Tratar instancias grandes mediante la implementación de una versión en paralelo de algoritmos híbridos del estado del arte, así como una versión en paralelo del método propuesto, que utilicen eficientemente los recursos computacionales en un ambiente de cómputo de alto rendimiento (cluster de procesadores).
- Realizar comparaciones entre el método propuesto y los algoritmos del estado del arte, utilizando los principales conjuntos de prueba que han sido considerados en la literatura.
- Divulgar los avances alcanzados.

1.5. Contribuciones

Las principales contribuciones de este trabajo son las siguientes:

- Se diseñó e implementó un método híbrido para el JSSP, incluyendo una estrategia de manejo explícito de diversidad que había sido utilizada en otros problemas NP-Difíciles. El método es competitivo frente a otros algoritmos del estado del arte.
- Se diseñó una nueva estrategia que combina el uso de diversidad y clusters que también es competitiva con los algoritmos del estado del arte y permitió superar el mejor valor conocido en una instancia.
- Se logró superar los mejores valores conocidos en 30 instancias de los principales conjuntos de prueba que se consideran en la literatura.
- En dos instancias, para las cuales no se conocía ninguna solución óptima, se consiguió generar una solución con el valor óptimo.
- Se propuso la paralelización del método desarrollado en este trabajo y de dos algoritmos del estado del arte para el JSSP, consiguiendo llegar a resultados muy promisorios en ejecuciones de 24 horas con 50 procesos.

1.6. Organización de la tesis

El resto de este trabajo está organizado de la siguiente manera.

En el Capítulo 2 se mencionan los conceptos generales de optimización y metaheurísticas para problemas de optimización combinatoria.

La descripción formal del JSSP se desarrolla en el Capítulo 3, en donde se mencionan también los principales algoritmos del estado del arte para este problema.

El Capítulo 4 describe el método propuesto, considerando dos variantes para el manejo de diversidad, así como la paralelización del algoritmo y la propuesta de paralelización para dos algoritmos del estado del arte.

En el Capítulo 5 se presentan los resultados obtenidos, mostrando comparativas del método desarrollado con dos de los algoritmos del estado del arte, considerando pruebas de ejecuciones de los algoritmos en secuencial y en paralelo.

En el Capítulo 6 se presentan las conclusiones y algunas sugerencias para continuar con la investigación de metaheurísticas para el JSSP.

Finalmente, en el Apéndice A se presentan los casos en los que se logró alcanzar o superar los mejores resultados conocidos hasta la fecha, indicando el nuevo valor encontrado, la diferencia con respecto al mejor valor conocido antes de realizar esta tesis y el método específico usado para encontrar dicho valor.

Capítulo 2

Optimización y Metaheurísticas: Conceptos Generales

Los problemas de optimización surgen de manera natural en problemas cotidianos, por ejemplo encontrar la ruta más corta para ir de un lugar a otro en una ciudad, organizar horarios escolares con las restricciones de disponibilidad en cada salón de una escuela, entre otros. En general, en un problema de optimización queremos encontrar la mejor opción entre muchas alternativas de solución para un problema específico. Cuando la dimensión del problema (número de variables a considerar) es pequeña y discreta, este tipo de problemas lo podemos resolver con cierta facilidad pues podríamos enumerar todas las posibles soluciones. Sin embargo, cuando se considera un mayor número de variables o la cardinalidad del dominio de cada variable es muy grande, es decir, el tamaño del espacio de búsqueda crece, en muchos problemas se vuelve más complejo poder encontrar la mejor solución en tiempos admisibles. Una manera de enfrentar esta problemática es mediante el uso de metaheurísticas, técnicas que no garantizan obtener la mejor solución, pero que generalmente logran obtener soluciones de buena calidad.

En este capítulo se presentan conceptos generales de optimización combinatoria, así como algunas de las principales metaheurísticas que se utilizan para resolver este tipo de problemas.

2.1. Problemas de optimización

En un problema de optimización se debe encontrar la mejor solución de acuerdo a un criterio dado. Cuando se establecen condiciones adicionales que las soluciones deben cumplir, se dice que es un problema de optimización con restricciones. De manera formal, podemos definir un problema optimización con restricciones de la siguiente manera:

Dada una una función $f : \mathbb{S} \to \mathbb{R}$, llamada función objetivo, un problema de optimización con restricciones consiste en encontrar el argumento que minimiza f dentro del conjunto de soluciones factibles, lo que podemos enunciar con la siguiente expresión:

$$s^* = arg \min_{s \in \mathbb{S}} f(s)$$

sujeto a

$$s \in \mathbb{X}$$

donde:
- $\mathbb{S}$ es conjunto de todas las soluciones, también llamado espacio de soluciones.

- La función f determina el criterio de optimización, es decir, que tan buena es una solución.

- $\mathbb{X}$ representa el conjunto de soluciones factibles.

Aunque en esta definición sólo consideramos los problemas de minimización, los problemas en que se debe maximizar una función también se pueden representar como un problema de minimización, si consideramos lo siguiente:

$$\max_x f(x) = -\min_x -f(x).$$

Por lo tanto, en las siguientes definiciones consideraremos siempre que estamos trabajando con un problema de minimización.

Decimos que una solución $s^* \in \mathbb{S}$ es un óptimo (o mínimo global) de f si se cumple que

$$f(s^*) \leq f(s), \forall s \in \mathbb{S}.$$

En este caso, s^* pertenece al conjunto de las mejores soluciones de todo el espacio de búsqueda.

Otro concepto que necesitamos es el de vecindad de una solución. Una vecindad es una función $N : \mathbb{S} \to 2^{\mathbb{S}}$ que asigna a cada solución $s \in \mathbb{S}$ un conjunto de soluciones (vecinos) en el espacio de búsqueda. Dada una vecindad N, podemos definir el concepto de óptimo local. Una solución $\hat{s}$ es un óptimo local, respecto a una vecindad N, si se cumple que

$$f(\hat{s}) \leq f(s), \forall s \in N(\hat{s})$$

.

2.1.1. Problemas de optimización combinatoria

Un problema de optimización combinatoria tiene la característica que el conjunto $\mathbb{X}$ está formado por un número finito de elementos. Ejemplos típicos de problemas de optimización combinatoria son asignación de tareas, planificación de horarios, el problema del agente viajero (TSP, Traveling Salesman Problem) o el problema de la mochila (KP, Knapsack Problem), entre otros (Talbi, 2009).

2.1.2. Complejidad Computacional

En el estudio de algoritmos se diseña un algoritmo para resolver un problema particular y se analiza la eficiencia del algoritmo en términos de la cantidad de recursos (tiempo o espacio) que requiere para resolver un problema. La complejidad en tiempo se refiere al número de pasos necesarios para resolver el problema. La complejidad en espacio se refiere a la cantidad de memoria que se utiliza para resolver el problema. Generalmente la complejidad (en tiempo o en espacio) se mide en términos del tamaño de la entrada del problema, denotada como n, por ejemplo para indicar el número de variables, y usualmente se especifica en términos del análisis del peor caso.

El objetivo del análisis de un algoritmo no es determinar el número exacto de pasos que se requieren, sino una cota asintótica. Para especificar esto, se utiliza la notación Big$-O$, que es una de las más populares en el análisis de algoritmos (Cormen y col., 2009).

La complejidad de un problema es equivalente a la complejidad del mejor algoritmo para resolver ese problema. Un aspecto importante de teoría de la computación es clasificar problemas dentro de clases de complejidad. Una clase de complejidad representa el conjunto de todos los problemas que pueden ser resueltos usando una cantidad dada de recursos computacionales (Talbi, 2009). Existen dos clases importantes de problemas: P y NP.

La clase de complejidad P representa el conjunto de todos los problemas de decisión que pueden ser resueltos de manera determinista en un tiempo polinomial. Un algoritmo (determinista) es polinomial para un problema si la complejidad del peor caso esta acotada por una función polinomial respecto al tamaño de la entrada. Los problemas que pertenecen a la clase P se consideran que son relativamente fáciles de resolver.

La clase de complejidad NP representa el conjunto de todos los problemas de decisión que pueden ser resueltos en tiempo polinomial con una máquina de Turing no determinista. Bajo la hipótesis P != NP, esto tiene la implicación de que dichos problemas no pueden ser resueltos en tiempo polinomial en una computadora determinista.

Al estudiar la complejidad computacional de los problemas, se suele utilizar el concepto de reducción polinomial. De manera informal, se dice que un problema A se reduce polinomialmente a un problema B si, para todas las instancias de entrada de A, es posible construir siempre una instancia de entrada para B, con una función de tiempo polinomial en el tamaño de la entrada de A, tal que la entrada de A es resuelta si y solo si la instancia de B es resulta.

Un problema A en NP es $NP-$Completo si para cualquier problema de la clase NP es posible hacer una transformación (reducción), en tiempo polinomial, al problema A. Por otro lado, decimos que un problema A es NP-difícil (NP-hard) si para cualquier problema C en NP es posible reducirlo a A en tiempo polinomial. Nótese que los problemas NP-difíciles no son necesariamente problemas de decisión, y por tanto, pueden no pertenecer a la clase NP. Si un problema de optimización B tiene una versión de un problema NP-Completo, entonces B es NP-Difícil (Cormen y col., 2009).

2.2. Métodos de Optimización

Debido a la importancia práctica que tienen los problemas de optimización, se han propuesto diferentes métodos de optimización para obtener buenas soluciones. Estos métodos pueden clasificarse en dos grandes áreas: algoritmos exactos y algoritmos aproximados . Los algoritmos exactos garantizan encontrar la solución óptima, pero generalmente sólo son útiles cuando el tamaño del espacio de búsqueda del problema, es decir, la cardinalidad de $\mathbb{S}$ y de $\mathbb{X}$, es pequeño. Los algoritmos aproximados no garantizan encontrar la solución óptima, pero suelen obtener soluciones de una alta calidad en un tiempo razonable. Por lo tanto, en los algoritmos aproximados se sacrifica la garantía de obtener la solución óptima, que sí ofrecen los métodos exactos, a cambio de obtener soluciones habitualmente de buena calidad en un tiempo reducido.

Entre los algoritmos aproximados se suelen distinguir los siguientes tipos:

- **Heurísticas Constructivas.** Este tipo de algoritmos trabajan con un conjunto de reglas (heurísticas) para generar una solución, habitualmente tomando en cada paso ciertas decisiones sobre las características de la solución, que en pasos posteriores no se deshacen. Se suele considerar información específica del problema para mejorar la calidad de las soluciones que se construyen. Estos algoritmos suelen ser muy rápidos, y aunque la calidad de la solución suele ser mejor que la obtenida al generar una solución aleatoria, suele ser inferior a la obtenida por los métodos de búsqueda local. Por ejemplo, en el TSP una heurística para construir una solución es elegir en cada paso la arista de menor peso que no cree ciclos con las aristas previamente elegidas.
- **Heurísticas de Trayectoria.** Los métodos de trayectoria trabajan con una solución inicial (generada de manera aleatoria o con alguna heurística constructiva) y la van mejorando de manera iterativa, habitualmente considerando a las soluciones que se encuentran cercanas, de acuerdo a una vecindad.
- **Heurísticas Poblacionales.** Estos métodos mantienen un conjunto de soluciones (población) durante todo el proceso de optimización, considerando diferentes tipos de operadores que actúan de manera iterativa sobre la población, para intentar encontrar mejores soluciones.
- **Metaheurísticas.** Al hablar de heurísticas, los autores se refieren a métodos hechos a medida para un problema específico. Aunque hay muchas heurísticas constructivas, no hay tantas heurísticas de trayectoria o poblacionales. En su lugar se usan las metaheurísticas, que son algoritmos aproximados y usualmente no deterministas (Alba, 2005). Las metaheurísticas son métodos para resolver problemas de optimización, que se pueden aplicar a diferentes problemas, adaptando sólo componentes básicos dentro del algoritmo; en la literatura, también suele referirse a estos métodos como heurísticas modernas (Michalewicz y Fogel, 2013).

De manera general, las metaheurísticas son estrategias que guían el proceso de búsqueda en el espacio de soluciones, de forma que la búsqueda de la mejor solución se haga de manera eficiente. Se suelen distinguir principalmente dos categorías: metaheurísticas de trayectoria y metaheurísticas poblacionales o algoritmos basados en poblaciones.

2.2.1. Exploración e Intensificación

Para lograr obtener resultados de una buena calidad las metaheurísticas hacen un balance entre etapas de exploración e intensificación en el espacio de búsqueda. En la etapa de exploración, también conocida como diversificación, el objetivo es encontrar zonas prometedoras en el espacio de búsqueda, es decir, encontrar las zonas en las que se encuentran soluciones de buena calidad.

En la etapa de intensificación, también conocida como explotación, la idea es realizar una búsqueda más exhaustiva en una zona específica del espacio de búsqueda. Habitualmente se intensifica en regiones promisorias que se encontraron mediante exploración.

Estas etapas no siempre son aisladas ni únicas en el proceso de búsqueda. Es común combinar, de manera iterativa, las dos etapas. El balance entre la exploración y la intensificación es una característica muy importante en las metaheurísticas, ya que para lograr obtener soluciones de buena calidad en corto tiempo, se deben identificar rápidamente regiones promisorias, con soluciones de alta calidad, y no gastar demasiado tiempo en realizar una intensificación excesiva en zonas que podrían no tener soluciones de buena calidad (Blum y Roli, 2003).

En los últimos años se han desarrollado algoritmos que tratan de modificar el balance entre exploración e intensificación con base en el criterio de parada establecido para el algoritmo. En algunos casos, el principio es comenzar explorando y paulatinamente pasar a fases de intensificación (por ejemplo con la estrategia RMDDC propuesta en Segura y col., 2016a), mientras que en otros, se mezclan fases de exploración e intensificación durante todo el proceso de optimización, tal como lo hace el algoritmo Saw-Tooth GA, propuesto en Koumousis y Katsaras, 2006.

2.3. Metaheurísticas de Trayectoria

Las metaheurísticas de trayectoria son métodos de optimización en los cuales se mantiene una solución activa durante toda la búsqueda. En el proceso de búsqueda se realizan modificaciones o perturbaciones para alcanzar otras soluciones. Las nuevas soluciones que se pueden convertir en soluciones activas se pueden determinar de diferentes formas. En algunas ocasiones se usan mutaciones o perturbaciones, aunque probablemente lo más habitual es usar una función de vecindad. De esta manera, se construye un camino, o trayectoria, en

el espacio de búsqueda con las soluciones que se van evaluando hasta encontrar las soluciones de mejor calidad.

El principal problema de las metaheurísticas de trayectoria es que pueden fácilmente converger a una solución o una región, en la cual ya no es posible mejorar realizando ligeras perturbaciones. Por ejemplo si todas las soluciones vecinas ya son de una calidad peor a la solución actual. Similarmente, en algunos casos los métodos pueden ciclarse en el camino que están haciendo.

La búsqueda local es la metaheurística de trayectoria más simple, en la cual se considera una función de vecindad y se selecciona de la vecindad a alguna solución que mejore a la actual, de manera iterativa, hasta alcanzar un óptimo local; en este caso se dice que la búsqueda ha convergido a un óptimo local. Para escapar o evitar los óptimos locales, se han propuesto diferentes metaheurísticas de trayectoria, entre las principales se encuentran:

- **Búsqueda Local Iterada.** Una vez que se alcanza un óptimo local, se realiza una perturbación más grande, de manera que se logre escapar de los óptimos locales. Una de las principales dificultades para usar este método es definir el tipo de perturbación que se realizará, ya que si la perturbación es demasiado fuerte, simplemente se estarán explorando regiones al azar en el espacio de búsqueda, y difícilmente se lograrán alcanzar las mejores soluciones, pues prácticamente no habrá intensificación.
- **Recocido Simulado.** En este método se permite elegir soluciones de una calidad inferior a la solución actual, con cierta probabilidad que estará determinada por un parámetro T que va cambiando durante la ejecución. Habitualmente, al inicio de la búsqueda se permite que las soluciones puedan empeorar con una alta probabilidad, y por lo tanto logren escapar de los óptimos locales; al final de la búsqueda, la probabilidad de elegir una peor solución debería ser muy baja, de manera que se mantengan soluciones de buena calidad. Elegir la forma en que este parámetro cambia durante el proceso de búsqueda es una de las principales dificultades de este método.
- **Búsqueda Tabú.** Para escapar de los óptimos locales, este método utiliza una memoria de corto plazo para evitar volver a soluciones que ya han sido exploradas. De esta manera, cuando se alcanza un óptimo local, se permite que soluciones de peor calidad sean seleccionadas, y se almacenan en la memoria para explorar nuevas zonas y evitar repetir el mismo camino que ya ha sido explorado. Dado que se dispone de una memoria limitada, es importante establecer un tamaño para la memoria que se utiliza en este método. El tamaño de esta memoria juega un papel muy importante para dirigir la búsqueda. Si se utiliza un tamaño pequeño, es posible que no sea suficiente para escapar de los óptimos locales y al final se termina ciclando la búsqueda. Por otro lado, si el tamaño de la memoria es demasiado grande, podría ser demasiado restrictivo y dificultaría alcanzar soluciones de buena calidad. Otro problema es que almacenar soluciones completas y compararlas en cada iteración, puede ser muy

costoso en memoria y tiempo de ejecución. Por ello, en muchas ocasiones sólo se almacenan atributos o características importantes de las soluciones, con el fin de que las comparaciones sean más rápidas y el espacio que se requiere sea menor.

- **Búsqueda Local Guiada.** De manera similar a la búsqueda Tabú, este método utiliza una memoria. En este caso, la memoria se utiliza para penalizar las características (atributos) que han aparecido de manera más frecuente en el proceso de búsqueda. La idea es forzar a explorar otras regiones del espacio para lograr escapar de los óptimos locales. La principal dificultad para usar este método consiste en definir cuáles son los atributos que se deben penalizar de manera que sean útiles para realizar la exploración.

Este tipo de métodos, también suelen conocerse (de manera no demasiado formal) como métodos de búsqueda local, ya que generalmente se concentran en una zona del espacio de búsqueda en donde realizan un proceso de intensificación para buscar las mejores soluciones. Si el método logra encontrar la trayectoria a la zona en donde se encuentra la mejor solución, se tendrán buenos resultados. Sin embargo, muchas veces es muy difícil encontrar la zona correcta, por lo cual, en muchas ocasiones se integran dentro de métodos poblacionales, con el fin de conseguir un mejor balance entre exploración e intensificación.

2.3.1. Búsqueda Tabú

En esta sección se describe con mayor detalle el método de búsqueda tabú, ya que es uno de los métodos usados en esta tesis. En esta sección se presenta de forma general y posteriormente se detallará su adaptación para el JSSP.

Este método considera un conjunto de memorias, para conseguir un balance adecuado entre exploración e intensificación. La memoria más usada es la que se denomina memoria de corto plazo, cuyo propósito principal es poder escapar de óptimos locales y a la vez evitar que el algoritmo se cicle. Además, se pueden usar memorias de medio plazo y de largo plazo, para promover intensificación y diversificación, respectivamente. Sin embargo, estas memorias adicionales no se suelen usar cuando la Búsqueda Tabú es integrada en métodos híbridos (y no se usan en esta tesis), por lo que no entramos en detalles sobre dichos tipos de memorias. A la memoria de corto plazo, se le llama lista tabú y es muy dependiente de la vecindad y del tipo de prohibición que se establece.

En el Algoritmo 1 se presenta el pseudocódigo de la búsqueda tabú. En su versión más general, en la búsqueda tabú se explora la vecindad de la solución actual y se elige al mejor vecino de la misma que no está prohibido por la memoria de corto plazo, de manera que el proceso de exploración de la vecindad suele ser el paso más costoso, ya que se debe realizar una evaluación de todos los vecinos. Cuando la vecindad es muy grande, se suelen utilizar estrategias que permitan una evaluación más eficiente de los vecinos.

Si la solución inicial es de muy mala calidad, la búsqueda tabú generalmente logrará encontrar una mejor solución en pocas iteraciones; si la solución inicial ya es de buena calidad, la

Algoritmo 1: Búsqueda Tabú

Entrada: $sIni$ Solución Inicial ;
Salida: $sElite$ Mejor solución generada

```
1  sActual = sIni;
2  LT = inicializarListaTabu();
3  MMP = inicializarMemoriaMedia();
4  MLP = inicializarMemoriaLarga();
5  sElite = sIni;
6  mientras No Criterio de Paro hacer
7      vecinos = generarVecinos() ;
8      mov = elegirMejorVecinoNoPenalizado(vecinos, LT) ;
9      Si mov es mejor que sElite
10         sElite = mov ;
11     actualizarListaTabu(mov) ;
12     actualizarMemoriaCorta(MMP) ;
13     actualizarMemoriaLarga(MLP) ;
14     Si Condiciones para Intensificar
15         Realizar Etapa de Intensificación(MMP) ;
16     Si Condiciones para Exploración
17         Realizar Etapa de Exploración(MLP) ;
18 devolver sElite ;
```

búsqueda tabú podría requerir un número grande de iteraciones para lograr encontrar una mejor solución.

Al inicio del proceso de búsqueda, dependiendo del problema, es relativamente fácil y rápido caer en un óptimo local. Una vez que se da esta situación, la búsqueda tabú sigue eligiendo al mejor vecino no prohibido, sin importar que la calidad de la solución sea peor.

La identificación del mejor vecino no prohibido es generalmente es el factor más importante a considerar, línea 7, ya que suele llevar el mayor tiempo de procesamiento para este algoritmo. Es deseable poder descartar fácilmente las soluciones que no sean factibles o que sean de baja calidad.

Una de las principales características de la búsqueda tabú es el uso de una memoria para forzar a buscar en nuevas zonas y permitir escapar de los óptimos locales, esta memoria (conocida como lista tabú) se suele manejar como una lista que almacena soluciones o atributos de las soluciones que estarán prohibidos. Para manejar la lista tabú (líneas 2 y 11), existen diferentes estrategias; por ejemplo almacenar la solución completa, los atributos de la solución, o los cambios que llevan de una solución a otra. Almacenar todas las soluciones visitadas puede llegar a tener un alto costo en espacio. Por otro lado, también se debe considerar la forma en que se validará si un vecino está o no prohibido. Si la lista es muy grande, esta validación podría llegar a ser muy costosa y tener un efecto significativo en el tiempo de ejecución del algoritmo.

Los vecinos penalizados serán aquellos que tengan atributos prohibidos por la lista tabú. A

veces la penalización puede llegar a ser demasiado estricta y se pueden descartar soluciones de buena calidad. Para prevenir estos casos, se suele utilizar un criterio de aspiración, que permita elegir soluciones buenas. Un posible criterio de aspiración consiste en no descartar a posibles soluciones que están penalizadas, pero que son soluciones muy prometedoras, por ejemplo, con un valor de función objetivo mejor que la mejor solución que ha sido encontrada.

Dependiendo del problema, algunas veces es útil considerar memorias adicionales a la lista tabú, para mantener un mejor balance entre exploración e intensificación. Para ello, se consideran memorias a medio plazo (líneas 3 y 12) y una a largo plazo (líneas 4 y 13).

2.4. Metaheurísticas Poblacionales

Las metaheurísticas poblacionales son métodos que trabajan con un conjunto de soluciones candidatas (población de individuos) a las cuales se les aplican diferentes operadores (selección, mutación, cruza, entre otros) para lograr mejorar la población y al final obtener individuos de buena calidad.

Existe una gran cantidad de algoritmos poblacionales que en algunos casos han surgido como analogía a procesos naturales, por ejemplo el algoritmo evolutivo y los algoritmos bioinspirados como Colonia de Hormigas, Enjambre de Abejas, Optimización por Enjambre de Partículas, entre otros.

2.4.1. Algoritmo Evolutivo

Una de las metaheurísticas poblacionales más populares es el algoritmo evolutivo, el cual se inspira en el proceso evolutivo que siguen las especies (Eiben y Smith, 2003). De manera general, en una de sus versiones más básica el algoritmo mantiene un conjunto de soluciones (individuos) en un conjunto llamado población. Los individuos más aptos se combinan entre sí para generar nuevos individuos y si los nuevos individuos son mejores, reemplazarán a los padres y el proceso se repite de manera iterativa.

Los individuos son evaluados respecto a una función de aptitud. Aunque es común que esta función de aptitud se corresponda con la función objetivo a minimizar, algunas veces se considera alguna variante de la función objetivo o se combina con alguna otra función para determinar la aptitud, con el fin de mejorar el proceso de búsqueda. La función de aptitud no es una característica exclusiva del algoritmo evolutivo y es común encontrarla también en otras metaheurísticas.

En el Algoritmo 2 se presenta el pseudocódigo del algoritmo evolutivo. El algoritmo comienza generando individuos que conforman la población inicial. Posteriormente, el algoritmo

Algoritmo 2: Algoritmo Evolutivo

Salida: $sElite$ Mejor solución generada

```
1  pActual = generarPoblacionInicial();
2  iteracion = 0
3  mientras No Criterio de Paro hacer
4      padres = seleccionarPadres(pActual) ;
5      hijos = generarHijos(padres) ;
6      hijos = aplicarMutacion(hijos) ;
7      evaluar(hijos) ;
8      pActual = reemplazarIndividuos(pActual, hijos) ;
9  sElite = Elegir la mejor solución de pActual ;
10 devolver sElite
```

realiza un conjunto de iteraciones, en donde en cada iteración consta de varias etapas: selección de padres, reproducción de individuos y selección de los individuos que permanecen en la población. Estas etapas se repiten hasta cumplir algún criterio de paro.

A continuación se describen las principales etapas en este algoritmo:

1. **Generación de la Población Inicial** (línea 1) . Lo más común es generar a los individuos de manera aleatoria. Algunas veces se trabaja con individuos que son generados mediante alguna heurística previa para mejorar la calidad de los individuos. Sin embargo, si los individuos son generados con alguna heurística que no sea suficientemente aleatoria, los individuos podrían tener demasiadas características en común, lo que puede llevar a una convergencia prematura de la población.

2. **Selección de Padres** (línea 4). En esta etapa se elige un subconjunto de individuos de la población, posiblemente con repetición, a partir de los cuales se realizará la combinación para generar nuevos individuos.

3. **Generación de Individuos** (también llamada etapa de reproducción, línea 4). En esta etapa se aplican diferentes operadores, generalmente de cruza y mutación, para combinar la información de los individuos que han sido seleccionados como padres. Los operadores para combinar la información suelen ser muy dependientes del problema y la representación que se utiliza.

4. **Selección de Individuos** (línea 8). Una vez que se han generado nuevos individuos, se debe elegir qué individuos permanecerán en la población. A esta etapa también se le conoce como reemplazamiento de individuos. La selección de los individuos que permanecen en la nueva población es un proceso importante ya que tiene un efecto importante en el proceso de búsqueda y está muy relacionado con efectos conocidos como la convergencia prematura o pérdida de diversidad.

A continuación se van a describir algunas posibles formas de hacer la selección de padres, reemplazamiento de individuos y se discutirán algunos de las principales criterios de paro.

Selección de Padres

Para poder generar nuevos individuos, la fase de reproducción suele requerir de individuos, típicamente al menos dos, a partir de los cuales se combina su información. Los padres se seleccionan a partir de los individuos de la población actual, en cada iteración del algoritmo evolutivo; esta selección puede ser con o sin repetición, es decir, un individuo puede ser seleccionado como padre más de una vez (en la misma iteración), por lo que en el proceso de reproducción se podría llegar a generar un individuo idéntico (clon) al padre. Los padres juegan un rol muy importante, ya que suelen determinar las características principales de los nuevos individuos que se generan. Por tal motivo, esta etapa tiene también un efecto importante en la exploración del espacio de búsqueda que realiza el algoritmo.

De manera informal, se dice que un esquema tiene alta presión de selección si los mejores individuos tienen probabilidades mucho más altas de ser seleccionados que los individuos de peor calidad, mientras que tiene baja presión de selección, cuando las probabilidades son cercanas a la uniforme.

Para hacer la selección se padres se suelen considerar diferentes estrategias, las más comunes son las siguientes:

- **Método de la Ruleta.** La probabilidad de seleccionar a un individuo es proporcional a su aptitud. De esta manera, los individuos que tienen una aptitud mayor, tienen más probabilidad de ser seleccionados. La idea principal es que se favorece que los mejores individuos, de acuerdo a su aptitud (que muchas veces coincide con el valor de la función objetivo) sean seleccionados para que participen en la fase de reproducción; individuos con baja aptitud mantienen cierta probabilidad de ser seleccionados, con el fin de que se pueda seguir explorando el espacio de búsqueda.
- **Selección por $k-$Torneo.** Se eligen al azar $k-$individuos y se selecciona al mejor; la selección puede ser con o sin reemplazo, es decir, un individuo puede ser considerado más de una vez en el mismo torneo. Cuando se tiene una selección sin reemplazo, los individuos no pueden ser seleccionados más de una vez en el mismo torneo, se disminuye considerablemente la probabilidad de seleccionar a los peores individuos; cuando $k > 1$ se elimina la posibilidad de que el peor individuo sea seleccionado. Valores grandes de k aumentan la presión de selección, ya que los mejores individuos tendrán una alta probabilidad de ser seleccionados; cuanto mayor sea el valor de k, menor será la probabilidad de selección de los peores individuos. La selección por torneo binario ($k = 2$) es una de las más comunes para el algoritmo evolutivo.
- **Selección Aleatoria.** Es un caso particular de la selección por torneo, es el caso de $k = 1$, por lo que se tiene una selección aleatoria, con lo cual se disminuye la presión de selección, ya que todos los individuos tienen la misma probabilidad de ser seleccionados.

Condiciones de Paro

Dependiendo del problema y de los recursos que se dispongan se suelen considerar las siguientes condiciones de paro, que determinan cuando termina su ejecución el algoritmo evolutivo.

- **Número de Generaciones** (o número de funciones de evaluación). Se establece un número máximo de iteraciones que debe realizar el algoritmo; algunas veces este criterio se establece de manera indirecta considerando un número máximo de llamadas a la función de evaluación. Si se considera un número demasiado bajo de iteraciones, el algoritmo podría no alcanzar a converger a una región. Por otro lado, cuando se realizan demasiadas generaciones, típicamente se alcanza un límite en el que el algoritmo converge y ya no logra mejorar, aunque esto depende de otros factores (principalmente del operador de reemplazamiento); algunas veces, cuando el algoritmo no llega a converger, hacer más generaciones no mejora la calidad de las soluciones que se obtienen.
- **Tiempo de Ejecución.** Se fija un tiempo máximo para la ejecución de todo el algoritmo, sin importar el número de generaciones que se realizan. Se tiene una situación análoga a la que se discute en el punto anterior; un tiempo demasiado corto podría limitar el número de generaciones que se realizan, y un tiempo muy largo puede ser costoso y llegar a un número muy grande de generaciones, que no necesariamente mejorarán la calidad de la solución que se obtiene.
- **Valor de Éxito.** Se establece un valor que se debe alcanzar en la función objetivo y cuando el algoritmo llega a ese valor se detiene la ejecución. Para este tipo de criterios de paro siempre es necesario combinarlo con alguna de las condiciones anteriores, ya que de otra forma el algoritmo podría quedarse ciclado y nunca terminar, en los casos en donde no logre alcanzar el valor mínimo.

Los criterios de paro más empleados son un número máximo de evaluaciones y un tiempo máximo de ejecución. Nótese que en muchos casos, especialmente en problemas con evaluaciones costosas, el tiempo de evaluación es prácticamente fijo y el costo del resto de pasos no es demasiado significativo, por lo que en dichos casos fijar el criterio de paro por número de evaluaciones o por tiempo es prácticamente análogo. Por otro lado, en otros casos el tiempo es heterogéneo o el tiempo asociado a otros pasos no se puede despreciar, por lo que hay mucha discusión sobre cómo elegir el criterio de paro y como realizar comparativas que sean justas.

En este trabajo, para los algoritmos implementados se eligió un criterio de paro por tiempo, ya que son algoritmos híbridos y en cada iteración el proceso de intensificación puede ejecutar un número diferente de pasos, con diferentes costos, por lo que establecer un criterio de paro sólo teniendo en cuenta el número de generaciones o evaluaciones no sería un criterio justo.

Métodos de Reemplazamiento

Algunos métodos clásicos de reemplazamiento son los siguientes:

- **Reemplazamiento Generacional.** En este tipo se selección se reemplaza a todos los individuos de la población con los individuos generados en el paso de reproducción. Para poder aplicar este tipo de selección, es necesario que en el paso de reproducción se genere un número de hijos igual al tamaño de la población que se desea mantener. De otra forma, el tamaño de la población disminuiría (o aumentaría) durante la ejecución. Una de las principales desventajas de este mecanismo de reemplazamiento es que no se asegura que la mejora solución encontrada se mantenga en la población, ya que en cada paso todos los individuos son reemplazados; si en el paso de reproducción no se logra generar una solución con la misma calidad (o superior) del mejor individuo encontrado, la solución se pierde.
- **Reemplazamiento Generacional con elitismo.** La nueva población se genera con los nuevos elementos creados, al que se adjunta además el mejor de la población anterior.
- **Reemplazamiento de los Peores.** Los mejores individuos reemplazan a los peores. Para hacer este tipo de selección, se unen el conjunto de la población actual y el conjunto de individuos generados (hijos) en la fase de reproducción, se ordenan de acuerdo a su función de aptitud y se eligen a los mejores para mantenerlos en la población para la siguiente iteración.

Los métodos de reemplazamiento anteriormente mencionados son bastante sencillos y pueden sufrir de algunos inconvenientes. En el caso del reemplazamiento generacional, no se tiene elitismo, es decir, no siempre la mejor solución se preserva y en muchos casos puede sufrir de divergencia por lo que prácticamente no se realiza intensificación. Por el contrario, el reemplazamiento generacional con elitismo, y especialmente el reemplazamiento de los peores, son métodos que suelen sufrir de convergencia prematura, es decir, que tras pocas generaciones todos los individuos están localizados en una región reducida del espacio de búsqueda, careciendo de una buena exploración.

Dado que el problema de convergencia prematura es muy común, se han desarrollado varios esquemas de reemplazamiento que tratan de controlar la diversidad de los individuos seleccionados. Algunos de los más populares son los siguientes:

- **Suma pesada de aptitud y diversidad.** En este tipo de métodos se modifica la función de aptitud, de manera que la función objetivo y la aportación a la diversidad se consideran como una suma pesada, comúnmente de la siguiente manera:

$$aptitud(s) = \alpha f(s) + (1 - \alpha) diversidad(s, P)$$

  La función f representa la función objetivo del problema, P es la población, y la función $diversidad$ determina una medida de que tan similar es un individuo respecto

a los demás individuos de la población; α es un parámetro que controla que tanta importancia se da a la función objetivo respecto a la diversidad.

El principal problema de esta estrategia es que introduce un parámetro (α) que es demasiado sensible al problema, e incluso a una instancia en particular. En general, este tipo de métodos de selección sólo retardan la convergencia algunas generaciones, pero después de un tiempo terminan convergiendo.

