Universidad Nacional de La Pampa, Facultad de Ingeniería

PROYECTO FINAL DE GRADO DE INGENIERÍA EN SISTEMAS

"Un algoritmo de evolución diferencial híbrido para resolver el problema de planificación de tareas flexible"

Autor: Franco Marcelo Morero – Grado de Ingeniero en Sistemas.

Directora: Dra. Salto Carolina

Presentado: General Pico, La Pampa. 2020 – Aprobado el día 28/09/2020

Jurados: Alfonso, Hugo. Becker, Pablo. Rivera, Belén. **Filiación**: Facultad Ingeniería UNLPam.

Resumen: El problema de planificación job shop flexible (FJSSP, en inglés) es uno de los problemas de optimización más desafiantes, con aplicabilidad práctica en ambientes de producción real. En esta tesis se propone un algoritmo de evolución diferencial (DE, en inglés) simple para resolver el problema en cuestión. Para representar una solución al FJSSP se adopta una representación de vectores reales, la cual requiere un mecanismo de conversión muy simple para obtener una planificación factible. Consecuentemente, el algoritmo DE continúa trabajando en el dominio continuo para explorar el espacio de búsqueda del problema que es de carácter discreto. Además, para mejorar la habilidad de búsqueda local y lograr un equilibrio entre la exploración y explotación, se incorpora un algoritmo de búsqueda local simple. También, se incluye paralelismo a las operaciones del DE para mejorar la eficiencia del algoritmo. Los resultados obtenidos confirman que el rendimiento del DE mejora en forma significativa al incorporar las propuestas incluidas en este estudio.

Además, los resultados de las pruebas muestran que este algoritmo es competitivo en comparación con la mayoría de los existentes.

Palabras clave: Evolución Diferencial, Metaheurísticas, FJSSP, Optimización.

Abstract: The flexible job shop scheduling problem (FJSSP) is one of the most challenging optimization problems, with practical usage in real production environments. In this thesis, a simple differential evolution (DE) algorithm is proposed to solve the problem. To represent a solution to the FJSSP, a representation of real vectors is adopted, which requires a very simple conversion mechanism to obtain feasible scheduling. Consequently, the DE algorithm continues to work in the continuous domain to explore the search space of the problem that is discrete in nature. In addition, to improve local search ability and strike a balance between exploration and exploitation, a simple local search algorithm is incorporated. Also, parallelism to the operations of the DE is included to improve the efficiency of the algorithm. The results obtained confirm that the performance of the DE improves significantly when incorporating the proposals included in this study.

Furthermore, the test results show that this algorithm is competitive compared to other ones present in the literature.

Key Words: Differential Evolution, Metaheuristics, FJSSP, Optimization .

UNIVERSIDAD NACIONAL DE LA PAMPA

FACULTAD DE INGENIERÍA

TESIS DE GRADO EN INGENIERÍA EN SISTEMAS

Un algoritmo de evolución diferencial híbrido para resolver el problema de planificación de tareas flexible

Autor

Franco Marcelo Morero

Fecha: julio, 2020

Director

Dra. Salto Carolina

Resumen

El problema de planificación *job shop* flexible (FJSSP, en inglés) es uno de los problemas de optimización más desafiantes, con aplicabilidad práctica en ambientes de producción real. En esta tesis se propone un algoritmo de evolución diferencial (*DE*, en inglés) simple para resolver el problema en cuestión. Para representar una solución al FJSSP se adopta una representación de vectores reales, la cual requiere un mecanismo de conversión muy simple para obtener una planificación factible. Consecuentemente, el algoritmo *DE* continúa trabajando en el dominio continuo para explorar el espacio de búsqueda del problema que es de carácter discreto. Además, para mejorar la habilidad de búsqueda local y lograr un equilibrio entre la exploración y explotación, se incorpora un algoritmo de búsqueda local simple. También, se incluye paralelismo a las operaciones del DE para mejorar la eficiencia del algoritmo. Los resultados obtenidos confirman que el rendimiento del DE mejora en forma significativa al incorporar las propuestas incluidas en este estudio. Además, los resultados de las pruebas muestran que este algoritmo es competitivo en comparación con la mayoría de los existentes.

Agradecimientos

A mi directora de tesis, Dra. Carolina Salto, quien con sus conocimientos, su paciencia y dedicación me guió para poder llevar a cabo este trabajo. Gracias por el tiempo prestado para consultar dudas, su ayuda indispensable para realizar el análisis estadístico de los datos de esta investigación y por la lectura y revisión de la tesis.

Al grupo de trabajo LISI, quienes colaboraron desinteresadamente en esta tarea.

A mis padres, mis hermanos y a toda mi familia, que me apoyan día a día para que pueda llegar a cumplir mis objetivos.

A la Universidad Nacional de la Pampa, donde cursé esta carrera de grado, que me permitió conocer a personas maravillosas, tanto formadores como amigos a quienes recordaré por siempre.

¡Eternamente gracias!

Índice general

Índice de figuras **3**

Índice de tablas **4**

1. Introducción **5**

1.1. Descripción del trabajo y justificación 5

1.2. Objetivos 6

1.3. Contribuciones 6

1.4. Metodología 7

1.5. Cronograma de trabajo 8

1.6. Organización de la tesis 8

2. Metaheurísticas **10**

2.1. Introducción 10

2.2. Algoritmos metaheurísticos 11

2.3. Clasificación de las metaheurísticas 12

2.3.1. Métodos basados en trayectoria 13

2.3.2. Métodos basados en población 15

2.4. Metaheurísticas paralelas 19

3. Evolución Diferencial **22**

3.1. Introducción 22

3.2. Algoritmo *DE*: propuesta de Storn & Price 23

3.2.1. Inicialización 24

3.2.2. Mutación 24

3.2.3. Recombinación 25

3.2.4. Selección 26

3.3. Descripción de *DE* 26

4. *DE* híbrido para *FJSSP* **28**
4.1. Introducción . . . 28
4.2. FJSSP: *Flexible Job Shop Scheduling Problem* . . . 29
4.3. Representación y evaluación de soluciones . . . 30
4.4. Decodificación de soluciones . . . 31
4.5. DE y búsqueda local . . . 32
4.6. *DE* y paralelismo . . . 33
4.7. *HDE* para el *FJSSP* . . . 35

5. Estudios experimentales **36**
5.1. Introducción . . . 36
5.2. Diseño experimental . . . 37
5.3. Resultados experimentales . . . 39
5.3.1. Resultados variando los parámetros F y Cr . . . 39
5.3.2. Resultados de *DE* con búsqueda local . . . 42
5.3.3. Resultados de *HDE* y su paralelismo . . . 44
5.4. Comparación de *HDE* con la literatura . . . 44

6. Conclusión **46**

Bibliografía **48**

Índice de figuras

4.1. Ejemplo del proceso de decodificación de una solución para una instancia del problema de FJSSP. . . . 32
4.2. Paralelización del algoritmo DE. . . . 34
4.3. Paralelización de la búsqueda local. . . . 34

5.1. Boxplot del error relativo del *DE* para distintos valores de F y Cr. . . . 41
5.2. Boxplot del error relativo del *DE* para distintos valores de F y Cr. . . . 42
5.3. Boxplot del error relativo del *HDE* para distintos valores de P_{BL}. . . . 43
5.4. *Speedup* por instancia del FJSSP. . . . 44

Índice de tablas

4.1. Tabla de tiempo de procesamiento 30

5.1. Instancias propuestas por Brandimarte 37

5.2. Valores de los parámetros 38

5.3. Mejores valores de C_{max} encontrados por el *DE* con diferentes valores de F y Cr para todas las instancias del FJSSP 40

5.4. Valores medios de C_{max} encontrados por el *DE* con diferentes valores de F y Cr para todas las instancias del FJSSP 40

5.5. Valores de C_{max} encontrados por el *HDE* con diferentes valores de P_{BL} para todas las instancias de FJSSP. 43

5.6. Comparación entre *HDE* y metaheurísticas basadas en población de la literatura 45

Capítulo 1

Introducción

1.1. Descripción del trabajo y justificación

Las técnicas de optimización juegan un rol importante en el campo de la ciencia y la ingeniería. En las últimas décadas, se han desarrollado varios algoritmos para resolver problemas de optimización complejos. La biología evolutiva o el comportamiento de enjambres inspiraron la mayoría de esos métodos. En este marco evolutivo o de inteligencia de enjambre se han propuesto varias clases de algoritmos que incluyen: algoritmos genéticos [18, 44], algoritmos metaheurísticos [54], evolución diferencial (DE) [29], optimización de colonias de hormigas (ACO) [30], optimización por enjambre de partículas (PSO) [33], algoritmo de colonia de abejas artificiales (ABC) [56], entre otros.

La evolución diferencial (DE) es un algoritmo evolutivo desarrollado para resolver funciones de espacios continuos. DE es un método de búsqueda estocástico basado en población, que a diferencia de otros algoritmos metaheurísticos poblacionales, enfatiza más la mutación que la recombinación. Al utilizar operaciones de bajo costo computacional (usualmente lineales), resulta muy eficiente y eficaz al resolver problemas de optimización.

DE ha sido recientemente usado para optimización global en una gran variedad de áreas de problemas tal como planificación [58], diseño ingenieril [59], problemas de optimización con restricción [11], entre otros [12, 13]. En particular, dentro de los ambientes de manufactura, nos encontramos con el problema de planificación job shop flexible (FJSSP). Está probado que este problema es NP-duro [4], debido a que se tiene que asignar cada operación de un trabajo a una máquina apropiada (problema de ruteo) y secuenciar las operaciones asignadas en cada máquina (problema de secuencia-

miento). El objetivo que se persigue es minimizar el tiempo necesario para completar todos los trabajos. A medida que aumenta la cantidad de trabajos, por consiguiente la cantidad de operaciones, y la cantidad de máquinas disponibles, la complejidad del problema crece y por consiguiente la complejidad computacional para resolverlo. La resolución de este problema con algoritmos como DE, trae aparejado tiempos de procesamiento altos. Una manera de hacer frente a estos tiempos de procesamiento altos es utilizar alguna técnica de paralelización que permita distribuir el trabajo en los distintos procesadores y/o cores de nuestro equipo informático. Una de las estrategias es utilizar una configuración paralela maestro-esclavos, utilizando una interfaz de programación de aplicaciones (API), como lo es openMP, para la programación multiproceso de memoria compartida.

1.2. Objetivos

El objetivo general de este trabajo es: diseñar un algoritmo DE híbrido para resolver el FJSSP utilizando estrategias de distribución del trabajo computacional.

Como objetivos específicos, para alcanzar el objetivo general, se plantean los siguientes:

- Realizar un análisis del estado del arte sobre algoritmos DE y propuestas paralelas.
- Realizar un análisis del estado del arte en la resolución del FJSSP con algoritmos evolutivos.
- Diseñar e implementar el DE híbrido para el FJSSP.
- Diseñar e implementar el DE híbrido paralelo para el FJSSP.
- Realizar la experimentación correspondiente y el análisis de los resultados obtenidos.

1.3. Contribuciones

La contribución de esta tesis es la obtención de un algoritmo de evolución diferencial híbrido, incorporando una búsqueda local, que permite resolver el problema de *job shop flexible* explotando el paralelismo en su ejecución. Esto es el resultado de mi

trabajo de investigación al incorporarme en el grupo LISI, dentro del marco del programa de Becas de Iniciación en Investigación de la UNLPam en los periodos 2019-2020 y 2020-2021, el cual se respalda con las siguientes publicaciones:

- Morero F., Salto C., and Bermudez C. Parallelism and Hybridization in Differential Evolution to Solve the Flexible Job Shop Scheduling Problem, Journal of Computer Science Technology, vol 20 (1) 2020.
- Alfonso H., Bermudez C., Morero F., Minetti G., y Salto C. Utilización de Sistemas Inteligentes para optimizar el diseño de redes de distribución de agua en General Pico - La Pampa, en Libro de actas del XXII Workshop de Investigadores en Ciencias de la Computación (WICC 2020), Universidad Nacional de Patagonia Austral, El Calafate, 2020.
- Morero F., Bermudez C., and Salto C. A Simple Differential Evolution Algorithm to Solve the Flexible Job Shop Scheduling Problem. CACIC 2019. ISBN: 978-987-688-377-1, pág 2-11, October 2019.

1.4. Metodología

A continuación se detalla la metodología que se adopta al realizar este trabajo, detallando las fases en las que se divide.

La primera fase consiste en la revisión de material bibliográfico relacionado con algoritmos evolutivos, haciendo especial énfasis en algoritmos de evolución diferencial, a fin de adquirir conocimientos necesarios sobre estas técnicas para, en etapas posteriores, poder concretar una propuesta algorítmica. Además, en esta etapa de revisión se estudia el estado del arte en la problemática abordada FJSSP mediante la solución con DE y sus variantes híbridas. Esta fase permite identificar particularidades de las distintas alternativas de solución que resulten más prometedoras para resolver el FJSSP, y que se puedan integrar en una propuesta posterior.

En una segunda fase, es necesario el análisis y estudio de posibles paralelizaciones de metaheurísticas, con un enfoque centrado en paralelizar algoritmos de evolución diferencial, con el fin de poder lograr un algoritmo híbrido paralelo con grandes ventajas en tiempo y utilización de recursos computacionales.

En una tercera fase, se continua con el diseño y la implementación del DE híbrido

paralelo para resolver el FJSSP siguiendo el paradigma procedural, haciendo hincapié en la representación del problema.

Concretada la fase anterior, la siguiente consiste en la puesta a punto del algoritmo desarrollado. Para lo cual se deben realizar ejecuciones preliminares y analizar los resultados de las mismas para determinar la mejor configuración de parámetros. Una vez definidos los mismos, se procede al desarrollo de la experimentación completa para finalizar con el análisis de los resultados obtenidos y su estudio estadístico.

En la fase final se realiza la escritura del informe del proyecto final incluyendo el desarrollo realizado y el análisis de los resultados obtenidos.

1.5. Cronograma de trabajo

A continuación se detallan las distintas actividades, y su planificación en el tiempo, que surgen para la concreción de los objetivos parciales.

1. Estudiar y analizar el estado del arte en relación a algoritmo DE.
2. Estudiar y analizar el estado del arte en relación a posibles paralelizaciones de metaheurísticas.
3. Diseñar la forma de paralelizar un algoritmo DE para resolver el FJSSP e implementar el algoritmo propuesto, evaluando alternativas de mejora.
4. Realizar la experimentación y posterior análisis de resultados.
5. Documentar el trabajo realizado.

Las duraciones estimadas (en horas) de las distintas actividades se muestran en la siguiente tabla:

Actividad	**1**	**2**	**3**	**4**	**5**
Duración	30	20	50	60	40

Tiempo total: 200 horas.

