UNIVERSIDAD AUTÓNOMA DEL ESTADO DE HIDALGO

INSTITUTO DE CIENCIAS BÁSICAS E INGENIERÍA

ÁREA ACADÉMICA DE INGENIERÍA

Desarrollo de un algoritmo híbrido basado en técnicas evolutivas y conceptos de autómatas celulares para minimizar el makespan en un problema de taller de flujo

T E S I S

QUE PARA OBTENER EL GRADO DE:

MAESTRO EN CIENCIAS EN INGENIERÍA INDUSTRIAL

P R E S E N T A:

ERICK SERAPIO MARTÍNEZ GÓMEZ

DIRECTOR DE TESIS: DR. JUAN CARLOS SECK TUOH MORA

CODIRECTOR: DR. JOSELITO MEDINA MARÍN

MINERAL DE LA REFORMA, HGO., ABRIL DE 2018

UNIVERSIDAD AUTÓNOMA DEL ESTADO DE HIDALGO
Instituto de Ciencias Básicas e Ingeniería
Institute of Basic Sciences and Engineering
Dirección
Dean

Mineral de la Reforma, Hidalgo, a 2 de abril del 2018

Número de control: ICBI-D/281/2018
Asunto: Autorización de impresión de tesis.

M. EN C. JULIO CÉSAR LEINES MEDÉCIGO
DIRECTOR DE ADMINISTRACIÓN ESCOLAR

Por este conducto, me permito comunicar a usted que, el jurado asignado al pasante de la Maestría en Ciencias en Ingeniería Industrial, **C. Erick Serapio Martínez Gómez**, con número de cuenta 245261, quien presenta el trabajo de tesis titulado "**Desarrollo e implementación de un algoritmo híbrido basado en técnicas evolutivas y conceptos de autómatas celulares para minimizar el makespan en un problema de taller de flujo**", después de revisar el trabajo en reunión de sinodales, ha decidido autorizar la impresión del mismo una vez realizadas las correcciones que fueron acordadas.

A continuación, se anotan las firmas de conformidad de los integrantes del jurado.

PRESIDENTE:	Dr. Norberto Hernández Romero
SECRETARIO:	Dr. Joselito Medina Marín
VOCAL:	Dr. Juan Carlos Seck Tuoh Mora
SUPLENTE:	Dra. Eva Selene Hernández Gress

Sin otro particular, reitero a Usted la seguridad de mi atenta consideración.

Atentamente
"Amor, Orden y Progreso"

Dr. Óscar Rodolfo Suárez Castillo
Director del Instituto de Ciencias Básicas e Ingeniería

UNIVERSIDAD AUTÓNOMA DEL ESTADO DE HIDALGO
ICBI
DIRECCIÓN

ORSC/MABC

Ciudad del Conocimiento
Carretera Pachuca - Tulancingo km. 4.5
Colonia Carboneras
Mineral de la Reforma, Hidalgo, México, C.P. 42184
Tel. +52 771 7172000 exts. 2231, Fax 2109
direccion_icbi@uaeh.edu.mx

www.uaeh.edu.mx

AGRADECIMIENTOS

A Dios por haberme prestado vida suficiente, por haberme guiado a lo largo de la Maestría, por acompañarme y ser mi fortaleza en los momentos de debilidad, por estar conmigo, por brindarme una vida llena de aprendizajes, experiencias y sobre todo felicidad.

A mis padres Rosa Gómez Badíllo y Anastacio Martínez García por ser el pilar fundamental en mi educación, por motivarme a seguir adelante en los momentos de desesperación, por creer en mí hasta el último momento, la motivación constante, comprensión y amor.

A mis hermanas Cristal Martínez Gómez y Yessica Rubi Martínez Gómez por estar conmigo y ser una parte muy significativa de mi vida, por los consejos en los momentos difíciles de la vida y apoyo en la unión familiar.

A mi director el Dr. Juan Carlos Seck Tuoh Mora por la acertada orientación durante las muchas asesorías en todas las etapas del trabajo, así como también por la fé que deposito en mi desde un comienzo, por la valiosa colaboración y aportación en mis dudas a lo largo del trabajo y por su valiosa amistad.

A mi codirector el Dr. Joselito Medina Marín por la paciencia desde mi inicio, por la dedicación de tiempo, por compartir sus conocimientos conmigo, sus anécdotas y sobre todo su valiosa amistad.

A mi novia la Psic. Edith Gutiérrez Solís por haberme apoyado incondicionalmente en las buenas y en las malas, sobre todo por su paciencia y amor.

Al Dr. Norberto Hernández Romero por la confianza al facilitarme la entrada al laboratorio de manufactura, su interés, así como en la revisión del presente trabajo.

A la Dra. Eva Selene Hernández Gress por sus valiosas observaciones, así como también por sus importantes sugerencias en el trabajo de investigación.

Al Coordinador del programa Maestría en Ciencias en Ingeniería Industrial, el Dr. Oscar Montaño Arango por el seguimiento de mi desempeño, atención y desarrollo de mi crecimiento personal en el programa.

Expreso mi gratitut de manera muy especial para el Consejo Nacional de Ciencia y Tecnología (CONACYT) por el apoyo brindado, con número de becario 23970.

ÍNDICE

Índice de Figuras

Índice de Tablas

Índice de Algoritmos

Resumen

Este trabajo está enfocado en proponer un algoritmo híbrido que dé solución al problema de taller de flujo mejor conocido como Flow Shop Scheduling Problem (FSSP). El propósito es encontrar una secuencia óptima (permutación válida de todas las tareas) para llevar a cabo un conjunto de trabajos que son procesados en la misma secuencia en un conjunto de máquinas o recursos compartidos. Este requiere de un período de tiempo ininterrumpido para su ejecución, y el objetivo es minimizar el valor de la función costo, o makespan.

Esta tarea es lograda mediante la adaptación de los modelos de optimización por cúmulo de partículas por sus siglas en inglés Particle Swarm Optimization (PSO) y evolución diferencial llamado también Differential Evolution (DE) que son parte de la familia de los Algoritmos Evolutivos (AE). En el FSSP una permutación es un esquema de representación de individuos para evaluar el valor objetivo (aptitud). Adicionalmente, conceptos de autómatas celulares conocidos como Cellular Automata (CA) son para dar una versión celular al algoritmo híbrido propuesto de los dos modelos combinados autocontenidos, logrando una estructura celular variable. La ejecución del algoritmo es mejorada al convertir su implementación computacional a lenguaje máquina. El algoritmo es probado en diversas instancias de E. Taillard del tipo FSSP, consiguiendo una ejecución en un tiempo computacional aceptable con la mejor combinación de parámetros de control para la generación de buenas soluciones. El algoritmo propuesto es capaz de calcular el valor óptimo conocido en más ocasiones que los modelos mencionados anteriormente.

Abstract

This work is focused on proposing a hybrid algorithm to solve the Flow Shop Scheduling Problem (FSSP). The aim is to find an optimal sequence (or permutation) to carry out a set of jobs that are processed in the same sequence in a set of machines or shared resources. This requires an uninterrupted period of time for its execution, and the goal is to minimize the value of the cost function, or makespan.

This task is achieved through the adaptation of Particle Swarm Optimization (PSO) and Differential Evolution (DE) models, which are part of the family of Evolutionary Algorithms (EA). In FSSP, a permutation is a representation scheme of individuals to evaluate the objective value (aptitude). In addition, Cellular Automata (CA) concepts are used to give a cellular version for the proposed hybrid algorithm of the two combined models mentioned above, attaining a variable cell structure. The execution of the algorithm is improved when converting its computational implementation to machine language. The algorithm is tested in various instances of E. Taillard of FSSPs, obtaining a suitable execution time with the best combination of control parameters for the generation of good solutions. The proposed algorithm is able to calculate the known optimum value in more occasions than the previous models.

Introducción

Antecedentes

En secuenciación de tareas un problema de taller de flujo, conocido como Flow Shop Scheduling Problem (FSSP) ha sido un tema de gran importancia en la investigación de operaciones (IO), donde el objetivo es determinar el orden de ejecución de las tareas que serán llevadas a cabo de manera secuencial en las máquinas, dentro de un proceso de producción en una industria en general (Dorronsoro, 2006), lo que permite optimizar una función objetivo por excelencia como lo es el tiempo máximo requerido para completar todas las tareas (makespan), minimizando los tiempos muertos o maximizando la utilización de las máquinas (Sánchez & López, 2005; Martínez, 2015).

> "Las técnicas de CE[1] son métodos de inteligencia computacional que emulan el proceso de evolución natural de los seres vivos, con el objetivo de resolver problemas de optimización, búsqueda y aprendizaje, inicialmente propuestas en la década de 1970, las técnicas de computación evolutiva se consolidaron como estrategias eficientes de resolución de problemas en la década de 1990 y hoy en día constituyen una de las alternativas más utilizadas para abordar problemas de optimización complejos" (Nesmachnow, 2014, párr.3)".

Las implementaciones informáticas que se rigen por estos modelos se conocen como AE. Dorronsoro (2006) afirma que un AE es un proceso iterativo que trabaja sobre un conjunto de individuos que componen una población, a los que se les aplica una serie de operadores de variación que permiten a la población de individuos evolucionar y mejorar.

---

1 La computación evolutiva (CE) es un campo de investigación emergente que provee nuevas metaheurísticas para la resolución de problemas de optimización donde los enfoques tradicionales hacen al problema computacionalmente intratable. Reflejando la relevancia industrial de estos problemas, se han reportado en la literatura una variedad de métodos basados en AE para la resolución de problemas de Scheduling (Pandolfi, Villagra y Leguizamon, 2009). Los resultados reportados mostraron que un enfoque evolutivo se comportaba sensiblemente mejor para problemas de gran tamaño, produciendo muy buenas soluciones y de forma rápida.

> Los AE son herramientas realmente útiles para la resolución de problemas complejos ya que trabajan rápidamente en grandes (y complejos) espacios de búsqueda gracias a que, al funcionar sobre una población de individuos, son capaces de operar sobre varias soluciones simultáneamente. Por tanto, los AE pueden realizar varios caminos de búsqueda distintos a la vez. (Dorronsoro, 2006, p.25).

Changsheng y Jigui (2008) mostraron cómo se pueden modificar las ecuaciones iniciales de posición y velocidad en el PSO para adaptarlas a resolver el FSSP, con el objetivo de minimizar el makespan. De manera similar Čičková y Ševo (2010) mencionan que Differential Evolution (DE) conocido como Evolución Diferencial, siendo otra clase de técnicas evolutivas, es un método que funciona extremadamente bien en una amplia variedad de problemas.

Por otro lado, desde que el concepto de los Autómatas Celulares (CA) fue propuesto en los años 40's por Von Neumann y Ulam, éste se ha convertido en una poderosa herramienta para modelar y analizar sistemas discretos.

Alba y Tomassini (2002) afirman que, tradicionalmente, la mayoría de los AE trabajan sobre una única población, aunque existe una reciente tendencia que consiste en estructurar la población mediante la definición de algún criterio de vecindad entre las soluciones intermedias manejadas por el algoritmo.

Justificación

Desde que el FSSP fue abordado por primera vez, a mediados de los años 50´s, siendo uno de los campos más investigados, debido a la gran dificultad que supone obtener dicha secuencia y la importancia de conseguirla, múltiples investigadores han intentado desarrollar modelos y estrategias para darle solución. Tanto las estrategias como los modelos han evolucionado a través del tiempo, convirtiéndose en la actualidad en un reto para las técnicas computacionales (Véles & Montoya, 2007). Así la optimización se ha convertido en un activo, pero desafiante campo (Yang *et al*., 2008).

Con la llegada de los sistemas computacionales, un problema de secuenciación de tareas puede ser trabajado con herramientas informáticas para la programación de corto plazo, observando su comportamiento, mejorando los procesos de secuenciación, en especial en aquellos procesos productivos donde se cuenta con múltiples trabajos y múltiples tareas que son desarrolladas por múltiples departamentos (Sánchez & López, 2005).

El interés ha aumentado por parte de algunos investigadores que han implementado algoritmos basados en el modelo de CE, debido a su robustez y amplia aplicabilidad. Esto ha establecido una aproximación para resolver el problema de buscar valores óptimos, esto mediante el uso de modelos computacionales basados en AE, como un intento de resolver tareas de optimización en ordenadores secuenciales. Esto aprovecha las características de las grandes y potentes redes de computadoras existentes en la actualidad. (Alba, Giacobini, Tomassini y Romero, 2002; Alba & Troya, 2000; Back, Fogel y Dorronsoro, 2006; Michalewicz, 1997).

> "Además que el comportamiento que muestra un AE al resolver los problemas, viene determinado por el equilibrio que mantiene al manipular las soluciones de una población, entre la exploración del campo de búsqueda y la explotación local de las mejores soluciones" (Dorronsoro, 2006, p.1).

Un punto en contra de los primeros AE es que sufren el problema de convergencia prematura y su solución fácilmente es atrapada en un óptimo local, principalmente por un decremento de la diversidad de sus individuos (Giacobini, Tomassini, Tettamanzi y Alba, 2005; Dorronsoro, 2006).

Con base en lo anterior surge la idea de mejorar a este tipo de algoritmos, para contrarrestar sus deficiencias, siendo una buena opción considerar un modelo híbrido.

Ya que estos modelos tratan de obtener la ventaja de un reducido costo computacional, obtenido con los métodos de optimización local, y la posibilidad de encontrar un óptimo global (Pandolfi *et al*., 2009).

La forma de la población es un factor determinante en el comportamiento de un AE, hasta el punto que cambiar la forma de la población puede llevar a mejorar el comportamiento numérico del algoritmo, mucho mejor que en el caso de cambiar la política de actualización del individuo o el tipo de selección aplicado (Pandolfi *et al*., 2009).

Dorronsoro (2006) investigó que las estructuras celulares simulan la evolución natural desde el punto de vista del individuo, logrando algoritmos cada vez más eficientes para resolver problemas complejos (tanto problemas de optimización como de búsqueda). Además, él sostiene que el efecto perseguido de estructurar la población consiste en mantener de la diversidad de soluciones durante más tiempo, mejorando de esta manera la capacidad de exploración del algoritmo en el espacio de búsqueda, mientras que la explotación puede también ser reforzada a veces muy fácilmente.

Objetivos

Objetivo General

Desarrollar, implementar y evaluar el desempeño de un algoritmo híbrido de estructura celular variable para minimizar el valor del makespan en un problema de taller de flujo.

Objetivos específicos

1. Analizar modelos y estrategias de solución al FSSP dentro del campo de los sistemas computacionales, entendiendo sus ventajas y limitantes.
2. Desarrollar un modelo de PSO que replique un proceso de cruce de atracción y repulsión de partículas para evaluar el makespan.
3. Desarrollar un modelo en DE que replique el proceso de transformación directa de variables enteras a continuas y el proceso de transformación inversa de variables continuas a enteras para evaluar el makespan.
4. Optimizar los modelos de PSO y DE a través de Matlab, para reducir el tiempo computacional requerido en la ejecución.
5. Utilizar Matlab para ejecutar simultáneamente (de forma paralela) en múltiples trabajadores (múltiples datos) el modelo optimizado, aplicado a los modelos de PSO y DE.
6. Investigar acerca del funcionamiento del modelo celular en los AEC y la teoría de modelos celulares para proveer una población de individuos con una estructura espacial, definida como un grafo conectado, en el que cada vértice es un individuo que se comunica con sus vecinos más cercanos.
7. Realizar un análisis paramétrico al modelo de PSO y DE, efectuando cambios continuos en los parámetros para determinar cómo cambia el valor del makespan, buscando la mejor combinación de parámetros que más se acerque a la solución óptima en un tiempo computacional adecuado.

Hipótesis

Es factible desarrollar e implementar un algoritmo hibrido AEC de optimización, mediante las metaheurísticas PSO, DE, y la teoría de AC, que pueda ser utilizado en un conjunto de problemas del tipo FSSP, para determinar un orden de ejecución de tareas y minimizar el valor del makespan.

Planteamiento del problema

Pandolfi *et al*. (2009) afirma que para llevar a cabo la minimización del makespan en la secuenciación de tareas para el tipo FSSP, nos encontramos con un problema de optimización combinatoria (POC) del tipo NP-Hard, para el que no se conoce un algoritmo exacto que proporcione una solución óptima en un tiempo computacional razonable, salvo para ejemplares del problema de dimensiones muy reducidas.

Cuando se utiliza un algoritmo de optimización basado en población para resolver un problema complejo, dos de los factores más significativos que afectan la capacidad del algoritmo son: (1) Estructura de comunicación, y (2) mecanismos de herencia y difusión de información. A partir de esto, se propone mejorar la relación entre intensificación (exploración) y diversificación (explotación), teniendo como objetivo desarrollar e implementar un algoritmo híbrido, que emplearía el modelo del PSO logrando una exploración más eficiente en el espacio de búsqueda, al igual que el un modelo celular aplicado al modelo DE, buscando una mejor solución explotando de las soluciones encontradas e influyendo en la cooperación de la población y en la auto-adaptación de los individuos. Esto a través de generar estructuras variables de individuos, habilitándolos para que sólo puedan interactuar e intercambiar información con sus vecinos más próximos dentro de los vecindarios de la población, así logrará mejorar sus propiedades y comportamiento del algoritmo. Realizando el menor esfuerzo computacional; superando así los algoritmos existentes y encontrar la solución óptima, dado que son algoritmos especialmente adaptados para la optimización (Dorronsoro, 2006, p.88).

Organización de la Tesis

La tesis se desarrolla a lo largo de los siguientes capítulos con la siguiente estructura:

- En el capítulo 1 se desarrolla una introducción a la programación de la producción, una breve definición de POC para exponer el problema de secuenciación de tareas del tipo FSSP, comprender su complejidad, la función costo de interés y así como distintos métodos para darle solución con sus respectivas ventajas y deficiencias.

- En el capítulo 2 se describe como funciona un AE que forma parte de las metaheurísticas basadas en la población, se presentan sus ventajas y deficiencias y además de una introducción del concepto de métodos híbridos mostrando una posible clasificación de estos.

- En el capítulo 3 se explica como funciona la metaheurística PSO, sus principales elementos, características y factores significativas que afectan su comportamiento, al igual que sus ventajas y deficiencias.

- En el capítulo 4 presenta el funcionamiento de la metaheurística DE que aplica un método de transformación de variables enteras a continuas en una representación vectorial interna para lograr una adaptación al modelo original, el funcionamiento de sus proceso evolutivo, su modelo canónico, mostrando ventajas y deficiencias, además de una breve explicación de su funcionamiento.

- En el capítulo 5 adicionalmente se menciona como funciona un CA aplicado a un AE, se explica como se estructura una población de soluciones en una malla toroidal, mencionando de igual manera la clasificación de los tipos de vecindarios en los AEC, los tipos de aplicación al ciclo reproductor, como se puede utilizar para su funcionamiento como modelo algorítmico además de sus ventajas y deficiencias.

- En el capítulo 6 se explica como se lleva a cabo la metodología de hibridación de un algoritmo adaptado de PSO para resolver el FSSP utilizada de base para la metodología mencionada en el capítulo 4 del algoritmo DE, que formará el híbrido AECA y el funcionamiento de su estructura, además se exponen los resultados obtenidos por la implementación computacional.

- En el capítulo 7 finalmente se muestran los comentarios finales de la comparación de los algoritmos mencionados anteriormente, mostrando características, cualidades, criterios de evaluación, una comparación de los modelos en función costo, conclusiones y trabajo futuro.

Capítulo 1

Programación de la producción

Desde hace mucho tiempo, uno de los mayores problemas que se presenta en la administración de operaciones de cualquier empresa, ya sea en el área de manufactura como en el área de servicios es precisamente la programación secuencial de actividades de los diferentes trabajos propuestos por los clientes (Ballesteros & Duque, 2013). Una empresa manufacturera debe planificar sus operaciones en distintos niveles y operar cada uno de dichos niveles, buscando la mejor eficacia y efectividad para todo el sistema en su conjunto (Osorio & Motoa, 2008).

En el marco de la dirección de producción y operaciones, la programación de la producción es una etapa clave en la gestión de operaciones, ya que en un corto plazo se toman las decisiones de carácter operativo, que buscan el cumplimiento de las metas propuestas mediante la asignación de los recursos disponibles a las tareas de producción requeridas. La programación de la producción ha sido tema de análisis por parte de los administradores y programadores de la producción (López, 2013; Wang & Liu, 2013).

El problema de la programación de la producción reviste un carácter complejo dada la cantidad de elementos que involucra y las múltiples interrelaciones existentes entre ellos; esto ha hecho que en torno a la solución del mismo se hallan desarrollado gran cantidad de trabajos (Osorio & Motoa, 2008).

1.1 Problemas de optimización combinatoria (POC)

De acuerdo a Pasteels, Deneubourg y Goss (1987) y Arito (2010) en los POC se busca un orden específico sobre un conjunto finito de variables discretas (o posiblemente de un conjunto infinitamente contable). Aarts y Lenstra (2003) y Sait y Youssef (1999) afirman que este orden específico de variables discretas que se desea encontrar, puede ser un número entero, un subconjunto de ellos, una permutación, o una estructura de grafo.

Los POC están presentes en diversos campos como la economía, el comercio, la ingeniería, la industria o la medicina. Sin embargo, a menudo estos problemas son muy difíciles de resolver en la práctica. El estudio de esta dificultad inherente para resolverlos, tiene lugar en el campo de la teoría de las ciencias de la computación, ya que muchos de ellos pertenecen a la clase de

problemas Np-Hard, problemas para los cuales no se conoce o tal vez no exista un algoritmo determinístico que los resuelva en tiempo polinomial (Arito, 2010; Garey & Johnson, 1979; Martínez, 2015).

Véles y Montoya (2007) mencionan que:

> Si el número de operaciones necesarias para que un algoritmo resuelva el problema es una función polinomial del tamaño del problema, el problema es fácil. Si esta función no es polinomial, se dice que el algoritmo es no polinomial y el problema se considera difícil. Aunque esta clasificación parezca sólo de interés teórico, en la práctica resulta de gran importancia. Saber si un problema se puede resolver en tiempo polinomial equivale a saber si es posible encontrar la solución óptima en unos cuantos segundos o minutos o si, por el contrario, son necesarios años o incluso siglos para hacerlo. En el último caso surge la necesidad de desarrollar estrategias para encontrar soluciones buenas a un costo computacional razonable. (p.154).

Los POC se han convertido en tema de investigación de muchos expertos, por lo que muchos investigadores han propuesto soluciones empleando diversos métodos de optimización, que van desde la aplicación de métodos exactos, y algoritmos aproximados como las heurísticas, y metaheurísticas, utilizando técnicas manuales o haciendo uso de software especializados (Ballesteros & Duque, 2013, p.39).

1.2 Secuenciación de tareas

Pandolfi *et al*. (2009) comenta que la secuenciación de tareas es un problema clásico, paradigma de la familia de POC y de satisfacción de restricciones de precedencia o tecnológicas que, debido a su presencia en el mundo real, ha despertado mucho interés entre los investigadores, ya que puede clasificarse como una multitud de tareas cotidianas ejecutadas durante períodos de tiempo determinados, que van desde la planificación, diseño de productos o servicios, así como la explotación y mantenimiento de sistemas, plantas y equipos (Galzina, Lujić y Šarić, 2012). Además, son computacionalmente complejos NP-Hard (para un número de máquinas mayor o igual a tres), ya que el tiempo requerido para calcular una solución óptima crece exponencialmente con el tamaño del problema (número de trabajos a asignar en los recursos de cómputo). (Martínez, 2015; Morton & Pentico, 1993; Nesmachnow, 2014; Pinedo, 1995).

Los recursos y trabajos en una organización pueden adoptar muchas formas diferentes. Los recursos pueden ser máquinas en un taller, pistas de aterrizaje en un aeropuerto, equipos en un sitio de construcción, las unidades de procesamiento en un entorno de computación, y así sucesivamente (Ballesteros & Duque, 2013). Los trabajos pueden ser operaciones en un proceso de producción, los despegues y aterrizajes en un aeropuerto, las etapas de un proyecto de

construcción, las ejecuciones de programas de ordenador, entre otros. Cada trabajo puede tener un nivel de prioridad determinado, una partida lo antes posible, tiempo y una fecha de vencimiento.

1.2.1 Tipología de secuenciación de tareas

En este tipo de problemas se desea encontrar un orden específico en el que las tareas deben ser efectuadas optimizando el tiempo de ejecución de la secuencia de producción total, lo que quiere decir que se busca tener las máquinas trabajando el mayor tiempo posible (Rodríguez, 2014).