- **Mejor No Penalizado (Best Not Penalized, BNP).** En este método se considera una selección elitista, pero se penaliza a los individuos que son demasiado parecidos a alguno ya existente en la población; la noción de demasiado parecido depende del tiempo transcurrido y del criterio de paro (número de generaciones o tiempo máximo de ejecución). Con esto se logra retrasar el problema de convergencia prematura. Aunque este método también introduce un parámetro, que determina cuando dos individuos son demasiado parecidos, éste no es tan sensible y evita la convergencia al menos hasta alcanzar las condiciones de paro que se establezcan. Este tipo de selección ha sido aplicado exitosamente en (Romero Ruiz y Segura, 2018) y se explicará con más detalle más adelante en este trabajo.

Modelos Poblacionales

A partir del algoritmo evolutivo, se han desarrollado diferentes variantes. Una de las modificaciones que se pueden hacer es el tipo de modelo poblacional que se utiliza. Teniendo en cuenta el número de individuos que se generan en el paso de reproducción, existen dos modelos poblacionales:

- **Modelo Generacional** (Generational Model). En cada iteración se genera un conjunto de hijos, usualmente un número igual al tamaño de la población, que compiten con los padres. Dado que se genera un gran número de individuos, es posible realizar una mayor exploración. Sin embargo, si la evaluación de los individuos es muy costosa, este modelo podría requerir demasiado tiempo de ejecución para converger.
- **Modelo de Estado Estacionario** (Steady-State Model). En cada iteración típicamente se generan sólo 1 o 2 hijos, dependiendo del operador de cruce que se esté utilizando. Se utiliza comúnmente cuando la función de evaluación es muy costosa, o cuando por alguna otra razón el proceso de convergencia es muy lento. En general, con este modelo se logra acelerar la convergencia, ya que los individuos de mejor calidad rápidamente entran a la población y pueden compartir sus atributos con los otros individuos de la población.

2.5. Metaheurísticas Híbridas

Usualmente, las metaheurísticas poblacionales logran realizar una buena exploración del espacio de búsqueda, pero suelen tener problemas para lograr intensificar en las regiones promisorias. Por lo anterior, es común utilizar metaheurísticas híbridas, que son aquellas que acoplan a una metaheurística otro método, como podría ser una estrategia exacta (para resolver un subproblema) u otra metaheurística. Entre las diferentes técnicas híbridas, los métodos que combinan una metaheurística poblacional con una metaheurística de trayectoria son muy comunes, ya que tratan de aprovechar las fortalezas de cada uno de estos tipos de técnicas.

Al combinar las estrategias, es importante establecer un balance en la forma en que se utilizan las diferentes estrategias. Una forma de hacerlo es estableciendo un criterio de paro cada vez que se hace uso de la metaheurística de trayectoria. Cuando la integración de la metaheurística de trayectoria se realiza de esta forma se usan principalmente dos tipos de criterios de paro:

- **Número de Iteraciones.** Se estable un número de iteraciones que se realizará en el proceso de mejora.
- **Tiempo de Ejecución.** Generalmente este se calcula de acuerdo al número de iteraciones que se desea que realice la metaheurística híbrida.

En las siguientes secciones se presentan dos métodos híbridos que se utilizan en este trabajo.

2.5.1. Algoritmo Memético (Híbrido del Evolutivo)

Una de las metaheurísticas híbridas más populares es el híbrido que surge integrando un algoritmo de trayectoria en el algoritmo evolutivo. Aprovechando la buena exploración que se logra en el algoritmo evolutivo, se utiliza un método de trayectoria para mejorar a los individuos de la población que se generan durante el proceso de búsqueda. De esta forma, se logra realizar una mejor intensificación, y se suelen conseguir mejores resultados.

A este tipo de combinación se la llama algoritmo memético de primera generación y se presenta su pseudocódigo en el Algoritmo 3. La principal diferencia, respecto al Algoritmo Evolutivo, la podemos encontrar en las líneas 2 y 9. En estos pasos se introduce una metaheurística de trayectoria para lograr mejorar la calidad de las soluciones en la población, lo que permite mejorar el proceso de intensificación en el algoritmo evolutivo.

Al considerar este tipo de algoritmo, se suele tener el mismo problema de convergencia prematura del algoritmo evolutivo. Debido a que se realiza una fase de intensificación, en algunos casos la población podría llegar a converger más rápidamente (en un menor número de iteraciones) si no se usan estrategias para evitarlo.

Algoritmo 3: Algoritmo Memético (Híbrido del Evolutivo)

Salida: $sElite$ Mejor solución generada

```
1  pActual = generarPoblacionInicial();
2  mejorarPoblacion(pActual);
3  iteracion = 0
4  mientras No Criterio de Paro hacer
5      padres = seleccionarPadres(pActual) ;
6      hijos = generarHijos(padres) ;
7      hijos = aplicarMutacion(hijos) ;
8      evaluar(hijos) ;
9      mejorarPoblacion(hijos);
10     pActual = reemplazarIndividuos(pActual, hijos) ;
11 sElite = Elegir la mejor solución de pActual ;
12 devolver sElite
```

Algoritmo 4: Path Relinking

Entrada: $solOri$, solución que se utilizará como inicio de la ruta
$solDst$, solución que se utilizará como fin de la ruta
Salida: $sElite$, Mejor solución encontrada

```
1  solC = solOri ;
2  sElite = Elegir a la mejor solución entre solOri y solDst ;
3  mientras solC! = solDst hacer
4      solC = cambiarSolucion(solC, solDst) ;
5      Si solC es mejor que solElite
6          sElite = solC ;
7  devolver sElite
```

2.5.2. Algoritmo Híbrido con Path Relinking

El método de Path Relinking (PR) genera una ruta, en el espacio de búsqueda, entre dos soluciones dadas. Se pretende que las soluciones generadas en la ruta, compartan características de las soluciones dadas, de manera que se logre encontrar mejores soluciones. Para generar estas rutas, se debe establecer una solución inicial, determinar el tipo de movimientos que se realizarán desde esta solución para ir introduciendo atributos de la solución guía, de manera que la solución candidata reduzca la distancia entre estas soluciones.

En el Algoritmo 4 se presenta el pseudocódigo del método genérico de Path Relinking. En la línea 4 se debe establecer la forma en que la solución candidata, $solC$ en el pseudocódigo, debe cambiar de manera que tenga una mayor similitud con la solución destino, $solDst$ en el pseudocódigo. Dependiendo del tipo de perturbación que se realice, podría no garantizarse que se generen soluciones factibles y en estos casos se debe decidir si se aplica alguna estrategia de reparación o si simplemente se ignora esa solución candidata.

La forma en que se define el método $cambiarSolucion$ (línea 4), es muy dependiente del problema y la forma en que se representan las soluciones. Este método, $cambiarSolucion$,

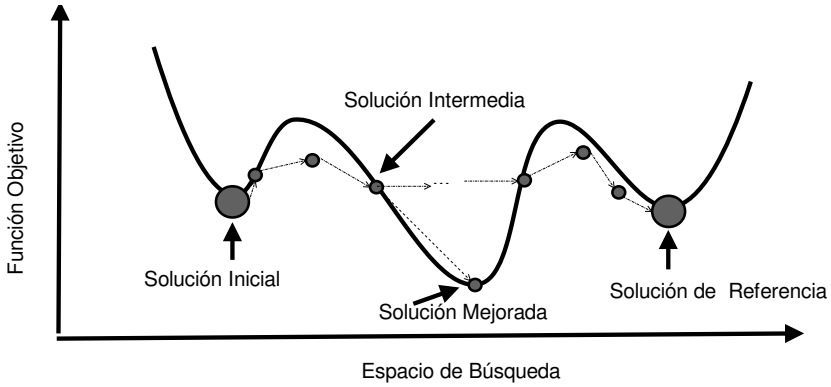


FIGURA 2.1: Ejemplo del método de Path Relinking Híbrido

establece la forma en que se muestrea la ruta entre las dos soluciones de entrada. Se debe tener en cuenta que si se muestrean individuos muy cercanos, el procedimiento podría llegar a ser muy costoso, por lo que generalmente se suele definir un parámetro para indicar la distancia o similitud mínima que habrá entre los individuos que se muestrean, de forma que se limita el número total de individuos generados.

En la Figura 2.1 se muestra un ejemplo gráfico del procedimiento de Path Relinking híbrido. En el ejemplo se muestra una posible ruta que se genera entre las soluciones (inicial y la de referencia). Cada uno de los puntos representa una solución candidata. El método de Path Relinking híbrido debe elegir una solución candidata y aplicar un proceso de mejora, generalmente una metaheurística de trayectoria. La idea central del algoritmo es que entre dos óptimos locales puede existir otro óptimo local de mejor calidad; por esta razón, una vez que se tienen los candidatos, se elige uno y se aplica el proceso de mejora, de manera que se intente que el candidato seleccionado se convierta en un óptimo local.

Aunque el algoritmo trabaja sólo con dos soluciones, esta estrategia se suele generalizar al considerar una población de soluciones, de manera que se tienen etapas similares a las del algoritmo evolutivo: etapa de selección para elegir a las dos soluciones para el procedimiento de PR, etapa de combinación (que en este caso es el proceso de Path Relinking), y etapa de reemplazamiento, en donde se determina si el nuevo individuo (generado por el proceso de Path Relinking) reemplazará a alguno de la población o si simplemente se ignorará.

En el Algoritmo 5 se muestra el pseudocódigo de un algoritmo poblacional híbrido, que utiliza el procedimiento de path relinking para generar nuevas soluciones explorando el espacio de búsqueda. El algoritmo comienzo generando una población inicial (línea 1), y cada una de las soluciones se le aplica una metaheurística de trayectoria (línea 2). De manera iterativa el algoritmo realiza las siguientes etapas: selección de padres (línea 5), combinación de las soluciones mediante una estrategia de Path Relinking (línea 7), mejora de las soluciones generadas considerando una metaheurística de trayectoria (línea 8) y una fase de actualización de la población con algún método de reemplazamiento (línea 9).

Al igual que el algoritmo evolutivo, este esquema de algoritmo poblacional híbrido con

Algoritmo 5: Path Relinking Híbrido

Salida: $sElite$ Mejor solución generada

```
1  pIni = generarPoblacionInicial();
2  pActual = mejorarIndividuos(pIni);
3  sElite = Elegir a la mejor solución en pActual;
4  mientras No Criterio de Paro hacer
5      p^1, p^2 = seleccionaPadres(pActual);
6      // generar rutas entre soluciones
7      muestras = combinarSoluciones(p^1, p^2);
8      cResMejora = mejorarSoluciones(muestras);
9      pActual = actualizarPoblacion(pActual, cResMejora) ;
10     sElite = Elegir a la mejor solución en pActual;
11 devolver sElite
```

path relinking tiene una gran flexibilidad para poder cambiar los componentes principales y combinarlos con diferentes estrategias para mejor el proceso de búsqueda del algoritmo.

2.6. Paralelización

Aunque el uso de metaheurísticas permite reducir significativamente el tiempo necesario para encontrar soluciones de buena calidad, algunos de estos métodos podrían aún requerir de una gran cantidad de recursos computacionales para poder obtener soluciones de alta calidad. Para problemas industriales o aquellos en donde se tiene una gran cantidad de variables, el tiempo podría llegar a ser un factor importante para lograr obtener una buena solución. Por ejemplo, en el TSP se ha logrado encontrar una solución óptima en una instancia con más de 30 mil ciudades considerando un criterio de paro de 10 días de ejecución utilizando un algoritmo híbrido (Segura y col., 2015).

El paralelismo es útil no solamente para reducir el tiempo para resolver un problema, sino también para mejorar la calidad de las soluciones que son generadas (Talbi, 2009). Algunos modelos de paralelización logran mejorar la calidad de las soluciones encontradas al considerar esquemas cooperativos entre diferentes procesos, ya que en muchos casos la paralelización produce un esquema que no es análogo al secuencial y puede tener implicaciones, por ejemplo, sobre el balance entre exploración e intensificación.

Existen diferentes modelos de paralelización de metaheurísticas, que utilizan distintos enfoques para realizar la paralelización (Alba, 2005; Sevkli y Aydin, 2007). En las siguientes secciones se describen los modelos de paralelización más comunes.

2.6.1. Paralelización en Metaheurísticas de Trayectoria.

Para los métodos de trayectoria, se distinguen principalmente tres modelos de paralelización:

- **Paralelización Multi-Inicio.** Se consideran diferentes ejecuciones, que pueden ser independientes o no, del método de trayectoria, con el fin de obtener mayor robustez en la calidad de las soluciones que se obtienen.
- **Paralelización de Movimientos.** En general, este tipo de paralelización no altera el comportamiento del método, ya que sólo se utiliza para realizar una evaluación en paralelo de los posibles movimientos que se pueden realizar desde una solución. El comportamiento del método en secuencial sería el mismo pero en un mayor tiempo de ejecución.
- **Paralelización de la Función de Evaluación.** Cuando la función de evaluación es computacionalmente costosa y puede ser vista como un cierto número de funciones de agregación o de funciones parciales, es posible implementar paralelización para acelerar la evaluación de un movimiento.

2.6.2. Paralelización en Metaheurísticas Poblacionales.

En los métodos basados en poblaciones, considerando también los algoritmos híbridos, se consideran los siguientes modelos de paralelización:

- **Ejecuciones Independientes.** Es el modelo más sencillo, no hay interacciones entre los procesos ya que las ejecuciones son independientes, y sólo al final se elige el mejor resultado de entre todas las ejecuciones realizadas. Se basa exclusivamente en que los optimizadores son estocásticos y se pueden iniciar con poblaciones diferentes.
- **Paralelización Global (Modelo Maestro-Esclavo).** Se define un proceso Maestro que se encarga de distribuir tareas a las operaciones en los otros procesos (esclavos). En general, el proceso maestro se encarga de realizar operaciones de selección y el resto de los procesos realizan operaciones de reproducción (recombinación, mutación y evaluación). En este modelo, la exploración del espacio de búsqueda es conceptualmente idéntica a la que se hace cuando se ejecuta el algoritmo de manera secuencial. Este modelo funciona bien cuando la tarea que deben realizar los procesos esclavos es computacionalmente costosa.

  En la Figura 2.2 se muestra una representación típica para este tipo de modelo de paralelización. En general, los procesos esclavos no tienen comunicación entre sí, pero si pueden existir dependencias en su ejecución, dependiendo de si en el proceso maestro se realizan o no pasos de sincronización.
- **Modelo Distribuido.** Este tipo de modelo también es conocido como modelo multipoblacional. Se manejan subpoblaciones de individuos, para las cuales se debe definir una política de migración entre las diferentes poblaciones, que incluye parámetros como probabilidad de migración, selección y reemplazamiento de individuos migrantes, topología, entre otros. Un ejemplo típico de este tipo de paralelización son los algoritmos híbridos que utilizan esquemas poblacionales basados en islas. En cada isla se

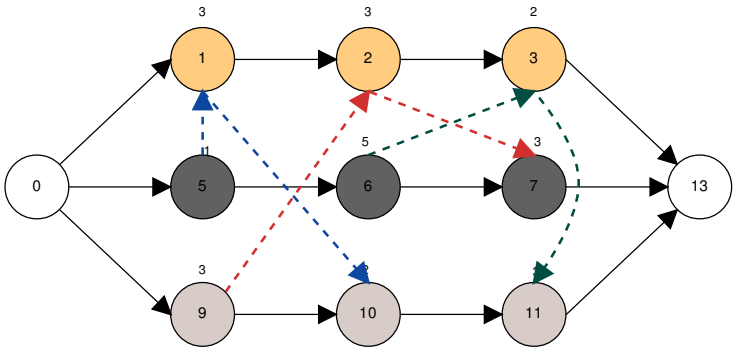


FIGURA 2.2: Ejemplo de modelo de Paralelización Maestro-Esclavo

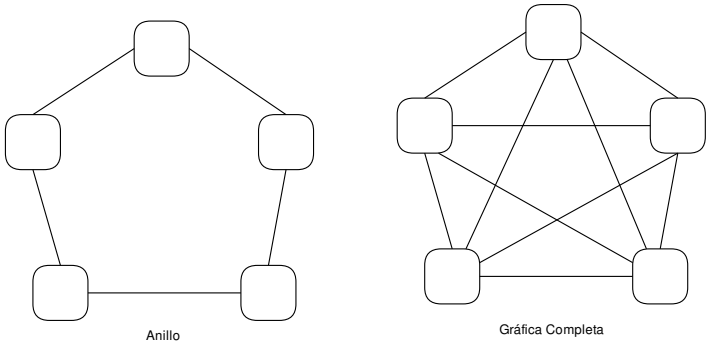


FIGURA 2.3: Ejemplos de modelos de Paralelización Distribuida.

ejecuta de manera independiente un algoritmo poblacional, aunque se mantiene cierta comunicación con los otras poblaciones (definida por las políticas de migración). En la Figura 2.3 se presentan dos topologías que se utilizan comúnmente en este tipo de paralelización; cada nodo de la gráfica representa una población de individuos, y las aristas definen quienes se comunican entre sí.

- **Modelo Celular.** También conocido como modelo de difusión, normalmente trabaja con una sola población, que se estructura en vecindades, y los individuos sólo pueden interactuar con sus vecinos. Así, cuando una buena solución es encontrada, esta se difunde lentamente hacia las otras vecindades, lo que puede retrasar los problemas de convergencia prematura. Al haber comunicaciones limitadas, solo se usa con individuos cercanos en la población, la paralelización en algunas arquitecturas se puede ver facilitada. En la Figura 2.4 se muestra un modelo de paralelización celular considerando una población de 9 individuos. A diferencia del modelo distribuido, en la gráfica cada nodo representa un individuo, y las aristas determinan los individuos con los que cada solución puede interactuar.
- **Modelos Híbridos.** Implementan características de diferentes modelos para tomar ventaja de cada uno. Combinaciones típicas son utilizar el modelo celular y el modelo de islas, cada isla incorporará los mecanismos del modelo celular.

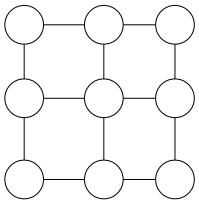


FIGURA 2.4: Ejemplo de modelo de Paralelización Celular.

En algoritmos evolutivos, uno de los modelos de paralelización más típicos es la paralelización global (maestro-esclavo), ya que por las características del algoritmo, la población puede descomponerse para manejarse de manera paralela. Típicamente, el proceso maestro maneja las etapas de selección y reemplazamiento, y los procesos esclavos se encargan de la evaluación de las soluciones; en algoritmos híbridos, los proceso esclavos comúnmente realizan la tarea de intensificación para cada solución, es decir, ejecutan la metaheurística de trayectoria. El proceso de recombinación (cruza y mutación) puede ser realizado por el proceso maestro o por los propios esclavos, dependiendo de las características de los operadores que se utilizan.

En esta tesis se usa un método de paralelización global. Para este modelo se distinguen dos tipos de Paralelización en Metaheurísticas Híbridas (Talbi, 2009):

- **Paralelización Síncrona.** Se tiene una coordinación entre todo el sistema, de manera que el proceso maestro se encarga de obtener la información de todos los procesos esclavos, para sincronizar su ejecución y actualizar la información; por ejemplo, la información de cual es la mejor solución encontrada hasta el momento. El trabajo que se debe realizar en una iteración se distribuye entre los esclavos, el proceso maestro espera hasta obtener los resultados de todos los esclavos para empezar la siguiente iteración. Este tipo de paralelización tiene un comportamiento idéntico al algoritmo en secuencial.
- **Paralelización Asíncrona** Cuando se considera paralelización asíncrona, no se tienen dependencias entre las ejecuciones de los procesos esclavos; el proceso maestro no requiere esperar a que todos los esclavos terminen para poder realizar una iteración del algoritmo.

Nótese que con el modelo asíncrono, la paralelización produce un algoritmo que no es análogo al esquema secuencial. La diferencia radica en que tras evaluar y realizar el proceso de mejora de un individuo, se pasa automáticamente al proceso de reemplazamiento, sin esperar a que el resto sean evaluados. En ese sentido se parece al modelo de estado estacionario, ya que se tiene un reemplazamiento en que se considera la población anterior y un individuo adicional. Sin embargo, tampoco es igual al modelo de estado estacionario, porque en este modelo sólo se considera un nuevo individuo en cada instante, mientras que en este caso se consideran múltiples individuos simultáneamente.

Capítulo 3

Estado del Arte

En este capítulo se presenta la definición formal del Problema de Planificación para Producción tipo Taller (JSSP, Job Shop Scheduling Problem) y algunos de los principales algoritmos del estado del arte para este problema.

3.1. Problema de Planificación para Producción Tipo Taller

3.1.1. Definición Formal

En el Problema de Planificación para Producción tipo Taller (JSSP, Job Shop Scheduling Problem) se tiene un conjunto de trabajos $J = \{J_1, \ldots, J_n\}$ y un conjunto de máquinas $M = \{M_1, \ldots, M_m\}$.

Cada trabajo $J_i \in J$ se compone de una secuencia con un orden predeterminado (restricciones de precedencia) de n_i operaciones $J_i = (O_{i,1}, \ldots, O_{i,n_i})$. Se tiene un total de $N = \sum_{i=1}^{n} n_i$ operaciones. En las instancias de prueba utilizadas en este trabajo $n_i = m, \forall i \in \{1, \ldots, n\}$ por lo que $N = n \times m$.

Para cada operación $O_{i,j}$ se tiene establecida la máquina $\mu_{O_{i,j}} \in M$ en la cual debe ser procesada por un tiempo fijo $d_{i,j}$ [1]. Cada máquina puede procesar a lo más una operación a la vez, y una vez que se comienza a realizar una operación, no se puede interrumpir este proceso (restricciones de recursos). Una planificación es una asignación de tiempos en los que se debe ejecutar cada una de las operaciones. Aunque existen diferentes medidas de desempeño que podrían funcionar como función objetivo, en este trabajo se considerará la medida más común en la literatura para este problema, minimizar el tiempo en el que se completa la última operación, también conocido como makespan o C_{max}.

Sea $O = \{O_0, O_{1,1}, \ldots, O_{n,m}, O_{N+1}\}$ el conjunto de todas las operaciones que deben ser planificadas; las operaciones O_0 y O_{N+1} representan operaciones ficticias, que tienen una duración 0, para marcar el inicio y el final, respectivamente, de los trabajos. Es posible asociar un número consecutivo a cada operación para utilizar un único índice, considerando lo

[1] Aunque el tiempo de asociado a cada operación podría ser un número real o entero, en las instancias consideradas en este trabajo se utilizan valores enteros no negativos.

siguiente $O_{i,j} = O_{(i-1)n+j}$. Utilizar un único índice permite facilitar la notación de información asociada a las operaciones, por ejemplo la duración (d_i, duración de la operación O_i) y la máquina asociada (μ_i, máquina asociada a la operación O_i). De acuerdo a la secuencia de cada trabajo, para cualquier operación $O_{i,j} \in O \setminus \{0, N+1\}$ se define la operación sucesora de $O_{i,j}$ en su respectiva secuencia de trabajo como: $JS[O_{i,j}] = O_{i,j+1}$, y la operación antecesora como $JP[O_{i,j}] = O_{i,j-1}$; para la primera operación $O_{i,1}$ de cada trabajo se tiene que $JP[O_{i,1}] = O_0$, y análogamente, para la última operación $O_{i,m}$ de cada trabajo se tiene que $JS[O_{i,m}] = O_{N+1}$.

Una planificación puede ser representada como un vector de los tiempos de finalización de todas las operaciones $(F_0, \ldots, F_N, F_{N+1})$.

Sea E_h el conjunto de operaciones que son procesadas en la máquina $h \in M$. Podemos formular el $JSSP$ como el siguiente problema de optimización:

$$C^*_{max} = \text{mín}\, makespan = \text{mín}\ C_{max} = \text{mín}\, F_{N+1} \tag{3.1}$$

sujeto a:

$$F_{JP[k]} + d_k \leq F_k; k \in \{1, \ldots, N+1\}, \tag{3.2}$$

$$F_j - F_i \geq d_i \vee F_i - F_j \geq d_j; \forall (i,j) \in E_h, \forall h \in M, \tag{3.3}$$

$$F_k \geq 0; k \in \{0, 1, \ldots, N+1\}. \tag{3.4}$$

La primera ecuación establece la función objetivo a minimizar: el tiempo en el que se completa la última operación. La segunda ecuación impone las restricciones de precedencia entre las operaciones de un mismo trabajo, la tercera ecuación establece la segunda restricción correspondiente a la precedencia entre las operaciones que se ejecutan en la misma máquina, de manera que sólo una operación es ejecutada a la vez en cada máquina, y la última ecuación implica que todos los tiempos deben ser no negativos.

Representación

Una forma de establecer una solución consiste en no utilizar los tiempos de inicio directamente, sino el orden en que se ejecutan las operaciones en cada máquina. De esta forma, representamos una planificación como $\pi = (\pi_1, \ldots, \pi_m)$, donde $\pi_h = (\pi_h(1), \ldots, \pi_h(|E_h|))$, corresponde a la permutación que determina el orden de procesamiento en la máquina h, a la cual se le llama modelo de gráfica-permutación (Nowicki y Smutnicki, 1996). Teniendo en cuenta esta representación, podemos calcular el tiempo de inicio de cada operación como el máximo del tiempo final de las operaciones de las que depende la misma.

Se define la gráfica dirigida $G = (O, R \cup E)$. Los vértices de la gráfica corresponden a las operaciones, y cada vértice tendrá un peso equivalente a la duración de la operación correspondiente. El conjunto R establece las relaciones de precedencia que se imponen en la secuencia de operaciones de cada trabajo:
$R = \{(O_0, O_{i,1}) | i = 1, \ldots, n\} \cup \{(O_{i,j}, O_{i,j+1}) | i = 1, \ldots, n, j = 1, \ldots, m-1\}$
$\cup \{(O_{i,m}, O_{N+1}) | i = 1, \ldots, n\}$.

En el conjunto E se consideran los arcos que representan las restricciones de recursos. Se relacionan aquellas operaciones que se deben ejecutar en una misma máquina:

$$E = \cup_{i=1}^{n} E_i$$

donde $E_i = \{O_k | \mu_{O_k} = m_i\}$.

Los tiempos de inicio de cada operación, dependen directamente del orden en el que sean planificadas en las máquinas, por lo cual, una planificación queda determinada si se establece el orden en el que las operaciones deben ser ejecutadas en cada una de las máquinas.

Dada una permutación π_i para cada máquina, se tiene una selección de las aristas en E_i, con lo cual es posible establecer una planificación para las operaciones. Las aristas no dirigidas del conjunto E son reemplazadas por aristas dirigidas que establecen la secuencia en que las operaciones serán procedas en cada máquina.

Determinar una planificación π es equivalente a elegir una gráfica $G(\pi) = (O, R \cup E(\pi))$, donde $E(\pi) = \cup_{i=1}^{n} E_i(\pi)$, y $E_i(\pi)$ es el conjunto de aristas que aparecen en la ruta dirigida que atraviesa todos los vértices asociados con E_i en el orden establecido en la permutación. Dada una permutación π, es posible generar la gráfica $G(\pi)$ que establece una dirección para las aristas no dirigidas de E_i en la gráfica G.

No todas las permutaciones generan soluciones válidas. Si la gráfica $G(\pi)$ es acíclica, entonces π es una planificación factible (Wang, 2012); si en la gráfica $G(\pi)$ existe algún ciclo la solución es no factible. La ruta más larga desde el vértice O_0 hasta O_{N+1} es llamada ruta crítica, y su longitud se corresponde con el makespan; el conjunto de operaciones continuas que se ejecutan en la misma máquina y están en la ruta crítica se denominan bloques críticos. Encontrar la mínima ruta crítica de todas las gráficas dirigidas $G(\pi)$ es equivalente a encontrar la solución óptima del JSSP, la solución correspondiente al mínimo makespan.

Notación

En la Tabla 3.1 se resume la notación utilizada en este trabajo para describir algunas propiedades y algoritmos para el JSSP.

Notación	Descripción
$O_{i,j}$	Operación $j-$ésima del trabajo i. Se le asocia un índice único considerando la siguiente regla: $O_{i,j} = O_{(i-1)n+j}$
O_i	Operación índice único: i
μ_i	Máquina asociada a la Operación O_i.
μ_{O_i}	Máquina asociada a la Operación O_i.
π	Conjunto de permutaciones que determinan el orden de procesamiento de cada uno de los trabajos.
π_i	Permutación que determina el orden de procesamiento en la máquina i.
s_i	Longitud de la ruta más larga desde O_0 hasta O_i (sin incluir la duración de O_i) en la gráfica $G(\pi)$, equivalentemente es el tiempo de inicio de la operación O_i.
q_i	Longitud de la ruta más larga desde la operación O_i hasta O_{N+1} (incluyendo la duración de la operación O_i) en la gráfica $G(\pi)$.
$JP[i]$	Operación predecesora en la secuencia de trabajo correspondiente a la operación O_i.
$JS[i]$	Operación sucesora en la secuencia de trabajo correspondiente a la operación O_i.
$MP[i]$	Operación predecesora en la secuencia de la máquina correspondiente a la operación O_i.
$MS[i]$	Operación sucesora en la secuencia de la máquina correspondiente a la operación O_i.
x, y	Se utilizan estas letras para denotar soluciones, es decir, planificaciones representadas como permutaciones que establecen el orden de ejecución en cada máquina.
x_i	Permutación que establece el orden de procesamiento de los trabajos en la $i-$ésima máquina.

TABLA 3.1: Notación utilizada para describir propiedades y algoritmos para el JSSP.

Ejemplo

Consideremos el ejemplo que se menciona en Wang, 2012. Típicamente se especifica el número de trabajos y de máquinas, junto con la secuencia y duración de las operaciones para cada trabajo. En el ejemplo que se muestra en la Tabla 3.2, se tiene que $n = 3$ y $m = 3$, es decir, se tienen 3 trabajos y 3 máquinas.

En este ejemplo, para el trabajo J_2 se tiene la secuencia $(m_1 \rightarrow m_3 \rightarrow m_2)$, por lo cual, J_2 primero se debe procesar en la máquina m_1 durante 1 unidad de tiempo, posteriormente se debe procesar en m_3 durante 5 unidades de tiempo y finalmente en la máquina m_2 durante 3 unidades de tiempo.

La gráfica correspondiente al ejemplo de la Tabla 3.2 se muestra en la Figura 3.1. Las aristas que relacionan operaciones que se ejecutan en la misma máquina se muestran con líneas punteadas de un mismo color, mientras que las aristas que relacionan las operaciones en la secuencia de cada trabajo se marcan con líneas continuas. De esta manera, cada renglón de

	Trabajos	Operaciones		
		1	2	3
Tiempo de Procesamiento	J_1	3	3	2
	J_2	1	5	3
	J_3	3	2	3
Secuencia de Procesamiento	J_1	m_1	m_2	m_3
	J_2	m_1	m_3	m_2
	J_3	m_2	m_1	m_3

TABLA 3.2: Ejemplo de instancia del JSSP 3×3, 3 trabajos y 3 máquinas.

la gráfica representa la secuencia de operaciones de cada uno de los trabajos. Arriba de cada vértice (nodo) aparece el peso asociado (duración de la operación).

Para obtener una solución, se debe hacer una selección y asignar una dirección en cada uno de los grupos de aristas correspondientes a las operaciones que se procesan en cada máquina, de manera que en la gráfica resultante no se generen ciclos; en cada grupo de aristas, correspondiente a cada máquina, la selección de aristas dirigidas debe ser tal que exista un camino que conecte todas las operaciones que son procesadas en la máquina correspondiente.

Para obtener una solución, se debe determinar una planificación de las operaciones, es decir, determinar el tiempo de inicio para cada operación. De acuerdo a la representación de gráfica-permutación, podemos obtener la planificación al considerar las permutaciones de operaciones para cada una de las máquinas. De esta forma, a partir de las permutaciones es posible determinar la selección y asignación de la dirección en cada uno de los grupos de aristas correspondientes a las operaciones que se procesan en cada máquina. Si en la gráfica que resulta al considerar una permutación para cada una de las máquina no se generan ciclos, se tiene una solución factible. Nótese que la permutación para cada máquina determina en cada grupo de aristas un camino en la gráfica que conecta todas las operaciones que son procesadas en la máquina correspondiente.

En la Figura 3.2 se muestra una posible solución a este problema, se puede comprobar que en la gráfica resultante no se tienen ciclos; si denotamos por x a la solución mostrada en la Figura 3.2, se tiene que $x = [x_1 = [O_{2,1}, O_{1,1}, O_{3,2}], x_2 = [O_{3,1}, O_{1,2}, O_{2,3}], x_3 = [O_{2,2}, O_{1,3}, O_{3,3}]]$ o equivalentemente la podemos denotar de las siguientes maneras: $x = [x_1 = [J_2, J_1, J_3], x_2 = [J_3, J_1, J_2], x_3 = [J_2, J_1, J_3]]$ o bien $x = [x_1 = [O_4, O_1, O_8], x_2 = [O_7, O_2, O_6], x_3 = [O_5, O_3, O_9]]$.

Una manera usual de presentar una solución es mediante un diagrama de Gantt, en el cuál se muestra de manera gráfica los bloques para cada operación y su planificación en las máquinas. En la Figura 3.3 se muestra la representación gráfica para la solución correspondiente a la Figura 3.2. El largo de cada bloque corresponde a las unidades de tiempo en el que debe ser procesada la operación.

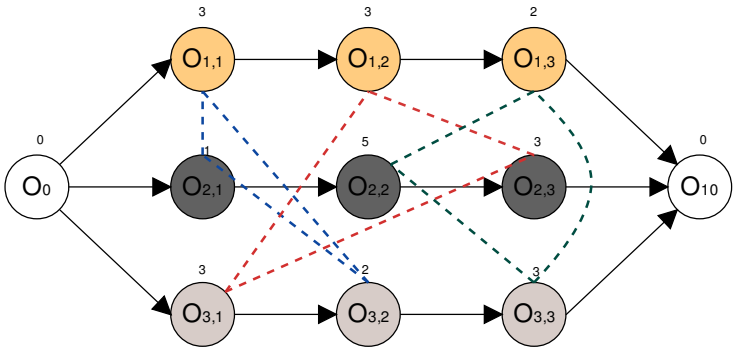


FIGURA 3.1: Ejemplo de representación gráfica de una instancia del JSSP.

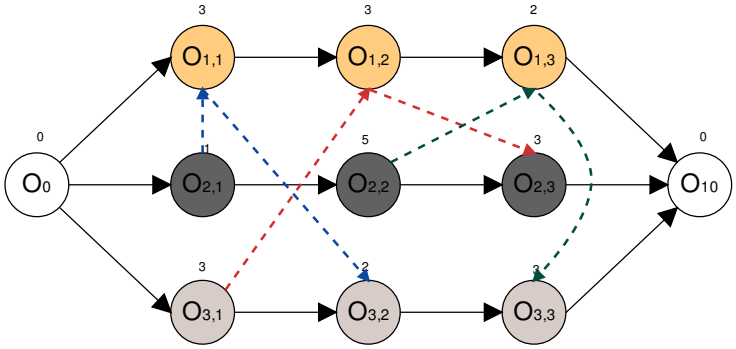


FIGURA 3.2: Ejemplo de una solución factible para la instancia mostrada en el Ejemplo 3.2.