1.6. Organización de la tesis

Los capítulos de esta tesis se organizan de la siguiente manera.

El Capítulo 2 contiene los fundamentos teóricos relacionados con metaheurísticas que ayudarán a la comprensión de la presente tesis. El mismo describe brevemente los algoritmos metaheurísticos, la clasificación de las metaheurísticas, tanto en métodos basados en trayectoria como en aquellos basados en poblaciones, y algunas estrategias para metaheurísticas paralelas.

El Capítulo 3 introduce al lector en los conceptos de la evolución diferencial, y presenta el algoritmo propuesto por Storn & Price, junto a sus 4 etapas: inicialización, mutación, recombinación y selección. Además se muestra el proceso iterativo del algoritmo *DE* en un formato de pseudocódigo.

El Capítulo 4 expone la propuesta del algoritmo *DE* híbrido para *FJSSP*. Se presenta una descripción del problema a tratar, el diseño de la representación y evaluación de soluciones utilizado, la decodificación de soluciones, el procedimiento de búsqueda local utilizado, la forma en que fue realizada la paralelización de *DE* y la búsqueda local, y por último el pseudocódigo del algoritmo de *DE* híbrido para *FJSSP*.

El Capítulo 5 exhibe los estudios experimentales llevados a cabo en este trabajo. Se describe el diseño experimental y los resultados experimentales obtenidos.

Por último, el Capítulo 6 presenta las principales conclusiones obtenidas de este trabajo.

Capítulo 2

Metaheurísticas

2.1. Introducción

El término *optimización* es muy amplio y se utiliza en una gran variedad de situaciones. Por ejemplo, un científico o ingeniero utiliza la optimización cuando pretende mejorar una idea, haciendo cambios, junto con el conocimiento obtenido en cada cambio [1]. En forma más especifica, existe un gran número de problemas de optimización en diferentes áreas de interés [2, 18].

Según García-Nieto y Alba en [19], todos los problemas de optimización comparten algunas características comunes. Al atacar un problema de optimización, es fundamental analizar las alternativas posibles a las distintas variables de decisión del problema a tratar y las restricciones factibles sobre las mismas. Además, es de suma importancia elegir y estudiar solo aquellas variables y restricciones que se consideren más adecuadas. Otro punto a tener en cuenta es seleccionar una medida de calidad para poder evaluar las posibles soluciones al problema.

Desde el punto de interés para esta trabajo final y en la inteligencia artificial en general, un problema dado es un *problema de optimización* cuando hay varias *soluciones candidatas* y una manera efectiva de medir la calidad de cada una de las soluciones. La *optimización* es el proceso llevado a cabo para tratar de encontrar la mejor solución candidata al problema, aprovechando al máximo un conjunto limitado de recursos (tiempo, espacio, etc.) [20].

Se llega a la solución cuando se encuentra un conjunto de variables de decisión y/o los valores adecuados para esas variables de decisión. Los valores se deben buscar dentro del dominio de las variables de decisión, siendo los posibles dominios *discretos*

o *continuos*. En el primero de los casos, la variable únicamente puede tomar aquellos valores que pertenecen a un conjunto finito. En el segundo, la variable puede tomar un valor fijo dentro de un intervalo determinado perteneciente al conjunto de los números reales. Un problema puede tener sólo variables continuas o sólo variables discretas o una combinación de ambos tipos de variables.

Los problemas de optimización son difíciles y complejos de resolver, por la definición del mismo, por el número de variables involucradas, por el dominio al que pertenecen las variables, o por una combinación de estas y muchas otras complejidades. Debido a que la dificultades y el costo computacional pueden ser tan elevados, muchas veces se considera suficiente con alcanzar una solución aproximada a la mejor solución conocida. Por esto, se utilizan algoritmos aproximados en vez de algoritmos exactos.

En este capítulo se presenta una introducción a los algoritmos metaheurísticos, luego se presenta una de las posibles calificaciones de las metaheurísticas: en métodos basados en trayectoria y en métodos basados en poblaciones, y por último se explican distintos conceptos de diseños para paralelizar las metaheurísticas.

2.2. Algoritmos metaheurísticos

Una gran cantidad de problemas de optimización son NP-Duros, [5] y llegar a una solución no es una tarea fácil. En muchas situaciones, una solución que aproxima a la óptima puede ser suficiente, si logra evitar perder tiempo, dinero o esfuerzo. Por esto, los algoritmos aproximados han tenido un especial interés en las últimas décadas. Estos se pueden dividir en, al menos, dos grandes grupos, los *heurísticos* y los *metaheurísticos*.

Los algoritmos heurísticos buscan dentro del espacio de soluciones e intentan encontrar una solución suficientemente buena, aunque no sea la óptima [21]. Claramente, en estos algoritmos se prefiere el bajo costo y esfuerzo computacional, frente a calidad y precisión de la solución. Usando una heurística, se eligen iterativamente soluciones localmente óptimas hasta lograr aproximarse lo mayor posible a la solución óptima global.

Las metaheurísticas aparecen en la década de los ochenta, presentando una nueva clase de algoritmos estocásticos y aproximados de alto nivel, capaces de hacer frente a problemas de optimización complejos [6]. Se utiliza un conjunto de operadores con un

diseño abstracto y de alto nivel, lo que evita especificar cuestiones correspondientes al problema a resolver. Si bien no se garantiza llegar a una solución óptima, logran alcanzar una solución cercana a la óptima. Estos algoritmos tienen la ventaja de obtener esa solución utilizando un mínimo de recursos computacionales [22].

El éxito de las metaheurísticas depende en gran parte de realizar un correcto balance entre *diversificación* e *intensificación*. El primer término, refiere a la capacidad de la técnica de explorar el espacio de soluciones. El segundo, es la habilidad para utilizar la experiencia acumulada en la búsqueda. Estos términos se conocen también como *exploración* y *explotación*, respectivamente. Es de suma importancia lograr el balance entre diversificación e intensificación para distinguir zonas del espacio donde hay soluciones de alta calidad, despreciando aquellas donde las soluciones son de baja calidad.

2.3. Clasificación de las metaheurísticas

Existen distintas estrategias para desarrollar un algoritmo metaheurístico, por lo que resulta de gran dificultad clasificar o agrupar los algoritmos por sus características. Podríamos considerar criterios para clasificar como la fuente en la cuál esta inspirada, el uso de una memoria histórica, el número de funciones objetivo que pueden optimizar simultáneamente, por la manera de generar las soluciones, entre otras.

Un criterio de clasificación aceptado es la cantidad de soluciones que las metaheurísticas pueden manipular en forma simultanea. De aquí surgen dos grandes grupos, por un lado los algoritmos basados en una única solución, llamados *métodos basados en trayectorias*, y por el otro los *métodos de búsqueda basados en poblaciones*.

Dentro del grupo de los métodos basados en trayectorias se encuentran modificaciones a algoritmos de búsqueda local, que pretenden realizar una exploración inteligente, simple y eficiente de las estructuras de vecindarios. Entre las técnicas más destacadas, podemos mencionar Recocido Simulado [23], Búsqueda Local Iterada [24], Búsqueda por Vecindario Variable [25].

En los métodos de búsqueda poblacionales se utiliza un conjunto de soluciones con el fin de explorar al mismo tiempo varias regiones del espacio de soluciones. Además, estos algoritmos emplean artilugios de aprendizaje explícito o implícito, con el objetivo de detectar correlaciones entre variables. Estas características son fundamentales para lograr descubrir rápida y eficientemente zonas del espacio con soluciones de alta calidad. Dentro de estos tipos de algoritmos encontramos a los Algoritmos

Evolutivos [26, 52], la Evolución Diferencial [27], los Algoritmos de Inteligencia Colectiva o de Enjambres [32, 34], Sistema Inmune Artificial [35].

En las siguientes sub-secciones se analizan de forma breve algunos de los algoritmos nombrados para cada uno de estos grupos representativos.

2.3.1. Métodos basados en trayectoria

Este tipo de algoritmos metaheurísticos inspecciona el espacio de soluciones marcando trayectorias sobre el mismo. La mayoría añade a la búsqueda local algún mecanismo para escapar de óptimos locales durante el proceso de exploración. Comúnmente, utilizan solo una solución, junto con un mecanismo que altera la trayectoria cuando detecta un óptimo local. Estos algoritmos son eficaces para hacer frente a diversos problemas de optimización en diferentes dominios.

Los métodos basados en trayectoria aplican iterativamente los procedimientos de generación y reemplazo de la única solución actual. En la etapa de generación, se crea un conjunto de soluciones $C(s)$ a partir de la solución actual s. En la etapa de reemplazo, se realiza un proceso de selección donde se elige una nueva solución del conjunto $C(s)$. Este proceso itera hasta que se cumple un criterio de finalización. En el algoritmo 1 se ilustra una plantilla genérica de alto nivel para este tipo de metaheurísticas.

Algoritmo 1 Plantilla de alto nivel para el método de trayectoria

Entrada: Solución inicial s_0

Salida: Mejor solución encontrada

1: $t = 0$
2: **repetir**
3: /* Generar soluciones candidatas a partir de s_t */
4: Generar ($C(s_t)$)
5: /* Seleccionar una solución de $C(s)$ que reemplace a la actual s_t */
6: s_{t+1} = Seleccionar($C(s_t)$)
7: $t = t + 1$
8: **hasta que** Se cumpla un criterio de finalización

Búsqueda local

La Búsqueda local es probablemente el método metaheurístico más antiguo y simple. Comienza con una solución inicial dada. En cada iteración, la heurística rem-

plaza la solución actual por otra que mejora el resultado de la función objetivo. La búsqueda finaliza cuando todos los posibles candidatos son peores que la solución actual, lo que significa que se alcanzó un óptimo local.

En general, la búsqueda local es un método muy fácil de diseñar e implementar y ofrece soluciones bastante buenas rápidamente. Es por esto que es un método de optimización ampliamente utilizado en la práctica. Una de las principales desventajas es que converge hacia óptimos locales; otra que el algoritmo puede ser muy sensible a la solución inicial.

La búsqueda local funciona bien si no hay demasiados óptimos locales en el espacio de búsqueda o si la calidad de los diferentes óptimos locales es más o menos similar. Si la función objetivo es altamente multimodal, como es el caso de la mayoría de los problemas de optimización, la búsqueda local generalmente no es un método efectivo para usar.

Recocido Simulado

La técnica de Recocido Simulado (*Simulated Annealing o SA* en inglés) es un método de búsqueda local con un mecanismo de escape de un mínimo local. Este algoritmo presentado por Kirkpatrick en [36] y Cerny en [37] se basa en los principios de la mecánica estadística, por lo que el proceso de recocido requiere calentar y luego enfriar lentamente una sustancia para obtener una estructura cristalina fuerte. La resistencia de la estructura depende de la velocidad de enfriamiento de los metales. Si la temperatura inicial no es lo suficientemente alta o se aplica un enfriamiento rápido, se obtienen imperfecciones (estados metaestables). En este caso, el sólido en enfriamiento no alcanzará el equilibrio térmico en cada temperatura. Cristales fuertes surgen de un enfriamiento cuidadoso y lento. El algoritmo SA simula los cambios de energía en un sistema sometido a un proceso de enfriamiento hasta que converge a un estado de equilibrio (estado de congelamiento estable). Este esquema fue desarrollado en 1953 por Metropolis [38].

SA se aplica tanto a problemas discretos como continuos. El algoritmo itera comparando la solución actual con una nueva. La técnica tiende a mantener la solución con alta calidad. No obstante, existe un parámetro de *Temperatura* que determina en qué grado se aceptan algunas soluciones de menor calidad con el fin de escapar de un óptimo local.

Búsqueda por Vecindario Variable

La Búsqueda por Vecindario Variable (*Variable Neighborhood Search o VNS* en inglés) fue propuesta por Pierre [39]. La idea básica de VNS es explorar sucesivamente un conjunto de vecindarios predefinidos para brindar una mejor solución. Explora de forma aleatoria o sistemática un conjunto de vecindarios para obtener diferentes óptimos locales y, a su vez, escapar de los óptimos locales. VNS explota el hecho de que el uso de varios vecindarios en la búsqueda local puede generar diferentes óptimos locales y que el óptimo global es un óptimo local para un vecindario determinado. Claramente, es de suma importancia tener cuidado al elegir el conjunto de vecindarios y el método de búsqueda local a utilizar para evitar una convergencia prematura.

Búsqueda Local Iterada

La técnica de Búsqueda Local Iterada *(Iterated Local Search o ILS* en inglés) fue presentada por Lourenço [24]. La esencia de la misma es construir iterativamente una secuencia de soluciones generadas por la heurística incorporada, lo que nos lleva a obtener soluciones mucho mejores que si solo se usaran repetidas pruebas aleatorias de esa heurística.

El lineas generales, ILS primero aplica una búsqueda local sobre una solución inicial. Luego, en cada iteración, el algoritmo explora un área reducida del espacio de soluciones, perturbando el óptimo local obtenido y obteniendo una solución intermedia. Al no utilizar una estructura de vecindario para realizar la perturbación, la solución intermedia puede encontrarse en cualquier zona del espacio de soluciones. Finalmente, aplica una búsqueda local a la solución intermedia. La solución generada es aceptada como la nueva solución actual cuando sea de mayor calidad. Este proceso itera hasta que se cumple algún criterio determinado de parada. El mecanismo de perturbación es importante para que el algoritmo tenga éxito. Una perturbación demasiado pequeña no será suficiente para alcanzar un óptimo global, mientras que un exceso en la misma puede provocar variadas búsquedas aleatorias.

2.3.2. Métodos basados en población

Los métodos de búsqueda basados en población, a diferencia de las técnicas por trayectoria, realizan la búsqueda en paralelo sobre varias regiones del espacio de soluciones, es decir, utilizando un conjunto de soluciones, conocido como *población*. Estos

métodos iteran inspeccionando varias zonas del espacio de soluciones. La alteración de la población es fundamental para llegar a soluciones cercanas a la óptima.

Este tipo de metaheurística comienza con una población inicial de soluciones. En la etapa de generación, se crea una nueva población de soluciones. En la etapa de reemplazo, se realiza un proceso de selección entre la población actual y la nueva. Este procedimiento itera hasta que se llega a un determinado criterio de parada. La generación y las fases de reemplazo pueden ser sin memoria, por lo que en ese caso esas etapas solo se basan en la población actual. Así también, se puede usar una memoria para realizar las dos fases nombradas con anterioridad. La mayoría de las metaheurísticas poblacionales son inspiradas en la naturaleza. En el algoritmo 2 se puede apreciar una plantilla de alto nivel para este tipo de métodos [62].