Dependiendo del uso de los recursos por parte de las actividades, podemos distinguir tres tipos de rutas que pueden seguir los trabajos (Alfonso & Barber, 2001; González-Fernández, 2011; Martínez, 2015).

a. Open Shop Scheduling Problem (OSSP): Cada una de las tareas de un trabajo se procesará sobre una máquina diferente, no existiendo relaciones de precedencia entre las operaciones, es decir no existe ninguna restricción en cuanto al orden de uso de los recursos por las actividades de cada uno de los trabajos. En caso que la máquina esté ocupada, existe la posibilidad de que dicha tarea se dirija a otra máquina, debido a la prioridad que puede tener dicha tarea sobre las demás (Rodríguez, 2014). El problema consiste en encontrar órdenes de los trabajos (ordenaciones de las operaciones pertenecientes al mismo trabajo) y órdenes de las máquinas (ordenaciones de las operaciones que han de ser procesadas en la misma máquina) que optimicen una cierta función objetivo (Martínez, 2015, p.30).
b. Job-Shop Scheduling Problem (JSSP): Cada trabajo, o conjunto de actividades, debe usar los recursos en un orden determinado (restricciones tecnológicas), en este caso también se añade una restricción de precedencia, y es que las tareas dentro de un determinado trabajo tienen un cierto orden y deben ejecutarse de forma secuencial. Los trabajos deben ser procesados, una sola vez, por recursos, con un orden, y durante un tiempo fijo dado. En estos problemas de secuenciación bajo ambientes JSSP con n trabajos y m máquinas, el número total de posibles soluciones es $(n!)^m$, si se tienen en cuenta todas las posibles alternativas, incluso las soluciones no realizables, por limitaciones físicas (Castrillón, Giraldo y Sarache, 2009; Yang *et al*., 2008). El problema consiste en encontrar un orden de tareas para cada máquina que optimice una cierta función objetivo.
c. Flow Shop Scheduling Problem (FSSP): En este caso existen restricciones de precedencia (un predecesor y un sucesor), y consisten en que las tareas de todos los trabajos deben utilizar las máquinas en orden secuencial (se considera que todas las máquinas están disponibles en un principio), es decir, que cada tarea del trabajo $j = 1,..., n$, se debe ejecutar en la máquina 1, la segunda tarea en la máquina 2, y así sucesivamente hasta la máquina m. Es un caso particular del Job-Shop. (Martínez, 2015, p.30; Rodríguez, 2014).

1.3 Flow Shop Scheduling Problem (FSSP)

Medina, Gradišar, Seck Tuoh, Hernandez y Nuñez (2016) definen que el FSSP consiste en encontrar la secuencia óptima (secuencia de trabajo válida de todas las tareas) para llevar a cabo un conjunto de trabajos multi-operación no relacionados (divididos en un conjunto de varias tareas llamadas operaciones) que son procesados en la misma secuencia en un conjunto de máquinas o recursos compartidos, y requiere de un período de tiempo ininterrumpido para su ejecución. Teniendo como consecuencia que algunas tareas puedan llegar a máquinas ocupadas, lo que genera colas de espera para ser procesadas. (Bárcena Diez, 2011; Martínez, 2015).

1.3.1 Descripción del problema

En el FSSP, dado el tiempo de procesamiento p_{jk} para cada trabajo j en cada máquina k, $1 <= j <= n$, $1 <= k <= m$, (Donde el tiempo de preparación y de cambio de operación ya están incluidos). Una secuencia de trabajos $S = (s_1, s_2, ..., s_n)$ es generada, donde un conjunto de n trabajos $(j = 1,..., n)$ serán procesados por un conjunto de m máquinas $(k = 1,..., m)$, así que el objetivo es encontrar el orden de una secuencia para el procesamiento de operaciones con el mínimo valor del makespan, Además que la configuración del FSSP tiene $n!$ posibles secuencias de trabajo y crece exponencialmente conforme se incrementa el número de tareas. (Medina, Hernández, Seck Tuoh y Martínez, 2016); Pérez, Jöns, Hernández y Ochoa, 2014).

Debido a que el problema de FSSP es esencialmente un problema de secuenciamiento de permutaciones, una secuencia de trabajo (permutación) puede ser usada como un esquema de representación de individuos, lo cual representa una forma natural de expresar la solución del problema (Pandolfi & Leguizamon, 2009; Ríos y Bard, 2010.).

> "La secuencia puede ser de producción lineal, donde una vez que se fabrica cierta cantidad de artículos de un tipo de producto, se preparan las máquinas para fabricar el otro producto, también existen secuencias de producción continua, donde las máquinas siempre están trabajando en la producción del mismo producto" (Garcia, 2008; Rodríguez, 2014, párr 6.)".

Sánchez y López (2005) explican que a menudo se producen conflictos entre diferentes trabajos que emplean los mismos recursos, debido a algunas restricciones tecnológicas del FSSP.

1.3.2 Restricciones

Fonseca, Martínez, Figueredo y Pernía (2014) y Sayoti y Essaid (2016) explican que el FSSP está sujeto a las siguientes restricciones que deben ser satisfechas:

- Solamente se cuenta con una máquina-herramienta de cada tipo (Ej.: un torno, un taladro, una fresadora, una rectificadora cilíndrica, etc.).
- No está permitido que dos tareas del mismo trabajo se procesen simultáneamente, ya que las relaciones de procedencia tienen que ser respetadas.
- Ningún trabajo debe ser procesado al mismo tiempo en la misma máquina.
- Ningún trabajo puede ser procesado más de una vez en la misma máquina.
- Cada trabajo es procesado hasta concluirse, una vez que se inicia una operación esta se interrumpe solamente cuando se concluye.
- La secuencia tecnológica dada para las máquinas es idéntica para todos los trabajos, y asume que la secuencia de los trabajos en las máquinas es también la misma.
- Todos los trabajos son independientes y están listos para su procesamiento en el tiempo cero.
- El makespan de todos los trabajos debe ser minimizado.

Estas restricciones producen conflictos y la solución consiste en decidir, para cada conflicto entre tareas y recursos, cuál de las tareas debe ejecutarse antes, si asumimos que cada tarea está disponible inicialmente y no existe un tiempo límite de entrega (Pandolfi *et al*., 2009).

1.3.3 Funciones objetivo para el FSSP

Chase, Aquilano y Jacobs (2000), al igual que López (2013) explican que los objetivos pueden adoptar formas diferentes. Un objetivo puede ser, la minimización del criterio de optimización objetivo por excelencia, más estudiada en los trabajos publicados y se le conoce con el nombre de makespan.

La minimización del tiempo que las tareas terminan tras su tiempo de fin límite (criterio conocido como minimización del tardiness, o del weighted tardiness si asignamos una determinada prioridad a cada tarea). Otras posibles funciones objetivo pueden ser el maximum lateness (en donde se intenta minimizar el máximo incumplimiento de tiempo de fin límite).Por último, el total flow time (que consistirá en minimizar la suma de los tiempos de finalización de todos los trabajos), (González, 2011, p.120).

1.3.4 Minimización del makespan

Hojjati y Sahraeyan (2009) y López (2013) mencionan que el makespan o C_{max} (como también se conoce en la literatura), es el tiempo máximo de finalización de todos los trabajos y este depende de el orden en que los trabajos sean relizados (Medina *et al*., 2016a).

Vallada y Ruiz (2011) consideran que minimizar el makespan equivale a maximizar la utilización de las máquinas, ya que se busca disminuir los tiempos de alistamiento y los tiempos de ocio de las máquinas. Hekmatfar, Fatemi-Ghomi y Karimi (2011) nos dicen que es un método efectivo para eliminar demoras y tardanzas en los trabajos, también lleva a la reducción del inventario en proceso, y minimiza los desórdenes de la planta por los trabajos no completados.

1.3.5 Análisis y representación

Por ejemplo, la tabla 1 muestra el tiempo de procesamiento correspondiente al tiempo de operación de cuatro trabajos y tres máquinas, donde cada trabajo tiene tres operaciones seriales, uno para cada máquina, tomado de (Medina *et al*., 2016b).

Tabla 1. Tiempo de procesamiento de cuatro trabajos y tres máquinas.

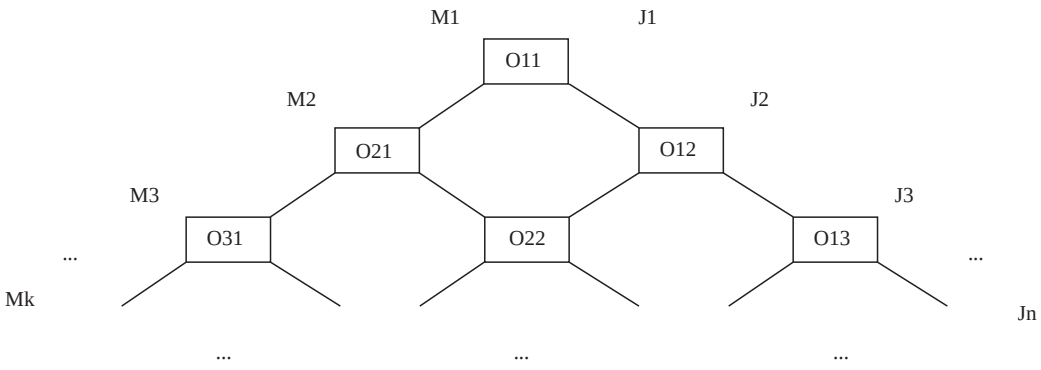
		Número serial de operaciones			
Artículos	*Jobs*	J_1	J_2	J_3	J_4
Número serial de operaciones	M_1	96	74	13	71
	M_2	90	57	5	23
	M_2	35	91	7	38

Elaboración propia basada en (Medina *et al*., 2016b).

> "La primera operación del primer trabajo no tiene dependencias de otras operaciones y comienza inmediatamente; sin embargo, las operaciones restantes dependen de la operación previa en la misma máquina, la operación previa en el mismo trabajo, o ambos" (Medina *et al*., 2016b, p.3)".

Lo anterior se muestra en la figura 1.

Figura 1. Dependencia de operaciones de trabajo de operaciones previas del mismo trabajo, y/o operaciones previas ejecutadas en la misma máquina.



Elaboración propia basada en (Medina *et al*., 2016b).

La figura 1 muestra la única dependencia entre las operaciones y el tiempo de procesamiento acumulado en caa operación, que representa el tiempo de procesamiento para todas las operaciones del primer trabajo, o la suma para las operaciones procesadas en la primera máquina.

Ya que el tamaño de la configuración del FSSP es *n!,* en este caso *n* tiene un valor de cuatro que representa el número de trabajos y el tamaño de posibles combinaciones válidas tiene un valor de 24. Para mostrar un ejemplo se presentan solo se presentan cuatro secuencias de trabajo seleccionadas aleatoriamente.

La tabla 2, 3. 4 y 5 muestran la secuencia escogida aleatoriamente, al igual que el cálculo del valor del makespan.

Tabla 2. Cálculo del makespan para la secuencia 1 3 2 4.

96	183	109	354
186	248	191	277
221	339	228	377

Valor de makespan M=377 unidades de tiempo.

Tabla 3. Cálculo del makespan para la secuencia 4 2 3 1.

254	145	158	71
244	202	207	94
379	293	300	132

Valor de makespan M=379 unidades de tiempo.

Tabla 4. Cálculo del makespan para la secuencia 3 2 1 4.

183	87	13	254
273	144	18	296
308	325	25	346

Valor de makespan M=346 unidades de tiempo.

Tabla 5. Cálculo del makespan para la secuencia 2 1 4 3.

170	74	254	241
260	131	288	283
295	222	340	333

Valor de makespan M=340 unidades de tiempo.

Elaboración propia basada en secuencias elegidas de manera aleatoria cuyo fin sólo es comparativo.

De acuerdo a los resultados obtenidos al ejecutar dichas secuencias de trabajo, la mejor secuencia de trabajo es la última 2 1 4 3, con un valor de makespan de 340 unidades de tiempo, esto demuestra que pudimos encontrar una buena solución, pero no podemos asegurar que es la óptima, ya que se necesitaría analizar todas las posibles combinaciones válidas y para problemas más grandes sería muy complicado encontrar su solución.

El pseudocódigo del algoritmo utilizado para resolver el FSSP se muestra a continuación.

Algoritmo 1. FSSP

```
Entradas: pop[i].x[iter], Np, t
Salidas: C_max(Gbest), Gbest, C_max pop[i].fitness
Begin
indice=0;
C_max pop[i].fitness= [0; 0; … 0] Np x n;
 for i=1 to Np
     Sec=P(i,:);
     C_max pop[i].fitness= calmakespan (Sec, t);
      if  pop[i].fitness[iter+1] < C_max(Gbest)
          C_max(Gbest)= pop[i].fitness[iter+1];
          indice=i;
      end
       Gbest= pop[i].x[iter] (indice, :);
 end
```

1.4 Solución al problema del FSSP

Matemáticamente, el problema del FSSP consiste en determinar un programa de producción factible, teniendo en cuenta las relaciones de precedencia y las restricciones impuestas en los trabajos a ser ejecutados en las máquinas, para poder observar tanto los tiempos de inicio como los de terminación de las diferentes operaciones relacionadas con los trabajos asociados, y que muestre una buena solución con uso de equipo de cómputo en tiempos razonables, establecidas como valor objetivo. (Hejazi & Saghafian, 2005; Véles & Montoya, 2007).

Martin (1992) al igual que Salazar (2009) y Ballesteros y Duque (2013) comentan que, debido a la gran importancia de los problemas de optimización a lo largo de la historia de la matemática aplicada, se han desarrollado múltiples métodos para tratar de resolverlos teniendo en cuenta el grado de su complejidad. El problema del tipo FSSP ha sido analizado aplicando diferentes tipos de técnicas, o métodos de optimización, ya que no pueden ser usualmente resueltos con métodos exactos. Por tal motivo, un esfuerzo significativo de investigación se ha enfocado en la aplicación de técnicas de computación inteligente, incluyendo Computación Evolutiva, Redes Neuronales, Lógica Difusa, Enjambres de Partículas, Evolución Diferencial y sus hibridaciones (Pecero & Reyes, 2014).

1.4.1 Métodos exactos

Salazar (2009) resalta la importancia de que los métodos garantizan encontrar la solución óptima para cualquier instancia de cualquier problema en un tiempo acotado. El inconveniente de estos métodos es que el tiempo necesario para llevarlos a cabo, aunque acotado, crece exponencialmente con el tamaño del problema. Esto provoca en muchos casos que el tiempo necesario para la solución del problema sea inabordable.

Generalmente, los métodos exactos necesitan tiempos exponenciales de computación cuando tratamos con instancias grandes de problemas complejos, ya que requieren un número exponencial de operaciones, lo cual la hace poco aconsejable. (Sánchez & López, 2005). De manera muy clara, los problemas NP-Hard no pueden abordarse de forma realista con técnicas exactas (Dorronsoro, 2006).

Estos algoritmos, entre los que se encuentran la programación dinámica y el algoritmo de ramificación y poda, implican un costo computacional en ocasiones tan elevado que hace que deban descartarse como alternativa de solución (Salazar, 2009, p.10).

1.4.1.1 Ramificación y Poda

Es utilizado para resolver problemas de optimización discreta, al igual que el diseño Vuelta Atrás, realiza una enumeración parcial del espacio de soluciones basándose en la generación de un árbol de expansión, cuya solución se expresa como una secuencia de decisiones y en cada decisión se elige entre un conjunto finito de valores. Este método no utiliza recursividad para generar el árbol y el árbol existe como una cola de nodos vivos.

Cuberos (2015) afirma que básicamente, en un algoritmo de Ramificación y Poda se realizan tres etapas:

La primera de ellas, Selección, se encarga de extraer un nodo de entre el conjunto de los nodos vivos. La forma de escogerlo va a depender directamente de la estrategia de búsqueda que decidamos para el algoritmo. En la segunda, la Ramificación, se construyen los posibles nodos hijos del nodo seleccionado en el paso anterior. Por último, la Poda, en la que se eliminan algunos de los nodos creados en la etapa anterior.

Esto contribuye a disminuir en lo posible el espacio de búsqueda y así atenuar la complejidad de estos algoritmos basados en la exploración de un árbol de posibilidades. Aquellos nodos no podados pasan a formar parte del conjunto de nodos vivos, y se comienza de nuevo por el proceso de selección. El algoritmo finaliza cuando encuentra la solución, o bien cuando se agota el conjunto de nodos vivos.

1.4.2.1 Ventajas y deficiencias

Dorronsoro (2006) enfatiza que la principal ventaja de la utilización de algoritmos exactos es que garantizan encontrar el óptimo global de cualquier problema, pero tienen el grave inconveniente de que en problemas reales (NP- Hard) su tiempo de ejecución crece de forma exponencial con el tamaño de la instancia, el cual no lo hace un método adecuado, así que su idoneidad queda descartada.

1.4.2 Métodos Heurísticos

Arito (2010) define que la concepción más común en Inteligencia Artificial, es interpretar que, heurístico es el calificativo apropiado para los procedimientos que, empleando conocimiento acerca de un problema y de las técnicas aplicables, tratan de aportar soluciones (o acercarse a ellas) usando una cantidad razonable de recursos. En un problema de optimización, aparte de las condiciones que deben cumplir las soluciones factibles del problema, en este se busca la que es óptima según algún criterio de comparación entre ellas.

> "Éstos se basan en reglas lógicas extraídas de la experiencia del usuario, para generar soluciones válidas en tiempos razonables. Aunque generalmente las soluciones así generadas son bastantes buenas, ningún heurístico puede garantizar la obtención del óptimo para el problema considerado. Además, la mayoría de estos algoritmos son difícilmente adaptables a problemas ligeramente diferentes a aquéllos para los que fueron concebidos" (Sánchez & López, 2005, p 5.).

Salazar (2009) asegura que los métodos heurísticos sacrifican la garantía de encontrar el óptimo a cambio de encontrar una buena solución en un tiempo razonable, suelen ser los métodos más rápidos, ya que generan una solución partiendo de una vacía a la que se le va añadiendo componentes hasta tener una solución competa, que es el resultado del algoritmo. Las soluciones ofrecidas son de muy baja calidad y encontrar métodos de esta clase que produzcan buenas soluciones es muy difícil ya que dependen mucho del problema, para su planteamiento se debe tener un conocimiento muy extenso del mismo.

1.4.2.1 NEH

Nawaz, Enscore Jr y Ham (1983) afirman que es un procedimiento constructivo de secuenciación muy eficiente que proporciona buenas secuencias para el problema del FSSP con el objetivo de minimizar el instante máximo de finalización de los trabajos.

Como es bien conocido, este procedimiento se puede resumir en dos fases:

- En la primera los trabajos se ordenan según un valor indicador y en la segunda, conocida como fase de inserción, se construye progresivamente la secuencia, incorporando cada trabajo, de uno en uno, en el orden previamente establecido, colocándolo en la mejor posición posible de la secuencia parcial generada, con el fin de minimizar el makespan parcial.
- Una vez colocado un trabajo, la posición relativa del mismo respecto a los demás trabajos ya considerados no se altera. (Companys & Ribas, 2012).

Deacuerdo a la segunda fase para la minimización del makespan, esta se resume en 4 sencillos pasos:

I. Ordenar los tiempos de trabajo, mediante sumas del tiempo de procesamiento en las máquinas.
II. Tomar los dos primeros trabajos y programarlos, con el fin de minimizar el makespan parcial, como si solo existieran esos dos trabajos.
III. Para $k=3$ a n, hacer el paso 4.
IV. Insertar el *k-ésimo* trabajo en el lugar, que minimiza el makespan parcial entre los k posibles.

1.4.2.2 Desventajas y deficiencias

Los heurísticos son fáciles en su implementación, sin embargo, las técnicas matemáticas consumen demasiado tiempo computacional, no garantizan encontrar el óptimo global, además no son capaces de producir soluciones de buena calidad para problemas de tamaño grande y requieren muchos supuestos que son difíciles de satisfacer en sistemas de manufactura reales. Además, son algoritmos muy complejos de definir para muchos problemas, debido a la exploración de soluciones no exhaustiva (Dorronsoro, 2006; Pérez, Sanchez, Hernández y Ochoa, 2014).

1.4.3 Métodos Metaheurísticos

Aarts y Lenstra (2003) afirman que debido a que la mayoría de los POC se clasifican como difíciles. Blum y Roli (2003) así como Martí (2003) aseguran que la investigación se ha concentrado en desarrollar algoritmos de aproximación. En las últimas décadas Véles y Montoya (2007) comentan que han emergido estrategias de alto nivel que planifican de manera estructurada la aplicación de varias operaciones para explorar espacios de búsqueda de elevada dimensión y complejidad intrínseca. Éstas consisten básicamente en la combinación de métodos heurísticos clásicos con estrategias eficientes de exploración, de una forma eficiente y efectiva. Estos métodos son comúnmente conocidos con el término metaheurísticas (Dorronsoro, 2006; Osman & Kelly, 1996).

> "Las metaheurísticas son métodos aproximados diseñados para resolver problemas de optimización combinatoria, en los que los heurísticos clásicos no son efectivos. Las metaheurísticas proporcionan un marco general para crear nuevos algoritmos híbridos, combinando diferentes conceptos derivados de la inteligencia artificial, la evolución biológica y los mecanismos estadísticos" (Véles & Montoya, 2007, párr. 7.).

Véles y Montoya (2007) combinan las metaheurísticas de forma diferente, de manera que dos conceptos clave para el desarrollo de un buen algoritmo de búsqueda son el equilibrio (generalmente dinámico) entre: Intensificación y diversificación. Este equilibrio entre estos dos aspectos contrapuestos es de gran importancia, ya que por un lado deben identificarse rápidamente las regiones prometedoras del espacio de búsqueda global y por otro lado no se debe malgastar tiempo en las regiones que ya han sido exploradas o que no contienen soluciones de alta calidad (Chicano, 2007). Estos métodos sacrifican la garantía de encontrar el óptimo a cambio de encontrar una "buena" solución en un tiempo razonable.

1.4.3.1 Algoritmos Genéticos (AG)

Los AG son técnicas de búsqueda y optimización basados en los principios de la genética y selección natural. Los AG están compuestos de un conjunto de individuos y diferentes tipos de reglas que aplican directamente sobre los individuos (Goldberg, 1989, p.38).

La población es un conjunto de individuos y cada individuo se representa por una cadena de símbolos, letras y/o números. La manera de generar estos individuos es de forma totalmente aleatoria, después, esta cadena es evaluada en una función costo con la finalidad de obtener una calificación para cada individuo, mediante la calificación otorgada al individuo se ordena y selecciona la población para obtener un conjunto con los individuos mejor calificados. Posteriormente, mediante el operador de cruce, seleccionamos de forma aleatoria los segmentos de información que van a compartir para generar nuevos individuos, cuando se termina de hacer el cruce de todos los individuos tenemos una nueva población que deberá tener una mejor calificación en la siguiente evaluación. Sin embargo, antes de someterlo a una nueva evaluación es necesario mutar de forma aleatoria algunos individuos de la población, la mutación consiste en seleccionar de forma aleatoria algunos individuos y posteriormente, en su cadena de genes, cambiar la información. (Hernández et al., 2014).

Hernández *et al*. (2014) menciona que:

Como todo algoritmo de solución, tiene sus pros y contras, sin embargo, la justificación para aplicar estos algoritmos a la solución de problemas en ingeniería es que han tenido éxito en aquellas aplicaciones en donde las herramientas matemáticas de optimización han fracasado.

A continuación, se enumeran las ventajas y desventajas de los AG.

Ventajas:

- **Implementación computacional**. Las operaciones computacionales para realizar un AG es a través de operaciones aritméticas, lógicas y de ordenamiento sencillas que son fáciles de implementar.
- **Información previa**. Los AG no necesitan información a priori del sistema que se desea optimizar. El AG genera múltiples soluciones de forma aleatoria, y si algunas de ellas mejoran, entonces son soluciones factibles que se tomarán en cuenta para evolucionar la población.
- **Optimización de sistemas no lineales**. El AG realiza una amplia exploración en el rango de búsqueda sin importar que el sistema sea de funciones discontinuas que no tengan derivada o que sea no convexo.
- **Paralelismo**. Los AG por su naturaleza son factibles de implementarse en clusters de cómputo paralelo, de forma tal que el tiempo de cómputo para evaluar la evolución de un individuo es el mismo que para evaluar toda la población.