Cuando en la gráfica G se realiza una selección para asignar una dirección a los arcos en el conjunto E, tal que se forman ciclos en la gráfica resultante $G(\pi)$, se obtiene una solución NO factible. En la Figura 3.4 se muestra un ejemplo de la gráfica que resulta con una solución no factible. En dicha solución, el ciclo se da con la secuencia de operaciones: $(O_1 \rightarrow O_2 \rightarrow O_9 \rightarrow O_{10} \rightarrow O_1)$. Por ello es habitual que los optimizadores basados en este tipo de representación utilicen mecanismos de reparación para convertir soluciones infactibles en soluciones factibles.

Tamaño del Espacio de Búsqueda

Con la representación elegida, dada una instancia con n trabajos y m máquinas, se tiene un espacio de búsqueda de $(n!)^m$, ya que para cada una de las m máquinas se debe elegir una permutación de tamaño n.

Debido a esto, sólo es posible utilizar métodos exactos en instancias pequeñas, con $n, m \leq 15$. Si se consideran tamaños más grandes, el espacio crece muy rápidamente; por ejemplo, si consideramos una instancias con 10 trabajos, $n = 10$, y 20 máquinas, $m = 20$, el tamaño del espacio de búsqueda sería de $(10!)^{20}$, aproximadamente $1{,}57 \times 10^{131}$.

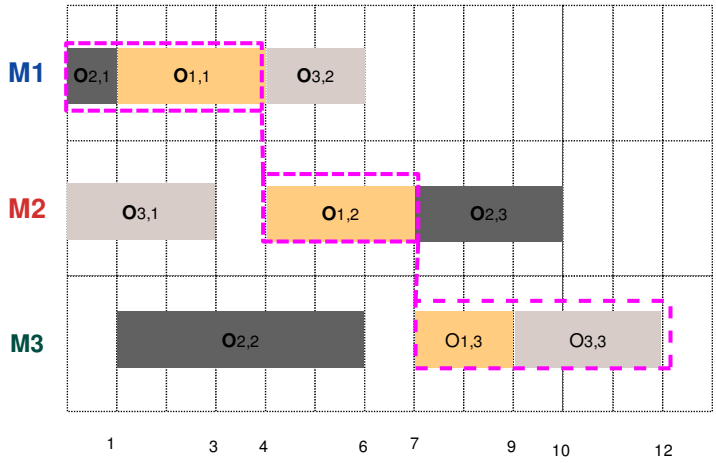


FIGURA 3.3: Ejemplo de una solución factible para la instancia mostrada en el Ejemplo 3.2.

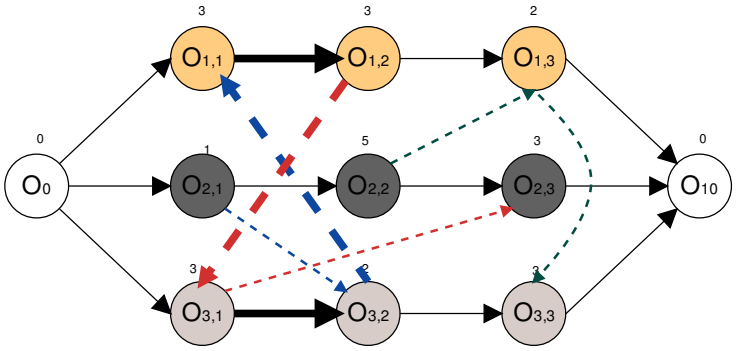


FIGURA 3.4: Ejemplo de una solución NO factible para la instancias mostrada en el Ejemplo 3.2 .

3.2. Similitud entre soluciones

Una buena exploración del espacio de búsqueda es una de las características deseables en los métodos de optimización. Para este fin, es necesario tener una medida para comparar que tan parecidas o similares son dos soluciones. En las siguientes secciones se introducen 2 medidas que han sido propuestas en la literatura (considerando la representación utilizada) para comparar la similitud entre dos soluciones.

Similitud considerando subsecuencia común más larga

La siguiente medida de similitud fue propuesta en Wang, 2012. Dadas dos soluciones $x = (x_1, \ldots, x_m)$, $y = (y_1, \ldots, y_m)$, consideramos la subsecuencia común más larga, SCL, entre cada una de las máquinas. Decimos que $SCL_i(x, y) = |SCL(x_i, y_i)|$ será el grado de $similitud_{SCL}$ entre x_i e y_i. De esta forma, podemos tener una medida de similitud entre dos soluciones, definida de la siguiente manera:

$$similitud_{SCL}(x, y) = \sum_{i=1}^{m} SCL_i(x, y)$$

Observamos que $similitud_{SCL} = n \times m$ soló si x e y son soluciones iguales. Además, N es una cota superior para el máximo valor en esta medida de similitud entre dos soluciones, ya que $1 \leq SCL_i(x, y) \leq n, \forall i \in \{1, \ldots, m\}$; $SCL_i(x, y) = n$ sólo sí $x_i = y_i$, es decir, los ordenes de procesamiento x_i e y_i son iguales.

La subsecuencia común más larga entre dos permutaciones se puede calcular de manera eficiente con programación dinámica (Cormen y col., 2009).

Disimilitud considerando el número de diferencias en la permutación

En Peng, Lü y Cheng, 2015 se considera la siguiente medida para comparar dos individuos. Dadas dos soluciones x, y consideramos el número de elementos que son diferentes en las permutaciones de las soluciones. Considerando este criterio, es posible definir una medida de similitud entre las soluciones:

$$NUM\_DIFF(x, y) = \sum_{i=1}^{m} diff_i(x, y)$$

donde,

$$diff_i(x, y) = \sum_{j=1}^{n} \mathbb{1}_{x_i[j]!=y_i[j]}$$

$diff_i$ es el número de elementos que son diferentes en la $i-$ésima máquina de las soluciones x e y.

A diferencia de la medida anterior $similitud_{SCL}$, NUM_DIFF se minimiza (con valor cero) cuando los individuos son idénticos, por lo que podemos considerar que NUM_DIFF es una medida de disimilitud entre los individuos. Nótese que esta medida es la suma de las distancias de Hamming de cada una las permutaciones asociadas a las diferentes máquinas, por lo que en esta tesis se denota como D_{HAM}.

3.3. Reparación de Soluciones

La representación de soluciones mediante permutaciones que representan el orden de procesamiento en cada una de las máquinas, no garantiza factibilidad. Dependiendo del tipo de operadores que utilizan las diferentes metaheurísticas para manipular las soluciones (por ejemplo cruza, combinación, vecindades, entre otros), podrían llegar a generarse soluciones no factibles (infactibles). En la gráfica correspondiente se tienen ciclos con lo cual se produce un bloqueo en las secuencias de trabajo.

Algoritmo 6: Verificación de Factibilidad de la solución

Entrada: $solucion$
Salida: Valor Booleano $solucionFactible$

```
1  O_J = {O_{1,1}, O_{2,1}, ..., O_{n,1}} ;
2  O_M = {π_1(1), π_2(1), ..., π_m(1)} ;
3  repetir
4      K = O_J ∩ O_M ;
5      O_J = O_J \ K ;
6      O_M = O_M \ K ;
7      Marcar todas las operaciones en K ;
8      para cada i ∈ K
9          Si JS[i] existe
10             O_J = O_J ∪ {JS[i]} ;
11         Si MS[i] existe
12             O_M = O_M ∪ {MS[i]} ;
13 hasta K == ∅;
14 devolver O_J ∪ O_M == ∅
```

En algunas metaheurísticas simplemente se ignoran las soluciones infactibles que se generan durante el proceso de búsqueda, pero es común que se utilice algún método para reparar, de manera que se vuelvan factibles, y poder considerar esas soluciones.

En las siguientes secciones se presenta un método para verificar si una solución es factible y una alternativa para reparar las soluciones cuando no cumplen factibilidad.

3.3.1. Verificación de Factibilidad

Debido a que no siempre se garantiza que las soluciones que se generan durante el proceso de búsqueda sean factibles, es necesario verificar si una solución que se genera, en un determinado momento de la búsqueda, es factible o no.

En el Algoritmo 6 se presenta el pseudocódigo para verificar si una solución es factible, propuesto en Lu y col., 2011. El algoritmo considera las secuencias de trabajo y las secuencias de procesamiento en las máquinas, en los conjuntos O_J y O_M, respectivamente.

En el conjunto O_J se consideran las operaciones para las que el algoritmo ya verificó que se pueden ejecutar sus predecesores en sus respectivos trabajos; en la línea 1 se consideran las operaciones iniciales de cada secuencia de trabajo. Análogamente, en el conjunto O_M se consideran las operaciones para las cuales el algoritmo ya verificó que es posible ejecutar sus predecesores en la secuencia de sus respectivas máquinas; en la línea 2 se consideran las operaciones iniciales en las permutaciones para las secuencias de las máquinas.

En cada iteración, se calculan las operaciones que no tienen predecesores pendientes (ni en la secuencia del trabajo ni en la secuencia de la máquina), y se marcan las operaciones que ya han sido procesadas, de manera que se va avanzando en las respectivas secuencias. Al

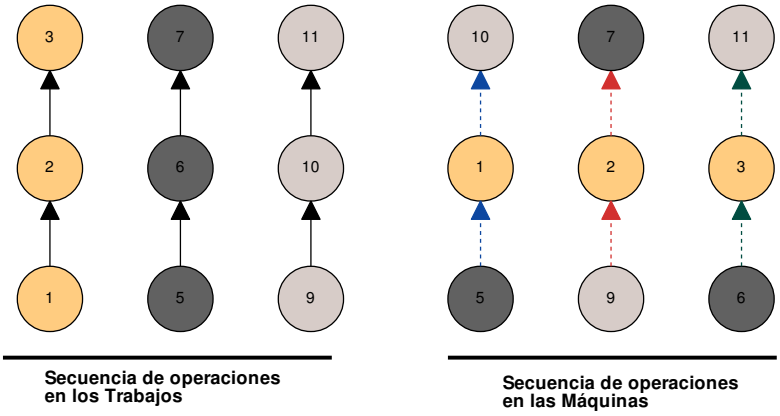


FIGURA 3.5: Secuencias de Procesamiento en una solución factible para la instancia del ejemplo de la Tabla 3.2.

final, si no quedaron pendientes operaciones en la secuencias de trabajo ni en las secuencias de máquinas, $O_J = O_M = \emptyset$, la solución es factible; en otro caso, $O_J, O_M \neq \emptyset$, se tienen operaciones que están bloqueadas por sus respectivas secuencias de trabajos y máquinas, y por lo tanto la solución no es factible.

El algoritmo es de complejidad $O(N)$, donde N es el total de operaciones, ya que es necesario procesar cada operación una sola vez. Esto se garantiza con el etiquetado de las operaciones ya visitadas, línea 7.

Ejemplo de Verificación de Factibilidad

Si consideramos la solución mostrada en la Figura 3.2 para el ejemplo de la Tabla 3.2, se puede verificar que la solución es factible utilizando el algoritmo anterior. En la Figura 3.5 se muestran las secuencias de las operaciones en los trabajos (izquierda) y en las máquinas (derecha). A partir de estas secuencias, en la Figura 3.6 se presenta una visualización gráfica de la ejecución del algoritmo. Cada línea divide las operaciones que son etiquetadas en cada iteración del algoritmo, por ejemplo etiquetando las operaciones O_5 y O_9 en la primera iteración, y la operación O_{11} en la última iteración.

Si aplicamos el algoritmo para la solución presentada en la Figura 3.4 para el ejemplo de la Tabla 3.2, se puede comprobar que el resultado del Algoritmo 6 será que la solución no es Factible. En la Figura 3.7 se presentan las correspondientes secuencias de trabajo para la solución considerada. En la Figura 3.8 se presenta la visualización gráfica de la ejecución del algoritmo. Se observa que sólo se realizan dos iteraciones; en la primera iteración se etiqueta a la O_5, y en la segunda iteración se etiqueta a la operación O_6. Después de estas dos iteraciones se tiene que $O_J = \{O_1, O_7, O_9\}, O_M = \{O_{10}, O_2, O_3\}$ por lo que $O_J \cap O_M \neq \emptyset$, por lo tanto la solución es No Factible.

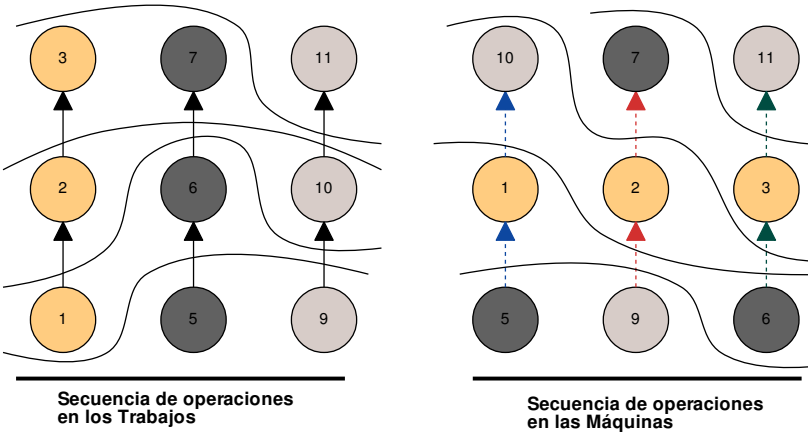


FIGURA 3.6: Verificación de Factibilidad en Solución Factible para la instancias del ejemplo de la Tabla 3.2.

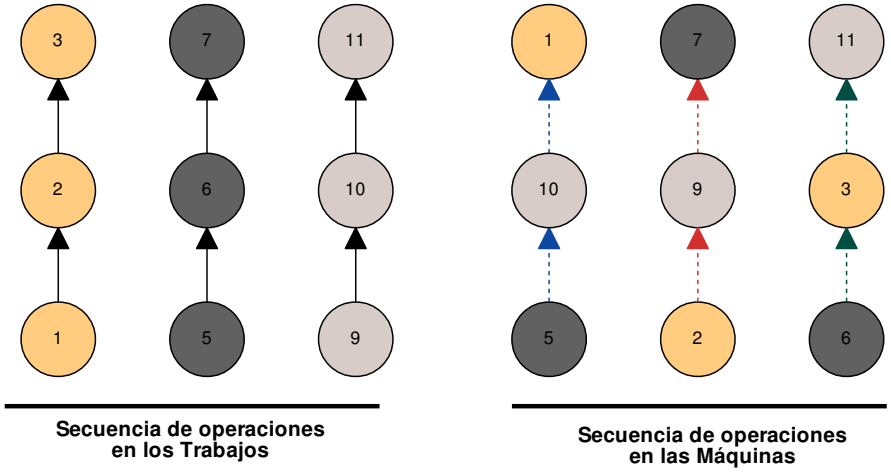


FIGURA 3.7: Secuencias de Procesamiento en una solución NO factible para la instancia del ejemplo de la Tabla 3.2.

3.3.2. Reparación considerando secuencia en los trabajos

En el JSSP, la secuencias para cada trabajo son fijas, son parte de los datos de cada instancia. Por lo tanto, las reparaciones sólo se pueden hacer modificando la secuencia en la que las operaciones son procesadas en cada una de las máquinas.

En el Algoritmo 7, se presenta un pseudocódigo, del método propuesto en (Lu y col., 2011), para reparar una solución. Este algoritmo considera la secuencia de trabajo de las operaciones que están en conflicto y las reordena en la máquina para eliminar el bloqueo.

El algoritmo comienza con los mismos pasos que el algoritmo de verificación. Si la solución ya es factible, el algoritmo termina sin generar ningún cambio en la solución. Si la solución no es factible, el algoritmo considera las operaciones en conflicto en el conjunto O_M, y las reorganiza de manera que la intersección con el conjunto O_J no sea vacía.

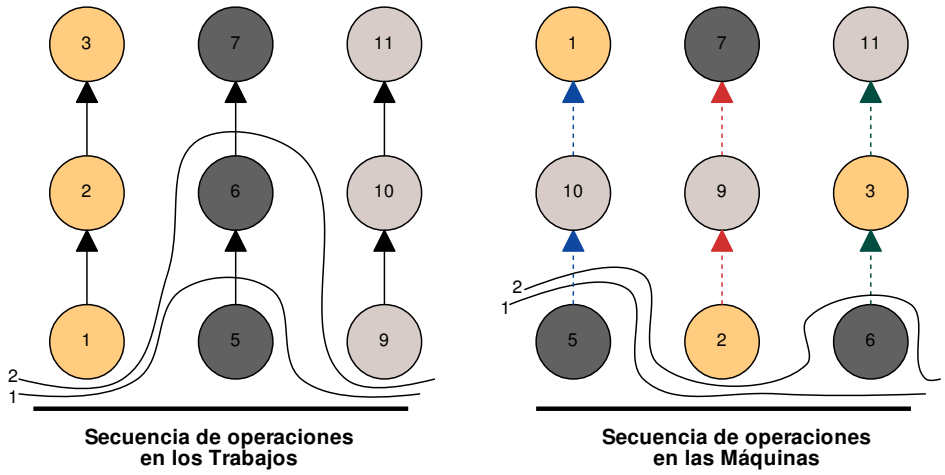


FIGURA 3.8: Verificación de Factibilidad en Solución No Factible para la instancia del ejemplo de la Tabla 3.2.

En la línea 18, si consideramos que para la operación seleccionada, O_k, la secuencia en su correspondiente máquina es:

$$\pi_{\mu_k} = (\pi_{\mu_k}(1), \ldots, \pi_{\mu_k}(c_{\mu_k} - 1), \pi_{\mu_k}(c_{\mu_k}), \pi_{\mu_k}(c_{\mu_k} + 1), \ldots, k, \ldots)$$

la posición de la operación O_k en la permutacion π_{μ_k} se denota por c_{μ_k}. Al reorganizar la operación O_k, se tiene que la nueva secuencia es

$$\pi_{\mu_k} = (\pi_{\mu_k}(1), \ldots, \pi_{\mu_k}(c_{\mu_k} - 1), k, \pi_{\mu_k}(c_{\mu_k}), \pi_{\mu_k}(c_{\mu_k} + 1), \ldots).$$

En la línea 19, se actualiza el conjunto O_M, ya que ahora la operación O_k no tiene predecesores pendientes en la máquina, y la operación $\pi_{\mu_k}(c_{\mu_k})$ ahora tiene como predecesor en la máquina a la operación O_K, por lo cual $\pi_{\mu_k}(c_{\mu_k})$ ya no debe pertenecer, en ese momento, al conjunto O_M. Una vez que se hace la reorganización de alguna de las operaciones en conflicto, el proceso se repite: se verifica si la solución ya es factible, si no lo es, se elige nuevamente alguna operación en conflicto para reorganizarla. El algoritmo termina cuando se logra verificar que las modificaciones han generado una solución que es factible.

En el proceso de reparación se pueden tener diferentes opciones para lograr generar una solución factible. En la línea 16 se tiene una elección aleatoria para seleccionar a la operación que se debe reorganizar con el fin de romper el bloqueo. En algunos casos se tienen varias opciones que pueden realizar mayores cambios en las permutaciones, respecto al número de intercambios de operaciones adyacentes que se necesitan para pasar de una permutación a otra. Cuando esto ocurre, consideramos que es preferible restringir la selección sólo a las operaciones que induzcan el menor cambio en la permutación, con el fin de que la operación de reparación no sea demasiado disruptiva, por lo cual en este tesis consideramos la variante propuesta en el mismo artículo de referencia (Lu y col., 2011), en donde se propone como

Algoritmo 7: Reparación de Soluciones Considerando la Secuencia de los Trabajos.

Entrada: $\pi = (\pi_1, \ldots, \pi_m)$ solución
Salida: π solución Factible

```
1  O_J = {O_{1,1}, O_{2,1}, ..., O_{n,1}} ;
2  O_M = {π_1(1), π_2(1), ..., π_m(1)} ;
3  repetir
4      repetir
5          K = O_J ∩ O_M ;
6          O_J = O_J \ K ;
7          O_M = O_M \ K ;
8          Marcar todas las operaciones en K ;
9          para cada i ∈ K
10             Si JS[i] existe
11                 O_J = O_J ∪ {JS[i]} ;
12             Si MS[i] existe
13                 O_M = O_M ∪ {MS[i]} ;
14     hasta K = ∅;
15     Si O_J ∪ O_M ≠ ∅
16         Elegir cualquier operación O_k del conjunto O_J ;
17         c_{μ_k} = índice de la operación O_k en la secuencia π_{μ_k} que pertenece al conjunto
            O_M ;
18         Reorganizar la operación O_k en la secuencia π_{μ_k} , moviendo O_k justo antes de
            π_{μ_k}(c_{μ_k}) ;
19         O_M = O_M \ {π_{μ_k}(c_{μ_k})} ;
20         O_M = O_M ∪ {O_k} ;
21 hasta O_J ∪ O_M = ∅;
22 devolver π
```

alternativa restringir la selección de operaciones a aquellas que producen un menor cambio en la permutación correspondiente. Representaremos por b_k el índice en la permutación π_{μ_k} de la operación O_k. Por lo que, ahora la elección en la línea 16 se hará de manera que se minimiza $b_k - c_{\mu_k}$. De esta manera aún se mantiene una reparación aleatoria, ya que puede haber más de una opción que minimice.

Ejemplo de Reparación

En la Figura 3.9 se muestra el caso cuando el algoritmo debe seleccionar una operación, línea 16. En el ejemplo, se tienen 3 opciones, ya que aún hay operaciones pendientes en cada uno de los trabajos. En este ejemplo, cada elección genera una solución factible diferente. Si consideramos la selección restringida a los movimientos de operaciones que inducen un menor cambio, se mantienen sólo dos opciones y se descarta la opción de mover la operación O_7, ya que cualquiera de los otros dos movimientos provocan un cambio menor.

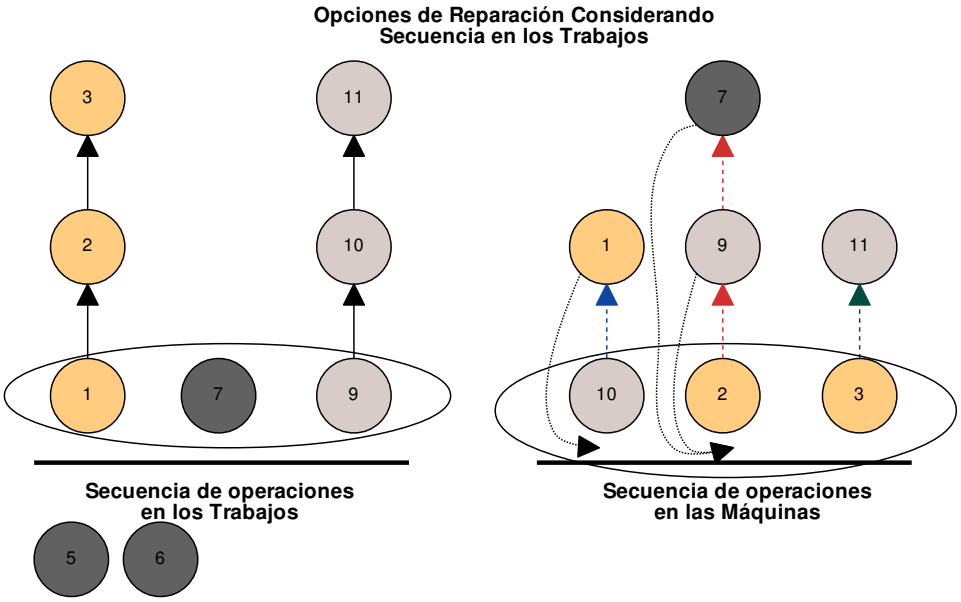


FIGURA 3.9: Reparación Considerando las operaciones en las Secuencias de Trabajos.

3.4. Principales Conjuntos de Prueba

Uno de los factores más importantes para poder evaluar y medir el rendimiento de los diferentes métodos para resolver un problema son los conjuntos de prueba (benchmarks), los cuales consisten en un conjunto de instancias de prueba del problema, con diferentes características, generalmente de diferentes tamaños. A partir de un conjunto de prueba, es posible realizar una comparación de los diferentes algoritmos para evaluar la calidad de los resultados obtenidos y el rendimiento computacional, considerando diferentes medidas de desempeño como son tiempo de ejecución y calidad de la solución obtenida.

Entre los principales conjuntos de prueba utilizados para comparar los diferentes métodos para el JSSP en (Hoorn, 2018) se mencionan los siguientes 8 conjuntos, donde la notación $n \times m$ indica que la instancia requiere procesar n trabajos en m máquinas.

- ft* (Fisher, 1963), 3 instancias, desde 6×6 hasta 20×5. Aunque este conjunto de prueba sólo consta de 3 instancias, una de ellas ($ft10$, 10×10) no fue resulta hasta 15 años después de que se publicara. Actualmente estas instancias aún se suelen utilizar para propósitos de comparación de tiempos de ejecución, principalmente en metaheurísticas de trayectoria. Este conjunto se considera fácil de resolver, y se conocen soluciones óptimas para todas las instancias del conjunto.
- la* (Lawrence, 1984), 40 instancias, desde 10×5 hasta 15×15. Para este conjunto de prueba ya se conocen todas soluciones óptimas para todas las instancias. Sin embargo, en algunas instancias aún es muy difícil alcanzar los valores óptimos (por ejemplo

las instancias $la29$ y $la40$), y solamente algunos métodos logran obtener soluciones óptimas. En general, este conjunto se considera de una dificultad de fácil a media.

- abz (Adams, Balas y Zawack, 1988), 5 instancias, desde 10×10 hasta 20×15. En este conjunto de prueba, se tienen instancias pequeñas, pero con una gran complejidad para poder solucionarse. Aún hay una instancia ($abz8$) para la cual no se ha logrado llegar a la cota inferior. Las instancias, $abz7$ y $abz9$ suelen clasificarse como difíciles de resolver.

- orb* (Applegate y Cook, 1991), 10 instancias, todas de 10×10. Este conjunto de prueba es el más fácil de resolver, para todas las instancias ya se conocen soluciones con los valores óptimos. Solo algunos artículos lo suelen utilizar para comparación.

- swv (Storer, Wu y Vaccari, 1992), 20 instancias, desde 20×10 hasta 50×10. En 5 instancias ($swv6-10$) no se ha logrado llegar a las cotas inferiores. Las últimos 5 instancias de este conjunto de prueba ($swv15-20$) suelen clasificarse como fáciles y generalmente no se utilizan para realizar comparaciones entre métodos, el resto de las instancias son consideradas como difíciles de resolver. En estos problemas el conjunto de máquinas es dividido en k $(1 \leq k \leq m)$ subconjuntos de igual tamaño, de manera que todos los trabajos son procesados considerando un orden parcial entre los subconjuntos. Eso significa que para todos los trabajos se tiene que primero deben ser procesados en las máquinas del primer subconjunto, antes de pasar a alguna máquina en el siguiente subconjunto y así sucesivamente. Para las instancias fáciles ($swv15-20$) $k = 1$, y para el resto de las instancias $k = 2$.

- yn (Yamada y Nakano, 1992), 4 instancias, todas de 20×20. Solo en 1 se ha conseguido la cota inferior conocida, es decir, se obtuvo una solución óptima, en las otras 3 no se ha logrado conseguir, aunque se tienen resultados cercanos.

- ta (Taillard, 1993), 80 instancias, desde 15×15 hasta 100×20. Este es uno de los conjuntos más utilizados para el JSSP. Para la mayoría de las instancias, 57, ya se han conseguido obtener los valores óptimos, sin embargo en 23 aún no se logran estos valores. En particular en las instancias de 20×20 y 30×20 no se ha logrado llegar a las cotas inferiores conocidas, por lo cual es posible que en algunas instancias aún se puedan encontrar mejores soluciones.

- dmu (Demirkol, Mehta y Uzsoy, 1998), 80 instancias, desde 20×15 hasta 50×20. Para este conjunto, en la mayoría de las instancias, 54, no se ha logrado alcanzar las cotas inferiores conocidas. En parte esto se debe a que las instancias de mayor tamaño ($dmu41 - dmu80$) presentan la características de que las máquinas están divididas en dos subconjuntos, de manera similar a las instancias en swv, es decir, se tiene que $k = 2$. En general, esta propiedad aumenta la complejidad de las instancias, para la mayoría de los algoritmos es difícil lograr resultados de alta calidad.

* Ya se han logrado obtener soluciones óptimas, aunque aún se utilizan para comparación en algunos artículos.

Con el objetivo de realizar nuestra validación experimental se consideraron los conjuntos más nuevos ya que, en general, son los que presentan más dificultad. Específicamente, se consideraron instancias de los conjuntos dmu, swv, y ta. Para las comparaciones realizadas en este trabajo, los mejores valores para los conjuntos de prueba se establecieron considerando los trabajos de Hoorn, 2018; Nagata y Ono, 2018; Gonçalves y Resende, 2014; *Personal Web Site Oleg S.*

3.5. Heurísticas Iniciales para resolver el JSSP

De acuerdo a (Cheng, Peng y Lü, 2016), las primeras propuestas para resolver el JSSP consideraban los siguientes dos tipos de estrategias:

- Procedimientos Heurísticos basados en reglas de prioridad para generar planificaciones. El tipo de reglas que se utilizan en este tipo de algoritmos pueden ser: elegir primero la operación de mayor duración, la operación de menor duración, entre otras.
- Heurísticas de desplazamiento de cuello de botella (Shifting Bottleneck Heuristic, Adams, Balas y Zawack, 1988). En estos métodos se realiza una asignación de las operaciones para cada máquina, considerando un orden establecido con base a la identificación de cuáles pueden crear cuellos de botella (de acuerdo a diferentes posibles criterios) y se realizan optimizaciones locales (por máquina) cada vez que se asigna una operación a una nueva máquina.

Estos métodos funcionaban bien en instancias pequeñas, sin embargo para instancias de mayor tamaño ($n, m \geq 15$) no logran obtener soluciones de buena calidad, por lo cual se empezaron a considerar metaheurísticas más complejas que permiten obtener soluciones de mejor calidad en un tiempo razonable. En las siguientes secciones se describirán los principales algoritmos aproximados que han sido propuestos para el JSSP. Primero se comentarán algunos métodos de trayectoria que han ido surgiendo, algunos de ellos se encuentran entre los métodos que actualmente obtienen los mejores resultados para el JSSP. En una sección posterior, se mencionarán los principales algoritmos híbridos que han logrado obtener buenos resultados, y que también están dentro del grupo de algoritmos que han obtenido los mejores resultados actuales para algunas instancias.

3.6. Metaheurísticas de Trayectoria para el JSSP

En las siguientes secciones se mencionan algunas de las principales metaheurísticas de trayectoria que han logrado obtener los mejores resultados en los principales conjuntos de prueba para el JSSP. En primer lugar se describirán de forma breve algunas de las estrategias más populares, y posteriormente se explicará en mayor detalle la estrategia que se decidió utilizar en esta tesis.

3.6.1. Algoritmos de Trayectoria Populares

Debido a la complejidad del JSSP, se han desarrollado diversas metaheurísticas de trayectoria para este problema. El recocido simulado fue una de las primeras metaheurísticas que se probaron para intentar resolver el JSSP, por ejemplo en (Van Laarhoven, Aarts y Lenstra, 1992). Con recocido simulado se logró resolver algunas de las instancias más fáciles de los principales conjuntos de prueba para el JSSP, sin embargo, instancias más grandes y de mayor dificultad no se lograban resolver. Taillard fue uno de los pioneros en proponer la búsqueda tabú (Taillard, 1994), para el JSSP. Posteriormente surgieron distintas variantes que proponían diferentes vecindades (Dell'Amico y Trubian, 1993; Nowicki y Smutnicki, 1996). Más tarde, surgieron propuestas que consideraban una búsqueda local guiada y diferentes estrategias de evaluación (Balas y Vazacopoulos, 1998; Eikelder y col., 1999).

A continuación se mencionan algunas de las estrategias más populares que han surgido en los últimos años.

TS/SA

TS/SA (Zhang y col., 2008) es una variante de la búsqueda tabú (TS) (Zhang y col., 2007), que será detallada en una sección posterior en este trabajo, en la que se utiliza una vecindad distinta (N6) y se incorporan características de recocido simulado. La idea es que el recocido simulado tiene propiedades de convergencia global, es decir, es posible demostrar que un algoritmo de recocido simulado converge a la solución óptima con probabilidad 1 si el tiempo de optimización es infinito. Por otro lado, la búsqueda tabú ha sido una de la metaheurísticas que mejores resultados han logrado obtener para el JSSP, por lo que la combinación de estas dos estrategias parece ser una buena idea.

Aunque este algoritmo logra obtener buenos resultados, sólo se mantiene competitivo con la TS propuesta anteriormente por los mismos autores, y en los resultados que se reportan en su artículo, no parece haber una diferencia significativa entre las dos variantes de la búsqueda tabú por lo que consideramos que el incremento de la complejidad en el diseño del algoritmo no queda completamente justificado.

GTA2017

GTA2017 es un algoritmo de Búsqueda Local Guiada (Shylo y Shams, 2018), que ha logrado una gran cantidad de nuevas cotas. El método utiliza una memoria de las mejores soluciones encontradas para predecir los componentes de las soluciones óptimas, para lo cual utiliza un modelo de regresión logística dentro de una búsqueda tabú. Este método, y una variante anterior (GTA2013), actualmente mantienen cerca del $50\,\%$ de las mejores soluciones conocidas para el conjunto de prueba dmu, el cual es uno de lo más difíciles en el JSSP.

IEBO

$IEBO$ es un Algoritmo de Búsqueda Local Guiada que considera soluciones parciales (Nagata y Ono, 2018). El método resuelve de manera iterativa el JSSP pero considerando una restricción adicional respecto al máximo valor que puede tener el makespan de las soluciones parciales. Se propone también una forma eficiente (utilizando programación dinámica) para hacer la exploración de la vecindad de las soluciones parciales. Dentro del método, se considera una etapa de intensificación local, mencionan que se puede utilizar cualquier método de trayectoria adaptándolo a trabajar con soluciones parciales; en el artículo consideraron la búsqueda tabú para su método propuesto. Reportaron 7 nuevos mejores valores conocidos, sin embargo las soluciones no están disponibles para su consulta en ningún sitio web; los valores que se reportaron en su artículo no han sido superados.

Otros métodos

En la literatura también aparecen algunos otros métodos que han sido competitivos, pero fueron superados por los métodos mencionados anteriormente. Entre estos métodos se encuentran:

- **GES** (Pardalos y Shylo, 2006) Algoritmo de Búsqueda que utiliza técnicas de Equilibrio Global.
- **AlgFix** (Pardalos, Shylo y Vazacopoulos, 2010), considera la propiedad de gran valle (big valley) que se presenta en el JSSP, considerando fases de intensificación y diversificación, en las cuales se realizan reinicios parciales.

Debido a que la búsqueda tabú es la que ha mostrado mejores resultados, en este trabajo se ha optado por elegir este método como mecanismo de intensificación, por lo cual en la siguiente sección se detalle de manera más específica la variante utilizada.

3.6.2. Búsqueda Tabú para el JSSP

En diferentes momentos, el JSSP ha sido abordado considerando diferentes métodos para resolverlo. La búsqueda tabú es uno de los métodos que más se ha estudiado (Taillard, 1994; Nowicki y Smutnicki, 1996; Nowicki y Smutnicki, 2005; Zhang y col., 2008).