Algoritmo 2 Plantilla de alto nivel para el método basado en población

Salida: Mejor solución encontrada

1: /* Generación de la población inicial */
2: Iniciar (P_t)
3: $t = 0$
4: **repetir**
5:     /* Generación de la nueva población */
6:     Generar (P'_t)
7:     /* Seleccionar la nueva población */
8:     P_{t+1} = Seleccionar-población($P_t \cup P'_t$)
9:     $t = t + 1$
10: **hasta que** Se cumpla un criterio de finalización

Las metaheurísticas basadas en poblaciones difieren en la forma en que realizan los procedimientos de generación y selección, como así también de la memoria que utilizan durante el proceso de búsqueda.

Algoritmos evolutivos

Los distintos algoritmos evolutivos se basan en la naturaleza como forma de inspiración para lidiar con problemas difíciles de optimización. En realidad, el proceso darwiniano de evolución natural es un problema de optimización. Particularmente, las especies que habitan nuestro planeta presentan una estructura que es resultado de

un proceso de adaptación de miles de años. La adaptación es la forma que tienen las especies para volverse óptimas en su entorno.

Diferentes escuelas de algoritmos evolutivos han progresado en forma independiente a partir de 1950:

- Algoritmos Genéticos (*Genetic Algorithms o GA* en inglés), desarrollados principalmente en Michigan, EE. UU., por J. H. Holland [42, 43].

- Estrategias Evolutivas (*Evolution Strategies* en inglés), desarrolladas en Berlin, Alemania, por I. Rechenberg [45, 46] y H-P. Schwefel [47, 48].

- Programación Evolutiva (*Evolutionary Programming* en inglés) por L. Fogel en San Diego, EE. UU. [49, 50].

Luego, al final de 1980, J. Koza [51] propone la Programación Genética (*Genetic Programming* en inglés). Cada uno de estos métodos constituye un enfoque diferente, pero todos están inspirados en los mismos principios de la evolución natural. Además, establecen los principios básicos para los distintos algoritmos evolutivos construidos y/o perfeccionados en los últimos años que persiguen el fin de resolver con una mayor precisión y eficacia los problemas de optimización complejos.

Los algoritmos evolutivos se han aplicado con éxito a muchos problemas reales y complejos. Son los algoritmos basados en población mas estudiados. Su éxito en resolver problemas difíciles de optimización en varios dominios promovió un campo de estudio conocido como Computación Evolutiva (*Evolutionary Computation o EC* en inglés) [52].

Este tipo de algoritmos son métodos iterativos que en el proceso utilizan el valor *objetivo* (mejor conocido como *fitness* en inglés) con el fin de realizar una búsqueda inteligente sobre el espacio de soluciones. Dada la gran relación que presentan este tipo de técnicas con los procesos biológicos, se denomina a las soluciones del problema como *individuos* y a cada iteración *generación*.

En cada generación se aplica sobre los individuos de la población P un conjunto de operadores con el fin de obtener nuevos individuos, los cuales reemplazarán a los de la población actual en la generación siguiente. Se repite este proceso hasta alcanzar un criterio de finalización. Los individuos que formarán parte de la población inicial son generados aleatoriamente desde el espacio de soluciones. Luego se aplica sobre

ellos un procedimiento de reproducción, utilizando usualmente la recombinación y la mutación, para luego seleccionar las nuevas soluciones.

El operador de *recombinación* utiliza un conjunto de individuos de la población actual para producir nuevos individuos. A esos nuevos individuos se le puede aplicar un procedimiento de *mutación* con el fin de provocarles cambios adaptativos. La *selección* es la encargada de elegir individuos teniendo en cuenta el valor obtenido por la función objetivo. La función objetivo $f(\cdot)$ es la encargada de evaluar un individuo x y determinar su calidad para resolver el problema. Se aplican dos procesos de selección, uno para escoger los individuos que serán padres en el proceso de recombinación y otro para elegir los individuos qué reemplazaran a los de la población actual en la próxima generación. El algoritmo 3 es un esquema genérico de este proceso.

Algoritmo 3 Plantilla de un Algoritmo Evolutivo

Salida: Mejor individuo o mejor población encontrada

1: /* Generación de la población inicial */
2: Generar ($P(0)$)
3: $t = 0$
4: **mientras** no se cumpla un criterio de finalización **hacer**
5:     /* Generación de la nueva población */
6:     Evaluar ($P(t)$)
7:     $P'(t)$ = Seleccionar ($P(t)$)
8:     $P'(t)$ = Reproducción ($P'(t)$)
9:     $P'(t)$ = Evaluar ($P'(t)$)
10:     $P(t+1)$ = Reemplazar ($P(t)$,$P'(t)$)
11:     $t = t + 1$
12: **fin mientras**

Los tres operadores mencionados son aplicados en la mayor parte de los algoritmos evolutivos. Sin embargo, la flexibilidad que presenta esta técnica permite incorporar otros mecanismos, como por ejemplo una búsqueda local, con el fin de abarcar todo el espacio de soluciones y lograr analizar las mejores regiones del espacio.

Existen diferentes subtipos de algoritmos evolutivos, que siguen el esquema general, pero con algunas modificaciones. Entre ellas, la Evolución Diferencial (DE) tiene una estructura algorítmica muy simple, y ha demostrado un muy buen nivel de desempeño al resolver una amplia variedad de problemas muy complejos. En el capítulo 3 se

explicará con mayor detalle y en profundidad este algoritmo.

Algoritmos de inteligencia colectiva

Existen algoritmos poblaciones inspirados en la naturaleza que utilizan un esquema de inteligencia colectiva al realizar la búsqueda en el espacio de soluciones. Cada elemento de la población en este método es un *agente* sin capacidad propia de ser inteligente. Sin embargo, al colaborar e interactuar con los distintos agentes de la población, se logra una búsqueda inteligente sobre el espacio.

Puede encontrar una gran variedad de algoritmos que utilizan la inteligencia colectiva. Un ejemplo es la Optimización basada en Colonia de Hormigas (*Ant Colony Optimization o ACO* en inglés) presentada por Dorigo [53, 55]. La técnica fue realizada inspirándose en la forma en que las colonias de hormigas buscan alimento. A medida que siguen el camino hacia su alimento, los agentes (en este caso las hormigas) dejan rastros de feromonas (información numérica) que es de utilidad para los demás agentes de la colonia para ubicar la fuente de alimentos.

Otro claro ejemplo de este tipo de algoritmos es la Optimización basada en Colonia de Abejas Artificiales (*Artificial Bee Colony o ABC* en inglés) presentada por Karaboga [56, 60] . Como en ACO, los agentes simulan el comportamiento de las abejas al buscar alimentos. En este caso, las abejas son quienes se encargan de explorar el espacio, tomando como calidad de la solución a la calidad del néctar.

Sistema inmune artificial

Aquí la biología jugo un rol como fuente de inspiración de la técnica de Sistema Inmune Artificial (*Artificial Immune System o AIS* en inglés), desarrollada pensando en simular el sistema inmune humano [61]. El método imita a través de un modelo matemático el comportamiento y las propiedades inmunológicas de las células. En esta técnica, algunos algoritmos utilizan modelos de inmunología simples, mientras que otros aplican conceptos muy finos y con fundamentos interdisciplinarios complejos.

2.4. Metaheurísticas paralelas

Por un lado, los problemas de optimización son cada vez mas complejos y sus necesidades de recursos son cada vez mayores. En la vida real, los problemas de opti-

mización son NP-Duros y requieren un alto tiempo de CPU y un gran uso de memoria para resolverlos. Aunque el uso de metaheurísticas reduce significativamente la complejidad computacional del proceso de búsqueda, continúa demandando mucho tiempo computacional en diversos problemas de distintos dominios de aplicación, donde la función objetivo y las restricciones asociadas al problema hacen un uso intensivo de recursos, y el tamaño del espacio de búsqueda es enorme. Además, cada vez se desarrollan metaheurísticas más complejas que requieren un mayor uso de recursos.

Por otro lado, el rápido desarrollo tecnológico en el diseño de procesadores, redes y el almacenamiento de datos ha hecho que el uso de la computación en paralelo sea cada vez más popular. Estas arquitecturas representan una estrategia efectiva para el diseño e implementación de metaheurísticas paralelas.

La computación paralela y distribuida se puede usar en el diseño e implementación de metaheurísticas por las siguientes razones:

- **Acelerar la búsqueda:** uno de los objetivos principales de aplicar paralelismo en una metaheurística es reducir el tiempo de búsqueda.

- **Mejorar la calidad de las soluciones:** algunos modelos metaheurísticos paralelos permiten mejorar la calidad de la búsqueda.

- **Mejorar la robustez:** una metaheurística paralela puede ser más robusta para resolver diferentes problemas de optimización y diferentes instancias de un problema dado. La robustez también esta dada por la sensibilidad de la metaheurística a sus parámetros.

- **Resolver problemas de gran escala:** las metaheurísticas paralelas permiten resolver instancias de gran escala de problemas complejos de optimización.

En términos de diseño de metaheurísticas paralelas, se identifican tres modelos paralelos principales:

- **Nivel algorítmico:** en este modelo, se utilizan metaheurísticas independientes o cooperativas autocontenidas. La paralelización del algoritmo es independiente del problema. Si las diferentes metaheurísticas fueran independientes, la búsqueda sobre el espacio de soluciones sería equivalente a ejecutar secuencialmente cada una de las metaheurísticas, en término de la calidad de las soluciones obtenidas. Sin embargo, el modelo cooperativo altera el comportamiento de las metaheurísticas y permite mejorar la calidad de las soluciones.

- **Nivel de iteración:** en este modelo, cada iteración de la metaheurística se paraleliza. Es independiente del problema y el comportamiento de la metaheurística no se altera. El objetivo principal que persigue este método es acelerar el algoritmo, reduciendo el tiempo de búsqueda.

  - **Nivel de iteración para metaheurísticas basadas en trayectoria:** en este modelo, el espacio de soluciones se descompone en diferentes particiones que generalmente tienen el mismo tamaño. Las particiones se generan y luego se evalúan de manera paralela e independiente.
  - **Nivel de iteración para metaheurísticas basadas en población:** al tratar con metaheurísticas basadas en población los modelos paralelos surgen de forma natural, ya que cada elemento que pertenece a la población (por ejemplo, abejas en ABC, individuos en EA, hormigas en ACO) es una unidad independiente. Las operaciones comúnmente aplicadas a cada uno de los elementos de la población se realizan en paralelo.

    En los algoritmos evolutivos, la población de individuos se puede descomponer y manejar en paralelo. Un método para realizar la paralelización es la de maestro-trabajador. El maestro realiza las operaciones de selección y de reemplazo que son generalmente procedimientos secuenciales pues requieren una gestión global de la población. Los trabajadores realizan la recombinación, la mutación y la evaluación. El maestro envía las particiones (subpoblaciones) a los trabajadores y estos le devuelven al maestro las nuevas soluciones y su valor de *fitness*.

- **Nivel de solución:** en este modelo, el proceso de paralelización maneja una única solución del espacio de búsqueda. Es dependiente del problema. En general, la evaluación de la(s) función(es) objetivo(s) o de las restricciones para una solución es la operación más costosa en una metaheurística. En este método, no se altera el comportamiento de la metaheurística. El objetivo principal es acelerar la búsqueda.

Capítulo 3

Evolución Diferencial

3.1. Introducción

La *Evolución Diferencial* (*DE*) es considerada un subtipo de algoritmo evolutivo. A pesar de tener una estructura algorítmica muy simple, ha demostrado un nivel elevado de desempeño a la hora de resolver una amplia variedad de problemas muy complejos [27]. Como toda metaheurística poblacional, *DE* realiza una continua transformación de las distintas soluciones de la población. Es eficiente y eficaz, pues utiliza operaciones de bajo costo computaciones (usualmente lineal) para realizar la transformación de las soluciones. Son estas operaciones quienes se encargan de guiar la búsqueda a las regiones más prometedoras del espacio de soluciones.

DE fue propuesto inicialmente por Storn y Price en un reporte técnico en 1995 [28] y en 1996 demostró su eficacia en una sesión especial de un congreso especializado en computación evolutiva [31]. En el mismo, se aplicó la técnica con un gran éxito al resolver problemas de optimización en espacios continuos, superando a otros algoritmos evolutivos. En la actualidad, se han desarrollado distintas modificaciones para mejorar sus resultados.

El éxito de *DE* no es solo por su simplicidad, sino también por su capacidad para resolver problemas de optimización. La simplicidad está en su estructura algorítmica, pues solo se requieren unas pocas líneas de código para implementar el método, teniendo así el investigador la capacidad de invertir su tiempo y esfuerzo en entender el problema y no en la técnica que le da solución. Otra de las grandes ventajas, es la muy baja complejidad espacial que presenta respecto a otros algoritmos de optimización. Esto otorga la ventaja de poder obtener un algoritmo escalable a soluciones de casi

cualquier número de variables, teniendo como limitante solo a la capacidad de la memoria de la computadora utilizada en la experimentación. Otro punto importante que vale la pena destacar es su capacidad para resolver correctamente una gran variedad de problemas complejos tanto discretos como continuos, con uno o varios óptimos, separables o no separables, etc.

La robustez y la habilidad para converger a una solución óptima (o cercana a la óptima) es una característica importante que tiene *DE*. Además, solo se requiere configurar un pequeño número de parámetros con respecto a otros algoritmos evolutivos. En *DE*, el investigador solo necesita manipular tres parámetros para controlar el desempeño del algoritmo. Ahora bien, tener éxito o fracasar en la búsqueda de una solución a un problema depende, en gran parte, de identificar los valores correctos para esos parámetros. Por tal motivo, en algunos artículos se evita la ardua tarea de seleccionar *a priori* los valores, aplicando un ajuste dinámico de los mismos [40, 41].

3.2. Algoritmo *DE*: propuesta de Storn & Price

La propuesta original de *DE* de Storn y Price [27] fue pensada para resolver, específicamente, problemas de optimización que tengan su dominio en el conjunto de los reales. La solución óptima (o una cercana) se obtiene aplicando, sobre una población, un proceso iterativo donde se genera una nueva población de soluciones. En cada paso de este proceso, las nuevas soluciones surgen a través de las perturbaciones causadas por los operadores de mutación y recombinación sobre las soluciones actuales.

En términos de *DE*, una solución es un *vector* de *D* valores reales y cada componente representa una variable o parámetro del problema. Por lo tanto, los vectores son puntos en el espacio de *D* dimensiones de números reales (R^D). *DE* debe encontrar la mejor solución realizando una búsqueda dentro de ese espacio.

DE obtiene la solución óptima (o una cercana a la misma) luego de finalizar un proceso de dos etapas. En la primera etapa, se genera una población de vectores. En la segunda etapa, *DE* itera hasta alcanzar un punto de corte (p. ej., el número máximo de generaciones, max_{gen}) realizando tres fases básicas que componen el proceso evolutivo: mutación, recombinación y selección.