Desventajas:

- **Función costo**. La única forma para evaluar el desempeño de las evoluciones en el AG es a través de una función de evaluación, o función-costo. En ciertas aplicaciones no es sencillo establecer una función-costo para optimizar un sistema multivariable.
- **Reglas**. No existen reglas para determinar el número de individuos en una población, qué tipo de selección aplicar, o cómo realizar la mutación.
- **Aplicaciones**. Los AG tienen una amplia gama de aplicaciones donde han logrado optimizar sistemas con éxito, pero esto no quiere decir que se puedan utilizar en cualquier aplicación y que logren un mejor desempeño que los métodos analíticos matemáticos.
- **Programación serie**. Cuando los AG genéticos se programan en plataformas de procesamiento serie, no son tan rápidos en su tiempo de ejecución y por lo tanto es difícil implementarlos en aplicaciones en línea.
- **Convergencia prematura**. Es probable que un individuo con un alto desempeño propague su código genético desde el arranque de la simulación, de tal forma que se pierde diversidad en la población y el AG cae un punto óptimo local.

1.4.3.2 Desventajas y deficiencias

Dorronsoro (2006) menciona que la principal ventaja de la utilización de las metaheurísticas es que muestran un buen resultado a los problemas NP-Hard en un tiempo computacional razonable,

lo cual lo hace adecuado para resolver este tipo de problemas, ya que además su implementación es muy sencilla y nos acerca a un mejor resultado que un heurístico simple.

La principal desventaja de las metaheurísticas al igual que los heurísticos, es que no garantizan encontrar el óptimo global, aunque puede ser posible su obtención, teniendo en cuenta esto, se considera idónea su adecuación.

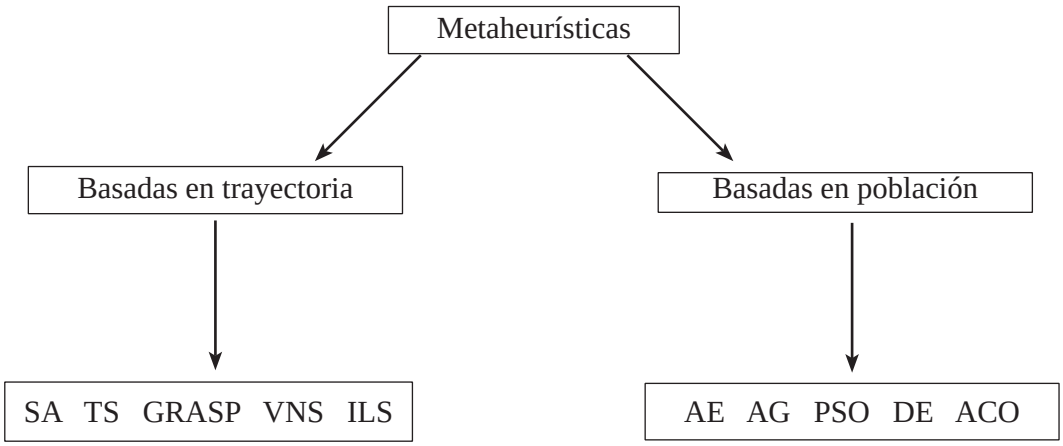
Capítulo 2

Clasificación de las Metaheurísticas

Las metaheurísticas pueden clasificarse de diferentes formas. Blum y Roli (2003), Chicano (2007) y Dorronsoro (2006) dan una clasificación que depende de un conjunto de características seleccionadas que las diferencian.

Dependiendo de las características que se seleccionen se pueden obtener diferentes taxonomías: Inspiradas en la naturaleza o no, con función objetivo estática o dinámica, con una o varias estructuras de vecindario, con memoria o sin ella y una de las clasificaciones más populares las divide en metaheurísticas basadas en trayectoria y basadas en población.

Figura 2. Clasificación de las Metaheurísticas.



Elaboración propia basada en (Chicano, 2007).

2.1 Metaheurísticas basadas en trayectoria

Chicano (2007) menciona que la principal característica de estos métodos es que parten de una solución y, mediante la exploración del vecindario, van actualizando la solución actual, formando una trayectoria. Mencionados en la figura 2 por sus siglas en inglés, se encuentra el recocido simulado (SA), la búsqueda tabú (TS), el procedimiento de búsqueda miope aleatorio y adaptativo (GRASP), la búsqueda con vecindario variable (VNS) y la búsqueda local iterada (ILS).

La mayoría de estos algoritmos surgen como extensiones de los métodos de búsqueda local simples, a los que se les añade algún mecanismo para escapar de los mínimos locales. Esto implica la necesidad de una condición de parada más elaborada que la de encontrar un mínimo local. Normalmente se termina la búsqueda cuando se alcanza un número máximo predefinido de iteraciones, se encuentra una solución con calidad aceptable, o se detecta un estancamiento del proceso.

2.2 Metaheurísticas basadas en población

Las metaheurísticas basadas en poblaciones son métodos que van construyendo un conjunto de individuos que representan el conjunto de soluciones (población) (Chicano, 2007). El procedimiento consiste en generar, seleccionar, combinar y reemplazar dicho conjunto de soluciones, en cada iteración, a diferencia de los métodos basados en trayectoria, que únicamente manipulan una solución del espacio de búsqueda por iteración. Su eficiencia y resultado depende fundamentalmente de la forma con la que se manipula la población en cada iteración.

Ya que dentro del presente trabajo se abordará el tema de Algoritmos Evolutivos como base fundamental del principio del algoritmo híbrido propuesto, se explicará con mayor detalle a continuación.

2.2.1 Algoritmos Evolutivos (AE)

Los AE son técnicas de optimización que trabajan sobre poblaciones de soluciones y que están diseñadas para buscar valores óptimos en espacios complejos, mediante el uso de modelos computacionales basados en procesos evolutivos (Dorronsoro, 2006; Pandolfi & Villagra, 2008). Conforman una de las metaheurísticas poblacionales, más ampliamente difundidas y estudiadas, probadamente eficientes en problemas de optimización, ya que realizan un buen balance entre una exploración del espacio de búsqueda y la explotación de sub- esquemas que se encuentran codificados dentro las soluciones.

En otras palabras, un AE es una técnica iterativa de aplicación probabilística de operadores de variación, como la recombinación de dos individuos, la aplicación de cambios aleatorios (mutación) en su contenido, y la utilización de una técnica de selección de soluciones que emula a la selección natural, respectivamente implementados sobre un conjunto de individuos.

Cada individuo en la población representa una solución candidata al problema a resolver, y tiene asociado una función costo que representa la adecuación de la solución para resolver el problema y se relaciona con la(s) función(es) objetivo considerada(s) en la variante del problema.

2.2.2 Principio de funcionamiento

Inicialmente la población se genera de forma aleatoria, o aplicando una heurística específica (y simple) para resolver el problema, esto introduce normalmente una gran cantidad de diversidad al comienzo del algoritmo. Tras la generación de la población inicial se calcula la función costo de cada uno de los individuos que la forman. Un AE procede de forma iterativa mediante la evolución de los individuos pertenecientes a la población actual.

Dorronsoro (2006) afirma que la reproducción sexual permite el intercambio del material genético de los cromosomas, produciendo así descendientes que contienen una combinación de la información genética de sus padres. Este es el operador de recombinación utilizado en los AE, algunas veces llamado operador de cruce debido a la forma en que los biólogos han observado a las cadenas de los cromosomas cruzándose entre sí.

Estrategias específicas se utilizan para seleccionar los individuos a recombinar (el operador de selección) y para determinar qué individuos se insertan en la población luego de aplicar los operadores evolutivos (el operador de reemplazo).

La recombinación ocurre en un entorno en el que la selección de los individuos que tienen que emparejarse es principalmente una función de la función costo del individuo, es decir, que tan bueno es el individuo comparado con los de su entorno.

La importancia de la mutación, que introduce aún más diversidad mientras el algoritmo se ejecuta, es objeto de debate. Algunos se refieren a la mutación como un operador de segundo plano, que simplemente reemplaza parte de la diversidad original que se halla podido perder a lo largo de la evolución, mientras que otros ven a la mutación como el operador que juega el papel principal del proceso evolutivo (Dorronsoro, 2006).

La condición de parada o criterio de terminación del AE usualmente involucra un número determinado de iteraciones (programado previamente) del algoritmo, un tiempo límite de ejecución, o encontrar la solución óptima al problema (o una aproximación a la misma) en caso de que se conozca de antemano (Dorronsoro, 2006), o la detección de una condición de convergencia. (Nesmachnow, 2014).

> "Finalmente, el AE retorna el mejor individuo (solución) encontrado en el proceso, tomando en cuenta la función costo considerada (Nesmachnow, 2014, p 21.)".

Parte de la evolución está determinada por la selección natural de individuos diferentes compitiendo por recursos en su entorno. Por tanto, algunos individuos son mejores que otros. Es deseable que aquellos individuos que son mejores sobrevivan y propaguen su material

genético. (Dorronsoro, 2006). La idea clave detrás de un AE es manipular o evolucionar el conjunto de soluciones tentativas del problema, guiando la búsqueda hacia mejores soluciones (potencialmente, hacia la solución óptima).

2.2.3 Ventajas y deficiencias

Dorronsoro (2006) afirma que, en la actualidad, el campo de los AE se encuentra en pleno crecimiento y evolución. Prueba de ello son las nuevas familias de AE que han surgido recientemente, como el PSO, DE, los algoritmos de estimación de distribuciones (AED), o la búsqueda dispersa (BD). (Laguna & Martí, 2003; Dorronsoro, 2006).

Nesmachnow (2014) sostiene que una de las principales características que propician el buen comportamiento de los AE es la lenta difusión de las mejores soluciones por la población, por lo que se mantiene la diversidad de soluciones entre los individuos por más tiempo, así se supera significativamente a las soluciones calculadas con heurísticas tradicionales, permitiendo hacer planificaciones de gran calidad en tiempos de ejecución reducidos. Siendo uno de los principales problemas el hecho que produce una lenta convergencia del algoritmo hacia el óptimo, disminuyendo de esta manera la eficiencia del algoritmo, lo cual representa su deficiencia.

Los AC han sido ampliamente estudiados en la literatura, pero no la posibilidad de importar las ideas existentes en el campo de los AC a los AE. Los AC son sistemas dinámicos formados por un conjunto de celdas que pueden estar en varios estados predefinidos. Estas celdas evolucionan en función de los estados de las celdas vecinas según un conjunto de reglas establecidas.

2.3 Métodos híbridos

Sedano (2013) aporta que los enfoques híbridos tratan de hacer frente a la complejidad de los fenómenos del mundo real, con un enfoque multidisciplinario y una pluralidad de técnicas, cobrando cada vez más importancia dentro del campo de las metaheurísticas. El objetivo de combinar técnicas es resolver problemas que un único método clásico no es capaz de resolver, tal es el caso de los sistemas complejos en biología, medicina, administración, ingeniería y redes sociales, entre otros.

Las metaheurísticas híbridas consisten en combinar dos o más algoritmos, diferentes metaheurísticas y de métodos de otros campos de la metaheurísticas. Para obtener sistemas que aprovechen más las ventajas de las estrategias individuales para conseguir un mayor beneficio, que por separado (sinergia). La combinación de estrategias que permitan la reducción de la complejidad del problema, un reducido costo computacional, el mejoramiento de las soluciones y la posibilidad de encontrar un óptimo global proporcionado por los métodos globales; son los enfoques más usados por los autores para hacer sus métodos competitivos, incorporando diferentes mecanismos con el objeto de mejorar la velocidad de convergencia y la calidad de los resultados. (Mercado, Pandolfi y Villagra, 2013; Talbi, 2009).

2.3.1 Clasificación de métodos híbridos

Talbi (2009) presenta una taxonomía de metaheurísticas híbridas y propone dos clasificaciones para este tipo de métodos: jerarquizadas y plana.

Las diferentes hibridaciones de metaheurísticas pueden clasificarse de un modo jerárquico en:

Combinación de bajo nivel: Los procedimientos heurísticos están embebidos unos en otros, de tal forma que para dos procedimientos dados una función del procedimiento continente se sustituye por el procedimiento embebido. Esta combinación se divide en:

- Serie: Un método se introduce dentro de otro como una función.
- Paralelo: Se tiene una población de soluciones de tal forma que sobre cada solución actúa un método que caracteriza por contener a otro método.

Combinación de alto nivel: los métodos heurísticos se combinan están autocontenidos, de forma que no existe interacción entre ellos. Se dividen en:

- Serie: Se tiene una única solución de tal forma que un método se aplica después del otro.
- Paralelo: Se tiene una población de soluciones de forma que cada método se aplica independientemente a cada solución.

Además, las metaheurísticas hibridas pueden organizarse en una clasificación plana de la siguiente manera:

Homogéneas o Heterogéneas

- Homogéneas: Todos los algoritmos combinados utilizan la misma metaheurística.
- Heterogéneas: Los algoritmos combinados utilizan diferentes metaheurísticas.

Globales o parciales

- Globales: En las hibridaciones globales, todos los algoritmos buscan en todo el espacio de búsqueda. El objetivo es entonces explorar el espacio más minuciosamente.
- Parciales: En las hibridaciones parciales el problema se descompone en subproblemas, cada uno definido en su propio espacio de búsqueda. Cada uno de los algoritmos se dedica a explorar uno de esos sub-espacios.

Los subproblemas están relacionados entre sí a través de restricciones entre las soluciones encontradas en cada uno. Por lo tanto, los algoritmos se comunican entre ellos para respetar estas restricciones y construir una solución global factible.

Especializados o generales

- Especializados: Se combinan algoritmos resuelven diferentes problemas de optimización.
- Generales: Todos los algoritmos resuelven el mismo problema de optimización.

Capítulo 3

Optimización por enjambre de partículas (PSO)

Morales Viscaya (2012) menciona que en 1995 Particle Swarm Optimization fue desarrollado por Kennedy y Eberhart, conocido como optimización por enjambre de partículas (PSO). Considerado como un AE, es un algoritmo de búsqueda usado como técnica basada en poblaciones, que se encuentra más en uso en diversas áreas de la ingeniería, con una gran potencialidad de aplicaciones, debido a que su implementación es muy sencilla, ya que utiliza pocos parámetros de control y sobre todo a que converge muy rápido a buenas soluciones, incluso con problemas no tan sencillos. Esta ha sido introducida como una técnica de optimización en espacios de números reales.

Este tipo de algoritmos se inspiraron en el comportamiento social de las bandadas de aves y su medio de intercambio de información para resolver problemas de optimización. En estas asociaciones existe un líder que condiciona el desplazamiento de la bandada. Además, cada individuo se guía basándose en su propio conocimiento (experiencia previa); en concreto, la toma de decisión por parte de cada individuo o partícula se realiza teniendo en cuenta un componente social, un componente individual, y su medio ambiente, mediante los cuales se determina el movimiento de esta partícula para alcanzar una nueva posición. (Changsheng & Jigui, 2008; Cagnina, Esquivel y Gallard, 2009).

3.1 Principio de funcionamiento

En el PSO el sistema es inicializado con una población de soluciones aleatorias y busca la solución óptima actualizando iteraciones (generaciones). Cada solución potencial dentro del espacio de búsqueda (equivalente a los individuos en los AG) es llamada una "partícula". Cada particula vuela en el espacio dimensional del problema con una velocidad que es dinámicamente ajustada de acuerdo a su propia experiencia de vuelo y a la de sus colegas. Por esta razón, las partículas están inclinadas a volar hacia una mejor área de búsqueda a lo largo de la evolución.

A medida que el algoritmo avanza en cada iteración o ciclo de vuelo, las partículas empiezan a concentrarse en zonas con soluciones de buena calidad del espacio de búsqueda, además la función objetivo es evaluada por cada partícula con respecto a su posición actual, indicando la función costo o la calidad de la misma. En cada ciclo de vuelo se actualizan las partículas considerando dos valores: p_{best}, el mejor valor alcanzado por la partícula hasta ese momento, y g_{best} el mejor valor encontrado por toda la población, al finalizar, el algoritmo devuelve la mejor solución visitada por algún individuo del enjambre. (Castillo *et al.*, 2014).

Figura 3. Principio de funcionamiento PSO.

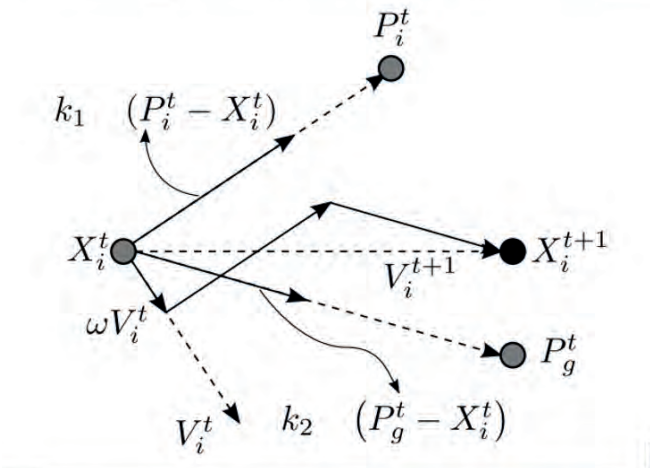


Figura tomada de (Martínez, Moreno, Cruz, y Seck- Tuoh, 2011).

3.2 Enjambre y partícula

Un enjambre es el conjunto de varias partículas que vuelan por el espacio de búsqueda de soluciones del problema durante la ejecución del algoritmo, que van desde la primera hasta el tamaño completo del enjambre.

Cada partícula es considerada como una estructura de datos que en su forma más simple consiste de tres vectores:

- ◊ Un vector posición x_i que representa la solución actual candidata de la partícula en el espacio de búsqueda, dentro del problema de optimización.
- ◊ Un vector velocidad v_i que almacena la dirección y magnitud hacia donde se moverá la partícula, además regula el movimiento en la posición de la partícula.
- ◊ Y un vector memoria p_i que almacena la localización de la mejor solución candidata encontrada por la partícula hasta el momento p_{best}.

La posición de la partícula es actualizada por:

Y su velocidad de acuerdo a

$$x^{(t+1)} = x_i^t + v_i^{(t+1)} \quad (1)$$

$$v_i^{(t+1)} = W * v_i^t + k_1\left(p_i^t - x_i^t\right) + k_2\left(p_g^t - x_i^t\right) \quad (2)$$

Donde k_1 y k_2 son números aleatorios uniformemente distribuidos que determinan el peso entre la posición de atracción p_i^t y p_g^t que es la mejor posición global encontrada por todas las partículas g_{best}.

Castillo, Mora, Rincón, y Ponsich, (2014) afirman que la calidad de la posición de cada partícula es determinada por una función costo, acorde con el problema a resolver, y que el movimiento de cada partícula sigue siendo influenciado por la mejor posición visitada por la partícula p_{best} y la mejor posición visitada por algún individuo del enjambre g_{best}.

3.2.1 Características de comportamiento (exploración y explotación)

Para promover una exploración amplia del espacio de búsqueda, las posiciones y velocidades iniciales asignadas a cada partícula son generadas de forma aleatoria. Conforme avanza el algoritmo, la velocidad y la posición cambian en función de la interacción social basada en la tendencia social de cada individuo a emular el éxito de otros individuos en la población. Esto es resultado de un fenómeno emergente denominado inteligencia de partículas (Castillo *et al*., 2014).

La relación entre la explotación de buenas soluciones (potenciada en la fase de selección) y la exploración de nuevas zonas del espacio de búsqueda (que se generan en la fase de evolución) que aplican los AE sobre el espacio de búsqueda es uno de los factores clave de su alto rendimiento con respecto a otras metaheurísticas. Además, esta relación entre exploración y explotación se puede mejorar de forma importante estructurando la población, lo que nos permite realizar una búsqueda descentralizada (Dorronsoro, 2006). También es posible afinar esta relación entre exploración y explotación con otros parámetros del algoritmo, como por ejemplo los operadores de variación utilizados, o la probabilidad de aplicarlos.

3.3 Factores significativos que afectan su capacidad

Yang *et al*. (2008) comenta que:

Cuando se utiliza algoritmo de optimización basado en la población para resolver problemas complejos, dos de los factores más importantes afectan significativamente la capacidad de los algoritmos son:

1. Estructuras de comunicación, y
2. Mecanismos de herencia y difusión de información.

Los dos factores podrían influir en la cooperación de la población y la auto adaptación del individuo, y afectar aún más el rendimiento de los algoritmos.

Además, afirma que cuando una bandada de aves sale a buscar alimento, dos simples e importantes estrategias son:

a. Buscar la región periférica alrededor del ave más cercana a la comida;
b. Juzgar la posición de la comida por su propia experiencia de vuelo.

Inspirada por las dos estrategias, el espacio de búsqueda del problema de optimización es considerado como un espacio de vuelo de las aves; cada ave es extraída como una partícula, para denotar una solución candidata; y la solución óptima a ser buscada para el problema es la comida para todas las partículas.

3.3.1 Robustez

Ya que dichos procedimientos no sólo deben proporcionar una solución factible, sino también robusta frente a las variaciones externas que puedan afectar a su rendimiento. Morales Viscaya (2012) afirma que el concepto de robustez de una solución o programa se refiere generalmente a su habilidad para afrontar perturbaciones aleatorias que ocurran en el tiempo de ejecución y permanecer aceptable (con un deterioro mínimo en su función objetivo). En la práctica, debido a las condiciones de trabajo dinámicas e inciertas, los planes difícilmente se corresponden con las previsiones y deben modificarse a menudo a causa de sucesos aleatorios (por ejemplo, una avería en una máquina o un trabajo urgente que debe insertarse inmediatamente). De este modo, cuanto más robusto es un plan o una secuenciación de tareas, más sencillo es reprogramarlo, por lo que resulta conveniente incorporar reglas de robustez en el propio desarrollo del método de resolución.

3.3.2 Ventajas y deficiencias

PSO tiene las ventajas de tener pocos parámetros de control, poder utilizar funciones costo más versátiles, pudiéndose implementar sin mayor complicación funciones no diferenciables, no lineales y/o discontinuas. Además, en sus diferentes variantes y/o modificaciones es una buena alternativa de solución comparada con técnicas como AG para problemas de optimización global con múltiples máximos y mínimos, discontinuidades y con soluciones determinísticas en tiempo no polinomial. (Hau, 2012; Sarmiento, 2012).

Ordóñez (2008) comenta que, pese a que la primera versión de PSO puede solucionar satisfactoriamente los problemas de optimización sencillos, al aplicarlo para problemas difíciles con mayor espacio de búsqueda, se presentan varias deficiencias, además de una rápida y prematura convergencia en los puntos medios óptimos y tiende a tener una explotación de soluciones débil.

Capítulo 4

Evolución Diferencial (DE)

Rammohan, Ponnuthurai, Pan y Mehmet (2011) redactan que Differential Evolution (DE) conocido como Evolución Diferencial, pertenece a la clase de técnicas evolutivas, ya que es un método de optimización propuesto por (Storn & Price, 1997), es un optimizador global estocástico, inspirado en la observación de los procesos de la naturaleza, diseñado originalmente para trabajar con variables continuas.

Involucra una búsqueda de una población de individuos, para cada iteración *g*, que difiere de otros AE por su mecanismo de generación de descendencia, ya que en DE una descendencia es generada usando diferencias vectoriales entre individuos de la población. (Čiková & Števo, 2010 ; Noroozi, Hashemi y Meybodi, 2011; Onwubolu & Babu, 2004; Onwubolu & Davendra, 2006).