A continuación vamos a describir las principales vecindades que se consideran para la representación del modelo de gráfica-permutación, descrito anteriormente en este trabajo.

Vecindades

Todas las vecindades que se consideran para la representación del modelo de gráfica-permutación se basan en realizar transformaciones en las rutas críticas que existen en la solución. Se tienen dos propiedades principales que son deseables en las vecindades:

- Propiedad de conectividad. A partir de cualquier solución inicial, es posible construir una secuencia de movimientos en la vecindad que nos lleven a una solución óptima.
- Propiedad de factibilidad. Todos los movimientos en la vecindad generan soluciones factibles.

Aunque es deseable que las vecindades cumplan con ambas, no siempre se puede garantizar o mantener estas propiedades.

En general, en las vecindades para el JSSP, se consideran todas las rutas críticas que se pueden generar a partir de una solución dada. Se ha demostrado que sólo el intercambio de operaciones que están en la ruta crítica pueden mejorar el makespan. Intercambiar operaciones que no están en la ruta crítica, no mejoran el makespan y pueden generar soluciones no factibles (Van Laarhoven, Aarts y Lenstra, 1992). En las siguientes secciones se describen las principales vecindades que se han sido utilizadas por diferentes metaheurísticas propuestas en la literatura.

N1

La vecindad N1 fue una de las primeras que se consideraron para el JSSP. Esta vecindad fue propuesta en (Van Laarhoven, Aarts y Lenstra, 1992) y en ella se se consideran todos los movimientos que surgen a partir del intercambio de operaciones adyacentes, que se ejecutan en la misma máquina, en la ruta crítica. Una de las principales ventajas de esta vecindad es que cumple con las dos propiedades mencionadas anteriormente. $N1$ tiene propiedades de conectividad y factibilidad. Además, se demuestra que si para una solución la vecindad N1 es vacía, entonces la solución es óptima (Eikelder y col., 1999); la vecindad es vacía sólo cuando todos los bloques tienen dimensión uno.

En la Figura 3.10 se muestra la representación de los posibles movimientos de esta vecindad en un bloque. Para este tipo de movimientos en (Taillard, 1994) se menciona un método para obtener una cota inferior en $O(1)$, lo que permite evaluar los movimientos rápidamente. Sin embargo, esta vecindad tiene una gran cantidad de movimientos, y una parte considerable de estos no llevan a ninguna mejora inmediata (intercambiar operaciones internas en los bloques).

N5

La vecindad N5 fue propuesta en (Nowicki y Smutnicki, 1996). Solo se consideran los vecinos que se producen con el intercambio de un solo arco en la ruta crítica. En particular, considera solo el intercambio de las operaciones extremas (inicial o final) de un bloque. Esta vecindad es más pequeña que otras vecindades, en particular los movimientos de esta vecindad son un subconjunto de N1.

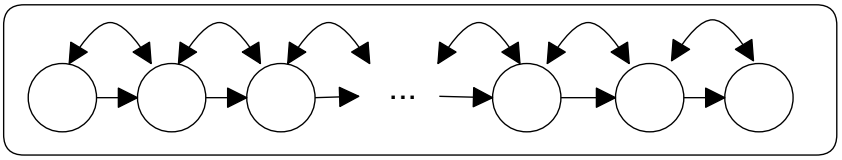


FIGURA 3.10: Vecindad N1

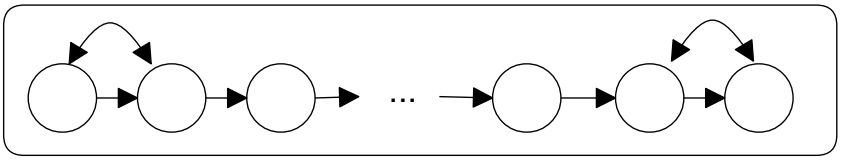


FIGURA 3.11: Vecindad N5

En la Figura 3.11 se muestra la representación de los posibles movimientos de esta vecindad en un bloque en la ruta crítica. Los movimientos a considerar son:
-Intercambiar la primera operación con la segunda (excepto para el primer bloque).
-Intercambiar la penúltima operación con la última (excepto para el último bloque).

N6

Esta vecindad fue propuesta en (Balas y Vazacopoulos, 1998). Se propone realizar movimientos que no se restringen solamente a intercambiar operaciones adyacentes. Los autores demostraron algunos teoremas que establecían condiciones suficientes para determinar cuando este tipo de movimientos podía producir soluciones no factibles, con lo que se lograba eliminar rápidamente este tipo de movimientos. Dado que se consideran movimientos más grandes (ya que no necesariamente se intercambian operaciones adyacentes), se logra conseguir soluciones de mejor calidad más rápidamente.

En la Figura 3.12 se muestra la representación de los posibles movimientos de esta vecindad en un bloque en la ruta crítica. En general se consideran dos tipos de movimientos:
- Mover una operación interna hacia el inicio del bloque.
- Mover una operación interna hacia el final del bloque.

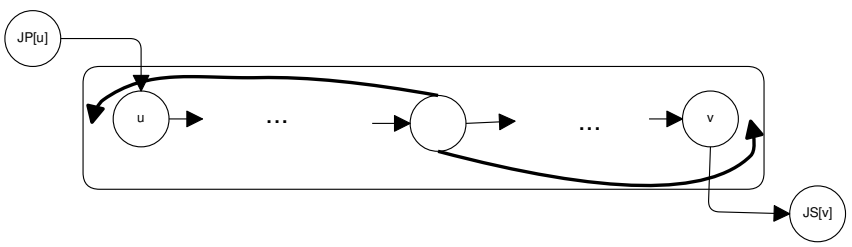


FIGURA 3.12: Vecindad N6

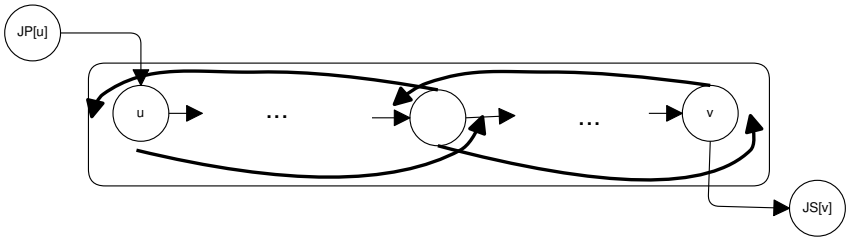


FIGURA 3.13: Vecindad N7

N7

La vecindad N7, propuesta en Zhang y col., 2007, se considera una extensión de la vecindad N6, en la cual los vecinos se generan al mover cualquiera de las operaciones internas a los extremos del bloque. La vecindad N7 se extiende al considerar también los vecinos que se generan al insertar las operaciones extremas (inicial o final) del bloque como operaciones internas.

En la Figura 3.13 se muestra la representación de los posibles movimientos de esta vecindad en un bloque en la ruta crítica. Movimientos a considerar:
- Insertar a la Derecha: si una ruta crítica contiene las operaciones u, v y JS[v] (sucesor de la operación v, en el mismo trabajo), insertar u a la derecha de v.
- Mover a la izquierda: si una ruta crítica contiene las operaciones u, v y JS[v], mover v a la izquierda de u.
- Insertar a la Izquierda: si una ruta crítica contiene las operaciones u, v y JP[u] (antecesor de la operación u, en el mismo trabajo), insertar v a la izquierda de u.
- Mover a la derecha: si una ruta crítica contiene las operaciones u, v y JP[u], mover u a la derecha de v.

Para esta vecindad, los autores demostraron teoremas que extienden a los propuesto para la vecindad N6, con lo cual se establecen condiciones suficientes para poder descartar movimientos que no son factibles.

Simplificación de la Vecindad

Además de proponer la vecindad N6, en (Balas y Vazacopoulos, 1998) los autores también propusieron un mecanismo para realizar una simplificación de la vecindad, después de que se produce un movimiento que empeora la calidad de la solución actual. Dado que se realiza un movimiento que empeora, es posible determinar en que parte de la ruta crítica se ha producido el incremento del makespan, por lo cual se puede restringir la búsqueda de mejores soluciones a una determinada parte de la ruta crítica. Con esto se logra acelerar significativamente la búsqueda, ya que la exploración de la vecindad es la etapa más costosa del proceso.

Algoritmo TS para el JSSP

En la búsqueda tabú, a partir de una solución inicial, que se establece como solución actual, se aplica un proceso de intensificación local. A la solución actual se le aplican pequeñas perturbaciones para generar nuevas soluciones llamadas vecinos. De manera iterativa, se elige al mejor vecino y se establece como solución actual. Este proceso se repite hasta cumplir algún criterio de paro. Para tratar de escapar de óptimos locales y a la vez evitar que la búsqueda se cicle, cuando se elige un movimiento, se almacenan los atributos asociados a ese movimiento con la finalidad de prohibir que se regrese a la solución anterior.

Una de las variantes de la búsqueda tabú que ha logrado mejores resultados, es la variante propuesta en (Zhang y col., 2007). En esta sección se describe el algoritmo propuesto en dicho trabajo. En el Algoritmo 8 se presenta el pseudocódigo general de esta variante. En esta variante, la vecindad utilizada es la $N7$, y adicionalmente se considera un mecanismo de simplificación similar al propuesto en (Balas y Vazacopoulos, 1998).

La Lista Tabú se implementa como una lista de atributos, en la cual se almacenan las secuencias y las posiciones de las operaciones involucradas antes de realizar el movimiento. De esta forma, si un movimiento provoca que se produzca alguna de las secuencias (en las mismas posiciones) que están en la lista tabú, el movimiento será prohibido. El tamaño de la lista tabú ha sido objeto de estudio, ya que tiene un efecto significativo en el proceso de búsqueda. Elegir un tamaño demasiado pequeño, podría no ser suficiente para prevenir los ciclos; por el contrario, un tamaño demasiado grande podría restringir demasiados movimientos, con lo cual no se tendría una buena intensificación. De manera general, se suele considerar un valor aleatorio entre límites que dependen del tamaño de la instancia. Al ser un valor aleatorio, se disminuye la probabilidad de que la búsqueda se cicle. En este trabajo estos límites se calcularon como $L_{min} = \lfloor 10 + n/m \rfloor$, $L_{max} = \lfloor 1{,}4 \times L_{min} \rfloor$ cuando $n \leq 2m$ y $L_{max} = \lfloor 1{,}5 \times L_{min} \rfloor$ en otro caso.

Tipos de Evaluación

Para determinar cual es el mejor movimiento se debe realizar una evaluación para cada vecino. Dado que en una vecindad se puede tener una gran cantidad de movimientos, realizar la evaluación de todos los vecinos suele tener un alto costo computacional, por tal motivo se han propuesto dos alternativas para realizar la evaluación: exacta y estimada.

Calcular el makespan de una solución, de manera exacta, se puede hacer con un algoritmo básico de etiquetado, en $O(N)$, recorriendo cada operación una vez para calcular los tiempos de inicio; el funcionamiento básico del algoritmo de etiquetado es comenzar con las operaciones que no tienen predecesores ni en los trabajos ni en las máquinas, asignar sus tiempos de inicio y ahora considerar a las operaciones sucesoras, el algoritmo es similar al de verificación de factibilidad que se presentó anteriormente en la Sección 3.3.1 de este trabajo.

Algoritmo 8: Búsqueda Tabú

Entrada: $solIni, maxSinMejora$
Salida: $sMejor$

```
1  sActual = solIni;
2  sMejor = sActual ;
3  LT = [] ;
4  numIt = 0 ;
5  mientras No Criterio de Paro hacer
6      evalAnt = sActual.makespan ;
7      mejorMov := encontrar el mejor vecino no prohibido o que cumpla el criterio de
         aspiración ;
8      Si Todos los vecinos están prohibidos por LT y ningún cumple el criterio de aspiración
9          mejorMov := elegir de manera aleatoria, con probabilidad uniforme, cualquier
             vecino ;
10     sActual = aplicarMovimiento(sActual, mejorMov) ;
11     Si sActual.eval <sMejor.eval
12         sMejor = sActual ;
13     Si evalAnt <sActual.makespan
14         simplificación de la vecindad de sActual para reducir posibles movimientos
             candidatos en la siguiente iteración. ;
15     tabuTenure = generar un valor aleatorio en el rango [L_min, L_max];
16     actualizarListaTabu(LT, mejorMov, numIt, tabuTenure) ;
17     numIt = numIt + 1 ;
18 devolver sMejor
```

En la evaluación exacta, se recalcula el valor del tiempo de inicio de todas operaciones. En Eikelder y col., 1999 se propuso el algoritmo BowTie para realizar la evaluación exacta de los vecinos, en el cual se consideran algunas mejoras para evitar recalcular los tiempos de inicio de todas las operaciones, y solo se consideran las operaciones que son afectadas por el movimiento.

La otra alternativa es utilizar una estimación del makespan de los vecinos. En la siguiente sección se detalla este enfoque, el cual es utilizado en este trabajo.

Evaluación por Estimación

La otra alternativa para evaluar a un vecino es utilizar una estimación del makespan.

Para realizar una evaluación por estimación, en Taillard, 1994 se propuso una estimación para movimientos que consisten en intercambios de operaciones adyacentes en bloques críticos (como $N1$ y $N5$). En Dell'Amico y Trubian, 1993 y en Balas y Vazacopoulos, 1998 se propusieron generalizaciones de esa estimación, para considerar los movimientos de otro tipo de vecindades (como $N3$ y $N4$), que consiste en intercambios hacia adelante o hacia atrás.

Algoritmo 9: Evaluación por Estimación.

Entrada: Q Secuencia de operaciones que se intercambian para obtener la nueva solución

Salida: Estimación de la evaluación del makespan

1 $l = |Q|$;
2 $a = Q[1]$;
3 $s'_a = \max(s_{JP[a]} + d_{JP[a]}, s_{MP'[a]} + d_{MP'[a]})$;
4 **for** $i = 2$ **to** l **do**
5 $\quad b = Q[i]$;
6 $\quad s'_b = \max(s_{JP[b]} + d_{JP[b]}, s'_a + d_a)$;
7 $\quad a = b$;
8 $b = Q[l]$;
9 $s'_a = \max(q_{JS[b]} + d_{JS[b]}, q_{MS'[b]} + d_{MS'[b]})$;
10 **for** $i = l - 1$ **to** 1 **do**
11 $\quad a = Q[i]$;
12 $\quad q'_a = \max(q_{JS[a]} + d_{JS[a]}, q'_b + d_b)$;
13 $\quad b = a$;
14 **devolver** $\max_{i=1,\ldots,l}(s'_{Q_i} + d_i + q'_{Q_i})$;

En el Algoritmo 14 se presenta el pseudocódigo del algoritmo de evaluación por estimación. Para realizar esta evaluación, el algoritmo considera la solución actual y la subsecuencia de operaciones (denotada por Q en el pseudocódigo) que se modifica cuando se realiza el movimiento. En el pseudocódigo, se denota por MP' y MS' a las operaciones predecesora y sucesora, respectivamente, en la secuencia que se produce cuando se realiza el movimiento.

En el primer ciclo (línea 4) se recalculan los tiempos de inicio, considerando la nueva subsecuencia que se produce si se realiza el movimiento. En el segundo ciclo (línea 10) se recalculan las longitudes de las rutas más largas desde las operaciones involucradas en el movimiento hasta la operación O_N, considerando un proceso análogo al primer ciclo.

Con las dos estimaciones realizadas, s' y q', se calcula la longitud de la ruta más larga que pasa por cada una de las operaciones involucradas en el movimiento, por lo que la estimación del makespan es el máximo valor de $s'_i + d_i + q'_i$ (línea 14).

En la Figura 3.14 se ejemplifica el movimiento que ocurre cuando la operación v se mueve a la izquierda de u; para los otros tipos de movimientos se tienen casos análogos. La evaluación por estimación consiste en iterar todas las operaciones (desde $u = Q[1]$ hasta $v = Q[l]$, considerando que hay un total de l operaciones entre u y v), y para cada operación se considera la ruta más larga que pasa por las operaciones sucesoras de sus respectivas secuencias de trabajo; de antemano los valores de la longitud de las rutas más largas que pasan por cada operación ya han sido calculadas (cuando se realiza la evaluación inicial de la solución, con una evaluación exacta). Al hacer un movimiento, se intenta reducir la ruta más larga, en la gráfica del modelo correspondiente, hacia la operación final (N). Sin embargo, esta evaluación sólo es una estimación ya que después de que se hace el movimiento, la ruta más larga

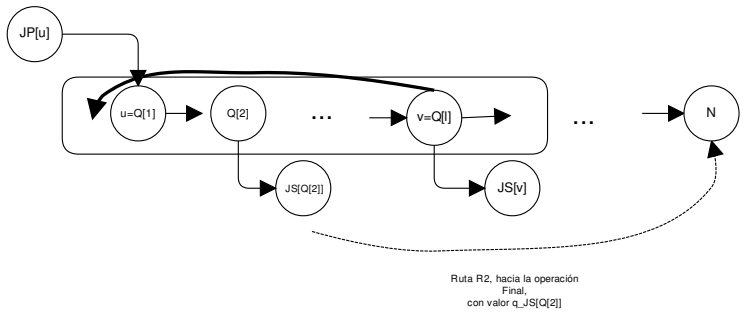


FIGURA 3.14: Representación de Evaluación por Estimación

no necesariamente tiene que pasar por todas las operaciones involucradas en dicho movimiento; cuando la nueva ruta crítica no contiene a las operaciones involucradas, entonces la estimación podría no coincidir con la evaluación exacta, ya que la estimación sólo considera la evaluación que se produciría cuando las operaciones involucradas en el movimiento sí están en la nueva ruta crítica.

3.7. Metaheurísticas Híbridas

En el desarrollo de algoritmos híbridos para el JSSP se han usado múltiples algoritmos poblacionales, como los basados en enjambres de partículas (Liu, Wang y Jin, 2007) o los basados en colonias de hormigas (Huang y Liao, 2008). Entre las diferentes estrategias, dos de las que mejores resultados han ofrecido están basadas en el algoritmo evolutivo y en el path relinking, las cuales han logrado obtener varios de los mejores resultados conocidos en los principales conjuntos de prueba para el JSSP. En esta tesis la validación experimental se realiza considerando estos dos tipos de estrategias, por lo que estos se explican en mayor profundidad. En las siguientes se describen los principales métodos híbridos que se han probado en el JSSP. Posteriormente se detallan los dos principales algoritmos híbridos del estado del arte para el JSSP, el HEA y TS/PR.

3.7.1. Algoritmos Híbridos Populares

Aunque los métodos de trayectoria han logrado obtener buenos resultados para el JSSP, también se han considerado metaheurísticas híbridas que intentan combinar las ventajas que ofrecen los esquemas poblacionales para lograr una mayor exploración con metaheurísticas de trayectoria para obtener una mejor intensificación en regiones promisorias. A continuación se describen brevemente los métodos híbridos más populares que se han propuesto para resolver el JSSP.

i-TSAB

Este método fue la referencia durante un gran período de tiempo (Nowicki y Smutnicki, 2005). El método utiliza como mecanismo de intensificación el algoritmo TSAB, que fue propuesto por el mismo autor unos años antes, la cual es una búsqueda tabú. El algoritmo trabaja con una idea similar al método de Path Relinking, utiliza dos soluciones, de un conjunto de soluciones promisorias, para explorar el espacio de búsqueda entre estas dos soluciones y generar una nueva solución. En el momento de su publicación, logró mejorar 91 de 112 instancias no resueltas.

BRKGA-JSP

Algoritmo evolutivo híbrido que utiliza una representación de claves aleatorias (random key representation (Gonçalves y Resende, 2014)). Considera una generalización de un método gráfico para resolver el JSSP. Propone una nueva vecindad, para la representación de claves aleatorias. Implementa un nuevo método de búsqueda local y en el momento de su publicación logró conseguir 57 nuevas cotas para los principales conjuntos de prueba. Aunque la mayoría de sus resultados ya han sido superados, aún hay algunas instancias en las cuales mantienen los mejores resultados, pero las soluciones no están disponibles para su consulta en ningún sitio web.

3.7.2. HGA

Este método también es un algoritmo evolutivo híbrido, en el cual se propone un método de selección para incrementar la diversidad en la población y evitar la convergencia prematura, de manera que se considera la aptitud del individuo y un valor de concentración (que depende de la similaridad que tiene el individuo con el resto de la población). El algoritmo también propone un nuevo operador de cruce, que extiende la idea del operador de cruce uniforme clásico, pero utilizado para conjuntos de permutaciones, en el que cada gen es una máquina completa (o una permutación completa). En este algoritmo también se propuso un operador de mutación (Wang, 2012).

Aunque el algoritmo obtenía buenos resultados considerando las cotas conocidas en el momento de su publicación, en el artículo correspondiente al algoritmo HEA se realizó la comparación contra el HGA, y se mostró la superioridad del HEA.

Operador de Cruce

En el Algoritmo 10 se muestra el pseudocódigo del operador de cruce que se utiliza en HGA. En este operador, primero se establecen dos conjuntos complementarios de máquinas, que se eligen de manera aleatoria con probabilidad uniforme. Para cada hijo, se heredan las secuencias completas de las máquinas de los padres. El primer hijo hereda del primer padre

Algoritmo 10: Operador de Cruce (recombinación) basado en las máquinas, utilizado por el algoritmo HGA.

Entrada: x, y Individuos seleccionados para aplicar recombinación.
Salida: h^1, h^2 nuevos individuos (hijos) generados a partir de x, y.

1 Dividir el conjunto $M = \{M_1, M_2, \ldots, M_m\}$ de máquinas en dos conjuntos complementarios A y B ;
2 Copiar en h^1 las cadenas de x correspondientes a las máquinas en A, y las cadenas de y correspondientes a las máquinas en B ;
3 Copiar en h^2 las cadenas de y correspondientes a las máquinas en A, y las cadenas de x correspondientes a las máquinas en B ;
4 **devolver** h^1, h^2

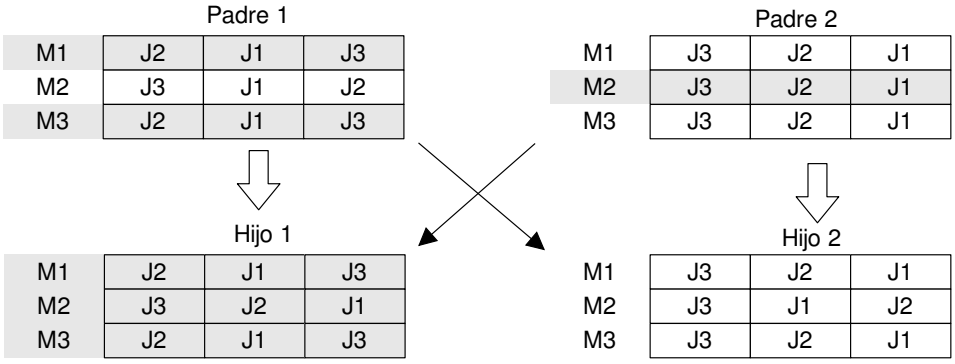


FIGURA 3.15: Ejemplo del operador de cruce por Máquinas Complementarias, utilizado en HGA.

todas las secuencias de las máquinas en el primer conjunto, y el resto las hereda del segundo padre; el segundo hijo es generado de manera análoga, hereda del segundo padre todas las secuencias de las máquinas del primer conjunto y el resto las hereda del primer padre. El operador no garantiza que las soluciones generadas sean factibles, por lo que es necesario realizar un paso posterior de reparación.

En la Figura 3.15 se muestra un ejemplo del operador de cruce utilizado en el algoritmo HGA. En este, se considera un conjunto de máquinas complementarias (elegido manera aleatoria con probabilidad uniforme), en la figura los conjuntos son $\{M_1, M_3\}$ y $\{M_2\}$. El primer hijo hereda las secuencias completas de las máquinas en el primer conjunto del primer padre, y el resto de las secuencias, para las otras máquinas, se heredan del segundo padre; el otro hijo se genera de manera análoga.

3.7.3. HEA

Este algoritmo fue propuesto en (Cheng, Peng y Lü, 2016). Es un algoritmo evolutivo híbrido, en el cual propusieron un nuevo operador de cruce y un mecanismo de reemplazamiento de individuos que toma en cuenta la calidad del individuo (el makespan) y la aportación a

Algoritmo 11: Operador de Cruce (recombinación) basado en la Subsecuencia Común más Larga (SCL), utilizado por el algoritmo HEA.

Entrada: x, y Individuos seleccionados para aplicar recombinación.
Salida: h^1, h^2 nuevos individuos (hijos) generados a partir de x, y.
1 Dividir el conjunto $M = \{M_1, M_2, \ldots, M_m\}$ de máquinas en dos conjuntos complementarios A y B. ;
2 Para cada máquina $M_a \in A$, copiar en h^1 la subcadena común más larga entre x, y manteniendo las operaciones en las posiciones de x, y copiar de y las operaciones faltantes en las restantes, manteniendo el orden de y. ;
3 Para cada máquina $M_b \in B$, copiar en h^2 la subcadena común más larga entre x, y manteniendo las operaciones en las posiciones de y, y copiar de x las operaciones faltantes en las restantes, manteniendo el orden de y. ;
4 **devolver** h^1, h^2

diversidad, medida como la mayor similitud a otro individuo en la población. En las siguientes secciones se detallan las principales características de este algoritmo.

Operador de Cruce

Dadas las soluciones x, y, se considera la Subsecuencia Común más Larga (SCL) en cada una de las máquinas. El primer hijo hereda las operaciones en la SCL en las mismas posiciones que se encuentran en el primer padre, el resto de operaciones se heredan manteniendo el orden relativo en la permutación del segundo padre. El segundo hijo se genera de manera similar; se heredan las operaciones en la SCL en las mismas posiciones que se encuentran en el segundo padre y el resto de operaciones se heredan considerando el orden relativo en la permutación del primer padre. El operador no garantiza que las soluciones generadas sean factibles, por lo que es necesario realizar un paso posterior de reparación.

En la Figura 3.16 se muestra un ejemplo del operador de cruce utilizado en el algoritmo HEA. A diferencia del utilizado en HGA, en este operador el cruce se realiza por cada máquina. En cada máquina se debe determinar la subsecuencia común más larga. El primer hijo hereda la subsecuencia manteniendo las posiciones del primer padre, y el resto de las posiciones se toman considerando el orden parcial en el segundo padre; el otro hijo se genera de manera análoga. En el ejemplo de la Figura 3.16, la subsecuencia común más larga es: J_2, J_3, J_4, por lo que el primer hijo hereda estas operaciones en las posiciones del primer padre, y las otras operaciones J_1 y J_5, se heredan en las posiciones restantes, considerando el orden parcial en el segundo padre.

Proceso de Intensificación

Considera la búsqueda tabú presentada en la sección 3.6.2 de este trabajo, solo modificando la forma en que se calcula el número de iteraciones que un movimiento estará en la lista

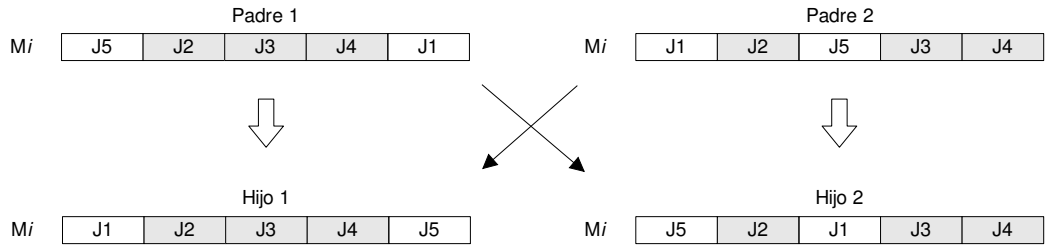


FIGURA 3.16: Ejemplo del operador de cruce por subsecuencia común más larga en cada Máquina, utilizado en HEA.

tabú (tabu tenure), asignándolo de la siguiente manera:

$$RL = \text{máx}\{(f - UB)/d_1, d_2\};$$

$$TabuTenure = tt + rand(RL);$$

donde:

- d_1, d_2, tt son parámetros constantes del algoritmo,
- f denota el makespan de la solución actual,
- $rand(RL)$ es un valor aleatorio entero que se genera con probabilidad uniforme en el rango $[0, RL]$,
- UB representa el valor de la mejor solución (mejor makespan) encontrada hasta ese momento.

La idea es que al inicio de la búsqueda, la diferencia entre f y UB será mucho mayor que el parámetro d_2, con lo que se permite que la búsqueda converja rápidamente a un óptimo local. Después de algunas iteraciones, la diferencia entre f y UB llegará a ser muy pequeña (ya que posiblemente se encontrará alguna zona plana que tenga diferentes soluciones de una calidad similar) y entonces el parámetro d_2 dominará esta etapa de la búsqueda tabú, con lo que se puede forzar a realizar una mayor intensificación.

Manejo de Diversidad

Para evitar el problema de convergencia prematura, en este algoritmo utilizan un mecanismo de control explícito de diversidad, específicamente en la fase de reemplazamiento. Se realiza una suma pesada entre el valor de la función objetivo (makespan) y la aportación a la diversidad, considerando la medida de similitud $similitud_{SCL}$, definida anteriormente en la sección 3.2.

Algoritmo 12: Estrategia de Actualización de la población en el algoritmo HEA.

Entrada: $P = x^1, \ldots, x^p$ Población de individuos.
$H = x^1, \ldots, x^h$ Población de hijos.
Salida: P Nueva población que combina la población P y los hijos H.

```
1 P' = P ∪ H
2 mientras |P'| > p hacer
3     x^u = argmin{g(i, P')|i = {1, ..., |P'|}} ;
4     P' = P' \ x^u ;
5 devolver P'
```

Para determinar qué individuos son seleccionados en la etapa de reemplazamiento, en este algoritmo se considera la calidad de la solución, medida por el makespan y la similaridad con otros individuos de la población. Dada una población P y un individuo $x^i \in P$, la aptitud del individuo se calcula como:

$$g(i, P) = \beta \tilde{A}(f(x^i)) + (1 - \beta)\tilde{A}(similitud_{SCL}(x^i, P))$$

donde:

- β es un parámetro que sirve para balancear entre la importancia dada al makespan y a la diversidad;
- $similitud_{SCL}(x^i, P) = \text{máx}\{similitud_{SCL}(x^i, x^j)|x^j \in P, j \neq i\}$, es el máximo grado de similitud del individuo x_i con otro individuo de la población P;
- $f(x^i) = makespan$ de la solución x^i;
- $\tilde{A}(y) = \frac{y_{max} - y}{y_{max} - y_{min} + 1}$ es una función para normalizar los valores de makespan y similitud; y_{max},y_{min} representan el máximo y el mínimo valor respectivamente, en la población P; en el primer término de $g(i, P)$ se hace referencia al makespan y en el segundo a la $similitud_{SCL}$. El último número 1 en el denominador se utiliza para evitar divisiones por 0.

En el Algoritmo 12 se presenta el pseudocódigo de la estrategia de reemplazamiento. En primer lugar se une a los individuos de la población anterior con la descendencia, y a continuación se itera, eliminando en cada iteración al individuo que minimiza la función previamente descrita. Esto se realiza hasta que la población tiene el número de individuos deseado.

Primero se calcula la aptitud de cada uno de los individuos, y al final se eligen a los individuos que minimizan la función g para permanecer en la población.

Algoritmo 13: Esquema general del HEA

Salida: x^* Mejor solución encontrada

1 $P = \{x^1, \ldots, x^p\} \leftarrow$ Inicialización de la Población ;
2 **for** $i = 1, \ldots, p$ **do**
3 $\quad x^i \leftarrow Búsqueda\_Tabu(x^i)$;
4 **mientras No** *Condiciones de Paro* **hacer**
5 $\quad$ Elegir de manera aleatoria, con probabilidad aleatoria, dos individuos, x^j y x^k, de P ;
6 $\quad (x^{p+1}, x^{p+2}) \leftarrow Operador\_Recombinación(x^j, x^k)$;
7 $\quad x^{p+1} \leftarrow Reparar(x^{p+1})$;
8 $\quad x^{p+2} \leftarrow Reparar(x^{p+2})$;
9 $\quad x^{p+1} \leftarrow Búsqueda\_Tabu(x^{p+1})$;
10 $\quad x^{p+2} \leftarrow Búsqueda\_Tabu(x^{p+2})$;
11 $\quad P = Actualización\_Población(P, \{x^{p+1}, x^{p+2}\})$;
12 x^* = Elegir la mejor solución de P ;
13 **devolver** x^*

Esquema General HEA

Los componentes anteriores se integran de una forma bastante estándar en un algoritmo evolutivo de estado estacionario, específicamente como se ilustra en el Algoritmo 13. En concreto, primero se genera una población inicial de manera aleatoria (línea 1) y se les aplica una búsqueda tabú para conseguir mejorar a las soluciones (línea 3). Posteriormente, de manera iterativa se realiza el proceso evolutivo, primero seleccionando de manera aleatoria con probabilidad uniforme (línea 5) a dos individuos que serán los padres a los cuales se les aplicará el operador de cruce (línea 6) para generar dos hijos. A los hijos se les aplica un operador de reparación para generar soluciones factibles (líneas 7 y 8) y después se utiliza el mecanismo de intensificación (búsqueda tabú) para mejorar dichas soluciones (líneas 9 y 10). Al final de cada iteración, se realiza el proceso de reemplazamiento (línea 11), considerando el Algoritmo 12 descrito anteriormente, para seleccionar a los mejores individuos.

Configuración de Parámetros

En la Tabla 3.3 se muestra la configuración de parámetros propuesta por los autores del algoritmo HEA. El parámetro más sensible es β, el cual determina la importancia que se le da a la función objetivo y a la diversidad (respecto al individuo con mayor similitud); este parámetro es difícil de calibrar, ya que dependiendo de la instancia, las características de la función de evaluación (landscape) podrían ser diferentes y tener un mayor o menor impacto respecto a la distribución de las mejores soluciones, y por tanto la ponderación con ese parámetro no necesariamente es la mejor para todas las instancias.

Parámetro	Descripción	Valor
tt	Constante 1 para Tabu Tenure en TS	2
d_1	Constante 2 para Tabu Tenure en TS	5
d_2	Constante 3 para Tabu Tenure en TS	12
α	Máximo de iteraciones sin mejora en TS	12000
β	Coeficiente para la función de aptitud	0.6
p	Tamaño de la población	30

TABLA 3.3: Configuración de Parámetros del Algoritmo HEA.

Algoritmo 14: Algoritmo TS/PR para el JSSP

Salida: $sElite$ Mejor Solución generada

$pIni = generarPoblacionInicial()$;
$P = mejorarIndividuos(pIni)$;
$cjtoPares = \{(S^i, S^j) | S^i, S^j \in P, S^i \neq S^j\}$
$sElite$ = Elegir a la mejora solución en P;
mientras No *Condiciones de Paro* **hacer**
    $(h^1, h^2) = seleccionarPadres(cjtoPares)$;
    $mejorCandidato_1 = pathRelinking(h^1, h^2)$;
    $mejorCandidato_2 = pathRelinking(h^2, h^1)$;
    $cResMejora_1 = aplicarTS_{fuerte}(mejorCandidato_1)$;
    $cResMejora_2 = aplicarTS_{fuerte}(mejorCandidato_2)$;
    $P = seleccionarMejores(P \cup \{cResMejora_1, cResMejora_2\})$;
    $cjtoPares = actualizarConjuntoPares(P)$;
    $sElite$ = Elegir a la mejora solución en P;
devolver $sElite$

3.7.4. TS/PR

Algoritmo Poblacional que incorpora un proceso de búsqueda tabú dentro del método de Path Relinking (Peng, Lü y Cheng, 2015). En el momento de su publicación, el método logró superar 49 de los mejores valores conocidos en los principales conjuntos de prueba.