DE depende de tres parámetros de control definidos por el investigador. Dos de los parámetros, F y Cr, son valores reales positivos. El tercer parámetro, N_P , es un valor entero positivo ($N_P > 0$).

A continuación se pasan a explicar con mayor detalle cada una de las operaciones en las distintas etapas mencionadas.

3.2.1. Inicialización

La población inicial P tiene N_P vectores $x_i = (x_{i,1}, x_{i,2}, ..., x_{i,D}) \in R^D (1 \leq i \leq N_P)$ distribuidos sobre el espacio de soluciones de manera aleatoria y uniforme. Cada componente $x_{i,j} \in R (1 \leq j \leq D)$ es una variable de decisión del problema a optimizar. Es común que, por las características propias del problema, el espacio de soluciones pocas veces sea exactamente R^D. Con frecuencia cada variable de decisión $x_{i,j}$ está acotada a un valor en el intervalo $[li_j, ls_j]$, donde li_j, $ls_j \in R$ son el límite inferior y superior, respectivamente.

El algoritmo distribuye inicialmente los vectores en el espacio del problema, asignando un valor a cada variable. El valor de la variable $x_{i,j}$ se obtiene, generalmente, aplicando la siguiente ecuación:

$$x_{i,j} = li_j + U(0,1) \times (ls_j - li_j) \tag{3.1}$$

donde $U(0,1)$ es un número aleatorio con distribución uniforme en el intervalo $[0,1]$. Con el fin de evitar cualquier perjuicio en el desempeño del algoritmo, $U(0,1)$ debe ser independiente de cualquier otro número aleatorio generado o por generar.

3.2.2. Mutación

En un contexto biológico, la mutación es un proceso súbito y espontáneo de variación en la información genética de un organismo vivo. En la computación evolutiva se apropió este término para definir un operador que cambia o perturba un elemento aleatorio del cromosoma. Por su parte, *DE* obtiene un *vector mutante*, también conocido como *vector donador*, aplicando el operador de *mutación diferencial*. Su uso es fundamental para el desempeño de *DE*, debido a que centra la búsqueda en las zonas más prometedoras del espacio de soluciones.

Uno de los esquemas más simples para implementar un operador de mutación diferencial es mostrado en la siguiente ecuación:

$$v_i^g = x_{r0}^g + F \times (x_{r1}^g - x_{r2}^g) \tag{3.2}$$

donde por cada *vector objetivo* $x_i^g(1 \leq i \leq N_P)$ de la población actual $P^g(0 \leq g \leq max_{gen}$ y P^0 es la población inicial), el operador crea un vector donador v_i^g. Para obtener el vector, primero, se elige aleatoriamente, de la población actual, un *vector base* x_{r0}^g diferente del vector objetivo. Luego se seleccionan de la población, de manera también aleatoria, otros dos vectores (x_{r1}^g y x_{r2}^g) diferentes entre ellos y diferentes a los dos anteriores (x_i^g y x_{r0}^g), y son utilizados para calcular el vector diferencia. Finalmente, el vector donador es obtenido sumando el vector diferencia escalado en un factor F al vector base.

El factor de escalado $F \in [0...1+)$ controla la amplificación de la diferencia entre los vectores x_{r1}^g y x_{r2}^g usados para la exploración del espacio, y así evitar que se produzca un estancamiento en el proceso de búsqueda. Comúnmente, F no es mayor que 1, aunque por definición no existe un límite superior para el parámetro. Al definir F, el algoritmo cambia automáticamente el tamaño del movimiento que van a realizar los vectores sobre el espacio. A lo largo de las generaciones, el vector diferencia adapta la búsqueda al paisaje (*landscape* en inglés) de la función objetivo. Claramente, existe una gran cantidad de posibles vectores diferencia cuando los valores de las variables están muy dispersos, pues si las variables están convergiendo, la cantidad de posibles vectores diferencia pasa a ser muy chica.

3.2.3. Recombinación

Luego de aplicar el operador de mutación, el vector donador es modificado a través del operador de recombinación, con el objetivo de incrementar la diversidad en la población. El operador intercambia componentes del vector donador v_i^g con el vector objetivo x_i^g. Al terminar con el proceso, se obtiene el *vector de prueba* u_i^g.

En la recombinación binomial, se analiza cada componente del vector objetivo en forma independiente para decidir si se realiza el intercambio por el del vector donador. La siguiente ecuación muestra cómo se obtienen las componentes del vector prueba:

$$u_{i,j}^g = \begin{cases} v_{i,j}^g & \text{si } r_j < Cr \vee j = j_r \\ x_{i,j}^g & \text{en otro caso} \end{cases} \tag{3.3}$$

donde $r_j = U(0,1)$ es un valor aleatorio uniformemente distribuido dentro del intervalo [0,1]. j_r es también un valor aleatorio pero uniformemente distribuido en el conjunto {1, ... ,*D*} y, finalmente, *Cr* es el parámetro de la probabilidad de recombinación de *DE*.

3.2.4. Selección

El tercer y último paso en el proceso evolutivo de *DE* es la operación de *selección*. Utilizando la función objetivo $f(\cdot)$, se compara el vector de prueba $u_{i,j}^{g}$ y el vector objetivo $x_{i,j}^{g}$. El mejor de ambos vectores es elegido para formar parte de la población de la próxima generación y el otro vector es descartado. La siguiente ecuación muestra cómo se realiza la operación:

$$x_i^{g+1} = \begin{cases} u_i^g & \text{si } f(u_i^g) \textit{ es igual o mejor que } f(x_i^g) \\ x_{i,j}^g & \text{en otro caso} \end{cases} \tag{3.4}$$

Cuando, por ejemplo, se está minimizando la función objetivo, el vector prueba integrará la nueva población si $f(u_i^g)$ es menor o igual a $f(x_i^g)$. En caso de que sean iguales, el operador se ve obligado a seleccionar el vector prueba con el fin de moverse en regiones planas del paisaje de la función objetivo.

El esquema de selección en *DE* compara al vector prueba y al objetivo que tienen el mismo índice y retiene el mejor de ambos. Esta competencia entre padres e hijos crea una nueva población con el mismo tamaño que la anterior y con un desempeño igual o superior a la actual. Como no existe un deterioro en la calidad de las soluciones, *DE* es un algoritmo evolutivo *elitista*. Esta propiedad es importante para alcanzar una convergencia al óptimo global [57].

3.3. Descripción de *DE*

El Algoritmo 4 muestra el proceso iterativo de *DE* en un formato de pseudocódigo. Cada una de las operaciones mencionadas en las secciones anteriores están aplicadas en el algoritmo: *inicialización* en la línea 1, *mutación* en línea 5, *recombinación* en línea 6 y *selección* en línea 7. El criterio de finalización de *DE* es libremente elegido por el investigador de acuerdo al problema a resolver. Podría ser un simple número de iteraciones (max_{gen}) especificado por el investigador o hasta alcanzar un número determinado de evaluaciones de la función objetivo o hasta lograr un valor objetivo específico. Cualquiera sea el criterio, su elección tiene una influencia directa sobre la mejor solución x_{best} obtenida por el algoritmo (línea 12). La función *agregar* (línea 8) incorpora un nuevo miembro en la población de la próxima generación.

Algoritmo 4 Pseudocódigo del algoritmo Evolución Diferencial (DE)

Entrada: F, Cr, N_p

Salida: x_{best}

1: inicializar(P,N_p)
2: $g \leftarrow 0$
3: **mientras** no se alcance la condición de finalización **hacer**
4:     **para** cada vector x_i^g de P^g **hacer**
5:         $v_i^g \leftarrow$ mutación(x_i^g, P^g, F)
6:         $u_i^g \leftarrow$ recombinación(x_i^g, v_i^g, Cr)
7:         $x_i^{g+1} \leftarrow$ selección(x_i^g, u_i^g)
8:         agregar(P^{g+1}, x_i^{g+1})
9:     **fin para**
10:     $g \leftarrow g + 1$
11: **fin mientras**
12: $x_{best} \leftarrow$mejor-solución(P^g)

Capítulo 4

DE híbrido para *FJSSP*

4.1. Introducción

El problema de JSSP (*Job Shop Scheduling Problem*) es uno de los problemas más importantes y difíciles en el campo de la programación de tareas. El problema FJSSP (*Flexible Job Shop Scheduling Problem*) es una extensión del JSSP clásico, en el que las operaciones pueden ser procesadas por cualquier máquina de un conjunto dado, en lugar de una máquina específica. En general, el FJSSP está mucho más cerca de un entorno de producción real y tiene una aplicabilidad más práctica. Sin embargo, el FJSSP es más complejo que el JSSP debido a su decisión adicional de asignar cada operación a la máquina apropiada además de las operaciones de secuenciación en las máquinas. Fue probado que FJSSP es NP-duro si cada trabajo tiene como mínimo tres operaciones y hay al menos dos máquinas [14].

Ante la búsqueda de soluciones a problemas de tan alta complejidad, en los últimos años el interés en las metaheurísticas híbridas ha aumentado considerablemente en el campo de la optimización. Los mejores resultados encontrados para muchos problemas de optimización clásicos o de la vida real se obtienen mediante algoritmos híbridos [9]. En este trabajo, será considerado un algoritmo de evolución diferencial híbrido para hacer frente al problema FJSSP planteado.

La resolución de un problema de optimización complejo y/o de alta dimensión puede beneficiarse de la distribución del computo. La idea es resolver pequeñas partes del problema y alcanzar la mejor solución en un tiempo razonable [8]. Esta idea puede ser llevada al algoritmo que resuelve el problema de optimización.

En este capítulo será descrito el problema FJSSP a resolver, el diseño de la repre-

sentación y evaluación de soluciones, la decodificación de soluciones, un algoritmo de búsqueda local, y por último la implementación de un algoritmo de *DE* híbrido para el problema de *FJSSP*.

4.2. FJSSP: *Flexible Job Shop Scheduling Problem*

En comparación con el JSSP (*Job Shop Scheduling Problem*) clásico, donde se requiere que cada trabajo se procese en una sola máquina, la asignación de tareas en FJSSP es más desafiante, ya que requiere una selección adecuada de una máquina de un conjunto de máquinas determinadas para procesar cada operación. Eso significa que el FJSSP consta de dos subproblemas. El primero es asignar cada operación a una máquina de un conjunto de máquinas capaces, y el segundo trata de la secuenciación de las operaciones asignadas en todas las máquinas.

El problema puede ser descrito de la siguiente manera. Dado un conjunto de trabajos independientes $J = \{J_1, J_2, ..., J_n\}$. Un trabajo J_i está formado por una serie de operaciones $O_{i1}, O_{i2}, ..., O_{in_i}$ que se realizarán una después de la otra de acuerdo con la secuencia dada. El conjunto de máquinas $U = \{M_1, M_2, ..., M_m\}$ es dado. Cada operación O_{ij} puede ejecutarse en cualquiera de los subconjuntos $U_{ij} \subseteq U$ de máquinas compatibles. Se tiene flexibilidad parcial si existe un subconjunto adecuado $U_{ij} \subset U$ para al menos una operación O_{ij}, en cambio, se tiene flexibilidad total si $U_{ij} = U$ para cada operación O_{ij}. El tiempo de procesamiento de cada operación depende de la máquina, por lo tanto, d_{ijk} es el tiempo de procesamiento de la operación O_{ij} cuando se ejecuta en la máquina M_k. No se permite la anticipación, es decir, cada operación debe completarse sin interrupción una vez iniciada. Además, las máquinas no pueden realizar más de una operación a la vez. Todos los trabajos y máquinas están disponibles en tiempo 0.

El problema es asignar cada operación a una máquina apropiada (problema de enrutamiento) y secuenciar las operaciones en las máquinas (problema de secuenciación) con el fin de reducir el tiempo de terminación (*makespan* en inglés). Esta medida es el tiempo necesario para completar todos los trabajos del problema y se define como $C_{max} = max_i\{C_i\}$, donde C_i es el tiempo de finalización del trabajo J_i. La tabla 4.1 muestra una instancia del problema FJSSP con 3 trabajos, 4 máquinas y 8 operaciones. Las filas y columnas corresponden, respectivamente, a operaciones y máquinas, y las entradas de la tabla son los tiempos de procesamiento requeridos. En este ejem-

plo, se tiene un escenario flexible parcial, una entrada de ∞ en la tabla significa que una máquina no puede ejecutar la operación correspondiente, es decir, no pertenece al subconjunto de máquinas compatibles para esa operación. Siguiendo el paradigma de los algoritmos evolutivos, llamaremos a cualquier solución del problema como un individuo o cromosoma.

4.3. Representación y evaluación de soluciones

Un punto de diseño fundamental en el desarrollo de metaheurísticas es la codificación (representación de una solución) a utilizar. En este caso, se consideró la propuesta por Bierwirth en [16] que es adecuada y relevante para el problema de optimización abordado, la cual se basa en una permutación con repeticiones. Una solución, S, es una permutación del conjunto de operaciones que representa un pedido tentativo para ordenar su programación, cada una representada por su número de trabajo. Por ejemplo, dado el vector $x_i^g = [0.6, -0.5, 0.4, -0.3, -0.1, 0.9, -0.7, 0.2]$, este será convertido a $[2, 1, 1, 3, 2, 2, 1, 3]$. Tal como se observa, el largo del vector x_i^g es equivalente a la cantidad de operaciones. Esta programación válida se corresponde con la secuencia de operaciones $S = [O_{21}, O_{11}, O_{12}, O_{31}, O_{22}, O_{23}, O_{13}, O_{32}]$.

Otro punto de diseño común es la definición de la función objetivo que guiará la búsqueda hacia soluciones "buenas"dentro del espacio de búsqueda. Al evaluar S, el valor objetivo es el tiempo de terminación de esta solución. Para calcular este valor, cada operación O_{ij} en S se asigna a una máquina viable M_k en el subconjunto U_{ij} con el tiempo de finalización más corto, y luego se actualiza la carga de M_k. Como solución inicial se utiliza un procedimiento aleatorio, principalmente porque no se conocen heurísticas de construcción de alto rendimiento para FJSSP.