4.1 Principio de funcionamiento

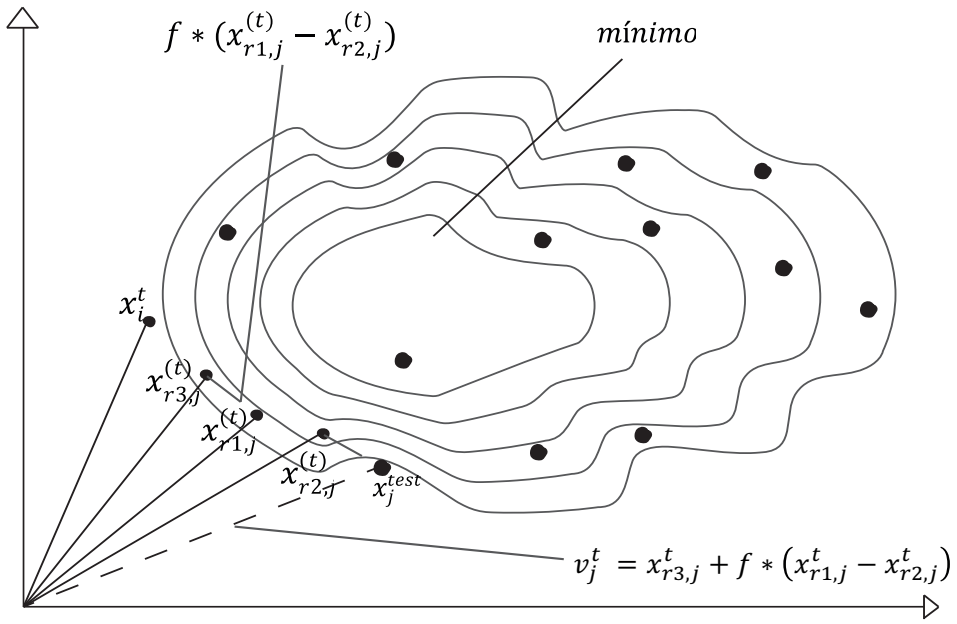
El modelo clásico DE trabaja a través de un ciclo de operadores de variación incluyendo mutación, cruce y operadores de selección después de la inicialización.

Čiková y Števo (2010), Noroozi, Hashemi y Meybodi (2011) y Onwubolu y Babu (2004) establecen que:

En DE cada individuo x_i, $i=1,2, ..., Np$, es representado como un vector real n-dimencional. Partiendo de una población P de Np individuos aleatoriamente inicializados al azar, en los rangos de búsqueda. La estrategia básica emplea la diferencia de dos vectores de parámetros (individuos) seleccionados aleatoriamente como el origen de variación aleatoria para un tercer individuo. Suponiendo una distribución de probabilidad uniforme, para generar nuevos individuos, agregando un vector diferencial, entre dos miembros de la población a un tercer miembro. Si el vector resultante arroja un valor de la función objetivo menor que un individuo determinado de la población, el vector recién generado reemplaza el vector con el que se comparó.

Un ejemplo bidimensional, que ilustra los diferentes vectores que forman una estrategia de trabajo es mostrado en la figura 4.

Figura 4. Principio de funcionamiento DE.



Elaboración propia basada en (Onwubolu, 2004).

4.2 Inicialización

Al igual que con todos los algoritmos de optimización evolutiva, DE trabaja con una población de soluciones, inicializada al comienzo del proceso evolutivo. Usualmente la inicialización aleatoria se utiliza para que cada individuo represente una solución candidata al problema dado. Cada individuo es evaluado con su valor de la función objetivo. (Čiková & Števo, 2010).

4.2.1 Evaluación

El proceso entero de reproducción continúa hasta que el último número de iteración (especificado por el usuario) es alcanzado. El valor del mejor individuo de cada iteración es reflejado a un vector historia, que muestra el progreso del proceso evolutivo.

4.2.2 Condición de parada

En su forma canónica, el único criterio de parada es al alcanzar el número máximo de iteraciones (representado por el parámetro *g*).

4.3 Mutación

La primera variante en DE funciona de la siguiente manera: para cada vector $v_{i,j}^t$ un vector ensayo *v* es generado de acuerdo a

$$v_j^t = x_{r3,j}^t + f * \left(x_{r1,j}^t - x_{r2,j}^t\right) \tag{3}$$

Donde:

$$i = 1,2 \dots Np, \quad j = 1,2 \dots d, \quad t = 1,2 \dots g$$

$$r_1, r_2, r_3 \in \{1,2 \dots Np - 1\}; \quad r_1 \neq r_2 \neq r_3 \neq i$$

Para cada individuo de la población otros tres diferentes individuos son escogidos (vectores r_1, r_2, r_3), que se refieren a tres vectores escogidos aleatoriamente de la población, y son asignados para cada valor del índice *i*. Ellos son mutuamente diferentes uno del otro y también diferentes del índice de ejecución *i*. (Čiková & Števo, 2010; Onwubolu & Davendra, 2006).

f es un factor real en el rango de [0.0, 1.0] y constante que controla la amplificación de la variación diferencial.

4.4 Cruce

Después del proceso de mutación, viene la formación de un nuevo individuo, que es también llamado vector prueba x^{test}. En el vector, un elemento después de otro es seleccionado; del individuo seleccionado actualmente $x_{i,j}^t$, y del vector ensayo $v_{i,j}^t$, y para cada valor del índice *j* (para cada vector parámetro) es generado un nuevo número aleatorio $rand_j$ [0,1], que es comparado con la constante de cruce C_r, que constituye una variable de control para el esquema DE, además representa un factor de cruce de valores reales en el rango de [0.0,1.0] que controla la probabilidad de que un vector ensayo $v_{i,j}^t$ sea seleccionado del vector mutado elegido aleatoriamente x_i^t, en vez del vector actual. Además es generado un índice aleatorio *k* que asegura un cambio de al menos un parámetro en el vector ensayo $v_{i,j}^t$. (Čiková & Števo, 2010).

Si $rand <= C_r$ o si $k=j$, la posición de cada elemento del x^{test} determinada del elemento vector ensayo $v_{i,j}^t$, de lo contrario, la posición del individuo se tomará del individuo seleccionado actualmente $x_{i,j}^t$.

Formalmente:

$$x_j^{test} = \begin{cases} x_{r3,j}^t + f * \left(x_{r1,j}^t - x_{r2,j}^t\right), & si\ rand_j\langle 0,1\rangle \leq c_r \vee j = k \\ x_{i,j}^t, & de\ lo\ contrario \end{cases} \tag{4}$$

Donde:

$$k \in \{1,2 \dots d\}, \quad c_r \in [0,1]$$

Generalmente f y C_r son parámetros de control que permanecen constantes durante el proceso de búsqueda y afectan la velocidad de convergencia y la robustez del proceso de búsqueda.

4.5 Selección

El esquema de selección en DE difiere de otros AE, con el fin de decidir si el nuevo individuo de prueba se convertirá en un miembro de la población de la siguiente iteración (t+1). El nuevo individuo se compara con el individuo actual seleccionado x_i^t . Si el nuevo individuo reduce el valor de la función objetivo, entonces es aceptado para la siguiente generación, de lo contrario, el individuo actual es conservado en la población (Hu, Xiong, Fang y Su, 2014).

$$x_i^{t+1} = \begin{cases} x^{test}, & si\ f_{cost}(x^{test}) \leq f_{cost}(x_i^t) \\ x_i^t, & de\ lo\ contrario \end{cases} \quad (5)$$

Por lo tanto, cada individuo de la población de prueba o temporal, es comparado con su contraparte en la población actual. El que tiene el menor valor de la función costo, se propagará a la población de la siguiente generación, como resultado, todos los individuos de la siguiente generación son tan buenos o mejores que sus contrapartes en la generación actual. El punto concerniente de interés en el esquema de selección en DE es que un vector prueba solo es comparado con un vector individuo, no con todos los vectores individuos en la población actual.

4.5.1 Estrategias

Price y Storn (2001) han sugerido diez diferentes estrategias de trabajo en DE y algunas pautas en la aplicación de estas estrategias para cualquier problema dado. Se pueden adoptar diferentes estrategias en el algoritmo DE dependiendo del tipo de problema para el cual se aplica. Las estrategias pueden variar en función del vector.

La convención general utilizada anteriormente es la siguiente: DE/**x**/**y**/**z**. **x** representa una cadena que denota el valor a ser perturbado; **y** es el número de vectores de diferencia considerados para la perturbación de **x**, finalmente **z** es el tipo de cruce que se utiliza (exp: exponencial; bin: binomial).

Ya que el esquema del algoritmo de trabajo por Price y Storn (1995) es la segunda estrategia en DE, es decir, DE/rand/1/bin. Por lo tanto, la perturbación puede ser en cualquier vector elegido al azar.

Tabla 6. Diez diferentes estrategias de trabajo

Estrategia	Formulación
DE/best/1/exp:	$v = x_{best}^t + f * (x_{r2}^t - x_{r3}^t)$
DE/rand/1/exp:	$v = x_{r1}^t + f * (x_{r2}^t - x_{r3}^t)$
DE/rand-to-best/1/exp:	$v = x_i^t + \ell * \left(x_{best}^t - x_i^t\right) + f * (x_{r1}^t - x_{r2}^t)$
DE/best/2/exp:	$v = x_{best}^t + f * (x_{r1}^t + x_{r2}^t - x_{r3}^t + x_{r4}^t)$
DE/rand/2/exp:	$v = x_{r5}^t + f * (x_{r1}^t + x_{r2}^t - x_{r3}^t + x_{r4}^t)$
DE/best/1/bin:	$v = x_{best}^t + f * (x_{r2}^t - x_{r3}^t)$
DE/rand/1/bin:	$v = x_{r3}^t + f * (x_{r1}^t - x_{r2}^t)$
DE/rand-to-best/1/bin:	$v = x_i^t + \ell * \left(x_{best}^t - x_i^t\right) + f * (x_{r1}^t - x_{r2}^t)$
DE/best/2/bin:	$v = x_{r5}^t + f * (x_{r1}^t + x_{r2}^t - x_{r3}^t + x_{r4}^t)$
DE/rand/2/bin:	$v = x_{best}^t + f * (x_{r1}^t + x_{r2}^t - x_{r3}^t + x_{r4}^t)$

Elaboración propia basada en (Onwubolu, 2004).

En el cruce exponencial, el cruce se realiza de acuerdo al número de variables a ser optimizadas hasta que este dentro de los límites de C_r. Para problemas de optimización discreta, la primera vez que un número elegido aleatoriamente entre 0 y 1 va más allá del valor de C_r, no se realiza ningún cruce y las variables permanecen intactas.

En el cruce binomial, el cruce se realiza en cada una de las variables, siempre que un número aleatorio escogido entre 0 y 1 está dentro del valor de C_r. Por lo tanto, los cruces exponenciales y binomiales producen resultados similares (Onwubolu & Davendra, 2004).

4.5.2 Ventajas y deficiencias

Muchas investigaciones han identificado que el algoritmo DE es uno de los más potentes algoritmos estocásticos de parámetros reales para problemas de optimización global, además la extracción de información de distancia y dirección de la población para generar desviaciones aleatorias da como resultado un esquema adaptativo con excelentes propiedades de convergencia (Čiková & Števo, 2010; Onwubolu & Babu, 2004; Onwubolu & Davendra, 2006).

Sin embargo, un problema de estancamiento aún existe en las variables de DE. Con el fin de superar la desventaja, ideas de mejora han aparecido poco a poco recientemente. Una de ellas es combinar múltiples operadores de mutación para equilibrar la capacidad de exploración y explotación. Otra más es desarrollar variantes en DE convergentes, en teoría para disminuir la probabilidad de ocurrencia del estancamiento, entre otras.

4.6 Modelo canónico

El algoritmo canónico DE funciona de la siguiente manera

1. Inicializar la población con una distribución uniforme
2. Hasta que una condición de finalización es cumplida, para cada individuo x_i en la población, hace en paralelo:

I. Selecciona tres diferentes individuos r_1 , r_2 y r_3 de la población actual, aleatoriamente.
II. Genera un vector de prueba *v* usando la ecuación (3).

$$v_j^t = x_{r3,j}^t + f * \left(x_{r1,j}^t - x_{r2,j}^t\right) \tag{3}$$

III. Crea un nuevo vector x_j^{test} utilizando un cruce binomial de acuerdo con la ecuación (4).

$$x_j^{test} = \begin{cases} x_{r3,j}^t + f * \left(x_{r1,j}^t - x_{r2,j}^t\right), & si\ rand_j\langle 0,1\rangle \leq c_r \vee j = k \\ x_{i,j}^t, & de\ lo\ contrario \end{cases} \tag{4}$$

IV. Evaluar el candidato x_j^{test}.

$$x_i^{t+1} = \begin{cases} x^{test}, & si\ f_{cost}(x^{test}) \leq f_{cost}(x_i^t) \\ x_i^t, & de\ lo\ contrario \end{cases} \tag{5}$$

V. Reemplazar el individuo $x_{i,j}^t$ con x_j^{test}, si el valor del nuevo vector creado x_j^{test} es mejor que el valor del individuo $x_{i,j}^t$.

En este algoritmo *f* y C_r son parámetros de control que desempeñan el mismo rol que la mutación y probabilidad de cruce en AE tales como AG.

Algoritmo 2. DE

```
Algoritmo DE
Entradas: Np, Maxiter, f, F, Cr, t
Salidas: Cmax(Gbest), Gbest
Begin
 n=size(t,2);
 iter=0;
 pop[i].x[iter]= Particulas (Np, n);
  pop[j].x[iter+1,j]= [0 0 … 0] Np x n;
 [Gbest, Cmax(Gbest), pop[i].fitness]= FSSP (pop[i].x[iter], Np, t);
    while iter < Maxiter
      for i= 1 to Np
    ind= 1 to Np
    ind(i)= [];
    Pad= randperm (Np-1, 3);
    individuos= ind(Pad);
    Padres= ([pop[i].x[iter](individuos))
   %conversión a números reales, hacia la transformación
     [Z]= numerosreales(pop[i].x[iter], n, F, Padres);
     pop[i].Vce= [0 0 … 0] (1, n);
     pop[i].Vt= [0 0 … 0] (1, n);
     pop[i].k= randi(n);
            for j= 1 to n
      A= rand;
       if A<= cr || j== k
         Pa= Z(individuos);
   %transformación inversa
         (pop[j].Vce[1,j),(pop[i].Vt[1,j])= vectorensayo(Pa,f,F,j);
         pop[j].x[iter+1,j]= vectorprueba(pop[j].Vce[1, j), j, n);
       else
         pop[j].x[iter+1, j]= pop[i].x[iter, j];
       endif
     endfor
     pop[j].x[iter+1]= permutacionValida(pop[j].x[iter+1], n);
     pop[i].fitness[iter+1]= calmakespan(pop[j].x[iter+1], t);
  %compara y actualiza el mejor costo personal
     if Cmax pop[i].fitness[iter+1]= < Cmax(pop[i].fitness)
         pop[i].x[iter]= pop[j].x[iter+1];
         Cmax(pop[i].fitness)= pop[i].fitness[iter+1];
     endif
  %compara y actualiza el mejor costo global
     if Cmax(pop[i].fitness)= < Cmax(Gbest)
         Cmax(Gbest)= Cmax(pop[i].fitness);
         Gbest= pop[j].x[iter+1];
     endif
  endfor
  iter=iter+1;
 endwhile
end
```

Capítulo 5

Algoritmos Evolutivos Celulares (AEC)

Existen múltiples familias de AE (la programación genética, GA, las estrategias evolutivas, la programación evolutiva, los algoritmos de estimación de distribuciones, etc.), pero muy pocos cuentan con versiones celulares, y estas surgieron como una evolución de un modelo de AC aplicado sobre un AE. (Dorronsoro, 2006; Whitley, 1993).

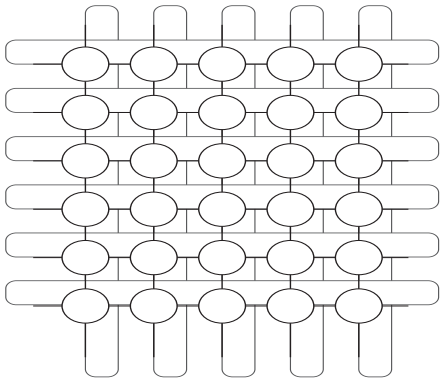
Un AEC puede verse como un AC con reglas de reescritura probabilísticas, en donde el alfabeto del AC es equivalente a todo el conjunto posible de individuos en el espacio de búsqueda, es decir, al número de posibles soluciones al problema. Por tanto, si vemos a los AEC como un tipo de CA, es posible importar herramientas analíticas y modelos o propuestas existentes en el campo de los CA al de los AEC y poder mejorar su rendimiento, su implementación es relativamente simple y para su aplicación no es necesario llamar a rigurosas razones matemáticas (Dorronsoro, 2006; Yang *et al*., 2008).

Las estructuras celulares (modelos celulares) simulan la evolución natural desde el punto de vista del individuo para proveer a la población de una estructura espacial de individuos (dispuestos conceptualmente, los cuales distribuye en la malla toroidal definida como un grafo conectado, en el que cada vértice es un individuo que se comunica con sus vecinos más cercanos. Estas estructuras logran algoritmos cada vez más eficientes para resolver problemas complejos, además, el efecto perseguido de estructurar la población consiste en prever algunas características interesantes respecto al mantenimiento de la diversidad poblacional durante más tiempo y el ajuste flexible de la presión selectiva y mayor eficacia, mejorando de esta manera la capacidad de exploración del algoritmo en el espacio de búsqueda, mientras que la explotación puede también ser reforzada a veces muy fácilmente (Bermudez *et al*., 2009).

5.1 Principio de funcionamiento

En un AEC la población está usualmente estructurada en una rejilla (o malla) bidimensional de individuos. En ella, los individuos (círculos) situados al borde de la malla están conectados con los individuos que están en el otro borde de la malla en su misma fila y/o columna, según el caso. Esta conexión está representada en la figura mediante una línea discontinua. El efecto conseguido es el de una malla toroidal, de forma que todos los individuos tienen exactamente el mismo número de vecinos. El conjunto de emparejamientos potenciales de un individuo se llama vecindario.

Figura 5. Distribución de la población modelo celular.



Elaboración propia basada en (Bermudez, Salto y Alfonso, 2009).

Los pequeños vecindarios solapados (de mismo tamaño y forma regularmente idéntica) con los vecindarios de los individuos más próximos de los AEC ayudan en la exploración del espacio de búsqueda, ya que la lenta difusión de las mejores soluciones inducida en la población con los vecindarios, dota al algoritmo de un tipo de exploración (diversificación), mientras que la explotación (intensificación) se da dentro de cada vecindario debido a la aplicación de las operaciones genéticas.

Las mejores soluciones se extenderán suavemente por toda la población, por lo que de esta forma logramos mantener la diversidad genética en la población por más tiempo. Esta suave dispersión de las mejores soluciones es una de las principales responsables entre el buen equilibrio de exploración y explotación que aplican en el espacio de soluciones al problema durante la búsqueda del óptimo. Es inmediato pensar que se puede regular esta velocidad de expansión de la mejor solución por la población (y por tanto el nivel de diversidad genética a lo largo de la evolución) modificando el tamaño del vecindario a utilizar, ya que el grado de solapamiento entre vecindarios crecerá con el tamaño del vecindario.

Los AEC están siendo aplicados con mucho éxito en los últimos años en la resolución de problemas complejos académicos e industriales, como problemas de logística o de ingeniería (Alba, Chicano, Dorronsoro y Luque, 2004; Dorronsoro, 2006; Folino, Pizzuti y Spezzano, 1998).

5.1.1 Estructura de vecindario

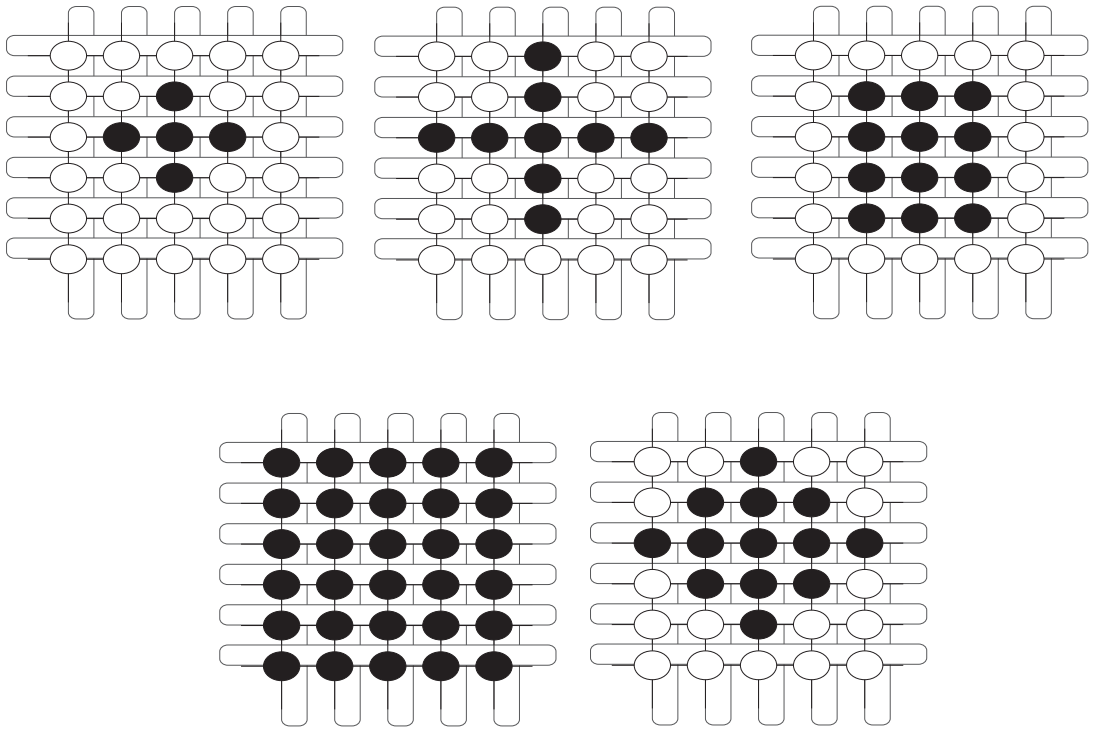
El AEC típico crea una población inicial de soluciones generadas aleatoriamente (individuos cargados en la malla en forma aleatoria), las cuales distribuye en la malla toroidal y calcula la función costo de cada una de tales soluciones. Luego comienza el proceso evolutivo, que recorrerá cada posición de la malla toroidal y determina el vecindario de cada uno de los individuos dependiendo de la posición.

La solución obtenida pasará a la nueva población en idéntica posición, se puede configurar para que ese pasaje se haga siempre o sólo cuando mejora la solución actual. Este proceso evolutivo se realizará hasta cumplir un determinado criterio de terminación o condición de parada, para intercambiar información dentro de sus vecindarios, y con sus vecinos más próximos en la población. La búsqueda es guiada mediante el valor calculado de función costo para cada una de las potenciales soluciones, hasta alcanzar un valor óptimo o una solución aceptable sea alcanzada. (Bermudez *et al*., 2009).

Sarma y De Jong (1996) distingue entre vecindarios lineales o compactos:

- Lineales (Lr). Los vecinos de un punto dado incluyen a las estructuras más cercanas elegidas sobre los ejes horizontal y vertical.
- Compactos (Cr). Los vecinos de un punto dado incluyen a las estructuras más cercanas, pudiendo describir formas compactas de cuadrados o rombos.

Figura 6. Tipos de vecindarios: Lineales y Compactos



Elaboración propia basada en (Bermudez et al., 2009).

5.2 AEC síncronos y asíncronos

Existen dos tipos distintos de AEC en función de cómo se aplique el ciclo reproductor a los individuos. Si el ciclo es aplicado a todos los individuos simultáneamente, se dice que el AEC es síncrono, puesto que los individuos que formarán la población de la siguiente generación son generados formalmente todos a la vez de forma paralela, es decir, que todos los individuos cambian de forma simultánea.

Por otro lado, si vamos actualizando los nuevos individuos en la población secuencialmente (individuos que formarán parte de la población de la siguiente generación) siguiendo alguna política determinada, la actualización de los individuos se hace una a la vez, tendremos un AEC asíncrono.

La política de actualización de los individuos tiene un marcado efecto en el comportamiento del algoritmo, al igual que otros parámetros como el tamaño y forma del vecindario. Además, el orden de visita de los individuos en el caso asíncrono también es un factor determinante en el comportamiento del algoritmo (Alba *et al*., 2006; Bermudez *et al*., 2009).

5.3 AEC como modelos algorítmicos

La forma de la población es un factor determinante en el comportamiento de un AEC, hasta el punto de que cambiar la forma de la población puede llevarnos a diferencias en el comportamiento del algoritmo mucho mayores que en el caso de cambiar la política de actualización del individuo o el tipo de selección, ya que son necesarios nuevos modelos algorítmicos que traten de beneficiarse de la lenta difusión de soluciones producida en la población de los AEC pero que al mismo tiempo traten de mitigar la pérdida de eficiencia y que lleven a cabo sobre la población un equilibrio entre exploración y explotación más adecuado, donde el ciclo reproductor tiende a potenciar la especialización de los individuos que lo componen (se fomenta la explotación dentro de estas zonas), manteniendo diversos caminos de búsqueda hacia soluciones diferentes.