En el Algoritmo 14 se presenta el pseudocódigo de TSPR. La etapa de reemplazamiento sólo considera una estrategia de reemplazo de los peores, no considera la diversidad de ninguna manera. El algoritmo utiliza el mismo método de intensificación que el algoritmo HEA, es decir la búsqueda tabú que se explicó en secciones anteriores en este trabajo.

Método de combinación por Path Relinking para el JSSP

En el Algoritmo 15 se presenta el pseudocódigo del procedimiento de path relinking para el JSSP. El primer paso es obtener el conjunto de operaciones que están situadas en posiciones diferentes en sus permutaciones correspondientes, en el algoritmo este conjunto se denota como NCS. Posteriormente, se van eligiendo elementos, de manera aleatoria con probabilidad uniforme, de este conjunto para hacer que la solución candidata (que en el primer paso es la solución inicial) sea más parecida (de acuerdo a la diferencia en la secuencia,

Algoritmo 15: Algoritmo Path Relinking para el JSSP

Entrada: $solIni, solRef$ Solución Inicial y solución guía para generar la ruta.
Salida: $mejorCandidato$ Mejor Solución en la ruta generada entre $solIni$ y $solRef$

```
1  solC = solIni;
2  candidatosRuta = ∅;
3  mientras similitud_DIFF(solIni, solC) < α hacer
4      Seleccionar de manera aleatoria un elemento solRef_i ∈ NCS(solC, solRef) ;
5      solC := Intercambiar el elemento de solC_i con otro en solC, de manera que la
         posición de solC_i queda igual que en solRef_i. ;
6  candidatosRuta = candidatosRuta ∪ {solC} ;
7  mientras similitud_DIFF(solC, solRef) > α hacer
8      solAnt = solC ;
9      mientras similitud_DIFF(solAnt, solC) < β hacer
10         Seleccionar de manera aleatoria un elemento solRef_i ∈ NCS(solC, solRef) ;
11         solC := Intercambiar el elemento de solC_i con otro en solC, de manera que la
             posición de solC_i queda igual que en solRef_i. ;
12     candidatosRuta = candidatosRuta ∪ {solC} ;
13 para cada sol ∈ candidatosRuta
14     sol = reparSolucion(sol) ;
15     sol = Búsqueda_Tabú_ligera(sol) ;
16 mejorCandidato = Elegir mejor solución en candidatosRuta.;
17 devolver mejorCandidato
```

considerando posición a posición). Las soluciones que serán muestreadas para ser consideradas como candidatas se determinan de acuerdo a los parámetros α y β, que establecen una distancia mínima entre las soluciones que se considerarán.

Al final, a las soluciones que se eligieron en el conjunto de candidatos, se les aplica una búsqueda tabú ligera, que corresponde a establecer el criterio de paro a un máximo de s_i iteraciones sin mejora.

En la Figura 3.17 se muestra un ejemplo del operador de Path Relinking utilizado en TSPR. En cada paso se va reduciendo la distancia de la solución candidata a la solución de referencia. Dependiendo de los valores de los parámetros α y β, se decide cuales de las soluciones muestreadas se considerarán. Como se puede notar, el procedimiento es aleatorio, ya que dependiendo del orden en que se eligen los elementos que son diferentes (entre la solución inicial y la solución de referencia), se pueden llegar a generar diferentes combinaciones para las soluciones candidatas. Adicionalmente, el procedimiento considera la reparación de las soluciones candidatas, y el procedimiento de reparación también tiene un componente aleatorio. Finalmente, la búsqueda tabú ligera, con un número muy limitado de iteraciones sin mejora, ayuda a que los candidatos lleguen a soluciones más promisorias de la región que se está explorando para mejorar la intensificación en el proceso de búsqueda.

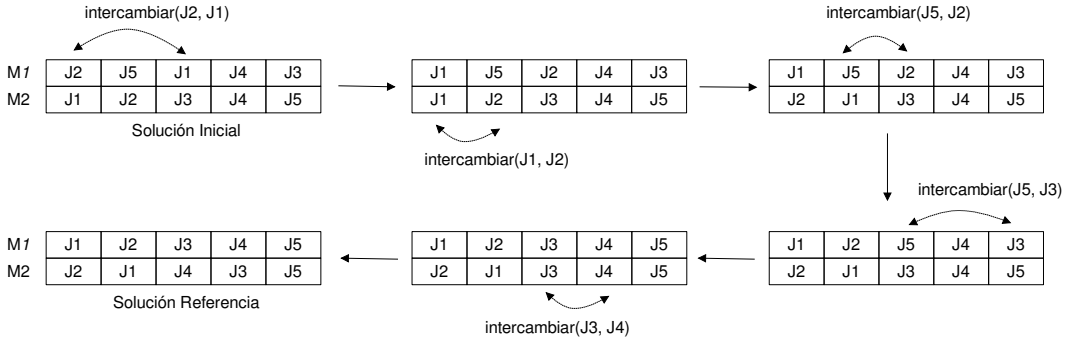


FIGURA 3.17: Ejemplo del operador de Path Relinking, utilizado en TSPR.

Parámetro	Descripción	Valor
α	Distancia mínima que hay entre las soluciones en $PathSet$ y las soluciones de inicio (S^i y S^G).	$\frac{dis}{5}$
β	Distancia mínima entre cualquier par de soluciones en $PathSet$.	$\max(\frac{dis}{10}, 2)$
s_i	Máximo número de iteraciones sin mejora para TS ligera	500
l_i	Máximo número de iteraciones sin mejora para TS pesada	12500
p	Tamaño de la población	30

TABLA 3.4: Configuración de Parámetros del Algoritmo TS/PR.

Configuración de Parámetros

En la Tabla 3.4 se detalla la configuración de parámetros propuesta por los autores del algoritmo. Se observa que el tamaño de población es el mismo valor, 30, que en el caso del HEA y el factor de intensificación (número de iteraciones sin mejora para TS) es similar, 12,000 en HEA y 12,500 para TS/PR.

El parámetro α establece la distancia mínima entre las soluciones candidatas y las soluciones de entrada ($solIni$ y $solRef$). El parámetro β se utiliza para definir la distancia que hay entre cada una de las soluciones candidatas que se agregan a $PathSet$. Estos dos parámetros establecen el número de candidatos que se considerarán en la ruta entre las dos soluciones. En la configuración propuesta en el artículo de referencia, se consideran 7 soluciones candidatas en la ruta. De esta manera se mantiene cierto grado de diversidad, ya que se evita que soluciones que son muy similares sean muestreadas en la ruta; si se considerarán estas soluciones que son demasiado cercanas, la búsqueda tabú podría llegar a la misma zona que la solución inicial y por tanto no se realizaría una buena intensificación en otras áreas.

Capítulo 4

Método Propuesto

En este capítulo se describe el método propuesto para el JSSP y los componentes que se utilizan en cada uno de los pasos del método.

4.1. Algoritmo Memético

La propuesta consiste en un algoritmo memético que permita mantener un control explícito de la diversidad. En el Algoritmo 16 se presenta el pseudocódigo del método propuesto, el cual es un algoritmo memético de estado estacionario (steady-state). El algoritmo comienza generando una población inicial de manera aleatoria, y a cada individuo se le aplica una búsqueda tabú para lograr obtener mejores soluciones iniciales. En cada generación (iteración) del algoritmo, se eligen a los padres y se aplica un operador de recombinación para generar a dos nuevos individuos (hijos), los cuales serán reparados y mejorados con la búsqueda tabú y agregados temporalmente a la población actual. Al final de cada generación, se realiza un proceso de actualización de la población, de manera que se eliminarán a dos individuos, que podrían ser los individuos generados, o alguno otro de la población.

Las novedades del método propuesto frente a lo publicado anteriormente son las siguientes: se incluyen dos formas diferentes de realizar el reemplazamiento, ambas basadas en realizar un control explícito de diversidad, se propone una nueva forma de calcular distancias entre individuos, se desarrolla un nuevo mecanismo para cruzar individuos y se detalla una forma de simplificar vecindades para implementar Tabu Search de forma eficiente. Además, se propone una paralelización del método. En las siguientes secciones se describe a detalle cada uno de los componentes del algoritmo.

4.1.1. Inicialización de la Población

La población inicial se genera considerando soluciones aleatorias que sean factibles. Para este trabajo se consideró una variante del algoritmo de inicialización propuesto en Dell'Amico y Trubian, 1993, en donde se consideran heurísticas (reglas de prioridad) para seleccionar el orden de planificación de las operaciones. En este trabajo se opta por una variante, en donde no se considera ninguna heurística para realizar la planificación de operaciones, y la

Algoritmo 16: Esquema general del método propuesto - Algoritmo Memético.

Salida: x^* Mejor solución encontrada

1 $P = \{x^1, \ldots, x^p\} \leftarrow$ Inicialización de la Población
2 **for** $i = 1, \ldots, p$ **do**
3 $\quad x^i \leftarrow Búsqueda\_Tabu(x^i)$
4 **mientras No** *Condiciones de Paro* **hacer**
5 $\quad$ Elegir de manera aleatoria, con probabilidad uniforme, dos individuos, x^j y x^k, de P
6 $\quad (x^{p+1}, x^{p+2}) \leftarrow Operador\_Recombinación(x^j, x^k)$
7 $\quad x^{p+1} \leftarrow Reparar(x^{p+1})$
8 $\quad x^{p+2} \leftarrow Reparar(x^{p+2})$
9 $\quad x^{p+1} \leftarrow Búsqueda\_Tabu(x^{p+1})$ $x^{p+2} \leftarrow Búsqueda\_Tabu(x^{p+2})$
10 $\quad P = Actualización\_Población(P, \{x^{p+1}, x^{p+2}\})$
11 x^* = Elegir la mejor solución de P ;
12 **devolver** x^*

construcción sólo considera el orden impuesto por las secuencias de trabajo; en el artículo citado el procedimiento que se propone construye la solución considerando los tiempos de inicio de las operaciones iniciales y los tiempos de finalización de las operaciones finales de las secuencias de trabajo. Aunque este enfoque ya ha sido sugerido y utilizado en otros artículos, no se suele mencionar el pseudocódigo explícito para realizar el paso de inicialización aleatoria de soluciones por lo que no es claro como realizaron los autores ciertos pasos. Debido a estas omisiones y a que en los algoritmos evolutivos ejecutados a largo plazo, las heurísticas de inicialización no suelen contribuir en gran medida en los resultados finales, se decidió usar un algoritmo más sencillo que no incorpora pasos heurísticos. En el Algoritmo 17 se presenta el pseudocódigo del método que se utilizó en este trabajo parar generar estas soluciones. El algoritmo considera el conjunto de soluciones $Planificables$, es decir, aquellas operaciones para las cuales sus operaciones predecesoras, en su respectiva secuencia de trabajo, ya han sido planificadas, por lo cual, al inicio del algoritmo, este conjunto estará conformado por las operaciones iniciales de cada uno de los trabajos. Posteriormente, de manera iterativa se realiza una selección aleatoria, con probabilidad uniforme, una operación del conjunto de $Planificables$, y se le asigna el orden en la máquina que corresponde (línea 8). Una vez planificada la operación, se verifica si ésta tiene un sucesor en la secuencia de trabajo, y si es así, se agrega al conjunto $Planificables$, pues para dicha operación ya no se tiene ninguna restricción para poder ser planificada.

Con este método de inicialización de soluciones aleatorias, es posible generar a todas las soluciones factibles, incluyendo a las óptimas. La probabilidad de generar una solución específica no es uniforme, ya que el orden en que se van incluyendo las operaciones en el conjunto $Planificables$ influye en la probabilidad que tiene una solución específica de generarse con este método. No obstante, es posible generar soluciones que sean suficientemente diferentes, y experimentalmente se vio que la distancia entre las soluciones generadas, utilizando esta estrategia, era alta.

Algoritmo 17: Generación Aleatoria de una solución Factible.

Entrada: Datos de una instancia del JSSP
Salida: $solicionFactible$

```
1  Planificables = {O_{1,1}, O_{2,1}, ..., O_{n,1}} ;
2  Solucion = [] ;
3  para cada m ∈ M
4      Solucion[m] = [] ;
5  repetir
6      k = Elegir de manera aleatoria, con probabilidad uniforme, una operación del
         conjunto Planificables ;
7      Planificables = Planificables \ {k} ;
8      Solucion[μ_k].append(k) ;
9      Si JS[k] existe
10         Planificables = Planificables ∪ {JS[k]} ;
11 hasta Planificables == ∅;
12 devolver Solucion
```

4.1.2. Implementación de la Búsqueda Tabú

Como mecanismo de intensificación para la propuesta se eligió una búsqueda tabú, la cual ha probado ser una buena alternativa para lograr soluciones de buena calidad en el JSSP. En particular, se eligió la búsqueda tabú con la vecindad N7 con evaluación por estimación, propuesta en Zhang y col., 2007, incorporándole las modificaciones al criterio de permanencia en la lista tabú (tabu tenure) propuestas en Cheng, Peng y Lü, 2016.

Para la implementación de la búsqueda tabú, se consideró el pseudocódigo presentado en una sección anterior de este trabajo, en el Algoritmo 8. Sin embargo, fue necesario tomar algunas decisiones de diseño, ya que en los artículos en donde se proponían los métodos o sus variaciones, no se especificaba claramente de que forma se trataban algunos casos. En particular se tienen dos puntos para los cuales se requirió tomar una decisión y que pueden afectar el funcionamiento de la búsqueda tabú: la selección de la ruta crítica que se considera para la exploración de la vecindad y el proceso de simplificación de la vecindad (después de que se ha realizado un movimiento).

Para la exploración de la vecindad N7, en este trabajo se consideraron todas las rutas críticas que se presentan en la solución, en vez de eligir una de manera aleatoria. Para algunas vecindades, por ejemplo en la N1, ya se había propuesto considerar todas las rutas críticas, y en otras vecindades, tal como la N5, se considera solo una ruta crítica que se selecciona de manera aleatoria. Seleccionar solo una ruta crítica puede acelerar la búsqueda tabú, ya que el número de movimientos que se deben considerar es mucho menor, pero puede tener como consecuencia no alcanzar siempre óptimos locales. En los artículos de referencia que usaban la N7 no era claro cuál de las dos estrategias utilizaban. En nuestro caso implementamos ambas y realizamos pruebas preliminares, decidiendo finalmente considerar todas las rutas críticas para formar la vecindad considerada en cada iteración.

La ventaja principal de la decisión tomada es que se tiene la certeza de que al terminar de ejecutar búsqueda tabú, al menos se habrá llegado a un óptimo local. Sin embargo, se tiene la desventaja de que el costo computacional es mucho más elevado, por lo que se realizan menos iteraciones de la parte global. Para reducir el costo computacional de explorar todas las rutas, se consideró una estructura de tipo conjunto (set), en la cual se almacenan los posibles movimientos que se pueden generar al considerar todas las rutas. En muchos cosas, las rutas críticas tienen bloques en común, de manera parcial o completamente iguales, por lo que se tienen movimientos repetidos. Con la estructura utilizada los movimientos repetidos se eliminan y solo es necesario evaluarlos una sola vez.

Simplificación de la Vecindad

Tal como se mencionó anteriormente, en la búsqueda tabú la idea principal es seguir una trayectoria en donde siempre nos movemos a la mejor solución, que no esté en la lista de prohibidos, en la vecindad. Sin embargo, dependiendo del tipo de vecindad que se considere, el número de movimientos podría ser demasiado grande, por lo cual existen implementaciones de Tabú Search que incorporan mecanismos para descartar aquellos movimientos que no son prometedores, o que de antemano se sabe que no pueden mejorar a la solución actual. Aunque con la evaluación por estimación se pueden descartar algunos de estos movimientos, aún es posible que la vecindad sea demasiado grande. Por tal motivo, algunos autores han propuesto alternativas para simplificar la vecindad, específicamente cuando se realiza un movimiento que ha producido un incremento en el makespan de la solución actual, es posible determinar en que parte de la ruta crítica se ha producido dicho incremento, y por tanto podemos simplificar la vecindad que se deberá considerar.

Este enfoque de simplificar la vecindad ya había sido aplicado anteriormente a la vecindad $N6$, en **Bazewicz1996** se puede revisar una justificación más detallada de la forma en que se realiza la simplificación. Debido a que los movimientos que se consideran en la vecindad $N7$ son similares a los de la $N6$, los autores de dicha vecindad (la $N7$) también mencionan que es posible realizar la simplificación en esta vecindad, sin embargo no se dan mayores detalles para la implementación.

En particular, debido a que la vecindad $N7$ considera también los movimientos que se producen al considerar operaciones internas en un bloque, es importante establecer la forma en que se debe realizar la simplificación. En el Algoritmo 18 se presenta el pseudocódigo para la simplificación de la vecindad $N7$ utilizado en esta tesis. Para la simplificación, se consideran dos casos, uno para los movimientos hacia adelante (forward interchange) y otro para los movimientos hacia atrás (backward interchange); cuando el movimiento consiste simplemente en intercambiar operaciones adyacentes, consideramos que dicho movimiento puede verse como cualquiera de los dos casos.

Dado el movimiento (U, V), sin importar de que tipo sea, si se produce un incremento del makespan, se tiene que el arco (V, U) estará en cualquiera de las nuevas rutas críticas (que se generan después de realizar el movimiento). Además, es posible dividir las ruta críticas

Algoritmo 18: Simplificación de la Vecindad.

Entrada:
sol : Solución actual ,
mov : Información del último movimiento realizado (tipo de movimiento, operaciones U, V involucradas, valores s_i y q_i de las operaciones operaciones involucradas (opU, opV) antes de realizar el movimiento).
Salida: vecindad simplificada

```
1  pos_izq = Falso
2  pos_der = Falso
3  Si mov.tipo ∈ { mover U a la derecha de V, insertar U a la derecha de V } , intercambiar U con V
4      Si s_{sol.opV} > s_{mov.opV} − d_u
5          pos_izq = Verdadero
6      Si q_{sol.opV} > q_{mov.opV} + d_u
7          pos_der = Verdadero
8  Si mov.tipo ∈ { mover U a la izquierda de V, insertar U a la izquierda de V } , intercambiar U con V
9      Si q_{sol.opU} > q_{mov.opV} − d_v
10         pos_der = Verdadero
11     Si s_{sol.opV} > q_{mov.opV} + d_v
12         pos_izq = Verdadero
13     Si pos_izq & !pos_der
14         sol.vecindad = calcularRutasCriticas(sol.op0, sol.opU)
15     Si pos_der & !pos_izq
16         sol.vecindad = sol.calcularRutasCriticas(sol.opV, sol.opN)
17 devolver Solucion
```

implicadas en dos partes: la parte izquierda, las rutas que van desde la operación 0 ($op0$, la operación ficticia que marca el inicio de los trabajos) hasta la operación U (opU), y la parte derecha, las rutas que van desde la operación V (opV) hasta la operación $N + 1$ (opN, la operación ficticia que marca el fin de los trabajos).

Dependiendo del tipo de movimiento, es posible validar si se cumplen las condiciones necesarias para determinar en qué parte de la ruta crítica se tiene el incremento. Cuando se cumplen las condiciones para determinar que el incremento pudo haber ocurrido en ambas partes (es decir, se tiene que $pos\_izq = pos\_der = Verdadero$), en ese caso no se realiza ninguna simplificación.

En caso de determinar que el incremento se produce en la parte izquierda ($pos\_izq = Verdadero$), la vecindad será reducida para incluir solamente los movimientos que se producen al considerar las rutas críticas que van desde la operación 0 hasta la operación U (incluida), de manera que el resto de movimientos (los de la parte derecha) serán descartados y por lo tanto se simplifica la vecindad. El otro caso es análogo, si se determina que el incremento se produjo en la parte derecha ($pos\_der = Verdadero$), la vecindad será reducida para incluir solamente los movimientos que se producen al considerar las rutas críticas que van desde la

operación V (incluida) hasta la operación $N + 1$.

4.1.3. Generación de nuevos Individuos

Una de las características principales de los algoritmos evolutivos (incluyendo el algoritmo memético propuesto en este trabajo), es la generación de nuevos individuos en cada iteración, mediante la combinación de información de individuos de la población. De manera general, para generar un nuevo individuo se realizan los siguientes pasos: se seleccionan los individuos que serán los padres, se aplica un operador de recombinación (cruza) para generar a los nuevos individuos, se les aplica un operador de reparación para obtener soluciones factibles y al final se realiza un proceso de mejora del individuo (generalmente mediante una metaheurística de trayectoria). En las siguientes secciones se detallan cada uno de los pasos.

Selección de Padres

Para seleccionar a los individuos a partir de los cuales se realizará la recombinación, se considera una selección aleatoria uniforme con reemplazo, por lo cual todos los individuos en la población tienen la misma probabilidad de ser elegidos. A diferencia de los modelos poblacionales que son generacionales, en donde la opción más común es hacer un torneo binario, en los modelos poblacionales de estado estacionario (steady-state), es común realizar este tipo de selección.

Dado que la selección es con reemplazo, un individuo puede ser elegido dos veces (en la misma iteración) como padre, por lo cual en este caso los hijos que se generan resultan en clones de los padres. Cuando ocurre este caso en el algoritmo, se mantiene sólo un hijo como clon del padre, y se selecciona un nuevo par de individuos para generar otro hijo. En el caso de los operadores que generan un par de hijos, en este caso sólo se conserva a uno de los hijos generados y se descarta el otro, de manera que al final el algoritmo siempre genera solamente dos hijos.

Operadores de cruce

El operador de cruce establece la forma en que se realiza la recombinación de la información de los padres. En la propuesta del esquema general del algoritmo memético detallado anteriormente, se tiene libertad para seleccionar cualquier operador de cruce que trabaje con dos individuos.

Para la representación de individuos utilizada, ya existen algunos operadores de cruza para el JSSP, en particular se tienen los operadores propuestos por el HEA (basado en la subcadena común más larga para cada máquina entre los individuos) y el HGA (basado en dividir

el conjunto de máquinas en dos conjuntos complementarios, y heredar para cada hijo las máquina en los conjuntos complementarios, de manera cruzada).

Aunque los operadores propuestos logran obtener resultados aceptables, en este trabajo se propone también un nuevo operador, el cual se describe a continuación.

Operador de Cruce basado en el orden parcial de las operaciones en cada padre

El principio que hay tras el diseño del nuevo operador de cruce es que uno de los aspectos más importantes a preservar en cada máquina es el orden entre las operaciones. Así, las operaciones que fueron asignadas en las primeras posiciones deberían quedar también en posiciones iniciales. Mientras que para las que estaban en posiciones finales, el operador debería tener tendencia a asignarlas en posiciones finales. En el caso del HGA, tal y como vimos se intercambian máquinas completas, por lo que no hay necesidad de generar un mecanismo para cruzar permutaciones. En el caso del HEA, sí se combinan permutaciones y se tiene el principio de preservar el orden identificando para ello la subsecuencia común más grande, pero se da importancia a que esa subsecuencia común permanezca exactamente en las mismas posiciones, lo cual consideramos que podría no ser tan relevante. Con el fin de proponer un operador que no dé importancia a la posición exacta en que está cada operación, sino más bien que trate de mantener un orden relativo entre las operaciones, se propone la alternativa descrita a continuación. El pseudocódigo del operador propuesto se presenta en el Algoritmo 19. El operador trabaja combinando las secuencias de operaciones en cada máquina de manera independiente, por lo cual el primer ciclo del algoritmo sirve para iterar el conjunto de máquinas. En cada máquina, se realiza un ciclo para ir seleccionando en cada paso una operación. En cada iteración se elige de manera aleatoria, con probabilidad uniforme, a alguno de los padres y se considera la primera operación que no ha sido seleccionada. Esta operación se copia al Hijo en la siguiente posición disponible en la máquina. El operador no garantiza que la soluciones generada sea factible, por lo que es necesario realizar un paso posterior de reparación.

En la Figura 4.1 se muestra un ejemplo del operador de cruce propuesto. El operador se aplica para generar la secuencia en cada máquina. En el primer paso (Figura 4.1a), se elige de manera aleatoria entre la primera operación disponible en la configuración (orden de procesamiento) de cada padre. En el ejemplo estas operaciones serían J5 en el Padre 1 y J1 en el Padre 2. En la figura del ejemplo se ha elegido al Padre 1, por lo cual la primera operación, para esa máquina, de la configuración para el Hijo será J5. Una vez que se elige la operación y se hereda al hijo, es necesario marcar la operación como no disponible, en la configuración de ambos padres, para ser seleccionada en pasos posteriores. En el segundo paso (Figura 4.1b), ahora se tiene que la primera operación disponible en la configuración del Padre 1 es J2, y para el Padre 2 aún esta disponible la operación J1. Nuevamente, la elección de qué padre elegir, para seleccionar la operación siguiente en el Hijo, se hace de manera aleatoria con probabilidad uniforme (se tiene una probabilidad de 0.5 de elegir al Padre 1, y 0.5 de elegir al Padre 2), en el ejemplo se considera que se ha elegido al Padre 2, por lo cual la siguiente operación en la configuración del Hijo es J1. Para determinar la configuración

Algoritmo 19: Operador de Cruce (recombinación) basado en el orden parcial de las operaciones en cada padre.

Entrada: x, y Individuos seleccionados para aplicar recombinación.
Salida: h nuevo individuo (hijo) generado a partir de x, y.

```
1  Etiquetar todas las operaciones en x, y como no visitadas. ;
2  para cada m ∈ M
3      k = 1 ;
4      mientras k ≤ n hacer
5          s = Seleccionar de manera aleatoria entre x_m y y_m ;
6          Considerar la primera operación O_s no visitada en la secuencia de operaciones
             de s ;
7          Marcar la operación O_s como visitada en x_m y en y_m ;
8          Copiar la operación O_s en la posición k de la máquina m en h ;
9          k++ ;
10 devolver h
```

completa en el Hijo, se eligen las operaciones restantes con un procedimiento análogo al descrito en los pasos anteriores. Sin embargo, en algunas ocasiones puede ocurrir un caso como el que se presenta en el ejemplo (Figura 4.1c) en donde realmente ya no importa la selección del padre, independientemente de cuál padre se selecciona, la configuración para el Hijo será la misma. Esto ocurre cuando la subsecuencia de las operaciones están presentes en el mismo orden parcial en cada Padre, en el ejemplo ambos padres tienen la subsecuencia de operaciones J2, J3, J4 . Debido a que las operaciones ya han sido seleccionadas, en el Hijo se mantiene también esa subsecuencia.

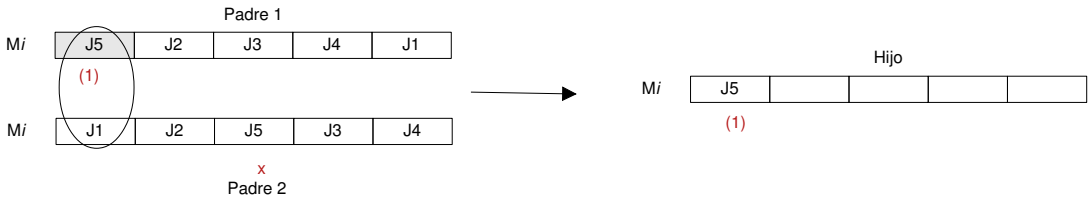
A diferencia de otros operadores, el operador propuesto solo genera un hijo por cada combinación de padres. Por tal motivo, en el esquema general del algoritmo evolutivo, el operador se ejecuta dos veces para generar un par de hijos.

Reparación de Soluciones

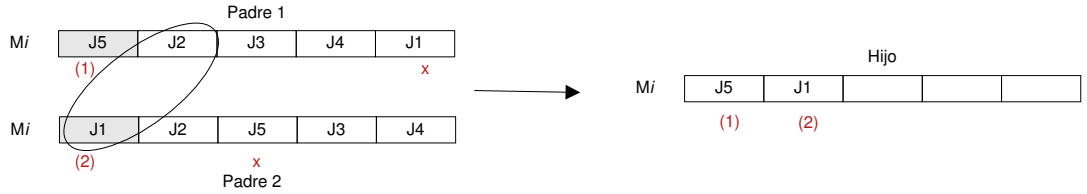
Debido a que los operadores de recombinación no garantizan generar soluciones factibles es necesario realizar un paso de reparación de las soluciones generadas. En este trabajo consideramos como operador de reparación el algoritmo propuesto en Lu y col., 2011, que ya ha sido descrito en la sección 3.3 de este trabajo.

Mejora de Hijos

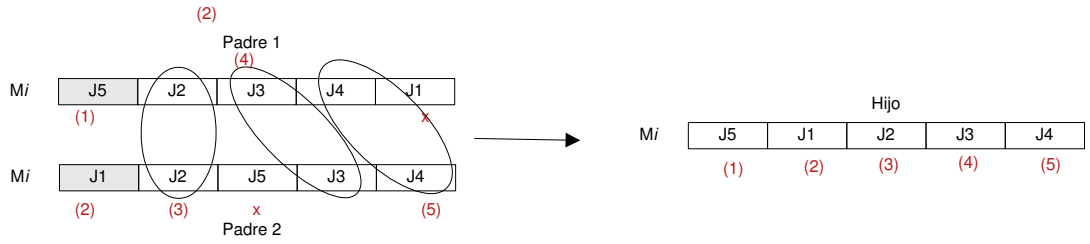
El último paso de la generación de hijos, correspondiente a la mejora de los individuos, se realiza considerando una búsqueda tabú, la cual ya ha sido explicada anteriormente en este trabajo. Cuando alguno de los hijos es un clon de los padres, la intensificación se realiza considerando la última solución visitada en la búsqueda tabú del padre. Para cada individuo se mantiene la mejor solución visitada y la última solución considerada en la búsqueda tabú.



(A) Primer paso.



(B) Segundo Paso.



(C) Pasos 3-5

FIGURA 4.1: Ejemplo del Operador de Cruce propuesto, basado en el orden parcial de las operaciones en cada padre..

De esta manera, los hijos que se generan como clones inducen un proceso de intensificación más fuerte (ya que la búsqueda inicia desde la última solución visitada en el padre).

4.1.4. Actualización de la Población

Una vez que se han generado los nuevos individuos, es necesario realizar un proceso de actualización de la población, de manera que se determine qué individuos permanecen en la población y cuáles serán eliminados; a partir de los p individuos de la población anterior, y de los 2 nuevos individuos creados, seleccionar los nuevos p individuos supervivientes que formarán la siguiente población. Existen diferentes alternativas para realizar este proceso de actualización tales como las estrategias clásicas descritas en la Sección 2.4.1 - Métodos de Reemplazamiento de este trabajo, pero carecen de control de diversidad, por lo que no son demasiado adecuadas para ejecuciones a largo plazo. Por ello, en esta tesis se usan la selección del Mejor Individuo no Penalizado (BNP, Best-Non Penalized) y una nueva estrategia basada en agrupaciones (clusters) de individuos que se explican en las siguientes secciones.

4.2. Manejo de la Diversidad

Tal como se comentó anteriormente en este trabajo, el problema de convergencia prematura surge cuando todos los individuos de la población se quedan en una misma zona del espacio de búsqueda, por lo cual ya no se realiza exploración y en muchos casos se tienen clones (copias) de los individuos. Para lograr un balance entre exploración y explotación, consideramos estrategias de actualización de la población (etapa de reemplazamiento de individuos) de tal forma que establezcan una penalización, de forma dinámica, que evite la selección de individuos similares. A continuación se dan algunas definiciones relacionados con la forma en que medimos la diversidad en una población, y en las secciones posteriores se detallan las estrategias de actualización de la población consideradas.

Para ambas estrategias, se considera un parámetro D que determinará la distancia (o similitud) mínima que puede haber entre dos individuos de la población. Este parámetro cambiará de manera dinámica durante la ejecución del algoritmo, de manera que al inicio del algoritmo se tendrá un valor suficientemente grande para forzar la exploración de nuevas zonas en el espacio de búsqueda, y al final un valor pequeño que induzca que se tenga un proceso de intensificación en las zonas promisorias encontradas. El valor inicial para este parámetro se inicializa de la siguiente forma:

$$D_{Ini} = 0{,}5 \times DCN_{av}(P_{ini})$$

donde

$$DCN_{av}(P) = \frac{1}{|P|} \sum_{i=1}^{|P|} DCN(P(i), P \setminus \{P(i)\}),$$

$$DCN(x, P) = \text{mín}\{distancia(x, y) | y \in P\}$$

P_{ini} es la población inicial después de que todos los individuos han sido mejorados mediante la búsqueda tabú. Para calcular la contribución a la diversidad, de un individuo respecto a una población, se considera la distancia al vecino más cercano ($DCN, DistanceClosestNeighbor$), de acuerdo a una medida específica, denotada como $distancia$ en la ecuación anterior.

4.2.1. Medidas de Disimilitud para el JSSP

A partir de las definiciones anteriores, aún cuando la medida de similitud es útil, para propósitos de comparación de individuos, es necesario tener una medida que se minimice cuando los individuos son parecidos, y que aumente si los individuos tienen mayores diferencias entre sí. Por lo anterior, en vez de utilizar las medidas de similitud entre individuos, para el algoritmo propuesto se utilizan medidas de disimilitud. A continuación se especifica la

forma en que se calcula la disimilitud considerando la métrica $similitud_{SCL}$ definida en la Sección 3.2.

Disimilitud considerando la subsecuencia común más larga

Dado que la máxima similitud, considerando la subsecuencia común más larga, es N, para obtener una medida de disimilitud basta considerar este valor máximo y restar la similitud entre las soluciones x, y , de la siguiente manera:

$$disimilitud_{SCL}(x, y) = N - similitud_{SCL}(x, y)$$

donde N es el número total de operaciones.

El rango de valores que pueden tomar las medidas $disimilitud_{SCL}$ y NUM_DIFF está acotado por $0 \leq NUM\_DIFF, disimilitud_{SCL} \leq N, N = n \times m$. NUM_DIFF sólo toma en cuenta la diferencia posición a posición, por tanto los casos extremos ocurren cuando todas las posiciones son iguales, o todas son diferentes. $disimilitud_{SCL}$ considera la subsecuencia común más larga, y aquí los casos extremos corresponden nuevamente a todas las posiciones iguales, o en el otro caso la subcadena común más larga será de al menos longitud 1.

Tras analizar las métricas anteriores, nos dimos cuenta de dos problemáticas. En primer lugar, las distancias entre cualquiera dos permutaciones de tamaño n, se mapean a los enteros $[0, n]$. Teniendo en cuenta que hay $n!$ permutaciones, este parece un rango bastante pequeño. En segundo lugar, en el caso del JSSP, aunque es relevante la posición asignada a cada operación, cuantificar simplemente si la operación queda en la misma posición o no, no parece suficiente para desarrollar una buena medida de similitud. Así, dada una permutación, intercambiar el primer elemento con el último debería producir una distancia mayor que intercambiar el primero con el segundo. Sin embargo, tanto en la medida de NUM_DIFF como en la $disimilitud_{SCL}$, esto no ocurre y en ambas la disimilitud es la misma. Por ello, se propone una nueva medida de disimilitud que trata de solventar estas dos problemáticas. En la siguiente sección se describe la medida propuesta.