Tabla 4.1: Tabla de tiempo de procesamiento

		M_1	M_2	M_3	M_4
J_1	O_{11}	∞	6	5	∞
	O_{12}	4	8	5	6
	O_{13}	9	5	∞	7
J_2	O_{21}	2	∞	1	3
	O_{22}	4	6	8	4
	O_{23}	9	∞	2	2
J_3	O_{31}	8	6	∞	5
	O_{32}	3	5	8	3

Algoritmo 5 Pseudocódigo del algoritmo de decodificación de soluciones

Entrada: $x = (x_1, x_2, ..., x_D)$

Salida: $S[D]$

1: $S[D] \leftarrow$ LPV(x)
2: **para** cada vector J_i de $\{J_1, J_2, ..., J_n\}$ **hacer**
3:     **para** cada posición j de $S[D]$ **hacer**
4:         **si** $S[j]$ es una operación de J_i **entonces**
5:             $S[j] \leftarrow J_i$
6:         **fin si**
7:     **fin para**
8: **fin para**

4.4. Decodificación de soluciones

En este trabajo, para mantener la simplicidad y las propiedades de DE en su configuración natural el algoritmo DE manipula vectores con valores reales. Consecuentemente, la decodificación de las soluciones consiste en convertir un vector de reales $x = (x_1, x_2, ..., x_D) \in R^D$ a la codificación propuesta por Bierwirth, descripta en la sección anterior, para obtener una solución S.

Para realizar esta conversión, la regla del valor de posición más grande (*largest position value* o LPV en inglés) presentada en [7] se emplea primero para construir una permutación de operaciones sin repeticiones al ordenar los valores del vector de reales en forma decreciente en conjunto con el vector de posiciones. Luego, se transforma el vector de permutaciones sin repeticiones al esquema de vector de permutaciones con repeticiones (solución) propuesto por Bierwirth. Para esto se itera sobre el conjunto $J = \{J_1, J_2, ..., J_n\}$ de trabajos y dentro sobre cada posición del vector de permutaciones, si la operación contenida en la posición tratada pertenece al trabajo J_i, se cambia el valor de $S[j]$ por el de J_i. En el algoritmo 5 se puede apreciar la especificación del proceso realizado.

En la Figura 4.1 se puede ver un ejemplo de este procedimiento para una instancia del problema FJSSP con 3 trabajos, 4 máquinas y 8 operaciones mostrada en la Tabla 4.1.

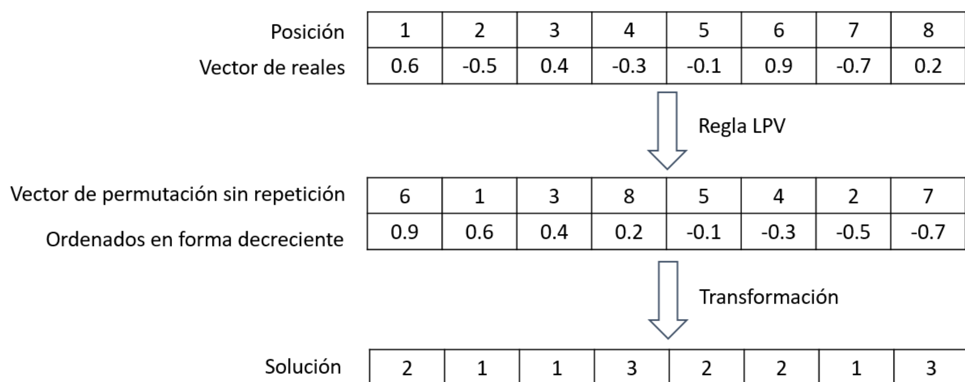

Figura 4.1: Ejemplo del proceso de decodificación de una solución para una instancia del problema de FJSSP.

4.5. DE y búsqueda local

Con el fin de mejorar la eficiencia de DE al resolver el FJSSP, se incorporó al algoritmo un procedimiento de búsqueda local simple, en el que se se seleccionan e intercambian al azar dos posiciones del vector de prueba. Si hay una mejora en la función objetivo, se acepta el intercambio, de lo contrario, no se considera. En el Algoritmo 6 se puede ver un pseudocódigo del procedimiento de búsqueda local.

Este procedimiento de búsqueda local se aplica a los vectores de prueba de la próxima generación (justo antes de la línea 10 del Algoritmo 4), lo que es beneficioso para evitar quedar atrapado en un óptimo local. Una característica importante de este procedimiento de búsqueda local es que no necesita una conversión hacia atrás, es decir, se aplica directamente sobre el vector de prueba.

La frecuencia de búsqueda local está controlada por la probabilidad P_{BL}. La búsqueda local se puede aplicar a cada individuo de la población o a solo unos pocos. Aplicar una búsqueda local a cada individuo de la población puede desperdiciar recursos sin proporcionar más información útil que hacerlo solo a una pequeña fracción de la población. El uso de una gran fracción de la población puede limitar la exploración del espacio de búsqueda al permitir que el algoritmo genético evolucione durante un pequeño número de generaciones. Un uso más selectivo de la búsqueda local puede mejorar la eficiencia de los híbridos. La probabilidad de búsqueda local puede afectar la velocidad de convergencia del algoritmo. Este efecto no debe ignorarse al decidir entre diferentes probabilidades de búsqueda local.

Algoritmo 6 Pseudocódigo de la Búsqueda local

```
1: para cada vector x_i de P hacer
2:     si random() < P_BL entonces
3:         j , k ← random(1,D)
4:         u_i ← intercambiar(x_i, j,k)
5:         si f(u_i) ≤ f(x_i) entonces        ▷ para un problema de minimización
6:             x_i ← u_i
7:         fin si
8:     fin si
9: fin para
```

4.6. *DE* y paralelismo

Actuando sobre la eficiencia del algoritmo, se incorporó un paralelismo a nivel de iteración, con el fin de acelerar el algoritmo reduciendo el tiempo de búsqueda. Como *DE* es una metaheurística basada en población se consideró un esquema de maestro-trabajador, donde los trabajadores realizan las operaciones de mutación, recombinación y evaluación de la función objetivo, y el maestro al recibir de los trabajadores las soluciones ya evaluadas, realiza el proceso de selección. En la figura 4.2 se puede observar cómo se realiza la paralelización para 4 CPUs.

Para la búsqueda local también se implementó un paralelismo a nivel de iteración, como se puede observar en la figura 4.3, donde los distintos vecinos son divididos en diferentes particiones del mismo tamaño. Una vez generadas las particiones, se aplica la búsqueda local a cada uno de sus individuos y luego se evalúan. Las operaciones en las particiones ocurren en forma paralela e independiente. Este esquema otorga grandes ventajas en cuanto a velocidad y aprovechamiento de recursos.

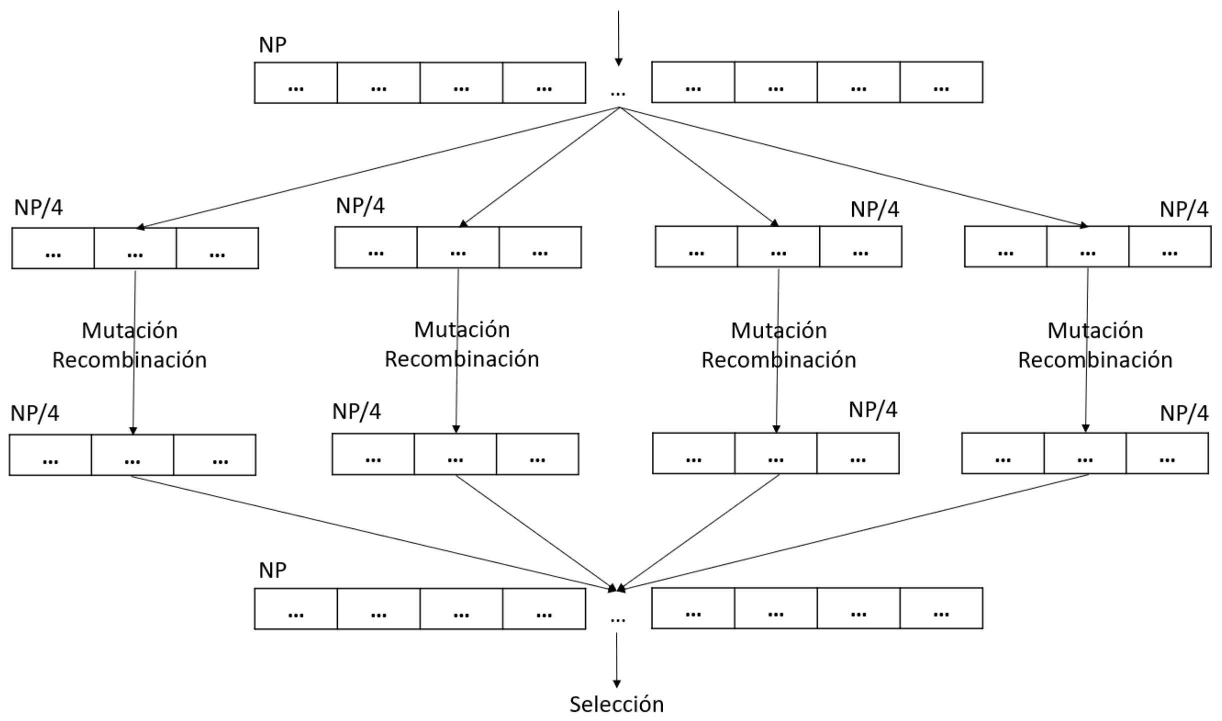

Figura 4.2: Paralelización del algoritmo DE.

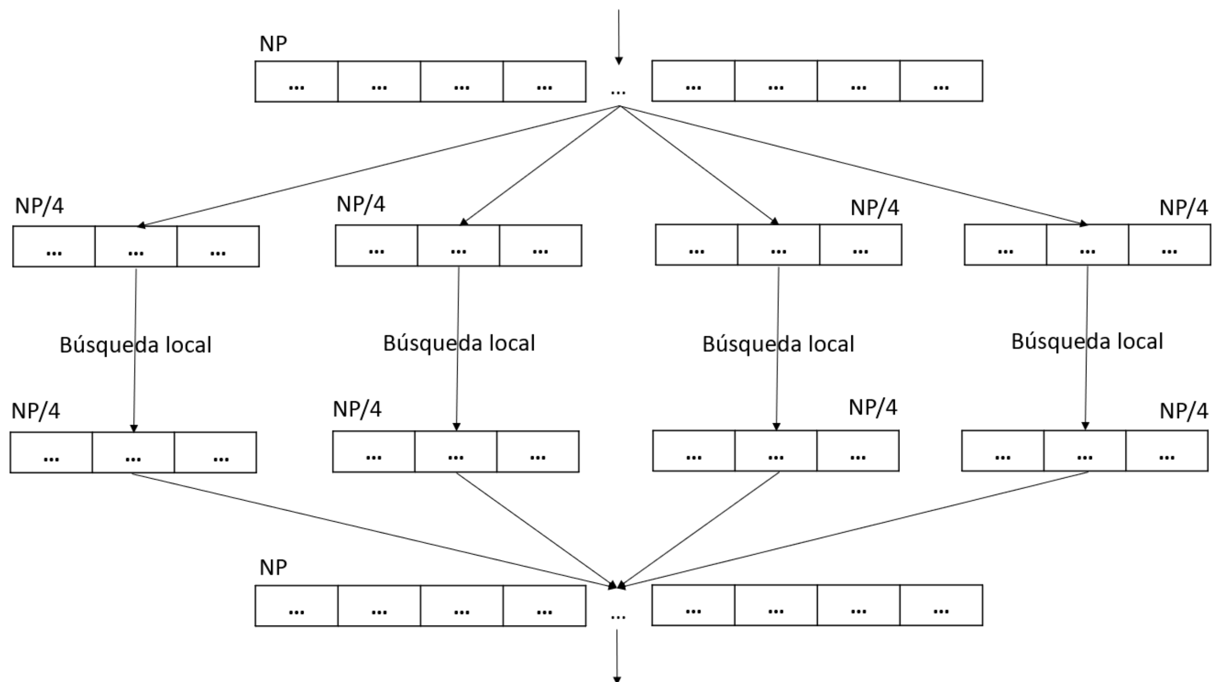

Figura 4.3: Paralelización de la búsqueda local.

4.7. *HDE* para el *FJSSP*

El algoritmo 7 presenta el pseudocódigo del algoritmo de *DE* híbrido para *FJSSP*. Como se puede observar el esquema general es el mismo que el presentado en el capítulo anterior, con la inclusión del proceso de búsqueda local. Además, se agrega como entrada la probabilidad de aplicar búsqueda local (P_{BL}), con el fin de permitir al investigador realizar cambios sobre el parámetro.

Algoritmo 7 Pseudocódigo de la Evolución Diferencial (DE)

Entrada: F, Cr, N_p, P_{BL}

Salida: x_{best}

1: inicializar(P,N_p)
2: $g \leftarrow 0$
3: **mientras** no se alcance la condición de fin **hacer**
4:     **para** cada vector x_i^g de P^g **hacer**
5:         $v_i^g \leftarrow$ mutación(x_i^g, P^g, F)
6:         $u_i^g \leftarrow$ recombinación(x_i^g, v_i^g, Cr)
7:         $x_i^{g+1} \leftarrow$ selección(x_i^g, u_i^g)
8:         agregar(P^{g+1}, x_i^{g+1})
9:     **fin para**
10:     búsqueda-local(P^g, P_{BL})
11:     $g \leftarrow g + 1$
12: **fin mientras**
13: $x_{best} \leftarrow$mejor-solución(P^g)

Capítulo 5

Estudios experimentales

5.1. Introducción

Para validar el algoritmo presentado en esta tesis, se adoptó un conjunto de instancias de prueba del FJSSP, otorgando grandes ventajas a la hora de comparar los resultados obtenidos frente a otros algoritmos. A fin de permitir una comparación cuantitativa de resultados, en este trabajo se utilizaron tablas comparativas que exponen los valores óptimos, los mejores obtenidos y las medias de los mismos, junto con el desvío estándar. Además se utilizó el error relativo, que es el cociente entre el error absoluto de una medida y el valor real de ésta, mostrados en forma de gráficos de caja (*boxplot* en inglés), los cuales son una forma de presentación estadística destinada, fundamentalmente, a resaltar aspectos de la distribución de las observaciones en una o más series de datos cuantitativos. El gráfico de caja es una buena alternativa a la presentación tradicional de datos medidos con escala cuantitativa, pues permite cotejar varias series de datos medidas con la misma escala y ubicadas en posiciones parecidas de ésta, siendo, en tal sentido, más claro y de mayor información que otros tipos de gráficos.

Dado que el objetivo del paralelismo es la reducción del tiempo real, se utiliza la aceleración como medida de comparación del algoritmo *DE* frente a su versión paralela. El tiempo debe incluir cualquier actividad que se realice en el proceso, por lo que la opción más prudente para medir el rendimiento de un código paralelo es considerar el tiempo entre el inicio y fin de todo el algoritmo para resolver el problema en cuestión. Para una comparación justa, la misma idea debe ser tomada en consideración en el caso secuencial.