Se calcula la función costo de cada uno de los individuos que conforman la población, posteriormente comienza el proceso evolutivo, donde se recorrerá cada posición de la malla toroidal y determina el vecindario de cada uno de los individuos dependiendo de la posición, después aplica los operadores de recombinación y mutación a los individuos seleccionados de tal vecindario.

Adicionalmente Dorronsoro (2006) menciona que, gracias a la lenta difusión de las mejores soluciones, la diversidad se mantiene por más tiempo en la población, formándose en ella pequeños nichos, y cada uno de estos nichos puede verse como un camino de explotación del espacio de búsqueda o agrupaciones de individuos parecidos, representando distintas regiones de búsqueda del algoritmo.

Por último, la solución obtenida pasará a la nueva población en idéntica posición, se puede configurar para que ese pasaje se haga siempre o sólo se actualizará sucesivamente cada celda cuando mejora la solución actual. Este proceso evolutivo se realizará hasta cumplir un determinado criterio de terminación.

5.3.1 Ventajas y deficiencias

En los últimos años, gracias al trabajo de unos pocos grupos de investigadores conscientes de las ventajas que implica utilizar este tipo de algoritmos, han ido apareciendo poco a poco publicaciones poniendo de manifiesto el buen comportamiento de los AEC en la resolución de problemas complejos sobre máquinas secuenciales (Alba *et al*., 2002; Alba & Troya, 2000; Folino *et al*., 1998). Los AEC están, recibiendo cada vez más interés por parte de la comunidad científica, y son ya muchos grupos de investigación los que han puesto de manifiesto su interés en este tipo de algoritmos.

Una de las principales características que propician el buen comportamiento de los AEC es la lenta difusión de las mejores soluciones por la población, por lo que se mantiene la diversidad de soluciones entre los individuos por más tiempo con respecto a otros tipos de AE.

Pero probablemente, esta característica constituye uno de los principales problemas de los AEC, ya que produce una lenta convergencia del algoritmo hacia el óptimo, disminuyendo de esta manera la eficiencia del algoritmo.

Capítulo 6

Implementación de un híbrido de un Algoritmo Evolutivo Celular Asíncrono (AECA)

6.1 Metodología para el modelado del algoritmo ATPPSO aplicado al FSSP

Un modelo de una alternativa de dos fases del algoritmo de PSO denominado por sus siglas en inglés como (ATPPSO), es propuesto para resolver el FSSP con el objetivo de minimizar el makespan.

6.1.1 Modelo del ATPPSO

Para un problema de FSSP consistente de *n* trabajos, de acuerdo con el carácter de este tipo de problemas, si la *i-ésima* posición de la partícula x_i, y su velocidad v_i, son ambos denotados como una permutación de todos los trabajos que deben satisfacer todas las diferentes restricciones (Lauriere, 1978), entonces el operador de cruce puede ser usado para redefinir el modelo del algoritmo original de PSO (Eberhart & Kennedy, 1995).

Ya que el comportamiento de una partícula es principalmente influenciado por su término de impulso, componente cognitivo, componente social. Para eso se tienen las siguientes fórmulas iterativas:

$$V_i(k+1) = V_i(k) \otimes P_{gbest} \otimes P_{ibest} \tag{6}$$

$$X_i(k+1) = X_i(k) \otimes V_i(k+1) \tag{7}$$

Donde $\otimes$ denota la operación de cruce. Al analizar las ecuaciones anteriores, se puede obtener que, cada partícula sigue dos "mejores" soluciones, la mejor solución personal y la mejor solución global. Al igual que el algoritmo original de PSO, este posee las propiedades de convergencia rápida, fácil de calcular y fácil de implementar.

Sin embargo, debido a que las velocidades de las partículas se aproximan rápidamente a cero, la velocidad de cada partícula y su posición actual tienen la misma permutación, las fórmulas iterativas presentadas anteriormente presentan la desventaja de ser fácilmente atrapadas en un óptimo local. Esto es principalmente por una disminución de la diversidad del enjambre.

Con el fin de superar ese problema, un proceso repulsivo es definido e introducido al algoritmo de ATPPSO, en que cada partícula es repelida lejos de su peor posición personal. Este proceso puede hacer que el enjambre busque en algunas regiones prometedoras sin alcanzar y escapar de un óptimo local. En este proceso la velocidad de cada partícula es actualizada a través de:

$$V_i(k+1) = V_i(k) \odot P_{iworst} \otimes P_{ibest} \tag{8}$$

Donde $\odot$ denota la operación de cruce inverso. Se puede ver que cada partícula es repelida lejos de su peor posición personal y atraída a su mejor posición personal simultáneamente. Esto puede mejorar la velocidad de convergencia en gran medida.

El proceso iterativo formado por las ecuaciones 6 y 7 es el proceso de atracción y el proceso de repulsión es construido por la ecuación 8. Estos dos procesos se ejecutan de forma alterna en el algoritmo ATPPSO.

De esta manera, nueva información será introducida continuamente al enjambre, que puede hacer que las partículas vuelen hacia otro espacio de búsqueda para mejorar la habilidad de exploración de cada partícula y abstenerse de promover la convergencia prematura del algoritmo.

Esto es muy parecido a como trabaja la naturaleza. Cuando las aves adquieren la mayor cantidad de alimento de un lugar, ellas se alejan para buscar los lugares que nunca han alcanzado.

6.1.2 Operadores de cruce del ATPPSO

El operador de cruce genera nuevas secuencias o descendientes combinando otras dos secuencias o padres. No debería haber elementos repetidos o faltantes, de lo contrario la secuencia no es válida.

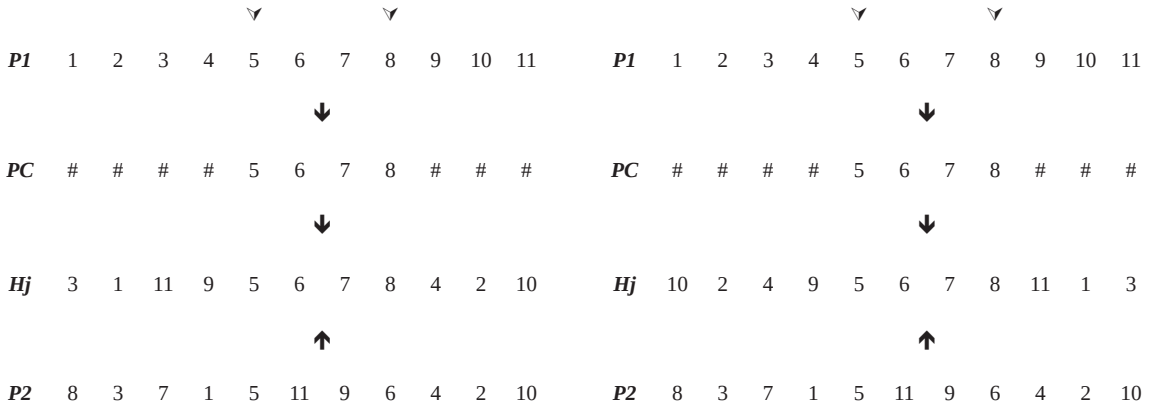
Muchos operadores de cruce diferentes generales y específicos han sido propuestos para el FSSP (Nearchou, 2004). Por simplicidad solo se adopta la segunda forma de los dos puntos de cruce descritos anteriormente.

En el proceso de atracción en primer lugar, un par de puntos de cruce son seleccionados aleatoriamente a lo largo del primer padre. En segundo lugar, los trabajos dentro del esquema de

punto de cruce son copiados en el hijo. En tercer lugar, los lugares restantes del hijo son llenados tomando en orden cada elemento legitimo del segundo padre (sin repetir valores).

En el proceso de repulsión un esquema de cruce de dos puntos es definido y usado para actualizar la velocidad de cada partícula. Al igual que en el proceso de atracción en primer lugar, un par de puntos de cruce son seleccionados aleatoriamente a lo largo del primer padre. En segundo lugar, los trabajos dentro del esquema de punto de cruce son copiados en el hijo. En tercer lugar, los lugares restantes del hijo son llenados tomando en orden inverso cada elemento legitimo del segundo padre (sin repetir valores).

Tabla 7. Operadores de cruce del modelo ATPPSO.



Elaboración propia basada en (Changsheng & Jigui, 2008).

6.1.3 Descripción del algoritmo

En respuesta al FSSP, cada tarea es asignada con un entero de 1 a n (n representa el número de tareas), que representa la operación correspondiente en el individuo.

En el algoritmo ATPPSO, cada individuo es entonces representado por un vector dimensional de enteros, representando el orden directo del procesamiento de tareas. Entonces, se crea una función para que la población inicial sea generada, de acuerdo a la ecuación 9.

$$P^{(0)} = x_{i,j}^{(0)} randperm\ d \tag{9}$$

Donde la función *randperm (d)* asegura el establecimiento de una permutación aleatoria de *d* enteros, por lo que es la permutación aleatoria de la secuencia de tareas.

El enjambre (población) es inicializado aleatoriamente con la posición actual cada partícula (secuencia de trabajo valida), siendo el mismo para la mejor y peor posición. Dando a cada

partícula una velocidad máxima inicial que es igual al orden inverso legitimo de su posición actual.

Cada individuo en la población es también asignado con su función costo, que representa el valor del makespan, se evalúa cada partícula del enjambre conforme a ese valor se establece el mejor global. Para cada iteración, cada partícula es actualizada de acuerdo al modelo del algortimo ATPPSO, agregando un valor de módulo, sí es un número par se ejecuta el cruce atractor y si es impar, se ejecuta el cruce repulsor, donde las nueva posición de las partículas se carga de acuerdo a dos tipos de cruces que se ejecutan de manera alterna. El criterio de terminación para las iteraciones es determinado de acuerdo a sí un número máximo de repeticiones es alcanzado, además actualiza la nueva población y la velocodad de las partículas. El contador de iteraciones iter comienza en cero, el número máximo de repeticiones es Maxiter, y el enjambre es pop.

Algoritmo 3. ATPPSO

```
Entradas: Np, Maxiter, Modulo, t
Salidas: C_max(Gbest), Gbest
Begin
 n=size(t,2);
 iter=0;
 pop[i].v[iter+1]= [0 0 … 0] Np x n;
 pop[i].x[iter+1]= [0 0 … 0] Np x n;
 pop[i].x[iter]= Particulas (Np, n);
 pop[i].v[iter]= pop[i].x[iter](:, end:-1:1);
 [Gbest, Cmax(Gbest), Cmax pop[i].fitness= FSSP (pop[i].x[iter], Np, t);
 pop[i].Pbest= pop[i].x[iter];
 Cmax(pop[i].Pbest)= pop[i].fitness;
 pop[i]. Pworst= pop[i].x[iter];
 Cmax(pop[i].Pworst)= pop[i].fitness;
  while iter <= Maxiter
   for i=1 to Np
    %Carga la velocidad de las partículas en el cruce atractor
     r=mod(iter, Modulo);
      if r~=1
          %Evalúa el proceso de cruce atractor
         pop[i].v[iter+1]= pop[i].v[iter] ⊗ Gbest⊗ pop[i].Pbest;
       else
         pop[i].v[iter+1]= pop[i]. Pworst ⊙ pop[i].v[iter]⊗pop[i].Pbest;
      endif
          %Carga la posición de las partículas en el proceso de atracción
      pop[i].x[iter+1]= pop[i].x[iter] ⊗ pop[i].v[iter+1];
      Cmax pop[i].fitness= calmakespan(pop[i].x[iter+1], t);
          %Compara y actualiza el mejor costo personal
      if Cmax pop[i].fitness < Cmax(pop[i].Pbest)
          pop[i].Pbest= pop[i].x[iter+1];
          Cmax(pop[i].Pbest)= Cmax pop[i].fitness;
      endif
          %Compara y actualiza el peor costo personal
      if Cmax pop[i].fitness > Cmax(pop[i].Pworst)
          pop[i]. Pworst= pop[i].x[iter+1];
          Cmax(pop[i].Pworst)= Cmax pop[i].fitness;
      endif
          Compara y actualiza el mejor costo global
      if Cmax pop[j].fitness < Cmax(Gbest)
          Cmax(Gbest)= Cmax pop[j].fitness;
          Gbest= pop[i].x[iter+1];
      endif
   endfor
   pop[i].x[iter]= pop[i].x[iter+1];
   pop[i].v[iter]= pop[i].v[iter+1];
   iter=iter+1;
 endwhile
end
```

6.2 Metodología para el modelado del algoritmo DE aplicado al FSSP

6.2.1 Inicialización

A partir de la nueva población de soluciones *P-PSO* obtenida a través del algoritmo ATTPSO, esta población servirá como la base del nuevo proceso evolutivo que utilizara el algoritmo DE.

6.2.2 Ciclo reproductivo

Este ciclo comprende el cruce y la mutación para crear individuos de la siguiente iteración. Para cada individuo x_i^t , *i = 1,2, ... np*, de la población otros tres diferentes individuos son escogidos (vectores r_1, r_2 y r_3). La diferencia de los dos primeros vectores da el vector diferencial *d* , que es multiplicado por la constante de mutación *f* y es añadida al vector r_3. Así se consigue el vector ensayo *v*.

Formalmente:

$$v_j^t = x_{r3,j}^t + f * \left(x_{r1,j}^t - x_{r2,j}^t\right) \qquad (3)$$

$$j = 1,2 \dots d\,, \qquad t = 1,2 \dots g$$

Después del proceso de mutación, viene la formación de un nuevo individuo, que es también llamado vector prueba x^{test} de modo que, un elemento tras otro es seleccionado del individuo seleccionado actualmente x_i^t y del vector ensayo *v*, y para cada par es generado un número aleatorio *rand[0,1],* que es comparado con la constante de cruce C_r . Si *rand <= C_r*, la posición relevante de x^{test} viene del elemento vector ensayo *v*, de lo contrario del individuo seleccionado actualmente x_i^t.

Formalmente:

$$x_j^{test} = \begin{cases} x_{r3,j}^t + f*\left(x_{r1,j}^t - x_{r2,j}^t\right), & si\ rand_j\langle 0,1\rangle \le c_r \vee j=k \\ x_{i,j}^t, & de\ lo\ contrario \end{cases} \qquad (4)$$

Donde:

$$i = 1,2 \dots Np, \qquad j = 1,2 \dots d\,, \qquad t = 1,2 \dots g$$

$$r_1, r_2, r_3 \in \{1,2 \dots Np-1\}; \qquad r_1 \neq r_2 \neq r_3 \neq i$$

Donde *k* es un índice aleatorio que asegura un cambio de al menos un parámetro en el vector prueba x^{test}.

El valor de la función objetivo para el vector prueba x^{test} es comparado con el valor de la función objetivo del individuo actual seleccionado y para la siguiente iteración es seleccionado el vector con la mejor función costo.

$$x_i^{t+1} = \begin{cases} x^{test}, & si\ f_{cost}(x^{test}) \leq f_{cost}(x_i^t) \\ x_i^t, & de\ lo\ contrario \end{cases} \quad (5)$$

Así que el proceso continúa en cada iteración para todos los individuos. El resultado es una nueva generación con el mismo número de individuos.

6.2.3 Selección apropiada de representación de individuos

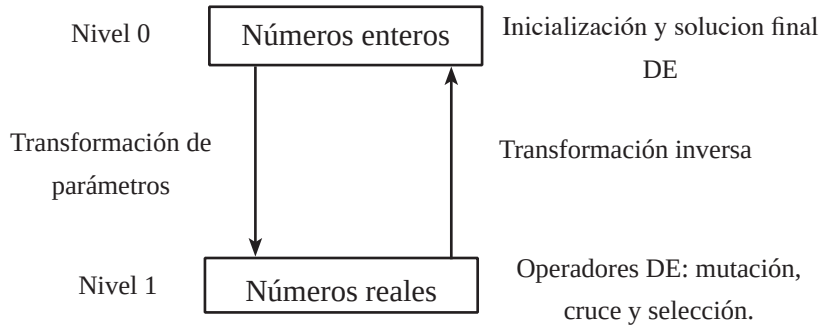
Por lo tanto, cada individuo de la población de prueba o temporal, es comparado con su contraparte en la población actual. El que tiene el menor valor de la función costo, se propagará a la población de la siguiente generación, como resultado, todos los individuos de la siguiente generación son tan buenos o mejores que sus contrapartes en la generación actual. El punto concerniente de interés en el esquema de selección en DE es que un vector prueba solo es comparado con un vector individuo, no con todos los vectores individuos en la población actual.

6.2.4 Transformación de los parámetros de individuos a números reales

Debido a que DE fue originalmente diseñado para resolver problemas con variables de tiempo continuo y la representación natural utilizada consiste en variables enteras, es deseable transformar números enteros en números reales. El método para la transformación fue presentado en (Onwubolu & Babu, 2004) para resolver el problema del agente viajero.

Resolver el problema del FSSP requiere variables discretas y secuencias ordenadas, en lugar de posición relativa indexada.

Figura 7. Transformación de parámetros y transformación inversa.



Elaboración propia basado en (Onwubolu & Babu, 2004).

Este método transforma las variables enteras en variables continuas para la representación interna de valores vectoriales, ya que en su forma canónica. El algoritmo DE es solo capaz de manejar variables continuas.

Donde r_i, *i=1,2,..., n* representa un número entero. La variable continua equivalente r_i se da como:

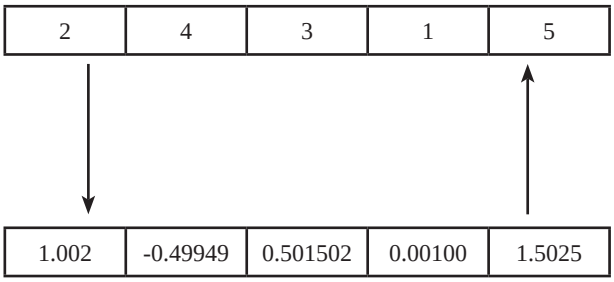
$$r_i = -1 + \frac{z_i * F * 5}{10^3 - 1} \tag{10}$$

Como resultado de la ecuación, algunos resultados resultarán negativos, pero no crea ningún problema de precisión.

Donde *F*, siendo el factor de escala, es dado por ejemplo. *F=100*, ya que encontró que este valor es adecuado para convertir virtualmente todos los números enteros en sus números reales positivos equivalentes.

La ecuación se utiliza para transformar cualquier variable entera en una variable continua equivalente. Sin esta transformación no es posible hacer movimientos útiles hacia el óptimo global en el espacio de solución, utilizando el mecanismo de mutación en DE, que trabaja mejor en variables continuas. La secuencia de tareas no es directamente usada en la representación interna en DE, más bien la secuencia es transformada en una forma continua. En esta técnica no se requiere redondeo o truncamiento, ya que esto a menudo genera un resultado menor al óptimo, porqué ningún intento es hecho durante la optimización para evaluar únicamente los sistemas realizables (Price & Storn, 2001).

Figura 8. Ejemplo de transformación de parámetros de individuos a números reales, para obtención de variable continua equivalente.



Elaboración propia

En la siguiente tabla se aprecia el proceso de transformación de parámetros de individuos a números reales en el problema de 5 trabajos, suponiendo que la secuencia esta dada anteriormente como: [2 4 3 1 5].

Tabla 8. Obtención de variable continua equivalente para una secuenci de cinco trabajos.

$$r_1 = -1 + \frac{2 * 100 * 5}{10^3 - 1} = 0.001001$$
$$r_2 = -1 + \frac{4 * 100 * 5}{10^3 - 1} = 1.002$$
$$r_3 = -1 + \frac{3 * 100 * 5}{10^3 - 1} = 0.501502$$
$$r_4 = -1 + \frac{1 * 100 * 5}{10^3 - 1} = -0.499499$$
$$r_5 = -1 + \frac{5 * 100 * 5}{10^3 - 1} = 1.5025$$

6.2.5 Transformación inversa de números reales a enteros

También se presenta un método de transformación inversa para transformar una población de variables continúas obtenidas tras la mutación en variables enteras para evaluar la función costo.

Las variables enteras son usadas para evaluar la función costo. El esquema de mutación de la población es absolutamente único. Después de la mutación de cada vector. El vector ensayo es evaluado para su función costo y así determinar si retenerlo o no. Esto significa que los valores de la función costo de los vectores actuales en la población necesitan ser también evaluados.

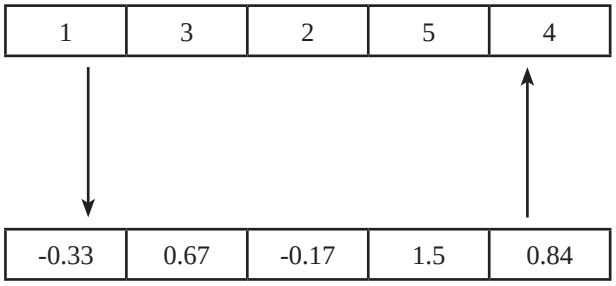
Estas variables vectoriales son continuas (a partir del esquema de transformación de parámetros) y tienen que ser transformadas en sus números enteros equivalentes. No es suficiente redondear estos valores para la clase de problemas como lo es el FSSP.

La técnica de transformación inversa es usada para convertir números reales a su equivalencia de número entero, siendo sólo un ejemplo comparativo sin tener relación con la figura 8.

El esquema se da como sigue:

$$z_i = \frac{(1+r_i)*(10^3-1)}{5*f} \tag{11}$$

Figura 9. Ejemplo de transformación de números reales a enteros.



Elaboración propia

La función de transformación inversa no es capaz de discriminar adecuadamente entre variables. Para asegurarse que cada número es discreto y único, algunas modificaciones son requeridas como sigue:

$$\alpha = int(z_i + 0.5) \tag{12}$$

$$\beta = \alpha - z_i \tag{13}$$

$$z_i^* = \begin{cases} (\alpha - 1) & si\ \beta > 0.5 \\ \alpha & si\ \beta < 0.5 \end{cases} \tag{14}$$

Donde z_i^* es el valor transformado usado para calcular la función objetivo.

Tabla 9. Ejemplo de obtención de nueva secuencia de procesamiento a través de las transformaciones.

$z_1 = \frac{(1 - 0.33) * (10^3 - 1)}{5 * 100} = 1.3386$

$\alpha_1 = int\ (1.3386 + 0.5) = 2$

$\beta_1 = 2 - 1.3386 = \ 0.66134$

$\beta_1 > 0.5$

$z_1^* = (2 - 1) = 1$

$z_2 = \frac{(1 + 0.67) * (10^3 - 1)}{5 * 100} = 3.3367$

$\alpha_2 = int\ (3.3367 + 0.5) = 4$

$\beta_2 = 4 - 3.3367 = \ 0.6633$

$\beta_2 > 0.5$

$z_2^* = (4 - 1) = 3$

$z_3 = \frac{(1 - 0.17) * (10^3 - 1)}{5 * 100} = 1.65834$

$\alpha_3 = int\ (1.65834 + 0.5) = 2$

$\beta_3 = 2 - 1.65834 = \ 0.34166$

$\beta_3 < 0.5$

$z_3^* = 2$

$z_4 = \frac{(1 + 1.50) * (10^3 - 1)}{5 * 100} = 4.9950$

$\alpha_4 = int\ (4.9950 + 0.5) = 5$

$\beta_4 = 5 - 4.9950 = \ 0.005$

$\beta_4 < 0.5$

$z_4^* = 5$

$$z_5 = \frac{(1 - 0.84) * (10^3 - 1)}{5 * 100} = 3.6763$$
$$\alpha_5 = int\,(3.6763 + 0.5) = 4$$
$$\beta_5 = 5 - 3.6763 = \ 0.3237$$
$$\beta_5 < 0.5$$
$$z_4^* = 4$$

6.2.6 Transformación de soluciones no-viables

Es importante mencionar que el esquema de conversión de la ecuación anterior, que transforma números reales después de las operaciones en DE, los números enteros no son suficientes para evitar la duplicación.

El uso del algoritmo DE no requiere la formación de soluciones factibles en caso de representación natural de un individuo. Por lo tanto, es necesario escoger un método apropiado de transformación de soluciones no factibles.

El problema de no factibilidad ocurre en 2 casos:

a. El parámetro del individuo x_j^{test} después de la transformación de números reales a enteros es menor que 1 o mayor que la permutación aleatoria de secuencia de tareas *d*. En este caso el parámetro relevante x_j^{test} es reemplazado por un nuevo parámetro generado aleatoriamente en el rango de 1 a *d*.
b. El individuo creado no compone la permutación aleatoria de enteros de secuencia de tareas *d*. En este caso se utiliza el enfoque de corrección que se presentó en (Brezina *et al*, 2009).