Disimilitud considerando la diferencia entre las posiciones en las permutaciones

Dada la importancia del orden relativo entre las operaciones, en lugar de simplemente verificar si una operación está asignada o no a la misma posición en ambas permutaciones, se procede a sumar la distancia en valor absoluto entre las posiciones en que quedó asignada cada operación en ambas permutaciones. De esta forma, dadas dos soluciones x, y la medida de disimilitud entre las soluciones queda definida de la siguiente manera:

$$D_{Pos}(x, y) = \sum_{i=1}^{m} \sum_{j=1}^{n} dpos_{i,j}(x, y)$$

donde:

- $dpos_{i,j}(x, y) = |j - pos(x_i[j], y_i)|$,
- $pos(a, b[]) :=$ posición del elemento a en la permutación b.

Esta medida de disimilitud es una suma de distancias de desviación entre permutaciones (permutation deviation distance) y ya había sido propuesta en «More distance functions for order-based encodings».

4.2.2. Selección del Mejor No Penalizado (BNP, Best-Non Penalized)

En esta estrategia se busca dar prioridad a los individuos de buena calidad, pero que no sean demasiado similares con algún otro individuo de la población. Para lograr esto, se considera un parámetro D de penalización, que establece la distancia mínima a la cual un individuo podrá ser seleccionado (considerado para la siguiente generación). El valor de D se actualiza durante la ejecución, considerando el tiempo en el que la ejecución debe terminar y el lapso en el que se ha ejecutado el algoritmo.

En el Algoritmo 20 se presenta el pseudocódigo de esta estrategia. En el primer paso, unimos a los individuos de la población actual con los hijos generados para conformar a la población P'. El segundo paso consiste en ordenar a los individuos de acuerdo a su función objetivo, de manera que demos prioridad de selección a los individuos que tengan una mejor aptitud (makespan). En los pasos 3 y 4 se selecciona al mejor individuo, se agrega a la nueva población ($nPob$) y se elimina de P'. El quinto paso corresponde a la actualización del valor del factor de penalización D, considerando el valor de D_{Ini}, calculado de acuerdo a la diversidad promedio de la población inicial (tal como se mencionó anteriormente), y los valores del tiempo transcurrido (T_{actual}) y el tiempo total de ejecución (T_{total}). Del paso 6 al 20, se elige de manera iterativa a los individuos restantes para completar la nueva población. Seleccionamos al mejor individuo de la población P', y verificamos si la contribución a la diversidad, con respecto a $nPob$, es mayor que el factor de penalización (D). Si la contribución es menor, es decir, el individuo está penalizado, se deberá descartar este individuo y considerar al siguiente mejor individuo en la población total. Si ya se han descartado todos los individuos, se elegirá al individuo que genera la mayor contribución a la diversidad, sin importar que esté penalizado. Lo anterior se repite de manera iterativa hasta que en la nueva población han sido seleccionado el número de individuos deseados, es decir, el tamaño de la población.

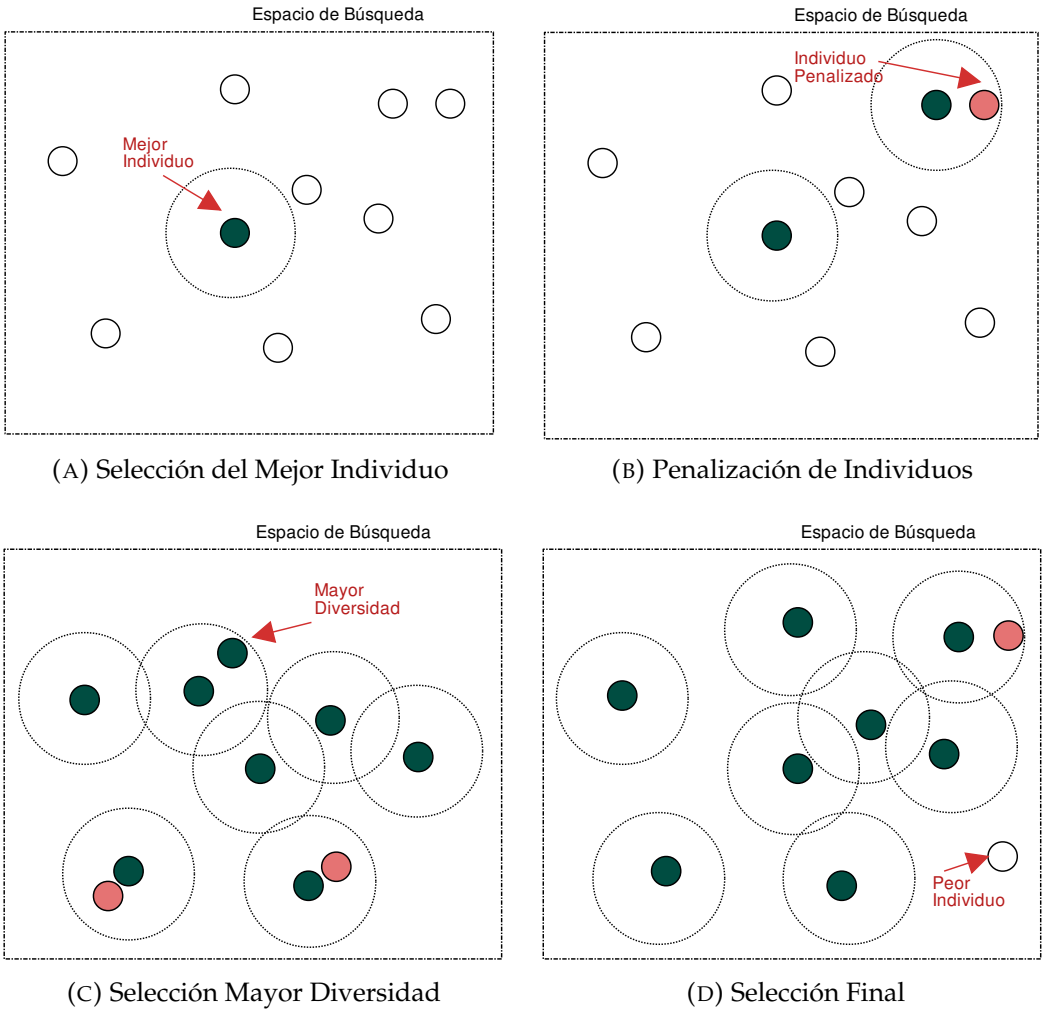
En la Figura 4.2 se presenta de manera gráfica el proceso de reemplazamiento BNP. Cada círculo representa un individuo. El primer diagrama (Figura 4.2a) representa el paso inicial de selección del mejor individuo. Una vez que se selecciona se establece un radio de penalización (en la figura está representado por un círculo con líneas punteadas), por lo que todos lo individuos que estén cercanos serán penalizados; esta situación se encuentra representada en el segundo diagrama (Figura 4.2b), en donde el individuo seleccionado ha penalizado

Algoritmo 20: Selección por Mejor No Penalizado (BNP, Best Not Penalized).

Entrada: $P = \{x^1, \ldots, x^p\}$ Población Actual , $H = \{x^{p+1}, x^{p+2}\}$ Hijos Generados
Salida: $nPob = \{x^1, \ldots, x^p\}$ individuos seleccionados para actualizar la población.

```
1  P' = P ∪ {x^{p+1}, x^{p+2}} ;
2  ordenar población P' de manera ascendente por la función objetivo (makespan) ;
3  nPob = {P'[0]} ;
4  P' = P' \ {P'[0]} ;
5  D = D_Ini − (T_actual / T_total) D_Ini ;
6  mientras |nPob| < p hacer
7      i = 0 ;
8      selected = False ;
9      mientras i < |P'| & !selected hacer
10         P'[i].DCN = DCN(P'[i], nPob) ;
11         Si P'[i].DCN > D
12             selected = True ;
13             x_sel = P'[i] ;
14         i++ ;
15     Si !selected
16         x_sel = P'[0] ;
17         x_sel.DCN = DCN(x_sel, nPob) ;
18         para cada x^k ∈ P'
19             x^k.DCN = DCN(x^k, nPob) ;
20             Si x^k.DCN > x_sel.DCN
21                 x_sel = x^k ;
22     nPob = nPob ∪ {x_sel} ;
23     P' = P' \ {x_sel};
24 devolver nPob
```

a otro que está a una distancia demasiado cercana, dentro del radio de penalización. Por tal razón, el individuo penalizado (en la figura marcado con rojo) no sería considerado para las siguientes iteraciones del proceso de reemplazamiento, a menos que todos los individuos restantes ya estén penalizados, tal como en el caso de la Figura 4.2c, en donde todos los individuos restantes ya están penalizados, por lo cual ahora se selecciona al individuo que aporte mayor diversidad, el que maximice el DCN, usando como conjunto de referencia a los supervivientes actuales. En el diagrama del ejemplo el individuo seleccionado es el de la parte superior del diagrama; aunque el individuo está penalizado, es seleccionado ya que es el que aporta más diversidad entre los individuos restantes que están penalizados. Finalmente, la Figura 4.2d representa el caso típico que se presenta, y que justo es el caso deseado para el proceso de reemplazamiento, en donde ni los peores individuos ni los individuos que son demasiado cercanos a uno ya seleccionado son considerados para la siguiente población.



(A) Selección del Mejor Individuo

(B) Penalización de Individuos

(C) Selección Mayor Diversidad

(D) Selección Final

FIGURA 4.2: Ejemplo del Proceso de Reemplazamiento considerando la estrategia BNP

4.2.3. Reemplazamiento considerando Agrupaciones (Clusters) de Soluciones

Uno de los principales inconvenientes de las estrategias de control explícito de diversidad es que suelen retrasar demasiado la fase de intensificación. Una manera de lograr intensificación es mediante cruza de individuos que son cercanos. Cuando se combinan individuos que no están demasiado alejados, algunos de los operadores de recombinación son capaces de mantener buenas características de ambos padres, con lo cual se logra obtener una mejor descendencia, y como resultado se obtiene una intensificación en una determinada zona del espacio de búsqueda.

Considerando lo anterior, se propone una estrategia de reemplazamiento que permite una mayor flexibilidad para seleccionar individuos que son similares, pero manteniendo un mínimo de diversidad. La estrategia de reemplazamiento considerando agrupaciones de individuos es una modificación a BNP que aplica dos factores de penalización. Estos dos factores se usan con el objetivo de que se puedan seleccionar algunos individuos cercanos entre sí aunque no demasiado (formando clusters de individuos) para promover intensificación, a

la vez que se fuerza a que algunos individuos seleccionados estén alejado para promover diversificación. De esta forma se busca diseñar un algoritmo que en cada fase realice pasos de exploración y pasos de intensificación, aunque también promueva mayor exploración al inicio y mayor intensificación al final.

En el Algoritmo 21 se presenta el pseudocódigo de la estrategia de reemplazamiento considerando agrupaciones de individuos. Para esta estrategia, se consideran 2 parámetros adicionales: $porD_{min}$ determina el porcentaje del factor de penalización para establecer la distancia mínima entre individuos que son seleccionables, y $porTmaxCl$ determina la fracción del tamaño de la población para estacler el máximo número de individuos que se podrán agrupar en una misma zona. El algoritmo comienza de manera similar a BNP, considerando la unión de los individuos de la población actual P y la población de hijos H. En el siguiente paso (línea 3) se inicializa la información de los clusters de los individuos, la cual será actualizada durante el proceso de selección. De manera similar a BNP, en esta estrategia consideramos al mejor individuo, y se agrega a la nueva población (paso 5) y se elimina de P'. Posteriormente, se calcula el factor de penalización D, considerando el tiempo transcurrido (T_{actual}) y el tiempo total de ejecución deseado (T_{total}).

Los siguientes individuos serán seleccionados considerando el siguiente proceso de manera iterativa. Primero se calcula la aportación a la diversidad, medida como el DCN, del siguiente mejor individuo en la población respecto a los individuos en la nueva población ($nPob$). Si la aportación es mayor al factor de penalización, el individuo es seleccionado inmediatamente. En otro caso, se verifica si la aportación es mayor a la distancia mínima y además si el cluster del individuo actual no está lleno, es decir, el número de individuos en el cluster es menor al tamaño máximo ($tamMaxCl$). Si ambas condiciones se cumplen, entonces el individuo es seleccionado, en caso contrario, se omite temporalmente al individuo y se intenta seleccionar al siguiente mejor. Cuando todos los individuos están penalizados (fueron omitidos), se realiza un proceso similar al que se tiene en BNP, es decir, se seleccionará al individuo que tenga una mayor aportación de diversidad a la población; nótese que cuando esto ocurre, el máximo número de individuos en un clúster podría llegar a superarse ya que en esta situación sólo se considera la máxima aportación a la diversidad (sin considerar el tamaño de los clusters), lo cual se decidió mantener de esa forma por simplicidad en el código y evitar realizar demasiadas comprobaciones que podrían aumentar el costo computacional de esta estrategia de reemplazamiento. Después de seleccionar al individuo, se realiza un paso de actualización de la información de los clusters (línea 32), de manera que la información de los clusters de cada individuo (en P' y en $nPob$) refleje la selección de los individuos que se encuentran en $nPob$; la actualización consiste en insertar a dicho individuo en los clusters de los individuos cuya distancia al mismo es menor a D.

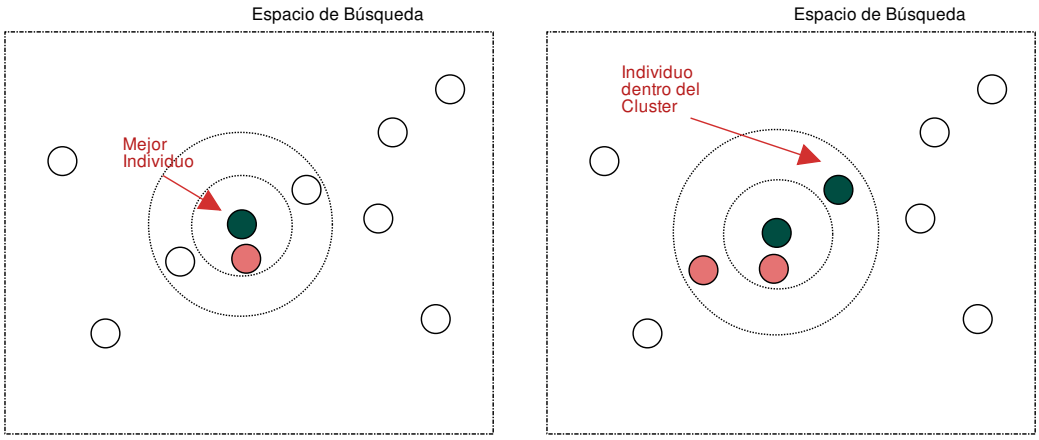
En la Figura 4.3 se ilustra el proceso de selección al considerar agrupaciones de individuos. En estos diagramas, cada uno de los círculos representa a los individuos de la población. A la izquierda (Figura 4.3a) se representa la selección del mejor individuo (señalado con una flecha), y los círculos representan la zona de penalización. El círculo más pequeño determina la zona en la cual los individuos son penalizados inmediatamente, ya que se encuentran a

Algoritmo 21: Estrategia de Actualización de la Población (Reemplazamiento) considerando Clusters.

Entrada:
$P = \{x^1, \ldots, x^p\}$ Población Actual ,
$H = \{x^{p+1}, x^{p+2}\}$ Hijos Generados,
$porD_{min}$ Fracción del factor de penalización para establecer Distancia Mínima ,
$porTmaxCl$ Fracción de la población para establecer el tamaño máximo por Cluster.
Salida: $nPob = \{x^1, \ldots, x^p\}$ individuos seleccionados para actualizar la población.

$P' = P \cup \{x^{p+1}, x^{p+2}\}$;
para cada $x \in P'$
    $x.cluster = \{x\}$
ordenar población P' de manera ascendente por la función objetivo (makespan) ;
$nPob = \{P'[0]\}$;
$P' = P' \setminus \{P'[0]\}$;
Actualizar información de cluster de individuos en $nPob$ y P' considerando al nuevo individuo agregado $P'[0]$;
$D = D_{Ini} - \frac{T_{actual}}{T_{total}} D_{Ini}$;
$D_{min} = porD_{min} \times D$;
$tamMaxCl = porTmaxCl \times p + 1$
mientras $|nPob| < p$ **hacer**
    $i = 0$;
    $selected = False$;
    mientras $i < |P'|$ & $!selected$ **hacer**
        $P'[i].DCN = DCN(P'[i], nPob)$;
        Si $P'[i].DCN > D$
            $selected = True$;
            $x_{sel} = P'[i]$;
        Si $!selected \ \& \ P'[i].DCN > D_{min} \ \& \ P'[i].cluster.size() < tamMaxCl$
            $selected = True$;
            $x_{sel} = P'[i]$;
        $i++$;
    Si $!selected$
        $x_{sel} = P'[0]$;
        $x_{sel}.DCN = DCN(x_{sel}, nPob)$;
        para cada $x^k \in P'$
            $x^k.DCN = DCN(x^k, nPob)$;
            Si $x^k.DCN > x_{sel}.DCN$
                $x_{sel} = x^k$;
    $nPob = nPob \cup \{x_{sel}\}$;
    $P' = P' \setminus \{x_{sel}\}$;
    Actualizar información de cluster de individuos en $nPob$ y P' considerando al nuevo individuo agregado x_{sel} ;
devolver $nPob$

una distancia muy cercana. El círculo de mayor tamaño, determina el área en la que se considerará que los individuos están en el mismo cluster. En la parte derecha (Figura 4.3b),



(A) Selección del Mejor Individuo (B) Selección considerando Cluster

FIGURA 4.3: Ejemplo del Proceso de Reemplazamiento considerando la estrategia basada en Agrupamientos (Clusters).

se representa la posible selección de un individuo dentro del cluster. Sin embargo, cuando el individuo es seleccionado, otros individuos que están dentro del cluster podrían quedar penalizados al considerar el tamaño del cluster; en el ejemplo se tiene que el tamaño máximo sería de dos individuos, por lo que el individuo que está también dentro del cluster, fuera del círculo más pequeño, quedaría penalizado.

Para esta estrategia de reemplazamiento, es necesario considerar algunas modificaciones al esquema general del algoritmo memético, de manera que se tenga un mejor aprovechamiento de las agrupaciones que se generan durante el proceso de reemplazo de individuos. En el Algoritmo 22 se presenta el pseudocódigo del algoritmo memético modificado para considerar las agrupaciones de individuos. Como novedad al esquema general, en $x^i.cluster$ se mantiene en todo momento a los individuos de la población que están a distancia menor a D.

La otra modificación del algoritmo memético se encuentra en el paso de selección de padres (línea 5). A diferencia del esquema general, en este caso inicialmente sólo se selecciona un individuo x^j de manera aleatoria con probabilidad uniforme en la población P. El otro individuo, x^k, se selecciona considerando la condición del paso 7. Primero se verifica si en el clúster de x^j hay algún otro individuo y se considera también una probabilidad de selección ($pSeleccion$), la cual cambia de manera dinámica durante la ejecución (línea 6); esta probabilidad de selección ($pSeleccion$) comienza en 0 al inicio de la ejecución y se aumenta linealmente de forma que la probabilidad llega a 1 al final de la ejecución.

Cuando ambas condiciones se cumplen (el cluster de x^j contiene más de un individuo y el valor aleatorio generado es menor a $pSeleccion$), se realiza la selección del otro padre de manera aleatoria, con probabilidad uniforme, considerando a los individuos dentro del cluster de x^j. En otro caso, cuando alguna de las dos condiciones no se cumple, la selección se realiza de manera aleatoria considerando de manera uniforme a los individuos de la población P.

Algoritmo 22: Algoritmo Memético modificado para favorecer cruce de individuos cercanos.

Salida: x^* Mejor solución encontrada

1 $P = \{x^1, \ldots, x^p\} \leftarrow$ Inicialización de la Población
2 **for** $i = 1, \ldots, p$ **do**
3 $\quad x^i \leftarrow Búsqueda\_Tabu(x^i)$
4 **mientras No** *Condiciones de Paro* **hacer**
5 $\quad x^j =$ Elegir de manera aleatoria, con probabilidad uniforme, un individuo de P ;
6 $\quad pSeleccion = Tiempo\_Transcurrido/Tiempo\_Total$
7 $\quad$ **Si** $x^j.cluster.size() > 1 \ \& \ rand(0,1) < pSeleccion$
8 $\quad\quad x^k =$ Elegir de manera aleatoria, con probabilidad uniforme, un individuo de $x^j.cluster$;
9 $\quad$ **En otro caso**
10 $\quad\quad x^k =$ Elegir de manera aleatoria, con probabilidad uniforme, un individuo de P ;
11 $\quad (x^{p+1}, x^{p+2}) \leftarrow Operador\_Recombinación(x^j, x^k)$
12 $\quad x^{p+1} \leftarrow Reparar(x^{p+1})$
13 $\quad x^{p+2} \leftarrow Reparar(x^{p+2})$
14 $\quad (x^{p+1}, x^{p+2}) \leftarrow Búsqueda\_Tabu(x^{p+1}, x^{p+2})$
15 $\quad P = Actualización\_Población(P, \{x^{p+1}, x^{p+2}\})$
16 $x^* =$ Elegir la mejor solución de P ;
17 **devolver** x^*

4.3. Paralelización

Debido a la complejidad del JSSP, generalmente las metaheurísticas de trayectoria (como tabu search) suelen tomar un número considerable de iteraciones para lograr encontrar buenas soluciones, por lo cual ejecutar el paso de mejora de los individuos, mediante la búsqueda tabú, es lo más costoso computacionalmente. Por lo anterior, se realizó una paralelización del algoritmo propuesto, considerando una arquitectura de memoria distribuida basado en el modelo de maestro-esclavo, para reducir el tiempo necesario para lograr realizar una mayor cantidad de generaciones. En los Algoritmos 23 y 24 se muestra el pseudocódigo que se consideró para los procesos maestro y esclavos, respectivamente. Los procesos esclavos se encargan de ejecutar la búsqueda Tabú, con el criterio de paro por número de iteraciones sin mejora, y el proceso maestro se encarga de realizar los pasos de recombinación y actualización de la población, de manera asíncrona. En particular, en el proceso maestro (Algoritmo 23) los hijos que son generados se mandan a procesos diferentes (líneas 13 y **??**). Otra diferencia importante, respecto al algoritmo secuencial, es que el reemplazamiento sólo se hace considerando un hijo, línea 16, por lo que el proceso maestro aplica el operador de reemplazamiento en cuanto uno de los proceso esclavos ha terminado de realizar la búsqueda tabú de alguno de los hijos.

Para tener una comparación justa, se realizó una paralelización del HEA y TSPR. En el caso de HEA, la paralelización se hizo de manera similar a la mencionada anteriormente para el método propuesto. Para el TSPR, se paralelizó todo el paso de reproducción, es decir, el paso

Algoritmo 23: Proceso Maestro para Algoritmo Memético en Paralelo.

Salida: x^* Mejor solución encontrada

```
P = {x^1, ..., x^p} ← Inicialización de la Población
for i = 1, ..., p do
    Enviar x^i a un proceso para realizar búsqueda tabú
Esperar a que todos los procesos terminen, y reemplazar individuos en P
mientras No Condiciones de Paro hacer
    Elegir de manera aleatoria, con probabilidad uniforme, dos individuos, x^j y x^k, de P
    (x^{p+1}, x^{p+2}) ← Operador_Recombinación(x^j, x^k)
    x^{p+1} ← Reparar(x^{p+1})
    x^{p+2} ← Reparar(x^{p+2})
    for h ∈ {p+1, p+2} do
        Si No Hay procesos disponibles
            Esperar a que un proceso termine ;
        Enviar x^h a un proceso disponible para realizar búsqueda tabú ;
        mientras Hay procesos esclavos que finalizaron hacer
            x^res = Solución devuelta por el proceso esclavo ;
            P = Actualización_Población(P, {x^res})
Esperar a que todos los procesos terminen, y actualizar individuos en P
x* = Elegir la mejor solución de P ;
devolver x*
```

Algoritmo 24: Proceso Esclavo para - Algoritmo Memético en Paralelo.

Entrada: x Una solución factible

Salida: x^* Mejor solución encontrada

```
x^res ← Búsqueda_Tabu(x)
devolver x^res
```

de Path Relinking en el esquema general. En el Algoritmo 25 se muestra el pseudocódigo que se consideró para los procesos esclavos para el caso específico de la paralelización de TSPR.

Algoritmo 25: Proceso Esclavo para TS/PR en Paralelo.

Entrada:
x Solución Inicial ,
y Solución Guía
Salida: x^* Mejor solución encontrada

1 $x^c = pathRelinking(x, y)$
2 $x^{res} =\leftarrow Búsqueda\_Tabu_{Pesada}(x^c)$
3 **devolver** x^{res}

Capítulo 5

Validación Experimental

Para validar el algoritmo propuesto, se consideraron 3 experimentos diferentes. En el primero se realizó un análisis de los componentes para obtener una buena parametrización del algoritmo. En el segundo experimento, se realizó una comparativa del método propuesto con dos algoritmos del estado del arte. En el tercer experimento se compararon versiones paralelas del método propuesto y de los dos algoritmos del estado del arte. En las siguientes secciones se detalla cada uno de los experimentos realizados.

Ya que la propuesta es un algoritmo estocástico, para cada experimento se realizaron 20 ejecuciones con los modelos correspondientes, y las comparaciones se realizaron considerando un conjunto pruebas estadísticas para determinar los casos en que se tiene una diferencia estadística significativa, tal como se mencionan en Segura y col., 2016a. En concreto, se realizaron las siguientes pruebas estadísticas. Primero se verificó si las muestras seguían una distribución normal, utilizando la prueba estadística de $Shapiro - Wilk$. En tal caso, se compararon las varianzas mediante la prueba estadística de $Levene$. Si las muestras tenían varianzas homogéneas (de acuerdo al resultado de la prueba estadística de $Levene$), se utilizó la prueba estadística $ANOVA$; si las varianzas no eran homogéneas, se utilizó la prueba estadística de $Welch$. Para muestras que no siguen una distribución normal (de acuerdo al resultado de la prueba estadística de $Shaphiro - Wilk$), se utilizó la prueba estadística no paramétrica de $Kruskal - Wallis$ para determinar si las muestras provenían de la misma distribución. Todas las pruebas estadísticas se realizaron considerando un nivel de significancia de $5\,\%$.

Para todos los experimentos se consideró un conjunto de 10 instancias, las cuales fueron elegidas de los principales conjuntos de prueba, considerando diferentes tamaños; el tamaño se denota como $n \times m$ donde n es el número de trabajos y m el número de máquinas. En la Tabla 5.1 se listan las instancias utilizadas para los experimentos. Se seleccionaron 6 instancias del conjunto dmu, dos instancias del conjunto swv y dos instancias del conjunto ta. Uno de los aspectos que se tuvo en cuenta a la hora de elegir las instancias fue el margen existente entre la cota inferior conocida (LB, Lower Bound) y el valor de la mejor solución conocida (UB, Upper Bound). En concreto, en todas las instancias elegidas, la UB y la LB difieren, con lo que era posible que se pudiesen mejorar las soluciones encontradas. Los valores de LB y UB fueron tomados de *Personal Web Site Oleg S.*

Instancia	Tamaño	LB	UB
dmu58	30×20	4319	4708
dmu60	30×20	4319	4739
dmu69	40×20	5423	5706
dmu70	40×20	5501	5888
dmu76	50×20	6342	6772
dmu77	50×20	6499	6778
swv06	20×15	1630	1669
swv12	50×10	2972	2977
ta42	30×20	1884	1941
ta44	30×20	1948	1982

TABLA 5.1: Conjunto de instancias utilizadas para los experimentos.

Para facilitar la interpretación de los resultados, se consideró un puntaje basado en las comparaciones de los algoritmos con el mejor método (método ganador), el cual es determinado considerando el menor promedio. De esta manera, para cada instancia se determina el método ganador (respecto a los métodos con que se están comparando), y el resto de métodos se compara con pruebas estadísticas para determinar si hay o no diferencia significativa. Cuando no hay diferencias estadísticas significativas, o el método es el ganador, se asigna un punto, en otro caso no se asigna ningún punto. De esta forma, el puntaje total nos permite determinar en cuantas instancias el método obtuvo resultados para los cuales no hay diferencias estadísticas significativas respecto al método ganador.

En todos los experimentos, los resultados de los modelos del método propuesto se denotan como: MAEDC_$C_{<Operador>}$_$D_{<Medida>}$_$< Estrategia >$ (Memetic Algorithm with Explicit Diversity Control) en la versión en secuencial, y como PMAEDC_$C_{<Operador>}$_$D_{<Medida>}$_$N < \#Procesos >$_$< Estrategia >$ (Parallel Memetic Algorithm with Explicit Diversity Control) en la versión en paralelo.

Para las ejecuciones de los algoritmos en secuencial (los modelos del método propuesto y los 2 algoritmos del estado del arte) se consideró un criterio de paro de 4 días (96 hrs). En las ejecuciones en paralelo se utilizaron 50 procesos, con un criterio de paro de 1 día (24 hrs). Los algoritmos fueron implementados en c++ con MPI, y compilados con mpic++ 7.4.0 y todas las pruebas fueron ejecutadas en un Intel(R) Xeon(R) CPU E5-2630 v4 (2.2 GH z - 32 GB de RAM), realizadas en el Cluster "Centro Estatal de Super Cómputo GTO, Puerto Interior".

5.1. Parametrización

Debido a que se tiene un número considerable de parámetros y a la limitación de recursos computacionales que se consideraron para los experimentos, sólo se estudia el efecto de algunos de los parámetros que consideramos más importantes, mientras que en otros se utilizó un valor fijo para todos los experimentos. Dado que la propuesta es un algoritmo híbrido, se tienen parámetros que son propios del mecanismo de intensificación (metaheurística de

Parámetro	Descripción	Valor
tt	Constante 1 para Tabu Tenure en TS	2
d_1	Constante 2 para Tabu Tenure en TS	5
d_2	Constante 3 para Tabu Tenure en TS	12
α	Máximo de iteraciones sin mejora en TS	100000

TABLA 5.2: Configuración de Parámetros para la Búsqueda Tabú.

trayectoria) y parámetros del algoritmo evolutivo. En este trabajo sólo nos enfocamos en la parte correspondiente al algoritmo evolutivo y se utilizó la configuración de parámetros para la búsqueda tabú que ha sido propuesta en otros trabajos. Todos los modelos considerados para los experimentos (los que surgen a partir del algoritmo propuesto) y los dos algoritmos del estado del arte, utilizan la configuración de parámetros para la búsqueda tabú (descritos en la Sección 3.7.3 - Proceso de Intensificación) con los valores detallados en la Tabla 5.2, los cuales corresponden a la configuración propuesta en (Cheng, Peng y Lü, 2016), cambiando únicamente el máximo número de iteraciones, aumentándolo de 12,000 hasta 100,000, ya que todos los experimentos consideraron ejecuciones a largo plazo. Para el TSPR se cambió también el criterio correspondiente a búsqueda tabú ligera, y se aumentó en la misma proporción que la búsqueda tabú pesada, pasando de 500 a 4,000 y de 12,500 a 100,000, respectivamente. El resto de parámetros se mantienen tal como se especifican en los artículos de referencia.

El método propuesto sólo considera un parámetro específico, que corresponde al tamaño de la población, el cual se fijó a un valor de 30, ya que este es el valor que consideran HEA y TSPR en su configuración de parámetros. Las estrategias de reemplazamiento consideran un parámetro adicional, que se utiliza para determinar el valor inicial del factor de penalización (D). Este parámetro se fijó como $0{,}5$ de la diversidad (medida como el DCN_{av}) en la población inicial. Finalmente, la estrategia de reemplazamiento considerando clústeres, considera dos parámetros adicionales: tamaño máximo del cluster y distancia mínima para penalización. Para todos los parámetros que consideraron valores fijos no se realizaron análisis exhaustivos, por lo que es posible que usando otros valores aún se mejoren más los resultados obtenidos. En la Tabla 5.3 se listan la configuración considerada para los parámetros considerados para el algoritmo propuesto.

5.1.1. Análisis de los componentes del Algoritmo

El primer experimentó consistió en realizar una comparación considerando diferentes operadores para cada uno de los de los principales componentes del algoritmo propuesto. En particular, el algoritmo propuesto tiene 3 componentes para los que se decidió analizar su impacto sobre el rendimiento: Operador de Cruce, Medida de similitud entre soluciones y estrategias de actualización de la población. Se consideró realizar estudios sobre estos componentes porque en esta tesis se proponen novedades para los mismos. Los operadores considerados para cada uno de los componentes del algoritmo son los siguientes:

Parámetro	Descripción	Valor
p	Tamaño de la población	30
d_{Ini}	Porcentaje de la diversidad inicial (DCN_{Ini}), utilizada para determinar el valor inicial del factor de penalización, calculado como: $D_{Ini} = d_{Ini} \times DCN_{Ini}$	0.5
$porTmaxCl$	Porcentaje de la población para determinar el tamaño máximo del cluster, calculado como: $tamMaxCl = porTmaxCl \times p + 1$	0.1
$porD_{min}$	Porcentaje del factor de penalización (D) para considerar la distancia mínima a la que puede ser seleccionado un individuo, calculado como: $D_{min} = porD_{min} \times D$	0.1

TABLA 5.3: Configuración de Parámetros para las estrategias de Actualización de la Población.

- Operadores de Cruce.
  - C_{LCS}, Cruce por subcadena común más larga, utilizado en HEA.
  - C_{Mch}, Cruce por máquinas complementarias, utilizado en HGA.
  - C_{Pos}, Cruce basado en la posición y el orden parcial de las operaciones en cada padre, propuesto en este trabajo.
- Medidas de similitud entre soluciones.
  - D_{LCS}, Medida basada en la longitud de la subcadena común más grande.
  - D_{Ham}, Medida basada en la distancia de Hamming.
  - D_{Pos}, Medida basada en la distancia entre las posiciones de las operaciones en cada máquina, propuesta en este trabajo.
- Estrategias de actualización de la población (Reemplazamiento).
  - BNP, Estrategia de selección del mejor No Penalizado (Best-Non Penalized).
  - $Cluster$, Estrategia de selección considerando agrupaciones (Clústeres) de soluciones, propuesta en este trabajo.