En este capítulo se describe el diseño experimental llevado a cabo para este traba-

Tabla 5.1: Instancias propuestas por Brandimarte

Instancia	Cantidad de trabajos	Cantidad de operaciones	Cantidad de máquinas	Óptimo
Mk01	10	55	6	40
Mk02	10	58	6	26
Mk03	15	150	8	204
Mk04	15	90	8	60
Mk05	15	106	4	172
Mk06	10	150	15	58
Mk07	20	100	5	139
Mk08	20	255	10	523
Mk09	20	240	10	307
Mk10	20	240	15	197

jo. Además se presentan los resultados experimentales obtenidos variando el factor de escalado (F) y la probabilidad de recombinación (Cr), agregando el procedimiento de búsqueda local y la mejora en tiempo del algoritmo gracias al paralelismo. Por último, se lleva a cabo una comparación de los resultados del algoritmo *HDE* frente a otros algoritmos competitivos presentes en la literatura.

5.2. Diseño experimental

En esta sección, se describe el diseño experimental utilizado en este trabajo. Se selecciona una amplia gama de instancias del FJSSP utilizadas en la literatura teniendo en cuenta su complejidad, que viene dada por el número de trabajos y máquinas, y la amplia variación de flexibilidad en la cantidad de máquinas disponibles por operación. En este sentido, fue considerado el conjunto de datos propuestos por Brandimarte [3] presentado en la tabla 5.1, ya que el número de trabajos varía de 10 a 20, el número de máquinas pertenece al rango [4,15] y el número de operaciones para cada trabajo puede tomar valores desde 5 a 15, en consecuencia, el número total de operaciones va desde 55 a 255. Teniendo en cuenta la flexibilidad esta oscila entre 1.43 y 4.10 [63].

Con respecto a la metodología seguida para analizar los resultados, primero, estudiamos el comportamiento de estos algoritmos con diferentes valores de Cr y F, considerando los mejores C_{max} encontrados y el error relativo de C_{max} contra el óptimo conocido en cada instancia. Estos análisis nos permiten determinar los mejores valores para los parámetros de control. En segundo lugar, determinamos el impacto de incorporar un procedimiento de búsqueda local a diferentes probabilidades P_{BL}. Para este propósito, tomamos en cuenta los mejores C_{max} encontrados, la tasa de aciertos (es

decir, la cantidad de veces que un algoritmo encuentra la mejor solución) y el error relativo de C_{max} contra el óptimo conocido en cada instancia. Finalmente, estudiamos el comportamiento del algoritmo de *DE* híbrido incluyendo el paralelismo con respecto al tiempo de ejecución en cada enfoque.

Los algoritmos considerados fueron programados en C++, por lo que su tiempo de ejecución es directamente comparable. Todos los algoritmos se compilaron en la misma computadora con los mismos indicadores de compilación y se ejecutaron en hardware homogéneo. La experimentación se llevó a cabo en un clúster conformado por cuatro máquinas con INTEL I7 3770K, 8 GB de RAM y el Slackware Linux con la versión 2.6.27 del núcleo. Para implementar la versión paralela de DE, se utiliza una interfaz de programación de aplicaciones (API) para computadoras paralelas de memoria compartida como es OpenMP [10].

Para este estudio, la configuración paramétrica del DE es la que se muestra en la tabla 5.2. El tamaño de la población, N_P, fue adoptado de trabajos anteriores y se establece en 50. Con respecto a la probabilidad de Cr y el factor F, se consideraron tres valores diferentes para cada uno de ellos 0.1, 0.5 y 0.9. Utilizar un valor de Cr alto implica que el vector prueba hereda muchos componentes del vector donador. En caso contrario, si es muy bajo, el vector prueba hereda muchos componentes del vector objetivo. Un valor intermedio como el de 0.5, produciría una situación intermedia e incrementaría la diversidad poblacional. Los valores extremos harían que la diversidad poblacional no fuese incrementada. El factor de escalado F controla la longitud del salto generado en la explotación del espacio de búsqueda. Un valor bajo lleva al algoritmo a intensificar la explotación de soluciones en algunas regiones del espacio de búsqueda, mientras que un valor alto implica una mayor diversificacion de las soluciones, explorando en mayor medida todo el espacio de búsqueda. Un valor intermedio hace un balance entre diversificación e intensificación. Para el parámetro restante, P_{BL}, también se analizaron tres valores 0.1, 0.5 y 0.7 (valores con baja, media y alta probabilidad), para estudiar cómo la frecuencia de aplicación de la búsqueda local impacta en

Tabla 5.2: Valores de los parámetros

Parámetro	**Valor**
N_P	50
F	0.1, 0.5, y 0.9
Cr	0.1, 0.5, y 0.9
P_{BL}	0.1, 0.5, y 0.7

los resultados arrojados por el *DE*.

Para hacer una comparación justa entre estos algoritmos, deben hacer el mismo esfuerzo computacional en cada ejecución. Puede lograrse si se ejecuta durante el mismo tiempo cada instancia del problema. De esta manera, el tiempo total ejecutado para una instancia se calcula teniendo en cuenta su tamaño. En la Ecuación 5.1 se explica cómo se calcula el tiempo total de ejecución, donde $\sharp O$ es el cantidad de operaciones para una instancia determinada.

$$TiempoEjecucion = \sharp O \times (\frac{\sharp O}{2}) \times 30 \quad (5.1)$$

Debido a la naturaleza estocástica de los algoritmos, fueron realizadas 30 ejecuciones independientes de cada prueba con el fin de recopilar datos experimentales significativos y aplicar métricas de confianza estadística para validar las conclusiones.

5.3. Resultados experimentales

En esta sección, analizamos la calidad de los resultados considerando los valores de C_{max} obtenidos para las distintas mejoras planteadas al algoritmo DE descrito en el capítulo anterior para resolver las instancias de FJSSP. En primer lugar se estudiará la influencia de los parámetros Cr y F, luego la incorporación del proceso de búsqueda local a través del parámetro P_{BL}, después el impacto en los tiempos al incorporar un paralelismo a nivel de iteración y por último se hace una comparación de los valores de C_{max} obtenidos por el algoritmo con los alcanzados por varios algoritmos competitivos presentes en la literatura.

5.3.1. Resultados variando los parámetros F y Cr

El primer análisis se centra en el efecto de usar diferentes valores de F y Cr en el rendimiento de *DE*, de valores bajos a altos (0.1, 0.5 y 0.9). Para este fin, se analizó la calidad de los resultados teniendo en cuenta los valores de C_{max} obtenidos para el *DE* básico al resolver las instancias FJSSP, es decir, se estudió cómo impacta en el rendimiento de *DE* el utilizar las distintas combinaciones de valores de parámetros.

La tabla 5.3 muestra los mejores valores de C_{max} obtenidos para el *DE* utilizando las combinaciones posibles de valores de F y Cr para cada instancia. La columna 2 presenta el mejor valor conocido de C_{max} (óptimo) para cada instancia. La última fila de

Tabla 5.3: Mejores valores de C_{max} encontrados por el *DE* con diferentes valores de F y Cr para todas las instancias del FJSSP

Instancia	Opt.	Mejor valor de C_{max}								
		F=0.1 Cr=0.1	F=0.1 Cr=0.5	F=0.1 Cr=0.9	F=0.5 Cr=0.1	F=0.5 Cr=0.5	F=0.5 Cr=0.9	F=0.9 Cr=0.1	F=0.9 Cr=0.5	F=0.9 Cr=0.9
Mk01	40	**40**	**40**	**40**	**40**	**40**	**40**	**40**	**40**	**40**
Mk02	26	**26**	27	27	**26**	27	**26**	27	27	27
Mk03	204	**204**	**204**	**204**	**204**	**204**	**204**	**204**	**204**	**204**
Mk04	60	**60**	**60**	65	61	**60**	**60**	62	63	62
Mk05	172	175	173	173	175	179	173	175	179	173
Mk06	58	65	59	63	66	69	59	67	70	61
Mk07	139	143	140	142	143	148	140	144	146	140
Mk08	523	**523**	**523**	**523**	**523**	**523**	**523**	**523**	**523**	**523**
Mk09	307	318	**307**	310	321	333	**307**	321	338	**307**
Mk10	197	237	210	221	238	245	206	240	246	215
		5/10	5/10	3/10	4/10	4/10	6/10	3/10	3/10	4/10

Tabla 5.4: Valores medios de C_{max} encontrados por el *DE* con diferentes valores de F y Cr para todas las instancias del FJSSP

Instancia	Valor medio de Cmax y desvío estándar medio								
	F=0,1 Cr=0.1	F=0.1 Cr=0.5	F=0.1 Cr=0.9	F=0.5 Cr=0.1	F=0.5 Cr=0.5	F=0.5 Cr=0.9	F=0.9 Cr=0.1	F=0.9 Cr=0.5	F=0.9 Cr=0.9
Mk01	40 $_{\pm0.00}$	40.26$_{\pm0.69}$	41.8$_{\pm0.48}$	40$_{\pm0.00}$	40$_{\pm0.00}$	40.06$_{\pm0.25}$	40$_{\pm0.00}$	40.03$_{\pm0.18}$	40.53$_{\pm0.73}$
Mk02	26.57 $_{\pm0.50}$	27$_{\pm0.00}$	27.8$_{\pm0.48}$	26.96$_{\pm0.18}$	27.33$_{\pm0.47}$	26.96$_{\pm0.18}$	27.03$_{\pm0.18}$	27.23$_{\pm0.43}$	27.33$_{\pm0.47}$
Mk03	204 $_{\pm0.00}$	204$_{\pm0.00}$	204$_{\pm0.00}$	204$_{\pm0.00}$	204$_{\pm0.00}$	204$_{\pm0.00}$	204$_{\pm0.00}$	204$_{\pm0.00}$	204$_{\pm0.00}$
Mk04	60 $_{\pm0.00}$	62.03$_{\pm1.42}$	66.93$_{\pm0.82}$	62.16$_{\pm0.53}$	65.76$_{\pm1.27}$	64.56$_{\pm1.90}$	63.76$_{\pm0.97}$	66.23$_{\pm0.81}$	65.33$_{\pm1.72}$
Mk05	176.33 $_{\pm0.66}$	177.2$_{\pm1.67}$	175.8$_{\pm1.07}$	176.5$_{\pm0.73}$	181.3$_{\pm0.96}$	173.6$_{\pm0.92}$	176.8$_{\pm0.69}$	181.2$_{\pm1.01}$	173.7$_{\pm0.94}$
Mk06	66.7 $_{\pm0.75}$	64.2$_{\pm2.52}$	66.13$_{\pm1.04}$	67.53$_{\pm0.62}$	70.83$_{\pm0.69}$	61.93$_{\pm1.36}$	67.96$_{\pm0.55}$	71.1$_{\pm0.54}$	63.53$_{\pm1.19}$
Mk07	144.3 $_{\pm0.53}$	143.8$_{\pm2.02}$	144.5$_{\pm1.27}$	144.5$_{\pm0.77}$	150.1$_{\pm0.94}$	141.8$_{\pm1.39}$	144.8$_{\pm0.79}$	150.1$_{\pm1.11}$	142.1$_{\pm1.12}$
Mk08	523 $_{\pm0.00}$	523$_{\pm0.00}$	523$_{\pm0.00}$	523$_{\pm0.00}$	523.5$_{\pm1.04}$	523$_{\pm0.00}$	523$_{\pm0.00}$	523.6$_{\pm0.95}$	523$_{\pm0.00}$
Mk09	323.6 $_{\pm2.44}$	316.1$_{\pm7.98}$	323.9$_{\pm6.23}$	326.3$_{\pm2.35}$	341.5$_{\pm2.87}$	307.5$_{\pm1.47}$	327.9$_{\pm2.32}$	342.3$_{\pm2.29}$	310.7$_{\pm3.46}$
Mk10	240.63 $_{\pm1.54}$	239.5$_{\pm7.10}$	226.8$_{\pm2.83}$	242.1$_{\pm1.85}$	251.2$_{\pm2.12}$	212.4$_{\pm2.99}$	242.3$_{\pm1.32}$	251.7$_{\pm1.86}$	219.5$_{\pm2.66}$

esta tabla muestra la relación entre el número de instancias resueltas de forma óptima con respecto al número total de instancias. En la tabla 5.4 se encuentran los valores medios de C_{max} junto con la desviación estándar media para los distintos valores de F y Cr.

Como se puede ver en la Tabla 5.3, el *DE* encuentra el óptimo en 3 instancias (MK01, MK03 y MK08) independientemente de las combinaciones de valores de F y Cr consideradas. Con un valor alto de F *DE* tiene menos posibilidades de encontrar las mejores soluciones para el FJSSP sin importar el valor de Cr usado, ya que la cantidad de valores óptimos alcanzados para las diferentes instancias es menor frente a otras posibles combinaciones de valores paramétricos (no más de 4 de las 10 instancias). El *DE* con $F = 0.5$ y $Cr = 0.9$ encuentra una mayor cantidad de veces valores de C_{max}

óptimos que el resto de las combinaciones (en 6 de las 10 instancias).

Una métrica importante es el error relativo de los mejores C_{max} obtenidos para cada instancia con respecto al óptimo. Esta métrica nos permite normalizar los datos de las diferentes instancias y, de esta manera, enfocar la atención en cómo las diferentes combinaciones de valores de F y Cr afectan al rendimiento de *DE* al resolver el FJSSP. Observando el *boxplot* del error relativo para las distintas combinaciones de parámetros de F y Cr presentado en la Figura 5.1 se puede ver que el algoritmo *DE* con $F = 0.5$ y $Cr = 0.9$ presenta la menor mediana del error. La caja de rango intercuartil muestra que la distancia entre el primer cuartil y el tercer cuartil es menor frente a otras posibles combinaciones de valores de parámetros. Los bigotes que se extienden desde la parte superior de la caja son claramente inferiores frente a los de las demás combinaciones. Por último, se puede observar que no existen valores atípicos, lo cual presenta una superioridad con respecto a las posibles elecciones de parámetros.

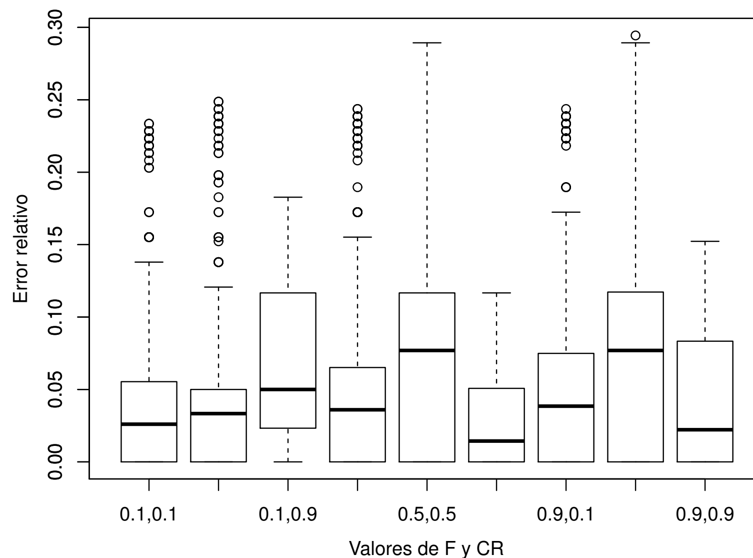

Figura 5.1: Boxplot del error relativo del *DE* para distintos valores de F y Cr.