1. Para cada secuencia, sí el número de elementos distintos *ed* de la permutación obtenida *d* es menor al número total de trabajos *n*, la permutación no es válida y se crea el vector *ve* (dimensión *n-d*) con los elementos no incluidos en la permutación aleatoria *d*, de otra manera la permutación si es válida.
2. Para cada elemento de la secuencia, encontrar al primer elemento repetido y reemplazarlo por el elemento del vector de elementos no incluidos *ve*. Para esto se define un índice de elementos repetidos *iv* con un valor inicial de *1*. En un ciclo desde 1 hasta *ed,* se crea el vector *ind* con un valor constante que encuentra las posiciones de elementos repetidos en la secuencia y se suman *sum = tamaño (ind,2),* ir al paso 3).
3. Si *sum > 1*. En un ciclo desde 1 hasta *sum-1, s*e recorre el vector con los elementos repetidos hasta el penúltimo valor y se reemplaza el elemento faltante con cada elemento repetido de la secuencia de acuerdo al índice *ind* del vector *ve,* el índice de elementos repetidos *iv* aumenta una unidad *iv = iv+1,* e ir al paso 4).

4. Si el índice de elementos repetidos *iv* es mayor que el número de elementos no incluidos en la permutación *tamaño (ve,2)*, continúa el ciclo para cada elemento de la secuencia, de otra manera se rompe el ciclo *break*.

6.2.7 Estructura de vecindario

La estructura de vecindario del algoritmo AECA usa diferentes estructuras (variables) asíncronas de vecindarios que permite diversificar la búsqueda de soluciones y evitar estancamiento en óptimos locales mediante mecanismos eficientes.

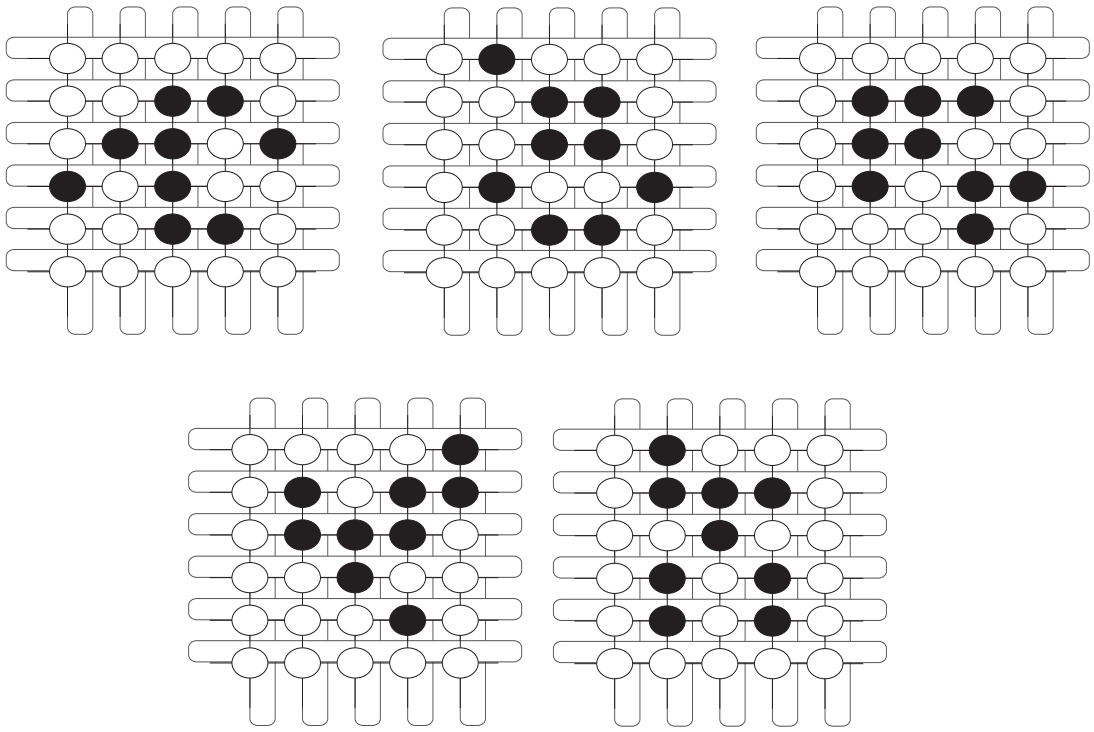
Ya calculado el valor de los diferentes costos de cada vecino generados para cada individuo en el algoritmo DE, estos vecinos residen en una población auxiliar, y se cargan en la malla toroidal, incluyendo al individuo original de la población actual del algoritmo ATPPSO, hasta que se halla computado la iteración completa. Donde la forma de la vecindad no importa dada la regla de evolución del algoritmo.

El vecino con menor función costo de la población auxiliar del algoritmo DE pasará a reemplazar al individuo de la población actual de manera asíncrona (procesamiento secuencial), si es que mejora la solución, donde el tamaño de vecindario se mantendrá estático, es decir que se mantendrá de acuerdo al parámetro ya definido en el algoritmo AECA y continuará hasta que se cumpla el criterio de terminación que es el número máximo de iteraciones.

Esto supone una implementación masivamente paralela real en que cada estructura reside en un procesador, ya que genera una carga de comunicación muy importante al pedir copias de las estructuras designadas como vecinas (o al menos de sus adecuaciones).

Dependiendo de la función costo de los individuos que conforman el vecindario definido para la malla entre toroidal del algoritmo ATPPSO y DE, se obtienen distintas estructuras, que pueden a su vez describir distinto comportamiento por su estrecha relación con la diversidad de la población.

Figura 10. Estructuras variables de vecindario del modelo celular del AECA.



Creación propia basada en (Bermudez *et al*., 2009).

6.2.8 Descripción del algoritmo

Ya inicializada la población de soluciones a partir del algoritmo ATPPSO, se omite la inicialización aleatoria de la población del algoritmo DE, al igual que el proceso inicial de evaluación de la función costo. Ya que se considera que la población de soluciones aleatorias generada inicialmente por el algoritmo ATPPSO, ya fue tratada y tubo la particularidad de generar un conjunto de soluciones posibles de buena calidad, a través de su proceso evolutivo, y la población actual se mejoró.

Dentro del algoritmo DE, inicialmente se agrega un ciclo que representa el número de vecinos que conformará un vecindario (8 de acuerdo al análisis de sensibilidad). Se ejecuta el ciclo reproductivo y las respectivas transformaciones mencionadas con anterioridad.

Una vez que se tienen todos los vecinos de cada individuo de la población del algoritmo DE, se selecciona al mejor, incluyendo al individuo original de la población actual del algoritmo ATPPSO y se forma un arreglo de costos, formando un nicho 9 de individuos parecidos, representando distintas regiones del espacio de búsqueda de ambos algoritmos.

Dentro del vecindario se obtiene la menor función costo y la posición que ocupa, a partir de eso comienza el proceso de evaluación de las soluciones, a manera de que con base en la posición que ocupe la menor función costo del algoritmo DE, el individuo correspondiente va a ir reemplazando la secuencia de trabajo (permutación) en la población en la misma posición, si este fuera de menor función costo, y así para cada individuo de la población, mejorará la población estructurándola a partir de un modelo celular.

Algoritmo 4. AECA

```
Entradas: Np, Maxiter, Modulo, f, F, Cr, nv
Salidas: Cmax(Gbest), Gbest
Begin
n=size(t,2);
iter=0;
 %Arreglo de partículas y velocidades del ATPPSO para cada permutación
pop[i].x[iter+1]= [0 0 … 0] Np x n;
pop[i].v[iter+1]= [0 0 … 0] Np x n;
 %Arreglo de vecinos y makespan para la evolución diferencial
pop[k].x[iter+1,k]= [0 0 … 0] Np x n x nv;
Cmax(pop[l].fitness)= [0 0 … 0] Np x n x nv;
pop[i].x[iter]= Particulas (Np, n);
pop[i].v[iter]= pop[i].x[iter](:, end:-1:1);
[Gbest,Cmax(Gbest),Cmax pop[i].fitness= FSSP (pop[i].x[iter], Np, t);
pop[i].Pbest= pop[i].x[iter];
Cmax(pop[i].Pbest)= Cmax pop[i].fitness;
pop[i]. Pworst= pop[i].x[iter];
Cmax(pop[i].Pworst)= Cmax pop[i].fitness;
while iter <= Maxiter

 %%Parte del ATPPSO

      for i=1 to Np
 %Carga la velocidad de las partículas en el cruce atractor
    r=mod(iter, Modulo);
     if r~=1
 %Evalúa el proceso de cruce atractor
       pop[i].v[iter+1]= pop[i].v[iter] ⊗ Gbest ⊗ pop[i].Pbest;
     else
       pop[i].v[iter+1]= pop[i]. Pworst ⊙ pop[i].v[iter] ⊗ pop[i].Pbest;
     endif
    %Carga la posición de las partículas en el proceso de atracción
       pop[i].x[iter+1]= pop[i].x[iter] ⊗ pop[i].v[iter+1];
       Cmax pop[i].fitness= calmakespan(pop[i].x[iter+1], t);
 %Compara y actualiza el mejor costo personal
     if Cmax pop[i].fitness < Cmax(pop[i].Pbest)
         pop[i].Pbest= pop[i].x[iter+1];
         Cmax(pop[i].Pbest)= Cmax pop[i].fitness;
     endif
 %Compara y actualiza el peor costo personal
     if Cmax pop[i].fitness > Cmax(pop[i].Pworst)
         pop[i].Pworst= pop[i].x[iter+1];
         Cmax(pop[i].Pworst)= Cmax pop[i].fitness;
     endif
 %Compara y actualiza el mejor costo global
     if Cmax pop[j].fitness < Cmax(Gbest)
         Cmax(Gbest)= Cmax pop[i].fitness;
         Gbest= pop[i].x[iter+1];
     endif
```

```
  endfor
     pop[i].x[iter]= pop[i].x[iter+1];
     pop[i].v[iter]= pop[i].v[iter+1];

 %%Parte ED

for j=1 to Np
 %Vecinos para la evolución diferencial
 for k=1 to nv
  ind=1 to Np
  ind(i)= [];
  Pad= randperm (Np-1, 3);
  individuos= ind(Pad);
  Padres= ([pop[i].x[iter](individuos));
 %conversión a números reales, hacia la transformación
  [Z]= numerosreales(pop[i].x[iter], n, F, Padres);
   pop[k].Vce= [0 0 … 0] (1, n);
   pop[k].Vt= [0 0 … 0] (1, n);
   pop[k].r= randi(n);
    for l=1 to n
    A= rand;
     if A<= cr || j== r
      Pa= Z(individuos);
 %transformación inversa
      (pop[k].Vce[1,l),(pop[k].Vt[1,l])= vectorensayo(Pa,f,F,l);
       pop[k].x[iter+1,k]= vectorprueba(pop[k].Vce[1,l),l,n);
     else
       pop[k].x[iter+1,k]= pop[i].x[iter,j];
     endif
   endfor
 %Pasar a permutación valida la nueva solución
      pop[k].x[iter+1,k]=(j,:,k)= permutacionValida(pop[k].x[iter+1,k
      (j,:,k),n);
      Cmax(pop[l].fitness)(j,1,k)= calmakespan(pop[k].x[iter+1,k]
      (j,:,k),t);
endfor
 %Una vez que se tienen todos los vecinos de la particula i, se
seleccionaalmejorincluyendoalindividuooriginal,formandounarreglo decostos
         aux= [reshape(Cmax(pop[l].fitness)(j,1,:),1,nv) Cmax pop[i].
fitness(j)];
 %Obtener la posicion del menor costo
      [menor,pmenor]= min(aux);
 %Ver si se mejoró el costo de la partícula actual
         if(menor < Cmax pop[i].fitness (j))
            pop[i].x[iter= pop[k].x[iter+1,k](j,:,pmenor);
            Cmax pop[i].fitness (j,1)=menor;
         endif
 %Ver si se mejoro el mejor costo de la particula actual
         if(menor < Cmax(pop[i].Pbest)(j))
            aux1= pop[k].x[iter+1,k](j,:,pmenor);
            pop[j].Pbest= aux1;
            Cmax(pop[i].Pbest)(j,1)=menor;
```

```
        endif
  %Ver si se mejoro el mejor costo global
        if(menor < Cmax(Gbest)
            Cmax(Gbest)= pop[k].x[iter+1,k](j,:,pmenor);
            Cmax(Gbest)= menor;
        endif
    endfor
    iter=iter+1;
endwhile
end
```

6.3 Implementación computacional

Se logró la optimización de los algoritmos de ATPPSO, DE y AECA, a través de la instrucción codegen que, aplicado a las funciones creadas, convierte el código a lenguaje máquina **mex** y así reduce considerablemente el tiempo computacional requerido al momento de ejecución.

Se utilizo la función **spmd**, que define un bloque de código que se ejecutará simultáneamente (de forma paralela) en múltiples trabajadores (múltiples datos) que deben ser reservados utilizando el comando **parpool**, aplicado a los algoritmos ya mencionados.

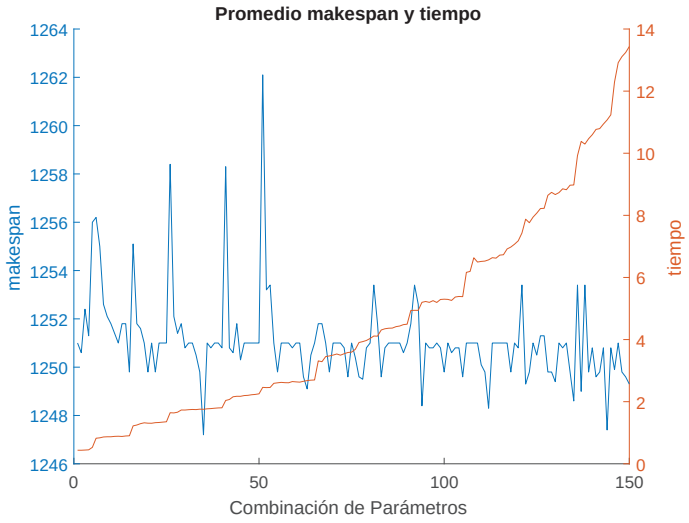
Los algoritmos optimizados han sido evaluados para 90 instancias del tipo FSSP publicados en E. Taillard, "Benchmarks for basic scheduling problems", EJOR 64(2):278-285, 1993.

Se realizó un análisis de sensibilidad de los parámetros (datos de entrada) de control que gobiernan a los algoritmos optimizados, para hacer estimaciones dentro de ciertos límites, sin que cambie la solución óptima. Esto se logró aplicando una serie de ejecuciones independientes (forma paralela) de pruebas paramétricas aplicadas a todas las instancias de E. Taillard, haciendo un estudio de comparación de los resultados obtenidos, así como su interpretación y obtención de la mejor combinación de parámetros.

Se apreciaron diferencias estadísticas en tanto a los promedios de tiempo de ejecución y makespan para las dieferentes instancias, en comparación a los algoritmos mencionados anteriormente, siendo mayor el tiempo de ejecución del modelo híbrido con la ventaja de obtener mejores resultados. Teniendo como metodología seleccionar el primer punto de cruce entre ambos (tiempo y makespan), de manera que el conjunto de parámetros pueda generar buenas soluciones en un tiempo computacional decente con respecto a la ejecución del algoritmo.

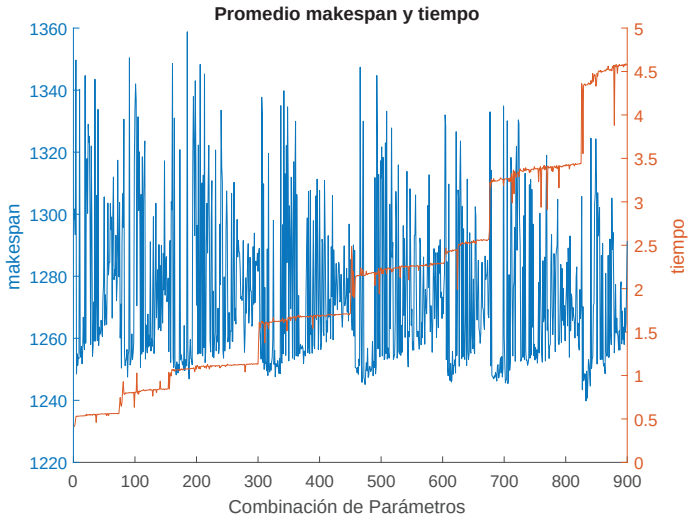
En las siguientes figuras se muestran instancias diferentes, con su primer punto de cruce, usado para la selección de parámetros de control. Estas muestran los resultados obtenidos de la cantidad de evaluaciones y el tiempo de ejecución correspondiente a las instancias especificadas en las figuras para los diferentes algoritmos estudiados, donde la aplicación se ejecuta de manera paralela 10 veces para cada instancia de prueba de acuerdo a la ecuación 15.

Figura 11. Análisis de sensibilidad de parámetros de control para el algoritmo ATPPSO para la instancia 7 de tamaño 100*10.



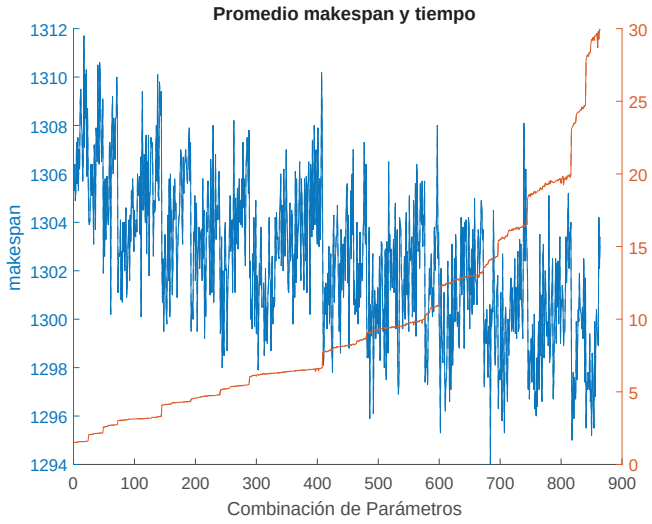
Elaboración propia

Figura 12. Análisis de sensibilidad de parámetros de control para el algoritmo DE para la instancia 9 de tamaño 50*10.



Elaboración propia

Figura 13. Análisis de sensibilidad de parámetros de control para el algoritmo AECA para la instancia 3 de tamaño 20*20.



Elaboración propia

La figura 11 muestra el análisis de sensibilidad de los parámetros de la instancia 7 de tamaño 100*10 (número de trabajos y máquinas) de E. Taillard. Para el algoritmo ATTPSO, teniendo en cuenta 3 parámetros de control, que son:

nv = Tamaño de la población de individuos.
Maxiter = Número máximo de iteraciones (criterio de finalización del algoritmo).
Mod = Módulo de control de cruces (atractor y repulsor).

La figura 12 muestra el análisis de sensibilidad de los parámetros de la instancia 9 de tamaño 50*10 (número de trabajos y máquinas) de E Taillard. Para el algoritmo DE, teniendo en cuenta 5 parámetros de control, que son:

Np = Tamaño de la población de individuos.
Maxiter =Número máximo de iteraciones (Criterio de finalización del algoritmo).
f = Constante de mutación, que afecta el nivel de estocastidad del proceso evolutivo.
F = Factor de escala, que controla la amplificación de la variación deferencial, afectando la convergencia del algoritmo.
C_r = Constante de cruce, que controla la probabilidad de que el vector prueba x^{test}, sea seleccionado, afectando la convergencia del algoritmo.

La figura 13 muestra el análisis de sensibilidad de los parámetros de la instancia 3 de tamaño 20*20 (número de trabajos y máquinas) de E Taillard. para el algoritmo DE, teniendo en cuenta los parámetros de control del algorirmo ATPPSO y DE, agregando uno, que es:

Nv = Vecinos para la Evolución diferencial.

Los limites utilizados para estos parámetros de control aplicados en los algoritmos se muestran en las siguientes tablas.

Tabla 10. Límites para los parámetros de control que gobiernan al algoritmo ATPPSO.

ATPPSO			
	Mínimo	Máximo	Paso
Np	30	150	30
Maxiter	1000	6000	1000
Módulo	1	5	1

Elaboración propia basada en los límites de los parámetros de control para el análisis de sensibilidad deATPPSO.

Tabla 11. Límites para los parámetros de control que gobiernan al algoritmo DE.

DE			
	Mínimo	Máximo	Paso
Np	30	120	30
Maxiter	200	4000	1000
f	0.1	0.9	0.2
F	100	200	50
Cr	0.1	0.9	0.2

Elaboración propia basada en los límites de los parámetros de control para el análisis de sensibilidad en DE.

Tabla 12. Límites para los parámetros de control que gobiernan al algoritmo AECA.

	AECA		
	Mínimo	Máximo	Paso
Np	30	90	30
Maxiter	2000	6000	2000
Módulo	3	4	1
f	0.05	0.1	0.05
F	100	200	100
Cr	0.1	0.5	0.2
Nv	4	10	2

Elaboración propia basada en los límites de los parámetros de control para el análisis de sensibilidad del AECA.

Con base en la observación de los resultados obtenidos por los algoritmos ATPPSO y ED, se pudieron acotar los límites de algunos parámetros para dar más holgura al parámetro de vecinos que podrían conformar el vecindario del modelo celular.

Los parámetros usados para comparar los 3 algoritmos son presentados en la tabla 11, para cada instancia del FSSP, 10 corridas independientes han sido calculadas, presentando los datos de cada problema, para cada conjunto de instancias (varios tamaños) se seleccionó el peor y el mejor resultado obtenido respectivamente.

La siguiente tabla representa la siguiente información:

Taillard = Nombre de la instancia.
*n*m* = Tamaño de la instancia (número de trabajos y máquinas).
Óptimo = Valor óptimo conocido de la instancia.
Mejor valor = Mejor valor encontrado por el algoritmo.
Peor valor = Peor valor encontrado por el algoritmo.
Valor promedio = Costo promedio.
Mediana
Desviación estándar
Tiempo = Tiempo de ejecución por el algoritmo.
% Error = Porcentaje de error.

$$\%\ Error = \frac{Promedio - Óptimo}{Óptimo} * 100 \tag{15}$$

Tabla 13. Resultados computacionales del conjunto de instancias para el algoritmo ATPPSO.

Problema			**ATPPSO**							
Instancia		n*m	Óptimo	Mejor valor	Peor valor	Valor promedio	Mediana	Desv.std	Tiempo	Error
Taillard	3	20*5	1081	1087	1098	1094.9	1098	4.998888765	2.01311933	1.28584644
Taillard	2	20*5	**1359**	**1359**	1366	1364.6	1366	2.951459149	2.04674186	0.4120677
Taillard	5	20*10	2326	2350	2394	2362.3	2360	12.50821952	2.67232262	1.56061909
Taillard	7	20*10	2226	2234	2274	2248.6	2247	10.18931903	2.65280456	1.01527403
Taillard	3	20*20	2326	2350	2394	2362.3	2360	12.50821952	2.67232262	1.56061909
Taillard	6	20*20	2226	2234	2274	2248.6	2247	10.18931903	2.65280456	1.01527403
Taillard	2	50*5	2724	2848	2890	2872.6	2882	15.74237029	5.33948413	5.45521292
Taillard	10	50*5	**2782**	**2782**	2789	2783.6	2783	2.011080417	5.41769739	0.05751258
Taillard	5	50*10	2986	3086	3172	3122.7	3124	26.29342968	5.80227747	4.57803081
Taillard	8	50*10	3039	3076	3152	3114.6	3108	24.76197802	5.77869962	2.48766041
Taillard	9	50*20	3765	3924	3995	3964.4	3970	26.1924501	7.04047586	6.6846071

Tabla 13. ***Continuación.***

Problema			**ATPPSO**							
Instancia	n*m	Óptimo	Mejor valor	Peor valor	Valor promedio	Mediana	Desv.std	Tiempo	Error
Taillard	3	50*20	3715	3815	3883	3847.3	3847	22.63748121	7.02810824	3.56123822
Taillard	9	100*5	5454	5492	5524	5506	5504	8	11.8208458	0.95342868
Taillard	1	100*5	5493	5495	5546	5513.2	5506	16.43708544	11.9512511	0.36774076
Taillard	5	100*10	5468	5660	5739	5698	5694	23.069942	12.3702829	4.20629115
Taillard	10	100*10	5845	5903	6016	5948.4	5939	41.47073935	12.5575886	1.76903336
Taillard	7	100*20	6346	6601	6796	6721.3	6730	62.7500332	14.4255786	5.91396155
Taillard	6	100*20	6437	6679	6790	6751.6	6762.5	35.37795924	14.4914474	4.88736989

Elaboración propia basada en los resultados computacionales obtenidos por el algoritmo ATPPSO.