Para realizar una primera comparativa la entre los diferentes modelos (combinación de Operador de Cruce, Medida de Similitud y Estrategia de Actualización) se consideró el promedio del Error Relativo calculado con la mejor cota inferior conocida (Hoorn, 2018; *Personal Web Site Oleg S.*), el cual se calcula como:

$$Error\ Relativo = \frac{Resultado - LB}{LB}$$

En la Tabla 5.4 se presenta la media de los promedios de los errores relativos en cada instancia para cada uno de los modelos que surgen a partir de las diferentes combinaciones

Algoritmo	Error Relativo Promedio
MAEDC_$D_{Pos}C_{Mch}$_Cluster	0.0515146
MAEDC_$D_{Pos}C_{LCS}$_BNP	0.0517771
MAEDC_$D_{Pos}C_{Mch}$_BNP	0.0519305
MAEDC_$D_{LCS}C_{Mch}$_BNP	0.0520921
MAEDC_$D_{Pos}C_{LCS}$_Cluster	0.0521987
MAEDC_$D_{LCS}C_{Mch}$_Cluster	0.0523206
MAEDC_$D_{Pos}C_{Pos}$_Cluster	0.0524822
MAEDC_$D_{Ham}C_{Mch}$_BNP	0.0525893
MAEDC_$D_{Ham}C_{Mch}$_Cluster	0.0530878
MAEDC_$D_{LCS}C_{Pos}$_BNP	0.0533448
MAEDC_$D_{Pos}C_{Pos}$_BNP	0.0534245
MAEDC_$D_{LCS}C_{LCS}$_BNP	0.0535862
MAEDC_$D_{LCS}C_{Pos}$_Cluster	0.0536124
MAEDC_$D_{LCS}C_{LCS}$_Cluster	0.0536570
MAEDC_$D_{Ham}C_{Pos}$_Cluster	0.0536986
MAEDC_$D_{Ham}C_{Pos}$_BNP	0.0538472
MAEDC_$D_{Ham}C_{LCS}$_Cluster	0.0540222
MAEDC_$D_{Ham}C_{LCS}$_BNP	0.0542218
HEA	0.0533543
TSPR	0.0536622

TABLA 5.4: Promedio de las medias en las 10 instancias de los Errores de los Algoritmos en Secuencial.

de operador de cruce (C_{LCS}, C_{Mch}, C_{Pos}), medida de similitud (D_{LCS}, D_{Ham}, D_{Pos}) y estrategia de actualización de población (BNP, Cluster). Se calcula el error relativo para cada ejecución, se obtiene el promedio por instancia, y en la tabla se reporta la media del error relativo promedio en todas las instancias. Los resultados se listan ordenados de manera ascendente por el promedio del error relativo, de manera que el modelo que obtuvo el menor error relativo promedio es MAEDC_$D_{Pos}C_{Mch}$_Cluster.

Para realizar un análisis más amplio, en las siguientes secciones se describen las comparaciones para analizar cada uno de los 3 componentes principales (estrategias de reemplazamiento, medidas de similitud y operadores de cruce) que se consideraron.

Comparación de estrategias de Reemplazamiento

La estrategia de reemplazamiento es el componente principal del método propuesto para lograr un control de la diversidad. Aunque ambas estrategias utilizan gestión explícita de la diversidad, la estrategia con Clústeres fuerza a realizar una mayor intensificación al relajar la condición para penalizar individuos que están cercanos.

La comparación de las estrategias de actualización, se realizó considerando todas las configuraciones que surgen a partir de fijar el operador de cruce y la medida de similitud. En la Tabla 5.5 se muestra la comparación de los modelos, variando únicamente la estrategia de reemplazamiento. Para hacer la comparación, se toman los resultados obtenidos por la

Configuración	BNP	Cluster
$D_{LCS}C_{LCS}$	10(5*)	9(5*)
$D_{LCS}C_{Mch}$	9(5*)	8(5*)
$D_{LCS}C_{Pos}$	10(5*)	9(5*)
$D_{Ham}C_{LCS}$	9(5*)	10(5*)
$D_{Ham}C_{Mch}$	10(8*)	9(2*)
$D_{Ham}C_{Pos}$	10(5*)	10(5*)
$D_{Pos}C_{LCS}$	9(5*)	9(5*)
$D_{Pos}C_{Mch}$	8(2*)	10(8*)
$D_{Pos}C_{Pos}$	8(1*)	10(9*)
Total	83(36*)	84(49*)

TABLA 5.5: Resultados de la comparación de las Estrategias de Actualización de la población.

configuración con BNP y se comparan mediante la prueba de método de ganador. La columna BNP indica el número de veces que el método con este tipo de selección fue el mejor promedio o no hay diferencias estadísticas significativas respecto a los resultados obtenidos con la estrategia con Clústeres, la columna Cluster es análoga. En cada renglón se indica el puntaje de acuerdo al número de veces en que el método no presenta diferencias estadísticas significativas con el método ganador (incluyendo las veces en que fue el método ganador), y entre paréntesis el número de veces en las cuales es el método ganador. Al final se agrega un renglón con la suma del puntaje obtenido por cada una de las estrategias de actualización al considerar las diferentes configuraciones.

En general se tiene que ambas estrategias obtienen un desempeño similar en todas las configuraciones, y aunque en dos casos los modelos $D_{Pos}C_{Mch}$ y $D_{Pos}C_{Pos}$ con la estrategia con Cluster consiguen obtener un puntaje más alto y ser el método ganador en más ocasiones, en 8 de las 10 instancias no se tienen diferencias estadísticas significativas con respecto a los resultados obtenidos con la estrategia BNP. Algo similar ocurre con el modelo $D_{Ham}C_{Mch}$, en donde la estrategia BNP logra ser el método ganador en 8 instancias pero no hay una diferencia estadística significativa con respecto a los resultados obtenidos con la estrategia con Cluster en 9 de las 10 instancias de prueba.

Se tiene que ambas estrategias de reemplazamiento, obtienen resultados similares al considerar diferentes configuraciones de Operadores de Cruce y Medidas de Similitud. Con base en los resultados, se confirma que a largo plazo, los resultados con BNP o con el método basado en clústeres son muy similares (con una muy ligera superioridad del método basado en clústeres) con lo que la mayor complejidad del método basado en clústeres no se justifica en este sentido. Sin embargo, como veremos posteriormente, si se analiza como evoluciona el makespan se podrá ver que en el caso del método basado en clústeres se requiere menos tiempo para llegar a valores aceptables, lo que puede ser beneficioso para casos en que se quiera detener la ejecución al llegar a cierto nivel de calidad.

Configuración	D_{LCS}	D_{Ham}	D_{Pos}
C_{LCS} BNP	6(3*)	4(1*)	9(6*)
C_{Mch} BNP	10(4*)	9(3*)	9(3*)
C_{Pos} BNP	9(2*)	9(1*)	9(7*)
C_{LCS} Cluster	6(0*)	5(1*)	10(9*)
C_{Mch} Cluster	8(2*)	6(0*)	10(8*)
C_{Pos} Cluster	9(0*)	8(0*)	10(10*)
Total	48(11*)	41(6*)	57(43*)

TABLA 5.6: Resultados de la comparación de las Medidas de Similitud.

Comparación de Medidas de Similitud

La medida de similitud entre soluciones afecta directamente la forma en que se penalizan las soluciones en cualquiera de las dos estrategias de actualización de la población. El rango de valores para las medidas D_{LCS} y D_{Ham} está acotado por $N = n \times m$, total de operaciones en la instancia, lo cual podría ser un rango pequeño si consideramos que el tamaño del espacio de búsqueda es $(n!)^m$. Por otro lado, el rango de valores para la medida D_{Pos} es más amplio, esta en el orden de $n^2 \times m$.

Para verificar el desempeño del método propuesto, al utilizar diferentes métricas, se consideraron todas las configuraciones que surgen al fijar el operador de cruce y la estrategia de reemplazamiento. En la Tabla 5.6 se resumen los resultados que se obtienen al comparar las configuraciones para los diferentes modelos. Cada renglón representa la configuración con los dos componentes que se fijaron, y en cada columna se listan los resultados obtenidos al realizar la comparación mediante el método de ganadores.

Para el caso de D_{LCS}, la configuración C_{Mch} y BNP logra ser la mejor en 4 instancias, y en el resto no hay diferencia estadística significativa con el método ganador, por lo cual obtiene un puntaje de 10. La medida D_{Pos}, en todas las configuraciones con la estrategia de reemplazamiento con Cluster logra al menos obtener resultados que no tienen diferencias estadísticas significativas (en la media) con el método ganador. Finalmente, la medida D_{Ham}, para el caso BNP, sólo en un modelo (C_{LCS} BNP) obtiene un puntaje que es muy inferior respecto a las otras métricas. En general, cuando consideramos los modelos en los cuales se utiliza la estrategia BNP, se tiene que no hay una diferencia tan grande en usar las diferentes medidas. Sin embargo, con la estrategia basada en Cluster se obtienen diferencias importantes al utilzar las diferentes métricas.

Se tiene que la medida de similitud D_{Pos}, es robusta ante diferentes configuraciones de operador de cruce y estrategias de actualización. D_{LCS} logra obtener mejores resultados considerando las configuraciones con BNP. La medida D_{Ham}, obtiene los peores resultados en todas las configuraciones. Esto también se refleja en los puntajes totales que se obtienen para cada una de las medidas. Por tanto, podemos concluir que la medida de similitud propuesta en esta tesis captura mejor las propiedades del problema, siendo capaz de alcanzar resultados más robustos.

Configuración	C_{LCS}	C_{Mch}	C_{Pos}
D_{LCS} BNP	7(2*)	10(5*)	6(3*)
D_{Pos} BNP	10(4*)	9(3*)	9(3*)
D_{Ham} BNP	8(6*)	8(3*)	4(1*)
D_{LCS} Cluster	5(2*)	8(6*)	6(2*)
D_{Pos} Cluster	7(2*)	8(7*)	6(1*)
D_{Ham} Cluster	8(0*)	9(6*)	7(4*)
Total	45(16*)	52(30*)	38(14*)

TABLA 5.7: Resultados de la comparación de los Operadores de Cruce.

Comparación de Operadores de Cruce

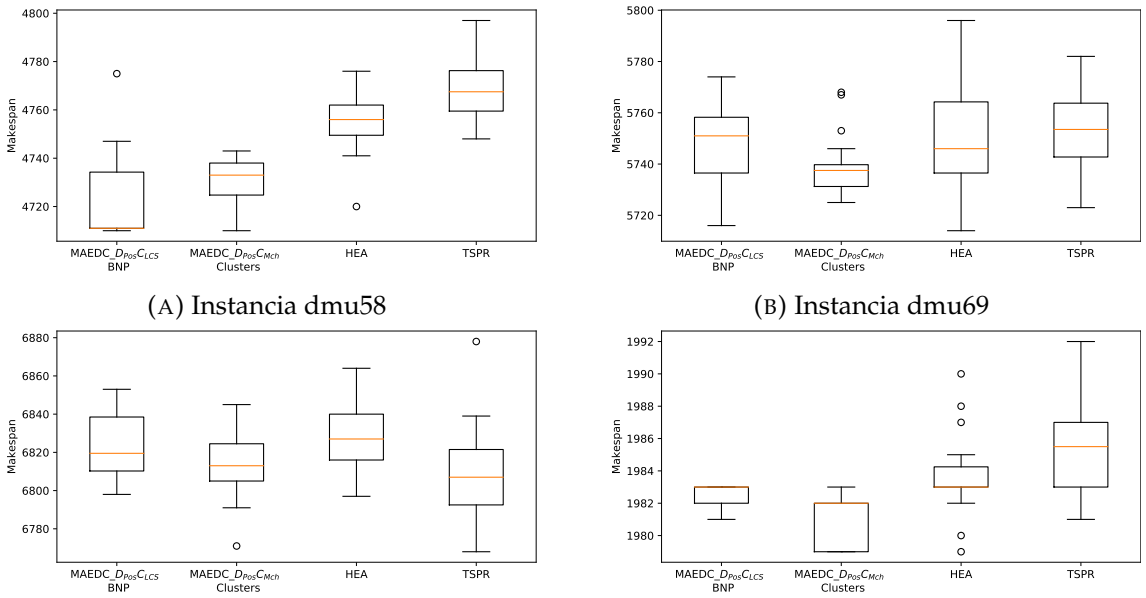
El Operador de Cruce es uno de los componentes principales de los algoritmos evolutivos, ya que determina la forma en que las soluciones se combinan para lograr explorar nuevas áreas o intensificar zonas en las que las soluciones que se cruzan son cercanas.

Para realizar la comparación entre los diferentes operadores de cruce se consideraron los modelos que surgen al fijar la estrategia de reemplazamiento y la medida de similitud. En la Tabla 5.7 se resumen los resultados que se obtienen al comparar las configuraciones que se obtienen al variar solamente el operador de cruce.

Los resultados para el operador C_{LCS} en general son buenos, y en la mayoría de las configuraciones logra obtener resultados que son similares a los del método ganador. Sin embargo, el operador C_{Mch} logró el mejor desempeño ya que en todas las configuraciones consigue obtener un puntaje alto (mayor o igual a 8), por lo cual con ese operador se consigue obtener, en la mayoría de los casos, resultados que son al menos similares (sin diferencia estadística significativa) a los del método ganador. Por lo anterior, se tiene que este operador es robusto ante diferentes configuraciones.

Por otro lado, el operador propuesto C_{Pos} obtuvo el puntaje más bajo en gran parte de las configuraciones, por lo cual esta propuesta no consigue ser competitiva con los operadores propuestos en otros algoritmos. Sin embargo, en todas configuraciones logra ser el método ganador en al menos una instancia, lo cual podría ser un indicador de que el operador tiene buenas características para algún tipo de instancias.

Si nos fijamos en el número de casos en que cada operador obtuvo la mejor media, vemos que C_{Mch} ganó en 30 casos, mientras que C_{LCS} y C_{Pos} sólo ganaron en 16 y 14 casos respectivamente. Con base en estos resultados, parece que no es demasiado prometedor realizar cruces combinando información de las permutaciones, sino directamente de las máquinas de forma completa. Igualmente, tanto la nueva propuesta (C_{Pos}) como C_{LCS} resultaron ser métodos ganadores en algunas instancias, por lo que de cara a futuro parece prometedor seguir investigando sobre métodos que sean capaces de combinar información de permutaciones en formas alternativas.



(A) Instancia dmu58

(B) Instancia dmu69

(C) Instancia dmu76

(D) Instancia ta44

FIGURA 5.1: Boxplot del Makespan obtenido en las ejecuciones en secuencial en diferentes instancias.

5.2. Comparación con el Estado del Arte - Algoritmos en Secuencial

Para el segundo experimento se consideraron los modelos que obtuvieron menor error relativo en cada una de las estrategias de reemplazamiento ($D_{Pos}C_{LCS}$ BNP y $D_{Pos}C_{Mch}$ Cluster), y se realizó una comparación con los dos algoritmos del estado del arte.

La Figura 5.1 muestra los boxplots del mejor makespan encontrado en cada ejecución, para 4 instancias; en el resto de la instancias de prueba se presenta un comportamiento similar a las 4 que se muestran en los diagramas. Para las instancias dmu58 (Figura 5.1a) y ta44 (Figura 5.1d) los boxplots muestran que el desempeño de las dos variantes del método propuesto es superior comparado con los algoritmos del estado del arte. Por otro lado, para las instancias dmu69 (Figura 5.1b) se puede verificar que el desempeño de los algoritmos es similar, y no hay una clara superioridad de alguno. En la instancia dmu76 (Figura 5.1c) se puede comprobar que TSPR tiene un mejor desempeño y en general logra mejores resultados, aunque tal como se muestra en el boxplot, puede llegar a tener ejecuciones malas, por lo que no es un método demasiado robusto.

En dos instancias (dmu58 y ta44), los boxplots del HEA muestran ejecuciones etiquetadas como atípicas, en las cuales el algoritmo logra resultados de alta calidad en dichas instancias, por lo cual vemos que este algoritmo (HEA) en algunas ocasiones sí logra obtener buenas ejecuciones, pero en general el algoritmo no es robusto.

		MAEDC				HEA		TSPR	
		$D_{Pos}C_{LCS}$ BNP		$D_{Pos}C_{Mch}$ Cluster					
Instancia	UB	$Best$	M_{av}	$Best$	M_{av}	$Best$	M_{av}	$Best$	M_{av}
dmu58	4708	4710	**4723***	4710	4730.7	4720	4754.8	4748	4768
dmu60	4739	4740	**4769**	4756	**4766.95***	4758	4772.7	4764	4778.1
dmu69	5706	5716	5748.75	5725	**5738.85***	5714	**5749.5**	5723	5752.5
dmu70	5888	5878	**5921.8**	5883	**5919.9***	5907	5929.2	5906	5933.95
dmu76	6772	6798	6822.95	6771	**6814.6**	6797	6829.4	6768	**6808.55***
dmu77	6778	6796	**6827.85***	6760	**6828.05**	6805	6845	6803	**6833.65**
swv06	1669	1667	**1677.25**	1670	**1676.85***	1674	1677.8	1672	1677.95
swv12	2977	2976	**2977.3***	2977	2980	2976	2979.85	2976	2979.6
ta42	1941	1939	**1942.5**	1939	**1942.4***	1940	**1944.35**	1940	1947.1
ta44	1982	1981	1982.4	1979	**1981.05***	1979	1983.65	1981	1985.45
Total Ganados		7(3*)		8(6*)		2(0*)		2(1*)	

TABLA 5.8: Resumen de los resultados obtenidos en 10 instancias en ejecuciones con los algoritmos en secuencial.

La Tabla 5.8 muestra un resumen de las comparaciones entre los modelos del método propuesto y dos algoritmos del estado del arte, HEA y TSPR. Las columnas $Best$ y M_{av} muestran el mejor makespan y el promedio del makespan, para cada uno de los algoritmos considerados. Para cada instancia se marca con un * el mejor promedio (el más bajo), y se establece que en esa instancia el algoritmo correspondiente es el método ganador. Adicionalmente, aparece subrayado el mejor valor obtenido por un algoritmo cuando este logró superar al mejor valor conocido, señalado en la columna UB (Upper Bound). Los algoritmos que no presentan diferencia estadística significativa, comparados con el método ganador, son resaltados en **negrita**. Al final de la tabla se incluye un renglón en el que se establece el total de veces en las cuales el método resultó ganador o sin diferencia significativa con el ganador, y entre paréntesis el número de veces en las cuales fue el ganador.

El modelo $D_{Pos}C_{LCS}BNP$ logra ser el ganador en 3 instancias, pero consigue obtener resultados que son similares (sin diferencia estadística significativa) a los del método ganador en otras 4 instancias. Sin embargo, el modelo $D_{Pos}C_{Mch}Cluster$ logra mejores resultados, ya que gana en 6 instancias y en 2 más logra resultados similares al método ganador. Con ambos modelos del algoritmo propuesto, se logró superar el mejor valor conocido en 5 instancias (diferentes para cada modelo) y en las demás obtienen resultados cercanos a la cota superior (UB).

El algoritmo HEA, consigue resultados similares al método ganador en dos instancias, pero en ninguna logra ser el método ganador. Sin embargo, este algoritmo logró superar los mejores valores conocidos en 3 instancias, pero en el resto de las instancias los resultados son inferiores a los que se obtienen con el método propuesto.

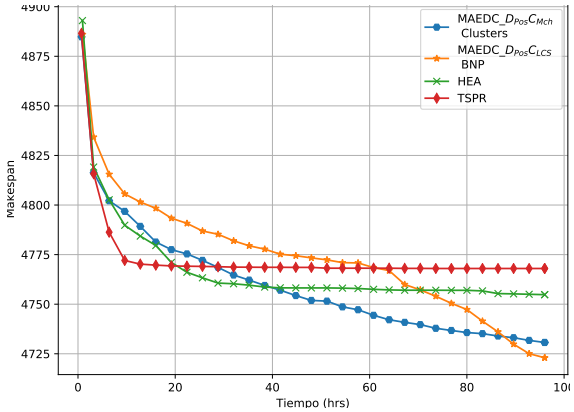
El otro algoritmo del estado del arte, TSPR, consigue obtener muy buenos resultados en las dos instancias (dmu76 y dmu77) con tamaño más grande, de 50×20, del conjunto de prueba utilizado. En particular, en la instancia dmu76 logró ser el método ganador, y en la dmu77 logra resultados que no tienen diferencia estadística significativa con el método ganador.

Los resultados anteriores se pueden entender mejor al analizar el comportamiento de los algoritmos, considerando la evolución de la función de aptitud y la evolución de la diversidad, medida como el DCN_{av}. En la Figura 5.2 se muestran las gráficas de la evolución en el tiempo del promedio del valor de la función objetivo (makespan) de la mejor solución obtenida y el promedio del DCN_{av} en las ejecuciones de los algoritmos.

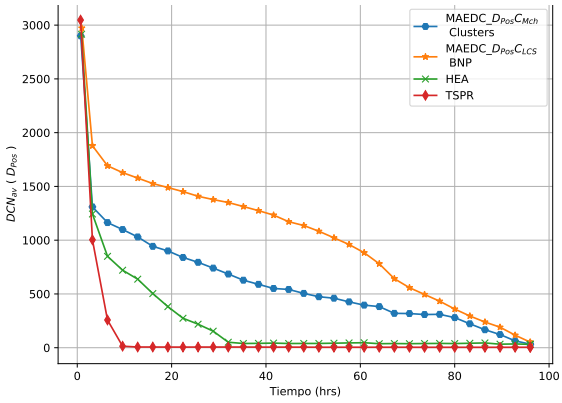
En las instancias dmu58 (Figura 5.2b) y ta44 (Figura 5.2h) los dos algoritmos del estado del arte tienen una convergencia prematura, ya que en menos de la mitad del tiempo pierden la diversidad en la población. Las dos configuraciones del método propuesto logran mantener diversidad durante toda la ejecución y al final consiguen obtener soluciones de mejor calidad (Figuras 5.2a y 5.2g).

En las Figuras 5.2c y 5.2d se observa el comportamiento típico de los algoritmos que se presenta en la mayoría de las instancias de prueba. Los dos algoritmos del estado del arte logran conseguir al inicio soluciones que son mejores respecto a las que se obtienen (en esa primera etapa de la ejecución) con las variantes del algoritmo propuesto. Sin embargo, una vez que los algoritmos HEA y TSPR convergen (pierden la diversidad en la población), lo cual ocurre generalmente antes de llegar a la mitad del tiempo de ejecución total, ya no logran mejorar las soluciones. Los modelos del método propuesto tardan una mayor cantidad de tiempo en lograr llegar a soluciones de alta calidad y sólo hasta el final de la ejecución logran obtener mejores soluciones. En particular, con la estrategia basada en clústeres se logra reducir el tiempo necesario para llegar a soluciones de alta calidad, aunque al final las soluciones que obtienen las dos estrategias de reemplazamiento consiguen resultados similares.

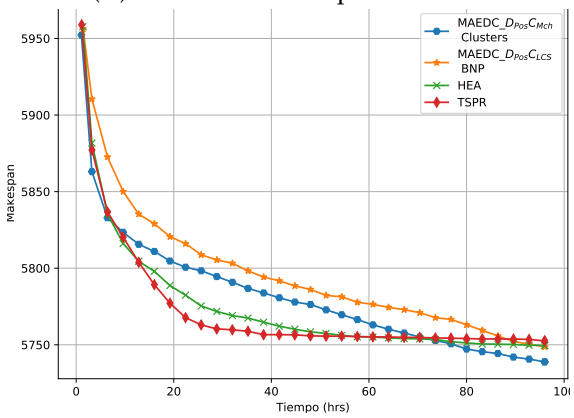
Por otra lado, en la instancia dmu76, en donde TSPR logra un buen desempeño (Figura 5.2e), cabe destacar que no presenta el problema de convergencia prematura y prácticamente se produce un descenso lineal de la diversidad durante todo el proceso de optimización (Figura 5.2b). Por ello, podemos ver que tanto HEA como TS/PR son menos estables pues existen casos en los que la cantidad de diversidad mantenida es apropiada, pero en otros casos, se pierde por lo que se desperdician gran parte de los recursos computacionales asignados.



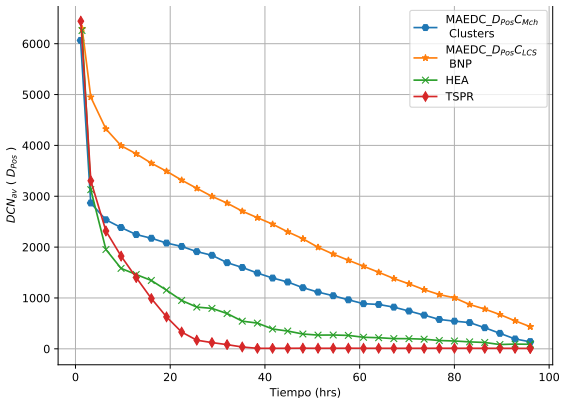
(A) Evolución makespan - dmu58



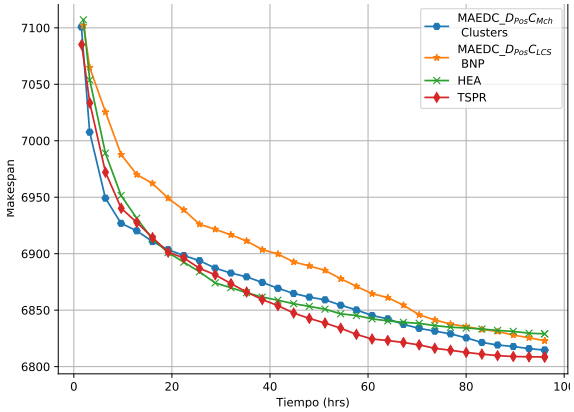
(B) Evolución DCN_{av} - dmu58



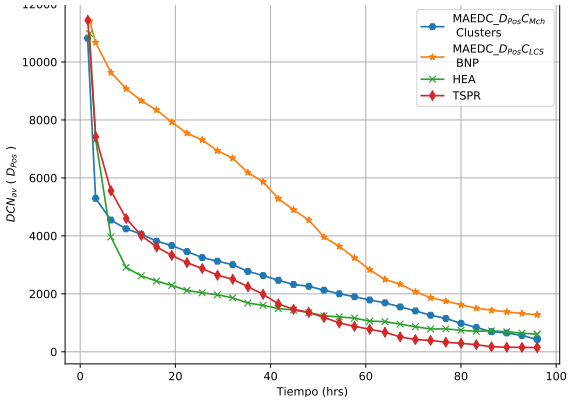
(C) Evolución makespan - dmu69



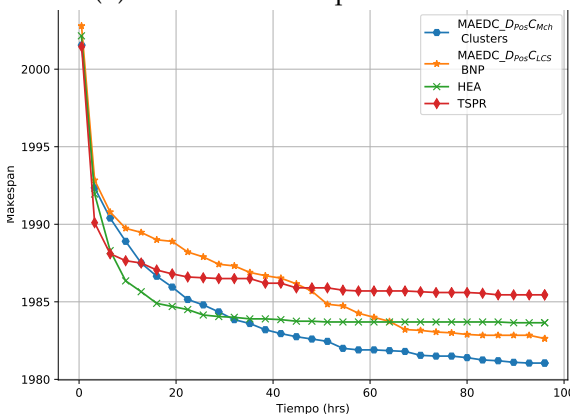
(D) Evolución DCN_{av} - dmu69



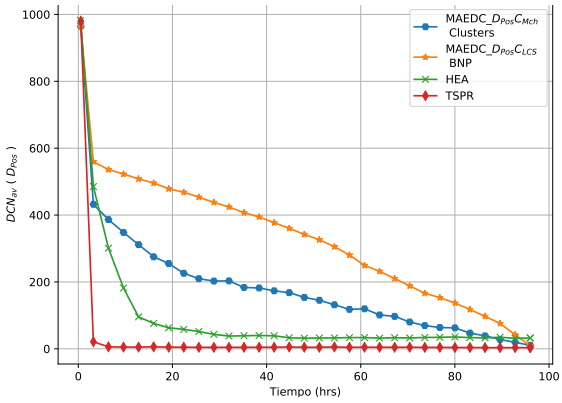
(E) Evolución makespan - dmu76



(F) Evolución DCN_{av} - dmu76



(G) Evolución makespan - ta44



(H) Evolución DCN_{av} - ta44

FIGURA 5.2: Evolución del makespan promedio de la mejor solución y DCN_{av} en 4 instancias de los algoritmos en secuencial.

Algoritmo	Error Promedio
PMAEDC_$D_{Pos}C_{LCS}$_N50_BNP	0.0029407
PMAEDC_$D_{Pos}C_{Mch}$_N50_BNP	0.0033073
PMAEDC_$D_{Pos}C_{Mch}$_N50_Cluster	0.0038040
PMAEDC_$D_{Pos}C_{LCS}$_N50_Cluster	0.0044401
PMAEDC_$D_{LCS}C_{Mch}$_N50_BNP	0.0045176
HEA_N50	0.0068178
TSPR_N50	0.0074394

TABLA 5.9: Error Relativo Promedio de los algoritmos en paralelo en las 10 instancias de prueba.

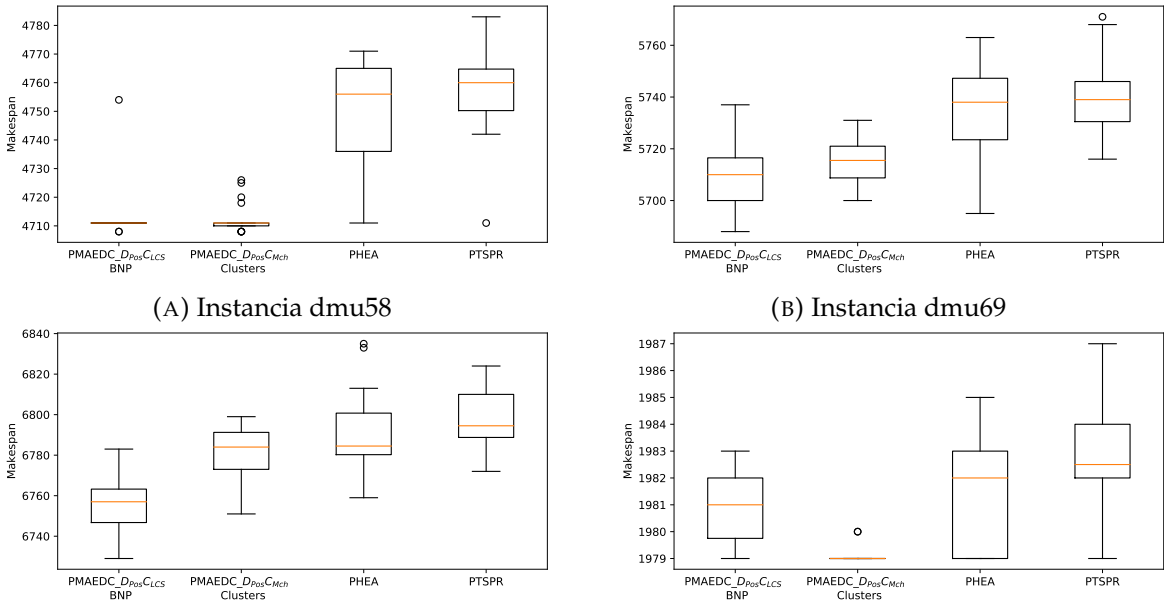
5.3. Comparación con el Estado del Arte - Algoritmos en Paralelo

Considerando los modelos en secuencial, el algoritmo propuesto logró obtener resultados prometedores que son competitivos con los resultados obtenidos por los algoritmos del estado del arte; en algunas instancias ya se lograba superar los mejores resultados conocidos. Sin embargo, se realizaron pruebas adicionales considerando mayores tiempos de ejecución, y se observó que los resultados podían seguir mejorando. Por lo anterior, se decidió realizar un experimento adicional, considerando una paralelización de los algoritmos. De esta manera, se logró reducir el tiempo necesario para lograr un número mayor de iteraciones.

Para el tercer experimento, se consideraron los 5 modelos que lograron mejor error relativo promedio y los 2 algoritmos del estado del arte (HEA y TSPR) para realizar comparaciones con ejecuciones de los algoritmo en paralelo. Para todas las ejecuciones de los algoritmos en paralelo se utilizaron 50 procesos, con un criterio de paro de 24 hrs.

En la Tabla 5.9 se muestra el error relativo promedio de los algoritmos en paralelo. Todos los algoritmos, los modelos del método propuesto y las versiones en paralelo de los dos del estado del arte (PHEA,Parallel HEA y PTSRP, Parallel TSPR), mejoraron respecto al error relativo promedio en las versiones en secuencial, lograron obtener un menor error relativo promedio, es decir, todos consiguieron mejorar su desempeño, al menos en cuanto a calidad de los resultados que se obtiene. Sin embargo, la diferencia entre los diferentes modelos y los algoritmos del estado del arte ya es muy clara. Mientras que error relativo promedio para los modelos del método propuesto se encuentra entre $0{,}003$ y $0{,}005$, las versiones de los algoritmos del estado del arte sólo alcanzan $0{,}006$ y $0{,}007$.

Se debe notar también que los dos modelos que consideran la métrica D_{Pos}, con BNP, son superiores al resto de los modelos. En general se tiene que el modelo con BNP da mejores resultados en paralelo cuando se utiliza una métrica adecuada. Dado que el rango de valores para D_{LCS} es menor, no se logran aprovechar los recursos de la mejor manera para realizar una buena exploración. Por otro lado, las versiones con Cluster promueven rápidamente la intensificación y por tanto realizan una menor exploración. Por tanto, en esquemas en paralelo, con tal cantidad de recursos no es necesario realizar demasiada intensificación y es preferible mantener mayor diversidad para promover la exploración del espacio de búsqueda y tener un mejor aprovechamiento de los recursos.



(A) Instancia dmu58

(B) Instancia dmu69

(C) Instancia dmu76

(D) Instancia ta44

FIGURA 5.3: Boxplot del Makespan obtenido en las ejecuciones en paralelo en diferentes instancias.

Para hacer un análisis más detallado del comportamiento de los algoritmos, se seleccionó el mejor modelo con cada una de las estrategias de actualización de la población y se realizó la comparación con las versiones en paralelo de los dos algoritmos del estado del arte.

En la Figura 5.3 se muestran los boxplot en 4 instancias del valor de makespan de la solución obtenida en las ejecuciones de los algoritmos. En general se observa que el desempeño de los dos modelos del algoritmo propuesto logran mejores resultados y son más robustos (el boxplot el más cerrado); por ejemplo para la instancia dmu58 (Figura 5.3a), el boxplot de ambas configuraciones del algoritmo propuesto es muy cerrado. Para las instancias dmu69 (Figura 5.3b) y dmu76 (Figura 5.3c), los boxplot están más abiertos, pero en general se tiene que el desempeño es mejor, aunque hay ocasiones que se tienen ejecuciones malas, estas son de una calidad de al menos la mediana de la versión en paralelo de los algoritmos del estado del arte.

En el caso de la instancia ta44 (Figura 5.3d) la versión con clústeres parece lograr un mejor desempeño, lo cual se puede explicar debido al tamaño de esta instancia. Dado que es una instancia más pequeña que las otras, la intensificación que se promueve con cluster logra que prácticamente en todas las ejecuciones se llegue a resultados de muy alta calidad.