Finalmente, se estudiará la distribución del número de evaluaciones realizadas por *DE* para encontrar los mejores valores de C_{max} para las diferentes combinaciones de F y Cr. Con este fin, la Figura 5.2 ilustra los resultados utilizando diez *boxplots* (uno por cada instancia). En general, se observa que el algoritmo *DE* con menor cantidad de evaluaciones es aquel con $F \in \{0.1, 0.5\}$ y $Cr = 0.9$, pero la combinación $F = 0.5$ y $Cr = 0.9$ supera a las demás desde el punto de vista de la calidad de resultados (ver Tabla 5.3). En consecuencia, serán considerados los valores de $F = 0.5$ y $Cr = 0.9$ en la experimentación restante.

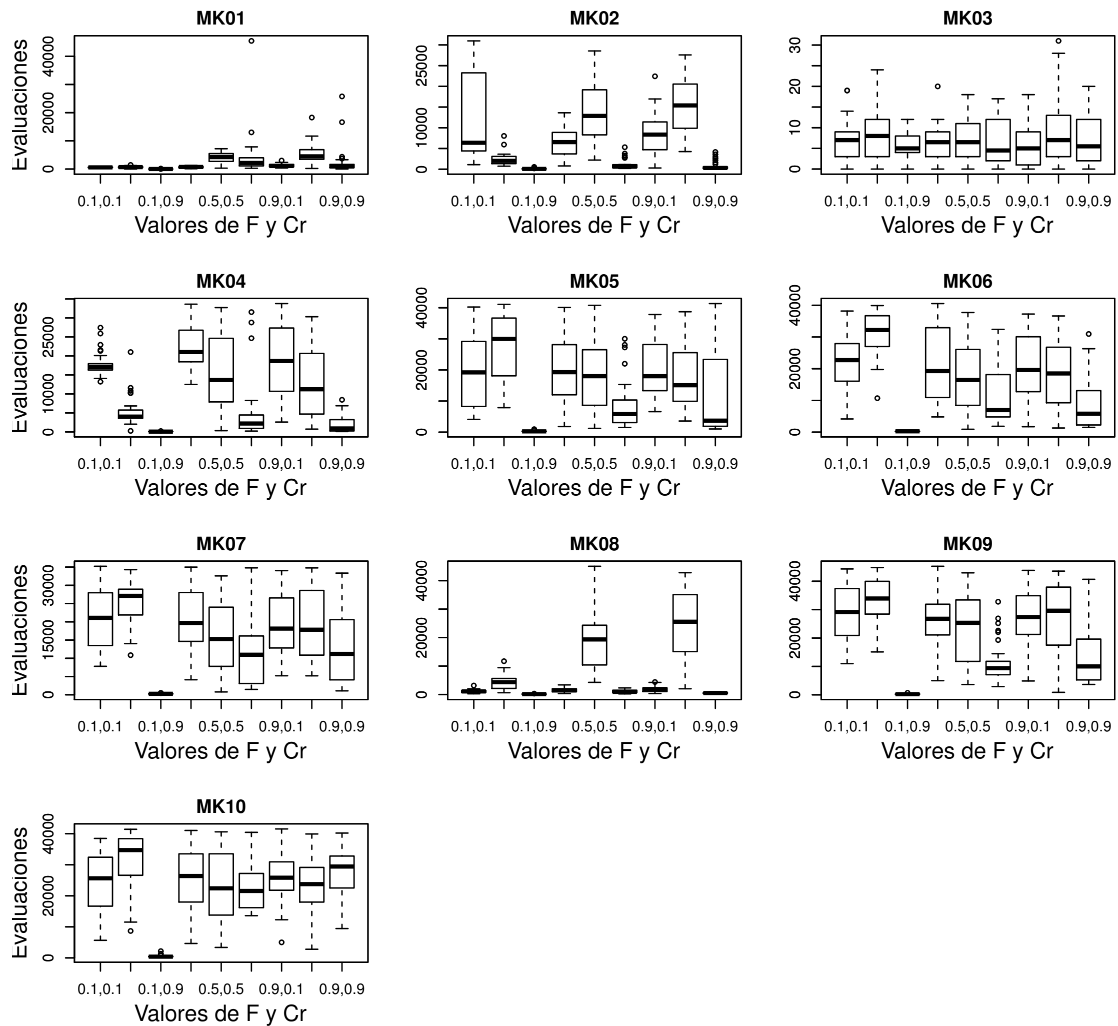

Figura 5.2: Boxplot del error relativo del *DE* para distintos valores de F y Cr.

5.3.2. Resultados de *DE* con búsqueda local

En esta sub-sección se analiza en detalle lo que sucede al introducir un procedimiento de búsqueda local en el algoritmo *DE* para resolver las instancias de FJSSP. El algoritmo resultante es llamado *HDE*. Para esos estudios, fueron considerados tres valores diferentes de P_{BL}: 0.1, 0.5 y 0.7 (de valores altos a bajos), es decir, se estudió cómo la frecuencia de la aplicación del procedimiento de búsqueda local impacta en el rendimiento de *HDE*.

La tabla 5.5 muestra los mejores y la media de los valores de C_{max} obtenidos para el algoritmo *HDE* con los diferentes valores de P_{BL}. El algoritmo *HDE* obtiene un número mayor de C_{max} óptimos con $P_{BL} = 0.5$ y con $P_{BL} = 0.7$. Pero el algoritmo *HDE* que aplica el procedimiento de búsqueda local con la frecuencia más alta ($P_{BL} = 0.7$) presenta valores de C_{max} medios más bajo para todas las instancias. Esto indica que el

Tabla 5.5: Valores de C_{max} encontrados por el *HDE* con diferentes valores de P_{BL} para todas las instancias de FJSSP.

Instancia	Optimos	Mejor valor de Cmax			Valor medio de Cmax y desvío estándar medio		
		$P_{BL} = 0.1$	$P_{BL} = 0.5$	$P_{BL} = 0.7$	$P_{BL} = 0.1$	$P_{BL} = 0.5$	$P_{BL} = 0.7$
Mk01	40	**40**	**40**	**40**	$40_{\pm 0.00}$	$40_{\pm 0.00}$	$40_{\pm 0.00}$
Mk02	26	27	**26**	**26**	$27.03_{\pm 0.18}$	$26.96_{\pm 0.18}$	$26.76_{\pm 0.43}$
Mk03	204	**204**	**204**	**204**	$204_{\pm 0.00}$	$204_{\pm 0.00}$	$204_{\pm 0.00}$
Mk04	60	**60**	**60**	**60**	$62_{\pm 1.31}$	$61.3_{\pm 0.70}$	$61.03_{\pm 0.66}$
Mk05	172	173	173	173	$174.3_{\pm 0.75}$	$173_{\pm 0.00}$	$173_{\pm 0.00}$
Mk06	58	63	62	61	$64.4_{\pm 0.56}$	$62.83_{\pm 0.53}$	$62.43_{\pm 0.62}$
Mk07	139	142	140	140	$143.2_{\pm 0.69}$	$142.0_{\pm 0.90}$	$141.8_{\pm 0.92}$
Mk08	523	**523**	**523**	**523**	$523_{\pm 0.00}$	$523_{\pm 0.00}$	$523_{\pm 0.00}$
Mk09	307	309	**307**	**307**	$313.1_{\pm 2.20}$	$310.0_{\pm 1.33}$	$309.0_{\pm 1.72}$
Mk10	197	226	225	224	$231.1_{\pm 2.41}$	$228.0_{\pm 1.36}$	$227.2_{\pm 1.38}$
		4/10	6/10	6/10			

algoritmo encuentra el valor óptimo o casi óptimo en la mayoría de las ejecuciones, lo cual queda claramente plasmado en el *boxplot* del error relativo de las distintas instancias contra los óptimos para los distintos valores de P_{BL} presentado en la figura 5.3, donde se observa que la mediana del error es menor con la frecuencia de búsqueda local más alta.

Para determinar que el *HDE* presenta una mejora en los valores de C_{max} obtenidos por el *DE*, se realiza una comparación de los valores de los errores relativos mostrados en la Figura 5.1 y los encontrados en la Figura 5.3. Se observa que el algoritmo *HDE* con $P_{BL} = 0.7$ presenta menor error relativo frente a los otros presentados en *DE*, lo que indica la ventaja de incorporar la búsqueda local dentro del marco de *DE*.

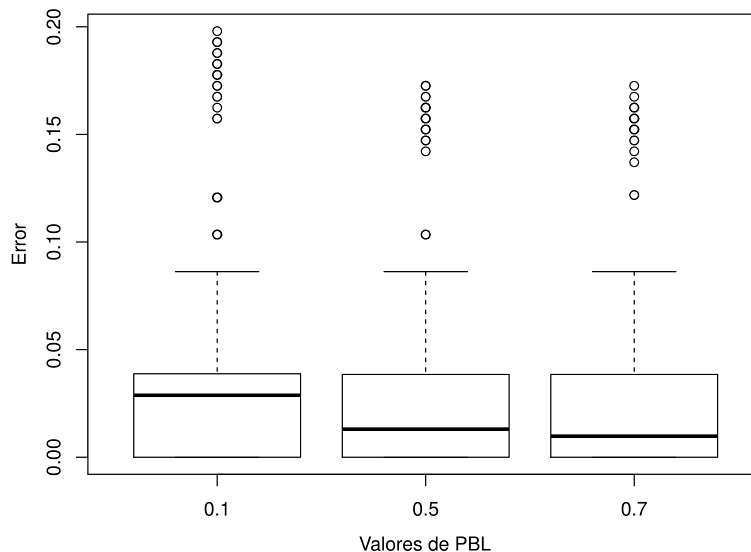

Figura 5.3: Boxplot del error relativo del *HDE* para distintos valores de P_{BL}.

5.3.3. Resultados de *HDE* y su paralelismo

En esta subsección se compara el *HDE* contra su versión paralela. La medida más importante de un algoritmo paralelo es el *speedup*. El *speedup* se define como la relación entre el tiempo de ejecución secuencial (tiempo de ejecución de *HDE*, en este caso) y el tiempo de ejecución paralelo. Para este análisis, fue considerado el *speedup* débil [15]. Por esa razón y siguiendo las mejores prácticas de Luque y Alba [17], el criterio de detención se basa en la calidad de la solución final lograda por los algoritmos, que se establece en los mejores C_{max} conocidos para cada instancia de FJSSP (consulte la columna óptimo de la tabla 5.1). En consecuencia, los valores de *speedup* solo se informan para las instancias para las cuales el algoritmo *HDE* obtiene el valor óptimo.

Una vez establecidos los tiempos de ejecución del algoritmo *HDE* y el algoritmo *HDE* paralelo, se calculan los valores de *speedup*. La Figura 5.4 muestra que el uso de la paralelización vale la pena, ya que permite acelerar el tiempo de ejecución con respecto al *HDE* secuencial sobre todas las instancias en un valor de 3 veces en promedio. El valor ideal de aceleración es 4, el número de núcleos disponibles por máquina, logrando así una aceleración lineal.

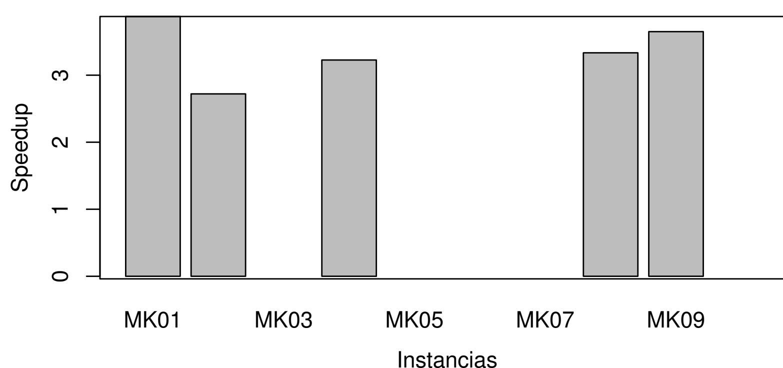

Figura 5.4: *Speedup* por instancia del FJSSP.

5.4. Comparación de *HDE* con la literatura

Finalmente, se presenta una comparación de los valores de C_{max} obtenidos por el *HDE* con los alcanzados por varios algoritmos competitivos presentes en la literatura destinados a resolver el FJSSP. Esto permite determinar qué tan buena es la metaheurística presentada en este trabajo. En esta comparación, las metaheurísticas basadas en población para resolver el FJSSP consideradas fueron:

I) hGA [64]: un algoritmo híbrido que combina la optimización por enjambre de partículas con un algoritmo genético.

II) BEDA [65]: un algoritmo de estimación de distribución basado en bi-poblaciones.

III) IACO [66]: una optimización basada en colonia de hormigas.

La Tabla 5.6 muestra que los valores de C_{max} obtenidos por el *HDE* son similares a los demás algoritmos para la mayoría de las diez instancias. Esta observación sugiere que el *HDE* propuesto en este trabajo es un algoritmo competitivo para resolver el FJSSP. Las comparaciones con respecto al esfuerzo computacional son difíciles de llevar a cabo porque la mayoría de los trabajos no informan el número de evaluaciones. En consecuencia, la eficiencia relativa de los algoritmos comparados es difícil de contrastar para obtener comparaciones significativas.

Tabla 5.6: Comparación entre *HDE* y metaheurísticas basadas en población de la literatura

	MK01	MK02	MK03	MK04	MK05	MK06	MK07	MK08	MK09	MK10
HDE	**40**	**26**	**204**	**60**	173	61	140	**523**	**307**	224
hGA	**40**	**26**	**204**	62	**172**	65	140	**523**	310	214
BEDA	**40**	**26**	**204**	**60**	**172**	60	**139**	**523**	**307**	206
IACO	**40**	**26**	**204**	**60**	173	60	140	**523**	**307**	208

Capítulo 6

Conclusión

En este trabajo se presentó un algoritmo *DE* simple para resolver el FJSSP. Se hizo una revisión del material bibliográfico relacionado con técnicas metaheurísticas que ayudan a la comprensión de la tesis. Se describe una introducción a los algoritmos metaheurísticos, luego se presenta una de las posibles calificaciones de las metaheurísticas: en métodos basados en trayectoria y en métodos basados en poblaciones, y por último se explican distintos conceptos de diseños para paralelizar las metaheurísticas. Así también se hizo una revisión de los conceptos de la evolución diferencial, y se presenta el algoritmo propuesto por Storn & Price, junto a sus 4 etapas: inicialización, mutación, recombinación y selección. Además se muestra el proceso iterativo del algoritmo *DE* en un formato de pseudocódigo.