Las dos veces en que el algoritmo logra alcanzar el valor óptimo se ve reflejado en negrita. En la instancia Taillard 2 para el tamaño de problema de 20 trabajos 5 máquinas y para la instancia 10 para el tamaño de problema de 50 trabajos 5 máquinas.

Tabla 14. Resultados computacionales del conjunto de instancias para el algoritmo DE.

Problema			DE							
Instancia		n*m	Óptimo	Mejor valor	Peor valor	Valor promedio	Mediana	Desv.std	Tiempo	Error
Taillard	1	20*5	**1278**	**1278**	1339	1303.5	1297	19.62566347	1.22274267	1.99530516
Taillard	2	20*5	**1359**	**1359**	1359	1359	1359	0	1.3460479	0
Taillard	2	20*10	1659	1682	1705	1690.8	1690	6.713171133	2.33996354	1.91681736
Taillard	7	20*10	1484	1486	1496	1491.9	1492	2.558211181	2.31142957	0.53234501
Taillard	10	20*20	2178	2183	2249	2216	2219	20.10527847	3.78422534	1.74471993
Taillard	6	20*20	2226	2233	2250	2244.7	2246	5.61842208	3.85093089	0.84007188
Taillard	8	50*5	2683	2704	2713	2707.3	2707	3.164033993	4.392969858	0.905702572
Taillard	10	50*5	2782	2783	2789	2785.8	2784	2.780887149	4.35849773	0.13659238
Taillard	3	50*10	2864	3006	3040	3024	3021	12.11977264	6.14139636	5.58659218
Taillard	4	50*10	3064	3140	3193	3171.1	3172.5	13.74732297	6.14090083	3.49543081
Taillard	8	50*20	3709	3895	3992	3967.1	3971.5	27.0819743	9.8096229	6.95870026
Taillard	3	50*20	3715	3861	3929	3901.6	3906	20.72143281	9.82241862	5.02288022

Tabla 14. ***Continuación.***

Problema			**DE**							
Instancia	n*m	Óptimo	Mejor valor	Peor valor	Valor promedio	Mediana	Desv.std	Tiempo	Error
Taillard 3	100*5	5175	5209	5224	5215.2	5214	4.341018826	10.0308393	0.77681159
Taillard 1	100*5	**5493**	**5493**	5527	5506.3	5505	10.99545361	10.0312257	0.24212634
Taillard 5	100*10	5468	5747	5808	5773.6	5772.5	18.6559493	13.5960164	5.58888076
Taillard 9	100*10	5875	6013	6086	6049.4	6053.5	21.15655927	13.6957021	2.96851064
Taillard 2	100*20	6241	6785	6868	6815.7	6812.5	24.51326172	20.7222683	9.20846018
Taillard 10	100*20	6465	6939	7005	6976.6	6977	20.07873391	20.8319535	7.91337974

Elaboración propia basada en los resultados computacionales obtenidos por el algoritmo DE.

Las tres veces en que el algoritmo logra alcanzar el valor óptimo se ve reflejado en negrita. En la instancia Taillard 1 para el tamaño de problema de 20 trabajos 5 máquinas y en la instancia 2 del mismo tamaño, y la última para la instancia 1 para el tamaño de problema de 100 trabajos 5 máquinas.

Tabla 15. Resultados computacionales del conjunto de instancias para el AECA

Problema			AECA							
Instancia		n*m	Óptimo	Mejor valor	Peor valor	Valor promedio	Mediana	Desv.std	Tiempo	Error
Taillard	1	20*5	**1278**	**1278**	1297	1291.3	1297	9.177871939	15.5683946	1.04068858
Taillard	2	20*5	**1359**	**1359**	1360	1359.2	1359	0.421637021	15.5418473	0.0147167
Taillard	3	20*10	1496	1506	1532	1513.6	1512	7.720103626	22.2994001	1.17647059
Taillard	9	20*10	**1593**	**1593**	1612	1600.2	1599	6.811754546	22.2294334	0.4519774
Taillard	8	20*20	2200	2206	2250	2220.7	2218.5	11.67190359	34.8778841	0.94090909
Taillard	9	20*20	**2237**	**2237**	2263	2247.6	2242.5	10.7723927	34.8196369	0.4738489
Taillard	8	50*5	2683	2686	2710	2703.3	2705	6.32543367	42.5679994	0.75661573
Taillard	1	50*5	**2724**	**2724**	2729	2724.5	2724	1.58113883	42.7669734	0.01835536
Taillard	6	50*10	3006	3094	3140	3110.6	3103	16.58781347	57.3838394	3.47970725
Taillard	4	50*10	3064	3096	3145	3125.2	3128.5	18.04192648	56.9365641	1.99738903
Taillard	8	50*20	3709	3846	3898	3868.7	3864.5	16.878981	89.7346222	5.61070015
Taillard	3	50*20	3715	3766	3861	3806.6	3800	33.49693187	89.8342664	2.46567968

Tabla 15. ***Continuación.***

Problema			**AECA**							
Instancia	n*m	Óptimo	Mejor valor	Peor valor	Valor promedio	Mediana	Desv.std	Tiempo	Error
Taillard 9	100*5	5454	5461	5524	5468	5504	3.4162953	104.17248	0.25670069
Taillard 2	100*5	5268	5271	5313	5310.2	5308	10.6521483	104.10531	0.20110075
Taillard 5	100*10	5468	5572	5684	5640.3	5646.5	29.95941699	126.613752	3.15106072
Taillard 10	100*10	5845	5903	5944	5910.2	5903	15.36084488	126.169485	1.11548332
Taillard 8	100*20	6481	6743	6876	6819.8	6828.5	34.80357581	190.550935	5.22758834
Taillard 6	100*20	6437	6644	6782	6706.5	6698	44.27000489	189.879257	4.18673295

Elaboración propia basada en los resultados computacionales obtenidos por el algoritmo DE.

El AECA demuestra ser superior alcanzando en 5 ocasiones el valor óptimo y se ve reflejado en negrita. En la intancia Taillard 1 para el tamaño de problema de 20 trabajos 5 máquinas y en la instancia 2 del mismo tamaño, en la instancia 9 de 20 trabajos 10 máquinas, en la instancia 9 de 20 trabajos 20 máquinas, y la última para la instancia 1 para el tamaño de problema de 50 trabajos 5 máquinas.

El programa se detuvo cuando se alcanzó el criterio de terminación, que es el número máximo de repeticiones (Maxiter). El máximo el tiempo de ejecución de la aplicación del algoritmo AECA es 190.5550935 segundos.

La siguiente tabla compara el enfoque propuesto entre los 3 algoritmos en función costo.

Tabla 16. Comparación del algoritmo AECA con los algoritmos ATPPSO y DE en función de makespan, tiempo y % de error en la literatura del FSSP.

Problema						Makespan			Tiempo			% Error		
Instancia														
n*m														
Óptimo						ATPPSO	ED	AECA	ATPPSO	ED	AECA	ATPPSO	ED	AECA
Taillard 1	Taillard 8	Taillard 9	Taillard 8	Taillard 9	Taillard 3	Taillard 2	Taillard 1							
20*5	50*5	20*20	20*20	20*10	20*10	20*5	20*5							
1278	2683	**2237**	2200	**1593**	1496	**1359**	**1278**	**1278**	2.04146159	1.22274267	15.5683946	1.228482	1.99530516	1.04068858
1278	2704	2256	2214	1609	1508	**1359**	**1278**	**1278**	2.01311933	1.3460479	15.5418473	1.28584644	0	0.0147167
1278	2704	2242	2222	1602	1509	**1359**	**1278**	**1278**	2.21185479	2.31491612	22.2994001	1.89839572	1.53074866	1.17647059
1278	2686	**2237**	2206	**1593**	1506	**1359**	**1278**	**1278**	2.23579685	2.30934786	22.2294334	2.37915882	1.07972379	0.4519774
5.46802077	5.39474491	2.66742098	2.66490551	2.23579685	2.21185479	2.01311933	2.04146159	1.22274267	15.5683946	1.228482	1.99530516	1.04068858		
4.39258523	4.392969858	3.79031777	3.816646607	2.30934786	2.31491612	1.3460479	1.22274267	15.5683946						
42.7669734	42.5679994	34.8196369	34.8778841	22.2294334	22.2994001	15.5418473	15.5683946							
0.49559471	0.99142751	1.81940098	1.47272727	2.37915882	1.89839572	1.28584644	1.228482							
0.31204112	0.905702572	1.28296826	1.568181818	1.07972379	1.53074866	0	1.99530516							
0.01835536	0.75661573	0.4738489	0.94090909	0.4519774	1.17647059	0.0147167	1.04068858							

Tabla 16. ***Continuación.***

Problema										**Makespan**			**Tiempo**			**% Error**		
Instancia	Taillard 5	Taillard 2	Taillard 9	Taillard 3	Taillard 8	Taillard 4	Taillard 6	Taillard 10	n*m	Óptimo	ATPPSO	ED	AECA	ATPPSO	ED	AECA	ATPPSO	ED	AECA
Taillard 1	5660	5284	5492	3815	3878	3106	3059	5903	100*10	5845	5903	5999	5903	12.5575886	13.6298996	126.169485	1.76903336	2.98545766	1.11548332
Taillard 2	5747	5284	5475	3861	3895	3140	3126	5999	100*10	5845	5903	5999	5903	12.3702829	13.5960164	126.613752	4.20629115	5.58888076	3.15106072
Taillard 3	5572	5271	5461	3766	3846	3096	3094	5903	100*10	5845	5903	5999	5903	11.880049	9.99897009	104.10531	0.5020001	0.33600018	0.20110075
Taillard 4	5468	5268	5454	3715	3709	3064	3006	5903	100*10	5845	5903	5999	5903	11.8208458	10.0131516	104.17248	0.95342868	0.55738907	0.25670069
Taillard 5	5660	5284	5492	3815	3878	3106	3059	5903	100*10	5845	5903	5999	5903	7.02810824	9.82241862	89.8342664	3.56123822	5.02288022	2.46567968
Taillard 6	5747	5284	5475	3861	3895	3140	3126	5999	100*10	5845	5903	5999	5903	6.98748985	9.8096229	89.7346222	5.61070015	6.75726588	5.61070015
Taillard 7	5572	5271	5461	3766	3846	3096	3094	5903	100*10	5845	5903	5999	5903	5.80167547	6.140900832	56.9365641	2.68929504	3.495430809	1.99738903
Taillard 8	5468	5268	5454	3715	3709	3064	3006	5903	100*10	5845	5903	5999	5903	5.82040079	6.1731556	57.3838394	3.71590153	4.88689288	3.47970725

Tabla 16. ***Continuación.***

Problema			Makespan			Tiempo			% Error		
Instancia	n*m	Óptimo	ATPPSO	ED	AECA	ATPPSO	ED	AECA	ATPPSO	ED	AECA
Taillard 8	100*20	6481	6766	7015	6743	14.5840261	20.8251937	190.550935	5.34639716	8.69310292	5.22758834
Taillard 6	100*20	6437	6679	6942	6644	14.4914474	20.7759956	189.879257	4.88736989	8.29578996	4.18673295

Elaboración propia basada en los resultados obtenidos por las comparaciones entre algoritmos.

Como se aprecia en la tabla 16 el algoritmo AECA llego a alcanzar en 5 ocasiones el valor del óptimo conocido de la respectiva instancia, en comparación con el ATPPSO y DE que sólo alcanzaron en 3 ocasiones el óptimo de la respectiva instancia. Aunque en función de tiempo computacional en comparación a los dos algoritmos anteriores AECA es mayor, el porcentaje de error del algoritmo en la mayoría de las ocasiones (excepto en dos) es menor un 88.88% con los algoritmos en comparación, dando a notar que el promedio de las soluciones obtenidas por el algoritmo son de muy buena calidad gracias a la estructuración de la población.

Capítulo 7

Comentarios finales

El AECA fue configurado para que trabaje en forma estática, ya que una vez definido el tamaño de la malla durante el proceso evolutivo, este permanecio con un tamaño seleccionado con 8 individuos como lo muestra la figura 10 y la actualización de los individuos se realizó en forma asíncrona (una por vez y en un determinado orden).

La población inicial fue generada en forma aleatoria con individuos distribuidos en una malla de estructura variable para la parte del algoritmo ATPPSO, y la solución obtenida posteriormente sirvió como población inicial para el proceso evolutivo del algoritmo DE.

El ciclo reproductivo del algoritmo AECA se llevó a cabo en dos partes, la primera parte se encargo de realizar una exploración de las mejores soluciones por parte de ATPPSO y la segunda parte, el algoritmo DE consistió en explotar las soluciones y formar una estructura variable de vecindario para cada individuo, estructurando la población del algoritmo.

La tabla 17 muestra que el algoritmo puede trabajar con una población de individuos *Np* pequeña tomando un valor de 30, obteniendo buenas soluciones con un módulo de 4, que representa de acuerdo a el algoritmo ATPPSO un número par o impar que aplica el cruce respectivo, para la parte del algoritmo DE se eligió una constante de mutación f con un valor bastante pequeño, ya que este afecta al nivel de estocastidad en el proceso evolutivo, un factor escala F que con un valor de 100 pudo controlar la amplificación de la variación diferencial, afectando de igual manera a la convergencia y a el vector en su proceso de mutación. Además, la constante de cruce C_r fue elegida con un valor pequeño de igual manera que *f*, al condicionar a la generación de cada trabajo con respecto al individuo actual o al individuo mutado.

El análisis de sensibilidad mostró que algorimo AECA trabaja muy bien con un número de vecinos *nv* de 8, que conformaron el vecindario, además que el algoritmo se hizo evolucionar con un criterio de finalización o *Maxiter* de 6000 iteraciones por cada parte, es decir 6000 iteraciones para la parte de ATPPSO y 6000 iteraciones para DE. Dando un total de 12000 iteraciones a la cual se somete el algoritmo para cada individuo.

Tabla 17. Mejores valores de los parámetros usados para el algoritmo AECA.

Parámetro	Valor
Np	30
Maxiter	6000
Módulo	4
f	0.1
F	100
Cr	0.1
Nv	8

Elaboración propia basada en los mejores valores para los parámetros utilizados en el AECA.

Por otra parte, para la ejecución de manera paralela, en la sesión principal de Matlab (cliente), se cargaron los argumentos de presentación y se crearon sesiones independientes (trabajadores / laboratorios) que no comparten memoria, además se crearon antes de necesitarlos, y se limpiaron al finalizar el trabajo, usando la instrucción **parpool** que determinó la cantidad de trabajadores a utilizar, para el caso particular fueron 10. Cada trabajador paralelamente se encargó de resolver un conjunto de instancias cada tamaño de problema (10 instancias), haciéndolo 10 veces para cada instancia de prueba, logrando llevar a cabo la ecuación número 15 para la obtención de resultados.

De acuerdo a dichos resultados obtenidos, la hipótesis planteada se acepta, ya que se considera factible la implementación del algoritmo AECA para resolver el FSSP, además el objetivo general de minimizar el valor del makespan, igualmente se cumple gracias a la réplica y optimización de los modelos de PSO y DE usados en el presente trabajo, adaptando un modelo celular variable para estructurar la población, seguido de un análisis parámetrico que sirvió para determinar la mejor combinación de parámetros para su ejecución a través del modo **matlabpool** y el método paralelo que utilizó trabajadores **spmd** (programa único de construcción de lenguaje de datos múltiples), un bloque de este método dividió el trabajo y datos entre trabajadores, existiendo comunicación entre estos al ejecutar las distintas instancias de Taillard a través de un proceso de cálculo en un tiempo de distribución contabilizado por separado, se recibieron los resultados, se guardaron y por último se limpiaron.

La aportación de este trabajo fue la investigación de un modelo híbrido entre dos metaheurísticas aplicado para un problema conocido como como taller de flujo o FSSP, donde el objetivo primordial es encontrar una secuencia óptima de ejecución de trabajos de pequeñas y grandes dimenciones en un tiempo computacional decente.

Conclusiones

La complejidad del FSSP y la búsqueda de un método adecuado para resolverlo, llevó a la réplica exitosa del algoritmo de ATPPSO que modifica las ecuaciones básicas del PSO tradicional. Con esto se decidió probar otro método clasificado como un AE poco trabajado en la literatura para resolver de igual manera el FSSP, esto llevó a replicar el algoritmo DE tradicional con transformaciones en su representación vectorial interna. Al notar ventajas y deficiencias de ambos algoritmos, se mitigaron dichas deficiencias con un algoritmo híbrido entre ambos métodos.

Por otra parte, los modelos celulares han sido un campo poco estudiado para resolver problemas académicos complejos o industriales.

Los resultados presentados en el capítulo 6, describen que el algoritmo híbrido AECA consistió en la implementación de dos metaheurísticos combinados autocontenidos, clasificado como de alto nivel en serie, que logró beneficiarse de la suave dispersión de las mejores soluciones producida en la población, a través de generar estructuras celulares variables, simulando la evolución natural desde el punto de vista de los individuos, habilitándolos para que sólo puedan interactuar e intercambiar información con sus vecinos más próximos dentro de los vecindarios de la población.

Gracias al análisis paramétrico de diferentes pruebas computacionales realizadas sobre los algoritmos ATPPSO y DE con diferentes instancias de prueba tomadas de la literatura de E. Taillard, se delimitó y se obtuvo una combinación de parámetros para AECA, que mejoró las propiedades y comportamiento del algoritmo, realizando el menor esfuerzo computacional, manteniendo la diversidad de soluciones durante más tiempo, mejorando de esta manera la capacidad de exploración del algoritmo en el espacio de búsqueda, mientras que la explotación fue también reforzada, generando soluciones satisfactorias para minimizar el valor del makespan, acercándonse al valor óptimo conocido con un porcentaje de error mínimo o en ocasiones igualarlo.

Se optimizó el AECA reduciendo el tiempo computacional de ejecución, gracias a la característica de generación de código, usando **codegen** y la instrucción **mex** que traduce el código de Matlab a código en lenguaje C. Y de igual manera a la ejecución del algoritmo de forma paralela en datos múltiples con un bloque de **spmd**, se vió igualmente optimizada al ser ejecutado para varias instancias del problema al mismo tiempo, ya que sin el tiempo de distribución contabilizado por separado por cada trabajador (definido con el comando **parpool**), el ejecutar todas las distintas instancias se necesitaría una cantidad de tiempo computacional grande.

El método propuesto posee las limitaciones de tiempo computacional de ejecución, que es mayor a los algoritmos en comparación, además que es necesario un equipo de cómputo con un mínimo de 10 núcleos para ejecutar el algoritmo.

Trabajo futuro

Las actividades propuestas como continuación de este trabajo se resumen en los siguientes puntos:

- Implementar el algoritmo AECA, probando diferentes estrategias de trabajo para DE, o con diferentes tipos de vecindarios como como los mencionados en el capítulo 5 para visualizar su comportamiento en la generación de soluciones de buena calidad.
- Este trabajo presenta la descripción de la implementación y los resultados, sin embargo, sería buena idea realizar un estudio del comportamiento del algoritmo AECA que permita observar la forma (comportamiento interno) en que realiza la búsqueda de buena soluciones.
- Ya que uno de los principales objetivos es minimizar el makespan, podría ser interesante aplicar este modelo para resolver otros problemas de scheduling y con diferente función costo, ajustando al algoritmo a sus restricciones.
- Aumentar la capacidad del algoritmo para trabajar con problemas de mayor tamaño, sin que afecte demaciado el tiempo computacional, así como probar con diferentes AE, para visualizar su comportamiento.

Referencias

Aarts, E., y Lenstra, J. K. (2003). Local search in combinatorial optimization. Eindhoven y Atlanta. Princeton University Press, 1-137.

Alba, E., Chicano, J. F., Dorronsoro, B., y Luque, G. (2004). Diseño de códigos correctores de errores con algoritmos genéticos. Actas del Tercer Congreso Español sobre Metaheurísticas, Algoritmos Evolutivos y Bioinspirados (MAEB), 51–58.

Alba, E., Giacobini , M., Tomassini, M., y Romero, S. (2002). Comparing synchronous and asynchronous cellular genetic algorithms. Springer-Verlag, Heidelberg, 601-610.

Alba, E., y Tomassini, T. (2002). Parallelism and evolutionary algorithms. IEEE transactions on Evolutionary computation, 443-462.

Alba, E., y Troya, J. M. (2000). Cellular evolutionary algorithms: Evaluating the influence of ratio. Proc. of the International Conference on Parallel Problem Solving from Nature VI (PPSN-VI), 29–38.

Alfonso Galipienso, M. I., y Barber Sanchís, F. (2001). Un modelo de integración de técnicas de clausura y csp de restricciones temporales: aplicación de problemas de scheduling. Tesis doctoral de la Universidad de Alicante. España.

Arito, F. L. (2010). Algoritmos de Optimización basados en Colonias de Hormigas aplicados al Problema de Asignación Cuadrática y otros problemas relacionados. Universidad Nacional de San Luis, Facultad de Ciencias Físico Matemáticas y Naturales, Departamento de Informática.

Back, T., Fogel, D., y Michalewicz, Z. (1997). Handbook of Evolutionary computation. Oxford University Press.

Ballesteros Silva, P. P., y Duque Vanegas, C. J. (2013). Aplicación del fraccionamiento de operaciones en una heurística constructiva en programación secuencial para asignación de varios trabajos a varias máquinas en paralelo. Universidad Tecnológica de Pereira, Scientia et Technica Año XVIII, 664-671.

Bárcena Diez, M. (2011). Minimización de costos y retrasos en la programación de la secuencia de despegues y aterrizajes en una pista. Ingeniería en Sistemas - Investigación de operaciones. Tesis de Maestría. Universidad Nacional Autónoma de México. México.

Bermudez, C., Salto, C., y Alfonso, H. (2009). Algoritmos Celulares con Operadores Específicos para Resolver un Problema de Ruteo de Vehículos. Laboratorio de Investigación en Sistemas Inteligentes. Facultad de Ingeniería – Universidad Nacional de La Pampa.

Brezina, I., Čičková, Z., Gežik, P., y Pekár, J. (2009). Modelovanie reverznej logistiky – optimalizácia procesov recyklácie a likvidácie odpadu. Bratislava: Ekonóm.

Blum, C., y Roli, A. (2003). Metaheuristics in combinatorial optimization: Overview and conceptual comparison. ACM Computing Surveys, 268-308.

Cagnina, L., Esquivel, S., y Gallard, R. (2009). Particle Swarm Optimization para Problemas de Secuenciamiento: Total Weighted Tardiness. Laboratorio de Investigación y Desarrollo en Inteligencia Computacional,.

Castillo Cruz, J. D., Mora Gutiérrez, R. A., Rincón García, E. A., y Ponsich Martínez, A. S. (2014). Adaptación de la técnica heurística optimización por enjambres de partículas para resolver un problema de empaquetamiento con restricciones de precedencia. Komputer Sapiens, Revista de Divulgación de la Sociedad Mexicana de Inteligencia Artificial, 7-11.

Castrillón, O. D., Giraldo, J. A., y Sarache, W. A. (2009). Solución de un problema Job Shop con un agente inteligente. Ingeniería y Ciencia, ISSN, 5(10), 75–92.

Changsheng, Z., y Jigui, S. (2008). An alternative two particle swarm optimization algorithm for flow shop scheduling problem. Expert Systems with Applications, 5162-5167.

Chase, R. B., Aquilano, N. J., y Jacobs, F. R. (2000). Administración de Producción y Operaciones: Manufactura y servicios. Bogotá: McGraw-Hill.

Chicano García, J. F. (2007). Metaheurísticas e Ingeniería del Software. Tesis Doctoral. Dpto. de Lenguajes y Ciencias de la Computación. Universidad de Málaga.

Čiková, Z., y Števo, S. (2010). Flow Shop Scheduling using Differential Evolution. Management Information Systems, 5(2), 008-013.

Companys, P., y Ribas Vila, I. (2012). El curioso comportamiento del método de inserción de la heurística NEH en el problema Fm|block|Cmax. 6th International Conference on Industrial Engineering and Industrial Management. XVI Congreso de Ingeniería de Organización. Vigo, July, 18-20.