En la Tabla 5.10 se resumen los resultados obtenidos considerando al mejor modelo obtenido con la la estrategia BNP, el mejor con la estrategia por Cluster y los dos algoritmos del estado del arte. De acuerdo al puntaje total (número de veces en que el método logra obtener resultados sin diferencia estadística significativa respecto al método ganador), se tiene que los dos modelos del algoritmo propuesto obtienen 8 puntos, sin embargo, si consideramos las veces en que cada uno fue el método ganador, se tiene que la estrategia BNP es superior a la basada en clústeres, ya que obtuvieron 7 y 3 puntos, respectivamente. Los dos

		PMAEDC				PHEA		PTSPR	
		$D_{Pos}C_{LCS}$ BNP		$D_{Pos}C_{Mch}$ Cluster					
Instancia	UB	$Best$	M_{av}	$Best$	M_{av}	$Best$	M_{av}	$Best$	M_{av}
dmu58	4708	4708	**4712.85**	4708	**4712.5***	4711	4748.5	4711	4757.65
dmu60	4739	4721	**4739.95***	4721	**4748**	4752	4769.2	4756	4767.65
dmu69	5706	5688	**5709.65***	5700	**5715.3**	5695	5735.7	5716	5739.8
dmu70	5888	5868	**5886.95***	5877	5896.95	5887	5915	5907	5926.4
dmu76	6772	6729	**6755.2***	6751	6781.55	6759	6790.75	6772	6797.45
dmu77	6778	6751	**6778.85***	6758	**6787**	6775	6821.4	6773	6816.05
swv06	1669	1667	**1669.75***	1667	**1671.05**	1667	1674.05	1669	1675.1
swv12	2977	2976	**2976***	2976	**2976.15**	2972	**2976**	2976	2977
ta42	1941	1939	1942.05	1939	**1940.7***	1939	1943.5	1943	1946.15
ta44	1982	1979	1981	1979	**1979.1***	1979	1981.15	1979	1982.45
Total Ganados		8(7*)		8(3*)		1(0*)		0(0*)	

TABLA 5.10: Resumen de los resultados obtenidos en 10 instancias en ejecuciones de los algoritmos en paralelo.

modelos del algoritmo propuesto lograron superar el mejor valor conocido en 9 instancias, y en la restante logran igualar el valor. El modelo con BNP logra obtener mejores resultados en 4 instancias (dmu69, dmu70, dmu76 y dmu77) que el modelo con la estrategia de reemplazamiento basada en cluster.

El algoritmo PHEA consiguió superar el mejor valor conocido en 8 instancias, y en particular en 1 instancia (swv12) logra conseguir una solución que tiene un valor de makespan igual a la cota inferior para esa instancia, por lo cual se tiene que la solución generada es óptima. Este método puede llegar a tener ejecuciones muy buenas (tal como sucedió en la instancia swv12), aunque en general los resultados no son demasiado robustos y la calidad es inferior a la obtenida por los modelos del algoritmo propuesto. Por otro lado, el algoritmo PTSPR obtuvo el mejor valor conocido en 3 instancias, pero aún así sus resultados fueron muy inferiores al de los otros algoritmos; específicamente este algoritmo no logró obtener resultados similares (sin diferencia estadística significativa) a los de ninguno de los métodos ganadores para las diferentes instancias.

La Figura 5.4 muestra las gráficas de la evolución en el tiempo del promedio del valor de la función objetivo (makespan) de la mejor solución obtenida y el promedio del DCN_{av} en las ejecuciones de los algoritmos.

En todas las instancias dmu (Figuras 5.4a, 5.4c y 5.4e), los dos modelos del algoritmo propuesto tienen un proceso de mejora continua, respecto al makespan de la mejor solución en la población, durante toda la ejecución ya que en todo momento se mantiene suficiente diversidad para seguir realizando exploración del espacio de búsqueda, lo cual permite aprovechar de una mejor manera los recursos computacionales.

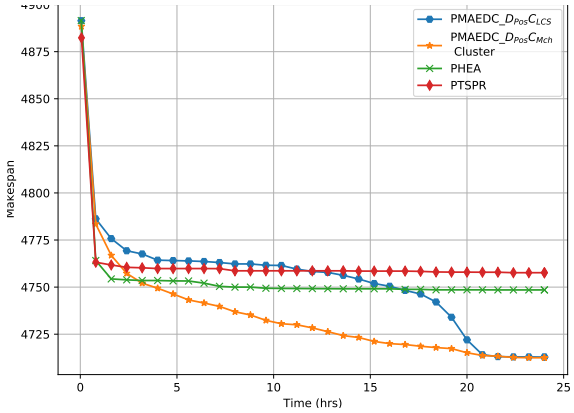
En las instancias de menor tamaño, dmu58 y ta44, se tiene que la estrategia basada en clústeres consigue obtener resultados de alta calidad (Figuras 5.4a y 5.4g) en la primera parte ejecución (primeras 5 hrs), que incluso son mejores que los mejores logrados por PHEA

PTSPR durante toda la ejecución. Mientras que la estrategia con Cluster consigue superar desde el inicio a PHEA y PTSRP, la estrategia con BNP sólo consigue superarlos hasta el final. En particular en la instancia dmu76, la estrategia con BNP logra superar claramente a la estrategia basada en clústeres, por lo que en general se tiene que con la estrategia BNP se logra encontrar mejores zonas y al final se consigue realizar una buena intensificación que permite obtener soluciones de alta calidad.

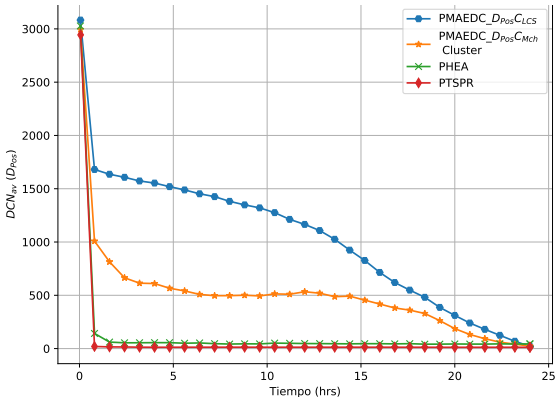
Respecto a la evolución de la diversidad (5.4b, 5.4d, 5.4f y 5.4h), medida como el el promedio del DCN_{av} en las ejecuciones de los algoritmos, en el esquema basado en cluster la cantidad de diversidad está por debajo del BNP debido a la formación de clústeres, pero sin la aparición de convergencia prematura.

Por otro lado, en los esquemas en paralelo de los algoritmos del estado del arte, PHEA y PTSPR, se acelera aún más la convergencia de los algoritmos, ya que por la naturaleza del algoritmo en paralelo, rápidamente surgen soluciones de alta calidad en una misma zona del espacio de búsqueda, y por la naturaleza de estos algoritmos estas soluciones son elegidas rápidamente con lo que se llega a tener convergencia prematura muy rápidamente.

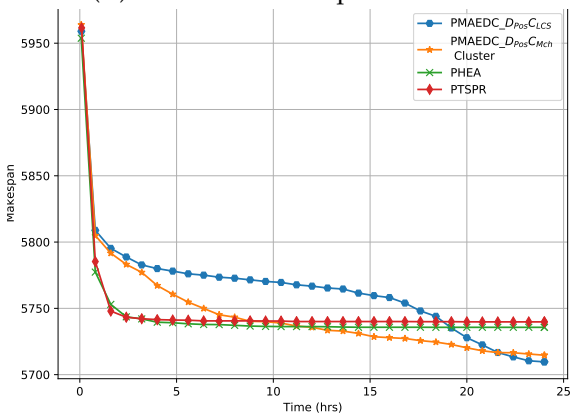
En conclusión, los análisis en paralelo han permitido mostrar que los esquemas basados en control explícito de diversidad son especialmente útiles cuando se dispone de una gran cantidad de recursos computacionales, por lo que pronosticamos que en el futuro, con la proliferación de máquinas cada vez más potentes que pueden realizar más iteraciones, este tipo de algoritmos será aún más popular. Cabe destacar también la importante contribución de alcanzar nuevas mejores soluciones en 9 instancias, siendo casos que se han utilizado como tests por varias décadas y que han sido tratados por multitud de métodos.



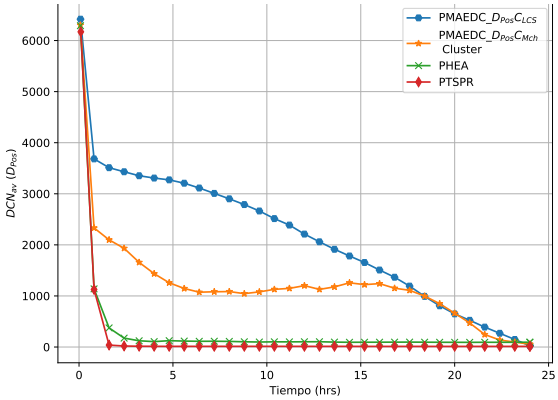
(A) Evolución makespan - dmu58



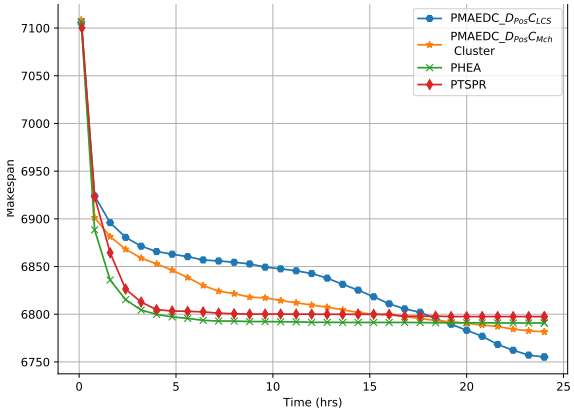
(B) Evolución DCN_{av} - dmu58



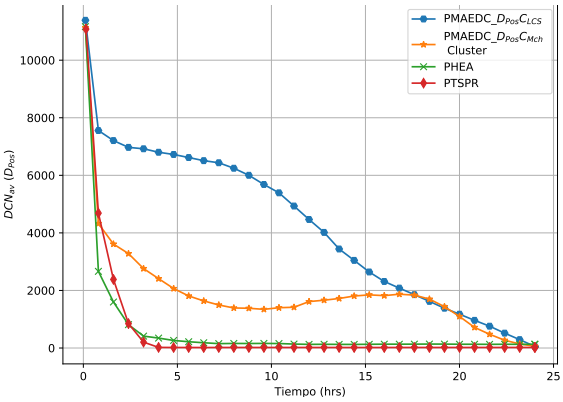
(C) Evolución makespan - dmu69



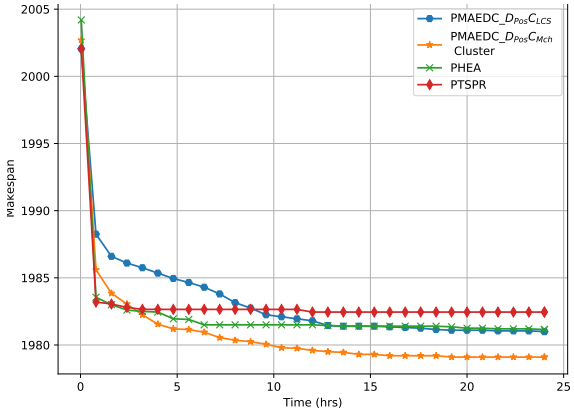
(D) Evolución DCN_{av} - dmu69



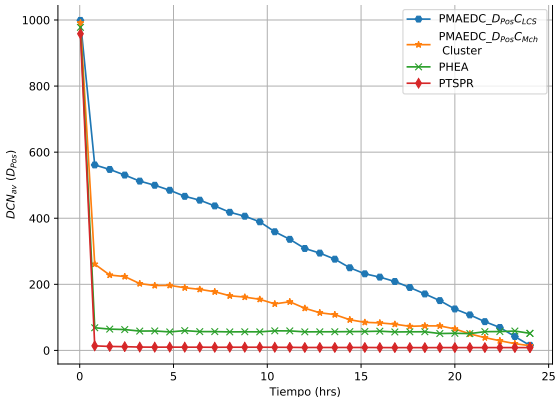
(E) Evolución makespan - dmu76



(F) Evolución DCN_{av} - dmu76



(G) Evolución makespan - ta44



(H) Evolución DCN_{av} - ta44

FIGURA 5.4: Evolución del makespan promedio de la Mejor solución y DCN_{av} en 4 instancias de los algoritmos en paralelo.

Capítulo 6

Conclusiones y Trabajo Futuro

6.1. Conclusiones

Las metaheurísticas han probado tener éxito en diferentes problemas NP-Difíciles. En este trabajo se desarrolló un método híbrido que considera un manejo explícito de la diversidad para resolver el $JSSP$. Las principales novedades del método desarrollado son la inclusión de estrategias de actualización de la población (reemplazamiento) que controlan la diversidad de forma explícita y el diseño de una nueva medida de similitud y de un operador de cruce. La validación experimental analiza los beneficios aportados por cada uno de los nuevos componentes, contrastando además los desarrollos realizados con dos algoritmos del estado del arte, mostrándose los beneficios aportados por el nuevo método desarrollado.

Con el control explícito de diversidad se logró obtener buenos resultados, sin embargo se requieren ejecuciones a largo plazo, por lo que es particularmente útil cuando se consideran instancias grandes o de una mayor dificultad, ya que en esos casos se requiere de una mayor cantidad de recursos computacionales, y es importante hacer un uso adecuado para evitar que los recursos sean desperdiciados.

En general, la mayor parte de esfuerzos de investigación se habían realizado en desarrollar buenos operadores de cruce y métodos de intensificación local. Sin embargo, los resultados encontrados en esta tesis muestran que es muy importante realizar un control explícito de la diversidad, particularmente relacionándolo con el criterio de parada y que es de vital importancia elegir una buena medida de similitud para lograr mantener suficiente diversidad en la población. En este trabajo se propuso una nueva medida de similitud entre soluciones, la cual considera un rango más amplio de posibles resultados que las usadas hasta la fecha, lo que permite establecer de una manera más precisa la cercanía entre soluciones.

El operador de cruza propuesto obtuvo resultados promisorios en instancias pequeñas, siendo capaz de ser el esquema ganador en algunas instancias. Sin embargo, no es adecuado para instancias de mayor tamaño. Los resultados mostraron que el operador de cruce adecuado dependía en gran medida de las características de las instancias, lo que abre nuevas posibilidades para el desarrollo de operadores adaptativos o que de alguna forma consideren de forma explícita el tipo de instancia que se está tratando.

Las dos estrategias de reemplazamiento, BNP y la basa en agrupaciones (Clusters) de individuos, mostraron ser efectivas para obtener buenos resultados. La estrategia con Clusters fue mejor en las pruebas con los algoritmos en secuencial, y en el caso de las ejecuciones en paralelo la estrategia BNP resultó superior. Esto nos dice que a corto plazo, o menores recursos computacionales, la mayor intensificación lograda con Clusters logra alcanzar mejores resultados rápidamente. A largo plazo, o con una mayor cantidad de recursos computacionales (como en el caso de cómputo en paralelo), al mantener una mayor diversidad con la estrategia BNP se logra un mejor aprovechamiento de los recursos, ya que hay mayor exploración y por tanto se logran alcanzar mejores zonas; en este escenario, cuando se tienen ejecuciones a largo plazo, o una gran cantidad de recursos computacionales, no es de particular interés obtener soluciones a un corto plazo, por lo cual también es preferible usar estrategias que mantengan mayor diversidad durante la ejecución.

Aunque con las ejecuciones en secuencial ya se lograban superar algunos de los mejores valores conocidos en las instancias de los principales conjuntos del prueba para el JSSP, con los algoritmos en paralelo se logró superar por un margen más amplio. El método propuesto, en su versión en paralelo, logró mejorar los resultados en 33 instancias de los principales conjuntas de prueba en la literatura para el JSSP y se consiguió alcanzar dos soluciones que son óptimas, en un caso con la versión paralela del HEA y en otro con una versión paralela del método propuesto. Este logro se ha verificado utilizando plataformas dedicadas a este problema en específico, y en las cuales el método desarrollado en esta tesis está listado actualmente como el método a superar.

6.2. Trabajo Futuro

Los métodos propuestos en esta tesis han proporcionado un avance importante en la resolución del JSSP. Aún así, aún se tiene un amplio margen de investigación para lograr mejores algoritmos.

Una posible línea de trabajo futuro consiste en mejorar las propuestas paralelas. Al respecto, cabe destacar que muchos optimizadores paralelos se centran en hacer crecer el tamaño de la población considerado. En nuestro modelo, hacer crecer la población de forma excesiva podría provocar un cuello de botella en el proceso maestro, por lo que considerar otros modelos de paralelización como los basados en islas o los celulares parece prometedor.

Los algoritmos que han tenido más éxito en el JSSP son híbridos que consideran una metaheurística de trayectoria y generalmente son el componente que requiere de mayores recursos computacionales (principalmente tiempo de ejecución). En este trabajo la investigación se centró principalmente en el diseño de metaheurísticas para resolver el JSSP, dejando en segundo término la complejidad de los algoritmos implementados. Sin embargo el tener implementaciones eficientes puede reducir significativamente el tiempo requerido para los diferentes procesos; en la implementación desarrollada se tienen dos componentes principales que repercuten de forma significativa en el tiempo de ejecución requerido para alcanzar

buenas soluciones. En el caso secuencial gran parte de los recursos son usados por la búsqueda tabú. El tiempo requerido por la misma tal vez podría reducirse considerando estructuras de datos avanzadas, por lo que sería de gran interés tratar de realizar mejoras al respecto. Por su parte, en el caso paralelo la estrategia de reemplazamiento puede convertirse rápidamente en un cuello de botella si se llegan a usar muchos procesos esclavos. Por ello, sería interesante también en rediseñar la misma para utilizar estructuras de datos que permitan acelerar los cálculos hechos en la misma.

Otro componente en cual aún hay posibilidades de mejor son las metaheurísticas de trayectoria, particularmente en las vecindades que se consideran para la representación utilizada. Las vecindades que se han propuesto tienen la desventaja de que no siempre es posible alcanzar soluciones que están cerca, con respecto a las diferentes medidas de similitud. En particular, soluciones que están a un solo movimiento de un óptimo local, no siempre es posible alcanzarlas con las vecindades que se tienen actualmente, ya que todas se basan en considerar movimientos internos dentro de un bloque. Si se consideran movimientos que consideran una operación dentro y una fuera del bloque, se puede realizar una mejor intensificación en zonas promisorias del espacio de búsqueda; considerar este tipo de vecindades incrementa significativamente el costo computacional que requieren los métodos y se tiene la desventaja de que muchos de estos movimientos no son factibles, y por tanto se tendría un desperdicio de los recursos al considerar una vecindad así. Por ello, un posible campo de investigación es estudiar diferentes tipos de vecindades y las condiciones necesarias que se deben cumplir para generar movimientos factibles.

Una gran cantidad de investigaciones se centran en el estudio de los operadores para manipular las soluciones con diferentes metaheurísticas. Sin embargo, las pruebas realizadas en este trabajo probaron que considerar buenas medidas de similitud puede ser suficiente para mejorar los algoritmos, particularmente para lograr un mejor control de la diversidad, por lo que una posible línea de trabajo es realizar un estudio más amplio de medidas de similitud para la representación utilizada, o bien considerar posibles medidas de similitud en otros tipos de representaciones de soluciones para el JSSP.

Finalmente, otro posible tema de investigación es el estudio del manejo de diversidad en otras metaheurísticas poblacionales, como las basadas en esquemas de Path Relinking. En este trabajo solo se consideró el esquema principal de los algoritmos evolutivos, sin embargo, otras estrategias poblacionales podrían también beneficiarse del control explícito de diversidad.

Apéndice A

Nuevos Valores Mejores Conocidos

En las pruebas realizadas se lograron encontrar nuevos valores mejores conocidos en diferentes instancias. En la Tabla A.1 se listan todas las instancias para las cuales se actualizó la solución con el mejor valor conocido en la plataforma (*Personal Web Site Oleg S.*). En la primera columna se indica el nombre de la instancia y entre paréntesis el tamaño (considerando la notación $n \times m$, n número de trabajos y m número de máquinas). Las siguientes dos columnas muestran las mejores cotas inferiores (LB) y superiores (UB) que estaban reportadas en la plataforma. En la columna $NuevoBKS$ se reporta el mejor valor conseguido por el método propuesto; en la columna $Diferencia$ se indica la resta entre la cota superior conocida antes de realizar este trabajo (UB) y el valor de la nueva mejor solución (Nuevo BKS). En la columna $Modelo$ se indica la configuración con la cual se consiguió obtener la solución con el valor reportado; el modelo se especifica considerando la notación:
$< Algoritmo > D\_$ <Medida de Similitud> $C\_$ <Operador de Cruce>
$N\_$ <Número de procesos en Paralelo> <Estrategia de Reemplazamiento>
$p\_$ <Tamaño de la población>.
En negritas se han resaltado los resultados en las instancias en donde se consiguió alcanzar un solución con un valor óptimo, ya que se ha alcanzado la mejor cota inferior conocida.

Con un signo de interrogación (?) se han señalado las instancias para las cuales existen publicaciones que mencionan haber alcanzado una mejor cota superior (UB) de la que se refiere en la tabla, sin embargo dichas soluciones no han sido publicadas en la plataforma mencionada; entre paréntesis se indica la referencia y la cota que es reportada en el artículo correspondiente.

Cuando el tamaño de población se incrementa, se tiene una pérdida de rendimiento ya que la estrategia de actualización de la población se vuelve un cuello de botella en el proceso maestro. Por tal razón, el número de generaciones que se hacen es inferior (aproximadamente la mitad), sin embargo al tener una población de mayor tamaño se consigue realizar una mejor exploración del espacio de búsqueda, lo cual incrementa la posibilidad de encontrar mejores zonas y por tanto alcanzar mejores soluciones. Por esa razón se realizaron algunas pruebas con tamaños de población de en torno a 30, mientras que en otros casos se llegaron a usar 100 individuos. Nótese que no se hicieron pruebas exhaustivas, sino que se realizaron ejecuciones para diferentes instancias tratando de probar diferentes parametrizaciones hasta que se superaba el mejor valor conocido.

Instancia (tamaño)	LB	UB	Nuevo BKS	Diferencia	Modelo
dmu19 (30x20)	3669	3765	3764	1	PMAEDC $D_{LCS}C_{Mch}$ N100 Cluster p30
dmu20 (30x20)	3604	3708	3703	5	PMAEDC $D_{Pos}C_{LCS}$ N100 BNP p30
dmu45 (20x15)	3001	3269	3266	3	PMAEDC $D_{Pos}C_{Mch}$ N100 BNP p100
dmu52 (30x15)	4065	4304	4303	1	PMAEDC $D_{Pos}C_{Mch}$ N100 BNP p30
dmu53 (30x15)	4141	4390	4378	12	PMAEDC $D_{Pos}C_{Mch}$ N100 BNP p100
dmu54 (30x15)	4202	4362	4361	1	PMAEDC $D_{Pos}C_{LCS}$ N100 BNP p100
dmu55 (30x15)	4140	4270	4263	7	PMAEDC $D_{Pos}C_{LCS}$ N100 BNP p30
dmu57 (30x20)	4302	4655	4653	2	PMAEDC $D_{Pos}C_{LCS}$ N50 BNP p30
dmu58 (30x20)	4319	4708	4701	7	PMAEDC $D_{LCS}C_{Mch}$ N50 BNP p30
dmu59 (30x20)	4217	4619	4616	3	PMAEDC $D_{Pos}C_{Mch}$ N100 BNP p30
dmu60 (30x20)	4319	4739	4721	18	PMAEDC $D_{Pos}C_{LCS}$ N50 BNP p30
dmu61 (40x15)	4917	5172	5171	1	PMAEDC $D_{Pos}C_{LCS}$ N50 BNP p30
dmu62 (40x15)	5033	5251	5248	3	PMAEDC $D_{LCS}C_{Mch}$ N100 BNP p30
dmu63 (40x15)	5111	5323	5313	10	PMAEDC $D_{Pos}C_{LCS}$ N100 BNP p30
dmu64 (40x15)	5130	5240	5226	14	PMAEDC $D_{Pos}C_{LCS}$ N100 BNP p30
dmu65 (40x15)	5105	5190	5184	6	PMAEDC $D_{Pos}C_{Mch}$ N100 BNP p30
dmu66 (40x20)	5391	5704	5701	3	PMAEDC $D_{Pos}C_{Mch}$ N100 BNP p30
dmu68 (40x20)	5426	5765	5763	2	PMAEDC $D_{Pos}C_{LCS}$ N100 BNP p30
dmu69 (40x20)	5423	5706	5688	18	PMAEDC $D_{Pos}C_{LCS}$ N50 BNP p30
dmu70 (40x20)	5501	5888	5868	20	PMAEDC $D_{Pos}C_{LCS}$ N50 BNP p30

Tabla A.1Nuevos mejores valores encontrados (continua . . .)

Instancia (tamaño)	LB	UB	Nuevo BKS	Diferencia	Modelo
dmu71 (50x15)	6080	6210	6207	3	PMAEDC $D_{LCS}C_{Mch}$ N100 BNP p30
dmu73 (50x15)	6001	6150	6136	14	PMAEDC $D_{LCS}C_{Mch}$ N100 BNP p30
dmu76 (50x20)	6342	6772	6718	54	PMAEDC $D_{Pos}C_{Mch}$ N50 BNP p30
dmu77 (50x20)	6499	6778	6747	31	PMAEDC $D_{Pos}C_{Mch}$ N50 BNP p30
dmu78 (50x20)	6586	6762	6755	7	PMAEDC $D_{Pos}C_{LCS}$ N100 BNP p60
dmu79 (50x20)	6650	6919	6910	9	PMAEDC $D_{Pos}C_{LCS}$ N100 BNP p100
dmu80 (50x20)	6459	6638	6634	4	PMAEDC $D_{Pos}C_{LCS}$ N100 BNP p100
swv06 (20x15)	1630	1669	1667	2	PMAEDC $D_{Pos}C_{LCS}$ N50 BNP p30
swv12 (50x10)	2972	2977	**2972**	5	PHEA N50 p30
ta34 (30x15)	1828	1829	**1828**	1	PMAEDC $D_{Pos}C_{LCS}$ N100 BNP p30
ta40 (30x15)	1651	1671 [1]	1670	1	PMAEDC $D_{Pos}C_{Mch}$ N100 Cluster p100 TS150k
ta42 (30x20)	1884	1941 [2]	1939	2	PMAEDC $D_{LCS}C_{Mch}$ N50 BNP p30
ta44 (30x20)	1948	1982 [3]	1979	3	PMAEDC $D_{Pos}C_{Mch}$ N50 BNP p30

TABLA A.1: Nuevos mejores valores encontrados.

[1] 1669? Gonçalves y Resende, 2014
[2] 1937? Gonçalves y Resende, 2014
[3] 1979? Hoorn, 2018

Bibliografía

Adams, Joseph, Egon Balas y Daniel Zawack (1988). «The shifting bottleneck procedure for job shop scheduling». En: *Management science* 34.3, págs. 391-401.

Alba, Enrique (2005). *Parallel metaheuristics: a new class of algorithms*. Vol. 47. John Wiley & Sons.

Applegate, David y William Cook (1991). «A computational study of the job-shop scheduling problem». En: *ORSA Journal on computing* 3.2, págs. 149-156.

Balas, Egon y Alkis Vazacopoulos (1998). «Guided local search with shifting bottleneck for job shop scheduling». En: *Management science* 44.2, págs. 262-275.

Blum, Christian y Andrea Roli (2003). «Metaheuristics in combinatorial optimization: Overview and conceptual comparison». En: *ACM computing surveys (CSUR)* 35.3, págs. 268-308.

Cheng, TCE, Bo Peng y Zhipeng Lü (2016). «A hybrid evolutionary algorithm to solve the job shop scheduling problem». En: *Annals of Operations Research* 242.2, págs. 223-237.

Cormen, Thomas H y col. (2009). *Introduction to algorithms*. MIT press.

Dell'Amico, Mauro y Marco Trubian (1993). «Applying tabu search to the job-shop scheduling problem». En: *Annals of Operations research* 41.3, págs. 231-252.

Demirkol, Ebru, Sanjay Mehta y Reha Uzsoy (1998). «Benchmarks for shop scheduling problems». En: *European Journal of Operational Research* 109.1, págs. 137-141.

Eiben, Agoston E, James E Smith y col. (2003). *Introduction to evolutionary computing*. Vol. 53. Springer.

Eikelder, Huub MM ten y col. (1999). «Sequential and parallel local search algorithms for job shop scheduling». En: *Meta-Heuristics*. Springer, págs. 359-371.

Fisher, Henry (1963). «Probabilistic learning combinations of local job-shop scheduling rules». En: *Industrial scheduling*, págs. 225-251.

Gonçalves, José Fernando y Mauricio GC Resende (2014). «An extended Akers graphical method with a biased random-key genetic algorithm for job-shop scheduling». En: *International Transactions in Operational Research* 21.2, págs. 215-246.

González, Oscar M y col. (2017). «A memetic algorithm for the capacitated vehicle routing problem with time windows». En: *2017 IEEE Congress on Evolutionary Computation (CEC)*. IEEE, págs. 2582-2589.

Hart, Emma, Peter Ross y David Corne (2005). «Evolutionary scheduling: A review». En: *Genetic Programming and Evolvable Machines* 6.2, págs. 191-220.

Hoorn, Jelke J van (2018). «The Current state of bounds on benchmark instances of the job-shop scheduling problem». En: *Journal of Scheduling* 21.1, págs. 127-128.

Huang, Kuo-Ling y Ching-Jong Liao (2008). «Ant colony optimization combined with taboo search for the job shop scheduling problem». En: *Computers & operations research* 35.4, págs. 1030-1046.

Koumousis, Vlasis K. y Christos P. Katsaras (2006). «A saw-tooth genetic algorithm combining the effects of variable population size and reinitialization to enhance performance». En: *IEEE Transactions on Evolutionary Computation*. ISSN: 1089778X. DOI: 10.1109/TEVC.2005.860765.

Lawrence, S (1984). «Resouce constrained project scheduling: An experimental investigation of heuristic scheduling techniques (Supplement)». En: *Graduate School of Industrial Administration, Carnegie-Mellon University*.

Lenstra, Jan K y AHG Rinnooy Kan (1979). «Computational complexity of discrete optimization problems». En: *Annals of Discrete Mathematics*. Vol. 4. Elsevier, págs. 121-140.

Liu, Bo, Ling Wang y Yi-Hui Jin (2007). «An effective PSO-based memetic algorithm for flow shop scheduling». En: *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)* 37.1, págs. 18-27.

Lu, Sun y col. (2011). «Infeasibility test algorithm and fast repair algorithm of job shop scheduling problem». En: *Journal of Southeast University (English Edition)* 27.1, págs. 88-91.

Michalewicz, Zbigniew y David B Fogel (2013). *How to solve it: modern heuristics*. Springer Science & Business Media.

Nagata, Yuichi e Isao Ono (2018). «A guided local search with iterative ejections of bottleneck operations for the job shop scheduling problem». En: *Computers & Operations Research* 90, págs. 60-71.

Nagata, Yuichi y Satoshi Tojo (2009). «Guided ejection search for the job shop scheduling problem». En: *European Conference on Evolutionary Computation in Combinatorial Optimization*. Springer, págs. 168-179.

Nowicki, Eugeniusz y Czeslaw Smutnicki (1996). «A fast taboo search algorithm for the job shop problem». En: *Management science* 42.6, págs. 797-813.

Nowicki, Eugeniusz y Czesław Smutnicki (2005). «An advanced tabu search algorithm for the job shop problem». En: *Journal of Scheduling* 8.2, págs. 145-159.

Pardalos, Panos M y Oleg V Shylo (2006). «An algorithm for the job shop scheduling problem based on global equilibrium search techniques». En: *Computational Management Science* 3.4, págs. 331-348.

Pardalos, Panos M, Oleg V Shylo y Alkis Vazacopoulos (2010). «Solving job shop scheduling problems utilizing the properties of backbone and "big valley"». En: *Computational Optimization and Applications* 47.1, págs. 61-76.

Peng, Bo, Zhipeng Lü y TCE Cheng (2015). «A tabu search/path relinking algorithm to solve the job shop scheduling problem». En: *Computers & Operations Research* 53, págs. 154-164.

Romero Ruiz, Emmanuel y Carlos Segura (2018). «Memetic algorithm with hungarian matching based crossover and diversity preservation». En: *Computación y Sistemas* 22.2.

Ronald, S. «More distance functions for order-based encodings». En: *1998 IEEE International Conference on Evolutionary Computation Proceedings. IEEE World Congress on Computational*

Intelligence (Cat. No.98TH8360). IEEE, págs. 558-563. ISBN: 0-7803-4869-9. DOI: 10.1109/ICEC.1998.700089. URL: http://ieeexplore.ieee.org/document/700089/.

Segura, Carlos y col. (2015). «A novel diversity-based evolutionary algorithm for the traveling salesman problem». En: *Proceedings of the 2015 Annual Conference on Genetic and Evolutionary Computation*. ACM, págs. 489-496.

Segura, Carlos y col. (2016a). «A novel diversity-based replacement strategy for evolutionary algorithms». En: *IEEE transactions on cybernetics* 46.12, págs. 3233-3246.

Segura, Carlos y col. (2016b). «Improving diversity in evolutionary algorithms: New best solutions for frequency assignment». En: *IEEE Transactions on Evolutionary Computation* 21.4, págs. 539-553.

Sevkli, Mehmet y M Emin Aydin (2007). «Parallel variable neighbourhood search algorithms for job shop scheduling problems». En: *IMA Journal of Management Mathematics* 18.2, págs. 117-133.

Shylo, Oleg. *Personal Web Site Oleg S.* URL: http://optimizizer.com/jobshop.php (visitado 31-05-2019).

Shylo, Oleg V y Hesam Shams (2018). «Boosting Binary Optimization via Binary Classification: A Case Study of Job Shop Scheduling». En: *arXiv preprint arXiv:1808.10813*.

Storer, Robert H, S David Wu y Renzo Vaccari (1992). «New search spaces for sequencing problems with application to job shop scheduling». En: *Management science* 38.10, págs. 1495-1509.

Taillard, Eric (1993). «Benchmarks for basic scheduling problems». En: *european journal of operational research* 64.2, págs. 278-285.

Taillard, Eric D (1994). «Parallel taboo search techniques for the job shop scheduling problem». En: *ORSA journal on Computing* 6.2, págs. 108-117.

Talbi, El-Ghazali (2009). *Metaheuristics: from design to implementation*. Vol. 74. John Wiley & Sons.

Van Laarhoven, Peter JM, Emile HL Aarts y Jan Karel Lenstra (1992). «Job shop scheduling by simulated annealing». En: *Operations research* 40.1, págs. 113-125.

Wang, Yuping y col. (2012). «A new hybrid genetic algorithm for job shop scheduling problem». En: *Computers & Operations Research* 39.10, págs. 2291-2299.

Yamada, Takeshi y Ryohei Nakano (1992). «A Genetic Algorithm Applicable to Large-Scale Job-Shop Problems.» En: *PPSN*. Vol. 2, págs. 281-290.

Zhang, Chao Yong y col. (2008). «A very fast TS/SA algorithm for the job shop scheduling problem». En: *Computers & Operations Research* 35.1, págs. 282-294.

Zhang, ChaoYong y col. (2007). «A tabu search algorithm with a new neighborhood structure for the job shop scheduling problem». En: *Computers & Operations Research* 34.11, págs. 3229-3242.