A continuación, se presenta una descripción del problema a tratar, el diseño de la representación y evaluación de soluciones utilizado, la decodificación de soluciones, el procedimiento de búsqueda local utilizado, la forma en que fue realizada la paralelización para acelerar el cálculo de *DE* y la búsqueda local, y por último el pseudocódigo del algoritmo del *HDE* para FJSSP, el cual sigue el esquema general de *DE*, con la inclusión del proceso de búsqueda local. Además, se agrega como entrada la probabilidad de aplicar búsqueda local (P_{BL}), con el fin de permitir al investigador realizar cambios sobre el parámetro.

Finalmente, se describe el diseño experimental llevado a cabo para este trabajo donde fueron seleccionadas una amplia gama de instancias utilizadas en la literatura teniendo en cuenta su complejidad. Además, se analiza la calidad de los resultados considerando los valores de C_{max} obtenidos para las distintas mejoras planteadas al algoritmo *DE* descritas anteriormente para resolver las instancias de FJSSP. En primer

lugar se estudia el efecto de usar diferentes valores de F y Cr en el rendimiento del algoritmo, de valores bajos a altos (0.1, 0.5 y 0.9). Se observa que la combinación $F = 0.5$ y $Cr = 0.9$ supera a las demás desde el punto de vista de la calidad de los valores de C_{max} obtenidos. Luego se analiza el impacto de la incorporación del proceso de búsqueda local a través del parámetro P_{BL}, donde fueron considerados tres valores diferentes: 0.1, 0.5 y 0.7 (de valores bajos a altos). Los resultados indican que el algoritmo *HDE* con alta probabilidad de aplicar el procedimiento de búsqueda local, puede encontrar las mejores soluciones para el FJSSP. Posteriormente se evalúa el efecto en los tiempos al incorporar paralelismo a nivel de iteración. Los resultados muestran que el uso de la paralelización permite acelerar el tiempo de ejecución con respecto al *HDE* secuencial sobre todas las instancias en un valor de 3 veces en promedio. Por último, se presenta una comparación de los valores de C_{max} obtenidos por el *HDE* con los alcanzados por varios algoritmos competitivos presentes en la literatura destinados a resolver el FJSSP. Los valores de C_{max} obtenidos por el *HDE* son similares a los demás algoritmos para la mayoría de las diez instancias. Como consecuencia, el algoritmo *HDE* propuesto en este trabajo ofrece buenas soluciones a este problema NP-duro de una manera eficiente.

En lo personal, me agradó la actitud de investigar y explorar en lo "desconocido" y en hacer cosas no imaginadas en un inicio, acudiendo al aprendizaje autodidacta con libros, papers, preguntando y solicitando consejos a mi directora y al grupo LISI, e incluso aprendiendo del modo ensayo/error en el proyecto desarrollado.

Aprendí que es muy diferente el estudio en las aulas, al de guiarse de un libro, y aún mas al de investigar académicamente, buscando nuevas propuestas para resolver problemas que no son triviales, expandiendo las fronteras del conocimiento, reflexionando críticamente y adaptándose a los formatos académicos.

Por último, reconocer que mi tesis no habría sido posible sin el apoyo de mi familia, amigos, y en especial a mi directora Carolina como así también a todo el grupo de investigación, que me sumaron a su equipo con una gran amabilidad y cordialidad desde el día uno. A todos ellos que me apoyaron, me dieron ánimos, conversamos, criticaron, etc, muchas gracias.

Bibliografía

[1] R. L. Haupt and S. E. Haupt *Practical genetic algorithms.* John Wiley & Sons, 2004.

[2] T. Mitchell. *Machine Learning*. McGraw Hill, 1997

[3] P. Brandimarte, "Routing and scheduling in a flexible job shop by tabu search," *Annals of Operations Research*, vol. 41, p. 157183, 1993.

[4] M. R. Garey, D. S. Johnson, R. Sethi, The complexity of flowshop and jobshop scheduling, Math. Oper. Res., vol. 1, no. 2, 1976, pp. 117–129.

[5] M. R. Garey, D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness.* , 1979

[6] C. Blum, A. Roli, Metaheuristics in combinatorial optimization: Overview and conceptual comparison, ACM Computing Surveys 35 (3) (2003) 268– 308

[7] Y. Yuan, H. Xu *Flexible job shop scheduling using hybrid differential evolution algorithms*. Computers & Industrial Engineering, Volume 65, Issue 2, June 2013, Pages 246-260

[8] J. M. Apolloni *Propuesta de Metaheurísticas Híbridas para Mejorar el Análisis de Datos en Bioinformática (Ph.D. Tesis)*. Universidad Nacional de San Luis, 2017.

[9] E.-G. Talbi. A taxonomy of hybrid metaheuristics. *Journal of Heuristics,*. 8:541–564, 2002.

[10] B. Chapman and G. Jost and R. van der Pas, *Using OpenMP: Portable Shared Memory Parallel Programming.*. MIT Press, 2007

[11] M. Ali, W.A Zhu, Penalty function-based differential evolution algorithm for constrained global optimization, Comput. Optim. Appl., 54, 2013, pp. 707-739.

[12] J.-G. Juang, S.-T. Yu, Disturbance encountered landing system design based on sliding mode control with evolutionary computation and cerebellar model articulation controller, Appl. Math. Model., 39, 2015, pp. 5862-5881.

[13] Z. Yang, Q. Yu, W. Dong, X. Gu, W. Qiao, X. Liang, Structure control classification and optimization model of hollow carbon nanosphere core polymer particle based on improved differential evolution support vector machine, Appl. Math. Model., 37, 2013, pp. 7442-7451

[14] M. R. Garey, D. S. Johnson, & R. Sethi, The complexity of flowshop and jobshop scheduling. *Mathematics of Operations Research,*. 1, 117–129, 1976.

[15] E. Alba, *Parallel Metaheuristics: A New Class of Algorithms.* Wiley, 2005.

[16] C. Bierwirth (1995). A generalized permutation approach to job-shop scheduling with genetic algorithms. Operations Research Spektrum. Vol. 17 (2). pp 87/92.

[17] G. Luque and E. Alba, *Parallel Genetic Algorithms: Theory and Real World Applications.* Springer Publishing Company, Incorporated, 2013.

[18] D. E. Goldberg. *Genetic Algorithms in Search, Optimization and Machine Learning*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1st edition, 1989.

[19] J. García-Nieto and E. Alba. Hybrid PSO6 for hard continuous optimization. *Soft Computing*. pages 1–19, 2014

[20] M. Dorigo and G. Di Caro. *New ideas in optimization*. 1999.

[21] J. Brownlee. *Clever Algorithms: Nature-inspired programming recipes.* Jason Brownlee, 2011.

[22] J. García-Nieto *Emergent Optimization: Design and Applications in Telecomunications and Bioinformatics (Ph.D. Thesis).* University of Málaga, 2013.

[23] P. J. M. Van Laarhoven and E. H. L. Aarts *Simulated annealing.* Springer, 1987.

[24] H. R. Lourenço, O. C. Martin, and T. Stützle. *Iterated local search.* Springer, 2003.

[25] P. Hansen, N. Mladenovic, J. Brimberg, and J. A. M. Pérez. *Variable neighborhood search.* In Handbook of Metaheuristics, pages 61–86. Springer, 2010.

[26] M. Gendreau and J-Y. Potvin *Handbook of metaheuristics.* volume 2. Springer, 2010.

[27] K. V. Price, R. Storn, and J. Lampinen *Differential Evolution: A practical Approach to Global Optimization.* Springer-Verlag, London, UK, 2005.

[28] R. Storn and K. V. Price. Differential evolution: A simple and efficient adaptive scheme for global optimization over continuous spaces. Technical report, TR95012-ICSI, 1995.

[29] R. Storn, K. Price Differential evolution—a simple and efficient heuristic for global optimization over continuous spaces. *Journal of Global Optimization.* 1997;11(4):341–359.

[30] A. Colorni, M. Dorigo, V. Maniezzo Distributed optimization by ant colonies. Proceedings of the 1st European Conference on Artificial Life; 1991; Paris, France. pp. 134–142.

[31] R. Storn and K. V. Price. Minimizing the real functions of the ICEC'96 contest by differential evolution. In *International Conference on Evolutionary Computation*,pages 842–844, 1996.

[32] J. Kennedy, R. Eberhart, and Y. Shi. *Swarm Intelligence*. Morgan Kaufmann, San Francisco, CA, 2001.

[33] R. C. Eberhart, J. Kennedy New optimizer using particle swarm theory. Proceedings of the 6th International Symposium on Micro Machine and Human Science; October 1995. pp. 39–43.

[34] C. Blum and X. Li. *Swarm intelligence in optimization*. Springer, 2008.

[35] D. Dasgupta. *Artificial immune systems and their applications*. Springer Publishing Company, Incorporated, 2014.

[36] S. Kirkpatrick, M. P. Vecchi, et al. *Optimization by simmulated annealing.* Science, 220(4598):671–680, 1983.

[37] V. Cerny. Thermodynamical approach to the traveling salesman problem: An efficient simulation algorithm. *Journal of optimization theory and applications* , 45(1):41–51, 1985.

[38] N. Metropolis, A. Rosenbluth, M. Rosenbluth, A. Teller, and E. Teller. Equation of state calculations by fast computing machines. *Journal of Chemical Physics,*. 21:1087–1092, 1953.

[39] P. Hansen and N. Mladenovic. Variable neighborhood search: Principles and applications. *European Journal of Operational Research,* . 130(3):449–467, 2001.

[40] A.K. Qin and P.N. Suganthan. Self-adaptive differential evolution algorithm for numerical optimization. In *Proceedings of the 2005 IEEE Congress on Evolutionary Computation* . , volume 2, pages 1785–1791, 2005.

[41] J. Brest, S. Greiner, B. Boskovic, M. Mernik, and V. Zumer. Self-adapting control parameters in differential evolution: A comparative study on numerical benchmark problems. *s. IEEE Transactions on Evolutionary Computation,* . 10(6):646–657, 2006.

[42] J. H. Holland. Outline for a logical theory of adaptive systems. *Journal of the ACM*. 3:297–314, 1962.

[43] J. H. Holland. *Adaptation in Natural and Artificial Systems*. University of Michigan Press, Ann Arbor, MI, 1975.

[44] J. H. Holland *Adaptation in Natural and Artificial Systems: An Introductory Analysis with Applications to Biology, Control, and Artificial Intelligence*. MIT Press; 1992.

[45] I. Rechenberg. Cybernetic solution path of an experimental problem. Technical Report, Royal Aircraft Establishment Library Translation No. 1112, Farnborough, UK, 1965.

[46] I. Rechenberg. *Evolutionstrategie: Optimierung technischer systeme nach prinzipien der biologischen evolution.*. Frommann-Holzboog, 1973.

[47] J. R. Schott. Fault tolerant design using single and multicriteria genetic algorithm optimization. PhD thesis, Department of Aeronautics and Astronautics, Massachusetts Institute of Technology, Cambridge, MA, 1995.

[48] H-P. Schwefel. Kybernetische evolution als strategie der experimentellen forschung in der stromungstechnik. Technical report, Diplomarbeit Hermann Fottinger Institut fur Stromungstechnik, Technische universit at, Berlin, Germany, 1965.

[49] L. J. Fogel. Toward inductive inference automata. In *Proceedings of the International Federation for Information Processing Congress*. Munich, 1962, pp. 395–399.

[50] L. J. Fogel, A. J. Owens, and M. J. Walsh. *Artificial Intelligence Through Simulated Evolution.*. Wiley, 1966.

[51] J. R. Koza *Genetic Programming.*. MIT Press, Cambridge, MA, 1992.

[52] T. Back, D. B. Fogel, and Z. Michalewicz, editors. *Handbook of Evolutionary Computation.*. Oxford University Press, 1997.

[53] D. Corne, M. Dorigo, F. Glover, D. Dasgupta, P. Moscato, R. Poli, and K. V. Price. *New ideas in optimization.*. McGraw-Hill Ltd., UK, 1999.

[54] P. C3P Report. 826. Caltech Concurrent Computation Program; 1989. On evolution, search, optimization, genetic algorithms and martial arts: towards memetic algorithms.

[55] M. Dorigo, M. Birattari, and T. Stützle. Ant colony optimization. *Computational Intelligence Magazine*. IEEE, 1(4):28–39, 2006.

[56] D. Karaboga and B. Basturk. A powerful and efficient algorithm for numerical function optimization: artificial bee colony (abc) algorithm. *Journal of Global Optimization,* . 39(3):459–471, 2007.

[57] G. Rudolph. Convergence of evolutionary algorithms in general search spaces. *In Proceedings of the Third IEEE Conference on Evolutionary Computation*, 1996.

[58] A. Gnanavel Babu, J. Jerald, A. Noorul Haq , V. Muthu Luxmi, T. Vigneswaralu, Scheduling of machines and automated guided vehicles in FMS using differential evolution, Int. J. Prod. Res., 48 ,2010, pp. 4683-4699.

[59] V.V. De Melo, G.L. Carosio, Investigating multi-view differential evolution for solving constrained engineering design problems, Expert Syst. Appl., 40, 2013, pp. 3370-3377.

[60] D. Karaboga. An idea based on honey bee swarm for numerical optimization.. Technical Report TR06, Erciyes University, 2005.

[61] L. N. De Castro and J. Timmis. *Artificial immune systems: a new computational intelligence approach.*. Springer Science & Business Media, 2002.

[62] E-G Talbi *Metaheuristics: from design to implementation*. Wiley, Hoboken, 2009.

[63] C. Bermudez, G. Minetti and C. Salto, *Improving Artificial Bee Colony Algorithm with Evolutionary Operators* Facultad de Ingeniería, Universidad Nacional de La Pampa, CONICET, Argentina, 2017.

[64] J. Tang, G. Zhang, B. Lin and B. Zhang, "A hybrid algorithm for flexible job-shop scheduling problem", *Procedia Engineering*, vol 15, pp. 3678 - 3683, 2011.

[65] L. Wang, S. Wang, Y. Xu, G. Zhou and M. Liu, "A bi-population based estimation of distribution algorithm for the flexible job-shop scheduling problem", *Computers & Industrial Engineering* , vol 62, no. 4, pp. 917 - 926, 2012.

[66] L. Wang, J. Cai, M. Li and Z. Liu, "Flexible job-shop scheduling problem using an improved ant colony optimization", *Scientific Programming*, pp. 1 - 11, 2017.