Cuberos Gallardo, M. (2015). Algoritmo de recocido simulado para la mejora de la eficiencia de una terminal intermodal. Dep. Organización Industrial y Gestión de Empresas II Escuela Técnica Superior de Ingeniería. Tesis de Maestría, Universidad de Sevilla.

Dorronsoro, B. (2006). Diseño e implementación de algoritmos geneticos celulares para problemas complejos. Universidad de Málaga, departamento de lenguajes y ciencia de la computación.

Eberhart, R., y Kennedy, J. (1995). A new optimizer using particle swarm theory. In: Proceedings of the sixth international symposium on micro machine and human science, Nagoya, Japan (pp. 39–43).

Folino, G., Pizzuti, C., y Spezzano, G. (1998). Combining cellular genetic algorithms and local search for solving satisfiability problems. Proc. of the IEEE International Conference on Tools with Artificial Intelligence, 192–198.

Fonseca Reyna, Y., Martínez Jiménez, Y., Figueredo León, Á., y Pernía Nieves, L. (2014). Influencia de los parámetros principales de un Algoritmo Genético para el Flow Shop Scheduling. Revista Cubana de Ciencias Informáticas, vol. 8, núm. 1, enero-marzo, 99-111.

Galzina, V., Lujić, R., y Šarić, T. (2012). adaptive fuzzy particle swarm optimization for flow-shop scheduling problem. Primjena adaptivnih neizrazitih rojeva estica u optimizaciji raspore ivanja proto ne proizvodnje, 1330-3651.

Garey, M. R., y Johnson, D. S. (1979). Computers and Intractability: A Guide to the Theory of NP-Completeness. San Francisco: W. H. Freeman.

Garcia, D. (2008). Ingenieria de organizacion en la empresa: Direccion de operaciones. Ediciones de la Universidad de Oviedo.

Giacobini, M., Tomassini, M., Tettamanzi, A., y Alba, E. (2005). Selection intensity in cellular evolutionary algorithms for regular lattices. IEEE Transactions on Evolutionary Computation, 489-505.

Goldberg, D. E. (1989). Genetic Algorithms in Search, Optimization, and Machine Learning. Addison Wesley.

Hau Lai, C. (2012). Diseño e implementación de algoritmos aproximados de clustering balanceado en PSO. Facultad de físicas y matematicas, departamento de ciencias de la computacion. Tesis de Maestría no publicada. Universidad de Chile.

Hejazi, S. R., y Saghafian, S. (2005). Flowshop scheduling problems with makespan criterion. International Journal of Production Research, 2895-2929.

Hekmatfar, M., Fatemi-Ghomi, S. M., y Karimi, B. (2011). Two stage reentrant hybrid flow shop with setup times and the criterion of minimizing makespan. Applied Soft Computing, 11, 4530-4539.

Hernández Romero, N., Medina Marín , J., y Seck Tuoh Mora, J. C. (2014). Introducción a Matlab para resolver problemas de ingeniería Aplicando Algoritmos Genéticos. Pachuca de Soto: Universidad Autónoma del Estado de Hidalgo.

Hojjati, S. M., y Sahraeyan, A. (Julio de 2009). Minimizing makespan subject to budget limitation in hybrid flow shop. International Conference on Computers & Industrial Engineering CIE 2009, 18-22.

Hu, Z., Xiong, S., Fang, Z., y Su, Q. (2014). A Convergent Differential E volution Algorithm w ith HiddenAdaptationSelectionforEngineeringOptimization.HindawiPublishingCorporation, Mathematical Problems in Engineering. http://dx.doi.org/10.1155/2014/135652, 11.

Laguna, M., y Martí, R. (2003). Scatter Search. Methodology and Implementations. Boston: Kluwer Academic Publishers.

Lauriere, L. (1978). A language and a program for stating and solving combinatorial problems. Artificial Intelligence, 10(1), 29–127.

López Vargas , J. C. (2013). Metodología de programación de producción en un flow shop híbrido flexible con el uso de algoritmos genéticos para reducir el makespan. Aplicación en la industria textil. Universidad Nacional de Colombia, Facultad de Ingeniería y Arquitectura, Departamento de Ingeniería Industrial.

Martí, R. (2003). rocedimientos metaheurísticos en optimización combinatoria. Matematiques, 3-62.

Martin, J. (1992). Experts Systems Solve a Scheduling Problem for Carpenter Technology Systems. APICS, 37-39.

Martínez, E. S. (2015). Modelado y análisis de secuenciación de tareas para el cálculo del makespan de los sistemas de manufactura flexible utilizando Redes de Petri. Tesis de Licenciatura no publicada. Universidad Autónoma del Estado de Hidalgo. México.

Martínez-Molina , M., Moreno-Armendáriz, M. A., Cruz-Cortés, N., & Seck Tuoh-Mora, J. C. (2011).Modeling Prey - Predator Dynamics via Particle Swarm Optimization and Cellular Automata. Springer-Verlag Berlin Heidelberg, Batyrshin and G. Sidorov (Eds.): MICAI 2011, Part II, LNAI 7095, 189-200.

Medina Marín, J., Hernández Romero, N., Seck Tuoh, J. C., y Martínez Gómez, E. S. (2016). Modeling and Simulation of Flow Shop Scheduling Problem thorough Petri Net Tools. International Journal of Computer,Electrical, Automation, control and Information Engineering, 10(5), 818-822.

Medina Marin, J., Gradišar, D., Seck Tuoh Mora, J. C., Hernandez Romero, N., y Nuñez Piña, F. (2016). A Petri Net Model to obtain the Makespan in the Flow Shop Scheduling Problem. Proceedings of the World Congress on Engineering and Computer Science, WCECS, San Francisco, USA, II, 19-21.

Mercado, V., Pandolfi, D., y Villagra, A. (2013). Hibridación de Metaheurísticas aplicadas al Problema de Ruteo de Vehículos. ICT-UNPA-70-2013, ISSN: 1852 - 4516.

Morales Viscaya, J. A. (2012). Un algoritmo híbrido para la optimización en paralelo de problemas continuos.Tesis de Licenciatura. Instituto Tecnológico de la Paz. México.

Morton, T., y Pentico, D. (1993). Heuristic scheduling systems. Wiley series in Engineering and technology management. John Wiley and Sons.

Nawaz, M., Enscore, E., y Ham, I. (1983). A heuristic algorithm for the mmachine, n-job flow-shop sequencing problem. The international Journal of Management Science, 11(1), 91-95.

Nearchou, A. C. (2004). The effect of various operators on the genetic search for large scheduling problems. International Journal of Product Economy, 88, 191–203.

Nesmachnow, S. (2014). Planificación de tareas en sistemas cluster, grid y cloud utilizando algoritmos evolutivos. Komputer Sapiens, Revista de Divulgación de la Sociedad Mexicana de Inteligencia Artificial, 18-22.

Noroozi, V., Hashemi, A. B., y Meybodi, M. R. (2011). CellularDE: A Cellular Based Differential Evolution for Dynamic Optimization Problems. Computer Engineering and Information Technology Department, A. Dobnikar, U. Lotrič, and B. Šter (Eds.): ICANNGA 2011, Part I, LNCS 6593 Amirkabir University of Technology, Tehran, Iran, 340-349.

Onwubolu, G. C., y Babu, B. V. (2004). New Optimization Techniques in Engineering (Vol. 141). Fiji Islands: Springer-Verlag Berlin Heidelberg.

Onwubolu, G., y Davendra, D. (2006). Scheduli ng flow shops using differential evolution algorithm. European Journal of Operational Research, Elsevier. Department of Engineering, The University of the South Pacific, P.O. Box 1168, Suva, Fiji, 171, 674–692.

Ordóñez Hurtado, R. H. (2008). Aplicación de la técnica pso a la determinación de funciones de lyapunov cuadráticas comunes y a sistemas adaptables basados en modelos de error. Facultad de ciencias físicas y matemáticas, departamento de departamento de ingeniería eléctrica. Tesis de Doctorado no publicada. Universidad dE Chile.

Osman, I. H., y Kelly, J. P. (1996). Meta-Heuristics: heory and applications. Kluwer Academic, 1-21.

Osorio, J. C., y Motoa, T. G. (2008). Planificación jerárquica de la producción en un job shop flexible. Escuela de Ingeniería Industrial y Estadística, Facultad de Ingeniería, Universidad del Valle ,Ciudad Universitaria, 158-171.

Pandolfi, D., Villagra, N. (2008). Algoritmos evolucionarios aplicados al problema de secuenciamiento del flow shop. Universidad nacional de san luis, Argentina.

Pandolfi, D., Villagra, A., y Leguizamón, M. G. (2009). Hibridización con búsqueda local de un algoritmo de estimación de distribución para la resolución del problema de secuenciamiento de Flow Shop. In XV Congreso Argentino de Ciencias de la Computación.

Pasteels, J. M., Deneubourg, J. L., y Goss, S. (1987). Self-organization mechanisms in ant societies. Trail recruitment to newly discovered food sources, Experientia Supplementum, 155–175.

Pecero, J. E., y Reyes, L. C. (2014). Programación de tareas, Scheduling e inteligencia artificial:Nuevos retos. Komputer Sapiens, Revista de Divulgación de la Sociedad Mexicana de Inteligencia Artificial, 1, 2-3.

Pérez, R., Sanchez, J., Hernández, A. y Ochoa, C. (2014). Un algoritmo de estimación de distribuciones para resolver un problema real de programación de tareas en configuración jobshop. Un enfoque alternativo para la programación de tareas. Komputer Sapiens - Revista de Divulgación de la Sociedad Mexicana de Inteligencia Artificial, 1, pp. 23-36.

Pinedo, M. (1995). Scheduling- Theory, Algorithms, and Systems. Prentice Hall International in Industrial and System Engineering.

Price, K., Storn, R., 2001, Differential evolution homepage (Web site of Price and Storm) as at 2001. http://www.ICSI.Berkeley.edu/ ~storn/code.html.

Rammohan, M., Ponnuthurai, S., Pan, Q.K. y Mehmet, T. (2011). Differential evolutionalgorithm with ensemble of parameters and mutation trategies. Applied Soft Computing. 11. 1679-1696. 10.1016/j.asoc.2010.04.024.

Ríos Mercado, R. Z., y Bard, J. F. (2010). Heurísticas para Secuenciamiento de Tareas en Líneas de Flujo. Programa de Posgrado en Investigación de Operaciones e Ing. Industrial, Universidad de Texas en Austin, 1-8.

Rodríguez Quiñones, T. A. (2014). Soluci´on de problemas tipo Flow-Shop mediante algoritmos evolutivos. Universidad Nacional de Colombia Facultad de Ingeniería, Departamento de Sistemas e Industrial.

Salazar Pinto, P. Y. (2009). Algoritmo híbrido autoconfigurado para optimización estructural. Bucaramanga: Universidad Industrial de santander, Facultad de ingenierías fisicomecánicas.

Sait, S. M., y Youssef, H. (1999). Iterative computer algorithms with applications in engineering: Solving combinatorial optimization problems. Piscataway: IEEE Computer Society Press, 1-248.

Sánchez, P., y López, S. (2005). Programación de tareas, un reto diario en la empresa. Anales de mecánica y electricidad, 24-30.

Sarmiento Ardila, C. J. (2012). Método Particle Swarm Optimization (PSO enjambre de partículas) aplicado al problema de múltiples objetivos del Job Shop Scheduling (JSP) o secuenciamiento de máquinas. Facultad de ingenierías físico-mecánicas, Escuela de estudios industriales y empresariales, Bucaramanga. Tesis de Licenciatura no publicada. Universidad Industrial de Santander.

Sayoti, F. y Essaid Rif, M. (2016) Golden Ball Algorithm for solving Flow Shop Scheduling Problem. Special Issue on Artificial Intelligence Underpinning. LAROSERI Laboratory, Dept. of Computer Science, Faculty of Sciences, University of Chouaib Doukkali, El Jadida, Morocco 15-18.

Sedano García, A. (2013). Metodología de sintesis óptima dimensional de mecanismos medinte algoritmos de optimización híbridos. Escuela técnica superior de ingenieros industriales y de telecomunicaciones .

Storn, R., Price, K. (1997). Differential evolution a simple evolution strategy for fast optimization. Dr. Dobb s Journal, April 1997, pp. 18–24, 78.

Talbi, E. G. (2009). Metaheuristics from design to implementation. University of Lille- CNRS - INRIA, 593.

Vallada, E., y Ruiz, R. (2011). A genetic algorithm for the unrelated parallel machine scheduling problem with sequence dependent setup times. European Journal of Operational Research, 211, 612-622.

Véles, M., y Montoya, J. (2007). Metaheurísticos: una alternativa para la solución de problemas combinatorios en administración de operaciones. 99-115.

Wang, S., y Liu, M. (2013). A genetic algorithm for two-stage no-wait hybrid flow shop scheduling problem. Computers & Operations Research, 1064-1075.

Whitley, D. (1993). Cellular genetic algorithms. Proc. of the Fifth International Conference on Genetic Algorithms (ICGA), 658.

Yang, S., Hongcheng, L., Liang, G., y Gouhui, Z. (2008). Cellular Particle Swarm Optimization. Journal of Information Sciences, 4460-4493.

Apéndice

Característica de generación de código codegen

Se puede generar código C/C++ y mex automáticamente desde Matlab. Con esta capacidad, se puede diseñar, implementar y probar software para aplicaciones integradas y de escritorio en el entorno Matlab, y luego traducir automáticamente los algoritmos a un código C / C ++ eficiente para su implementación en sistemas integrados y de escritorio.

Criterios de consideración para la generación de códigos de algoritmos de Matlab:

- Producir un código, legible, eficiente y compacto a partir de los algoritmos de Matlab para su implementación en sistemas integrados y de escritorio.

Generar funciones Mex de algoritmos Matlab:

- Acelerar algoritmos de Matlab,
- Verificar el código C generado dentro de Matlab.

Que característica de generación de código usar:

- Generar las funciones Mex del código de Matlab y en la línea de comando en el Codificador de documentación de Matlab.
- Generar el código C de Matlab en la línea de comando en el Codificador de documentación de Matlab.

Generar código de Matlab para sistemas integrados y de escritorio permite realizar diseño, implementación y prueba de software por completo dentro del espacio de trabajo de Matlab. Además, permite generar código C eficiente, legible y compacto automáticamente a partir de los algoritmos de Matlab.

Dentro del entorno de desarrollo de MATLAB, se puede:

- Verificar que los algoritmos son adecuados para la generación de código.

- Generar el código C / C ++ automáticamente, lo que elimina la necesidad de hacerlo manualmente, traduzca los algoritmos MATLAB y minimice el riesgo de introducir errores en el código.
- Modificar el diseño en el código MATLAB para tener en cuenta los requisitos específicos
- de escritorio y aplicaciones integradas, como el tipo de datos, administración, uso de memoria y velocidad.
- Probar el código generado y verificar fácilmente que los algoritmos modificados son equivalentes funcionalmente a sus algoritmos MATLAB originales.

Instrucción mex en Matlab

El término mex significa "ejecutable de Matlab", que Puede llamar a sus propias subrutinas C, C ++ desde la línea de comandos de Matlab como si fueran funciones integradas. Estos programas, llamados archivos binarios mex, son subrutinas vinculadas dinámicamente que el intérprete de Matlab carga y ejecuta.

El archivo mex contiene solo una función o subrutina, y su nombre es el nombre del archivo mex. Para llamar a un archivo mex, use el nombre del archivo, sin la extensión de archivo.

Los nombres de archivo mex compilan y vinculan uno o más archivos fuente de C, C ++ en un archivo binario mex, que se puede llamar desde Matlab. los nombres de archivo especifican los archivos de origen. También crea archivos ejecutables para el motor independiente Matlab y las aplicaciones de archivo .mat.

Matlab selecciona automáticamente un compilador, si está instalado, en función del idioma de los argumentos de los nombres de archivo.

Para llamar a un archivo mex, use el nombre del archivo, sin la extensión de archivo. El archivo mex contiene solo una función o subrutina, y su nombre es el nombre del archivo mex. El archivo debe estar en su ruta Matlab.

Algoritmo 5. Generación de código mex

```
function [Cmax(Gbest), Gbest, optimo]= AECA(Np, Maxiter, Modulo, f, F, cr,
nv, labindex, instancia) %#codegen
%codegen AECA -args {30, 6000, 4, 0.1, 100, 0.1, 8, labindex, instancia}
-report
```

Función spmd en Matlab

El programa único de construcción de lenguaje de datos múltiples (**spmd**) permite intercalar sin problemas la programación en serie y en paralelo. La instrucción Spmd permite definir un bloque de código para que se ejecute simultáneamente en varios trabajadores. Las variables asignadas dentro de la sentencia **spmd** en los trabajadores permiten el acceso directo a sus valores desde el cliente por referencia a través de objetos compuestos.

En un bloque de **spmd**, se tiene acceso a todos los trabajadores individualmente y controla lo que se ejecuta en ellos, cada trabajador tiene un labindex único.

El aspecto "programa único" de **spmd** significa que el código idéntico se ejecuta en varios trabajadores y las partes etiquetadas como bloques de **spmd** funcionan en los trabajadores. Cuando el bloque **spmd** está completo, el programa continúa ejecutándose en el cliente.

El aspecto de "datos múltiples" significa que aunque la sentencia **spmd** ejecuta código idéntico en todos los trabajadores, cada trabajador puede tener datos diferentes y únicos para ese código. Así, múltiples conjuntos de datos pueden ser acomodados por múltiples trabajadores.

Las aplicaciones típicas apropiadas para **spmd** son aquellas que requieren la ejecución simultánea de un programa en múltiples conjuntos de datos, cuando se requiere comunicación o sincronización entre los trabajadores, transferir datos entre ellos y utilizar arreglos codificados entre ellos.

Los trabajadores que ejecutan una declaración de **spmd** operan simultáneamente y se conocen mutuamente. Al ejecutar una instrucción **spmd** en una agrupación paralela, toda la salida de la línea de comandos de los trabajadores se muestra en la ventana de comandos del cliente. Debido a que los trabajadores son sesiones Matlab sin visualizaciones, ninguna salida gráfica.

Se debe crear una matriz distribuida antes del bloque **spmd**, dividida como matriz codistribuida en los trabajadores, y una matriz codistribuida, dividida en trabajadores, accesible como matriz distribuida en el cliente. Algunas funciones reconocerán las entradas de matriz distribuidas y ejecutarán en paralelo

Para ejecutar las sentencias en paralelo, primero debe abrir un grupo de trabajadores de Matlab utilizando parpool o tener sus prefijos paralelos para permitir el inicio automático de una agrupación.

Para llevar a cabo la ejecución en paralelismo explícito, varias instancias del algoritmo AECA corren en varios procesadores, con memorias separadas y ejecutan simultáneamente un solo comando o función de Matlab. Esto se logra dividiendo el cada conjunto de trabajos (instancias) en una sola tarea, cada tarea reside en un procesador, ya que genera una carga de comunicación muy importante al pedir copias de las tareas designadas como vecinas (o al menos de sus adecuaciones).

Nuevas construcciones de programación incluyen bucles paralelos y matrices distribuidas, que describen el paralelismo.

Un ejemplo de sintaxis puede ser como sigue:

```
Syntax:
Solicitar trabajadores
       parpool(trabajadores);
% ejecutar en el cliente
spmd
% ejecutar en todos los workers el proceso de cálculo
end
% ejecutar en el cliente
%guardar los resultados de los trabajadores
      d=[a{:}];
      Save('labIndex.out', 'd', '-ascii' );
%limpiar
     delete(gcp('nocreate') );
```

Algoritmo 6. Optimización del AECA

```
% --- Executes on button press in pushbutton1.
function pushbutton1_Callback(hObject, eventdata, handles)
corridas=str2num(get(handles.edit1,'string'));
Np=30;
Maxiter=6000;
Modulo=4;
f=0.1;
F=200;
Cr=0.1;
Nv=8;
aux=[Np Maxiter Modulo f F Cr Nv];
aux=aux';
combinaciones=combvec(Np,Maxiter,Modulo,f,F,Cr,Nv);
matcomb=combinaciones';
ncol=10;
reng=8;
%Selecciona el número de trabajadores a utilizar
nucleos= get(handles.popupmenu1,'Value');
%Selecciona el conjunto de problemas a resolver a resolver
problema= get(handles.popupmenu2,'Value');
problemafinal= get(handles.popupmenu3,'Value');
parametros=kron(matcomb,ones(corridas,1));
tamat=size(parametros,1);
%calcular las estadísticas
estadisticos=8;
while problema<=problemafinal
 TabladeresultadosPSOED=zeros(tamat,ncol,nucleos);
        Optimos=zeros(1,nucleos);
        Final=zeros(estadisticos,ncol
        Matriz=zeros(tamat,ncol);
        MatvalPSOED=zeros(tamat,1);
        Time=zeros(tamat,1);
        Promediovalores=zeros(tamat,1);
        Promediotiempos=zeros(tamat,1);
        val=zeros(tamat,1);
        parpool(nucleos);
spmd
       for i=1:tamat
           %Ejecuta el AECA para cada conjunto de instancias
        switch  problema
                    case 1
                    case 2
                    case 3
                     case 4
                     case 5
                     case 6
                     case 7
                     case 8
                     case 9
        endswitch
```

```
MatvalPSOED(i,1)=val;
Promedioval=mean(MatvalPSOED(i,:),labindex);
Promediotime=mean(Time(i,:),labindex);
 Matriz(i,1)=Promedioval;
                Matriz(i,2)=Promediotime;
                TabladeresultadosPSOED(:,:,labindex)=Matriz;
endfor
            Promediovalores=Matriz(:,1);
            Promediotiempos=Matriz(:,2);
            Optimos(1,labindex)=optimo;
        endspmd

        VFinal=[Promediovalores{:}];
        TFinal=[Promediotiempos{:};
        Error=zeros(1,tamat)
        for i=1:tamat

 Error(1,i)=((Bestm(1,i)-Toptimo(1,i))/Toptimo(1,i))*100
        endfor
        delete(gcp('nocreate'));
        problema=problema+1;
endwhile
end
```

Algoritmo 7. Análisis de sensibilidad

```
% --- Executes on button press in pushbutton1.
function pushbutton1_Callback(hObject, eventdata, handles)
%Selecciona el conjunto de instancias a resolver a resolver
instancia= get(handles.popupmenu1,'Value');
instanciasmax= get(handles.popupmenu4,'Value');
%Selecciona el conjunto de problemas a resolver a resolver
problema= get(handles.popupmenu2,'Value');
problemafinal= get(handles.popupmenu3,'Value');
mtPSOED=zeros(864,10);
mmPSOED=zeros(864,10);
matrizPSOED=zeros(864,9);
while problema<=problemafinal
    if instancia<=instanciasmax
         nombregraficatime=[nommatt num2str(problema),num2str(ins-
                                                              tancia)];
        switch  instancia
                  case 1
                  case 2
                  case 3
                   case 4
                   case 5
                   case 6
                   case 7
                   case 8
                   case 9
        endswitch
 %ordeno la matriz con respecto a la columna del tiempo
                MtPSOED=sortrows(PSOED,9);
                matrizPSOED(:,1:7)= MtPSOED(:,1:7);
%Matriz que acumula la secuencia definida por el tiempo
                mmPSOED(:,:)= MtPSOED(:,8:9:end);
%Matriz que define la secuencia ordenada de acuerso al makespan
                mtPSOED(:,:)= MtPSOED(:,9:9:end);
%Matriz que define la secuencia ordenada de acuerso al tiempo
%Ciclo para calcular el promedio de los 10 workers
                for i=1:864
                    matrizPSOED(i,end-1)=mean(mmPSOED(i,:));
                    matrizPSOED(i,end)=mean(mtPSOED(i,:));
                endfor
                set(handles.uitable1, 'Data', matrizAECA);
                % aporximaciones obtenidas
                figure(1);
                hold on
                %makespan
                x1= matrizAECA(:,end-1);
                %tiempo
                x2= matrizAECA(:,end);
                % Grafica la solucion real
                title('Promedio makespan y tiempo')
```

```
                xlabel('Combinación de Parámetros')
                yyaxis left
                ylabel('makespan')
                plot(x1)
                yyaxis right
                ylabel('tiempo')
                plot(x2)
                hold off
                j=figure(1);
                print(j, '-dpdf', '-r300', nombregraficatime)
                instancia=instancia+1;
    else
        problema=problema+1;
            instancia=1;
    endif
endwhile
end
```