Universidad Central "Marta Abreu" de Las Villas

Facultad Matemática, Física y Computación

Centro de Estudios de Informática

Algoritmo multi-objetivo para la solución de problemas de secuenciación de tareas tipo Job Shop

Tesis en opción al Título Académico de Máster

Autor: Erick David Rodríguez Bazán

Tutoras: Dra. Yailen Martínez Jiménez
MSc. Beatriz M. Mndez Hernndez

Santa Clara

2016

"Año 58 de la Revolución"

Hago constar que la presente Tesis fue realizada en la Universidad Central "Marta Abreu" de Las Villas como parte de la culminación de estudios de la Maestría en Ciencia de la Computación, autorizando a que el mismo sea utilizado por la Institución, para los fines que estime conveniente, tanto de forma parcial como total y que además no podrá ser presentado en eventos, ni publicado sin autorización de la Universidad.

Erick David Rodríguez Bazán — Fecha
Autor

Los abajo firmantes certificamos que el presente trabajo ha sido realizado según acuerdo de la dirección de nuestro centro y el mismo cumple con los requisitos que debe tener un trabajo de esta envergadura referido a la temática señalada.

Erick David Rodríguez Bazán — Fecha
Autor

Carlos Morell Pérez, Dr.C — Fecha
Jefe de laboratorio

Yailen Martínez Jiménez, Dr.C. — Fecha
Tutor

DEDICATORIA

Dedicatoria

AGRADECIMIENTOS

Agradecimientos

RESUMEN

Durante las últimas décadas, los algoritmos multi-objetivo han captado considerable atención. El problema de secuenciación tipo Job Shop, uno de los principales problemas de optimización, también ha generado el interés de la comunidad científica. Sin embargo, la mayoría de los algoritmos multi-objetivo que se han desarrollado para este problema están basados en enfoques no profesionales, es decir, la mayoría de ellos combinan sus objetivos y resuelven el problema multi-objetivo a través de enfoques basados en un solo objetivo, exceptuando a un reducido grupo de investigadores que utilizan algoritmos basados en la Frontera de Pareto. Además, gran parte de la literatura solo permite optimizar entre dos y tres objetivos, y estos están predefinidos de antemano. En esta tesis se implementa un algoritmo multi-objetivo usando sistemas multi-agente y Aprendizaje Reforzado basado en la Frontera de Pareto que permite optimizar el makespan, el tiempo de inactividad de las máquinas, la tardanza total y la robustez. El algoritmo fue implementado de manera genérica para si se desea, agregar un nuevo objetivo, solo sea necesario incluir una nueva clase. El algoritmo propuesto se aplicó a un conjunto de instancias para optimizar los objetivos tratados y finalmente los resultados experimentales muestran que el algoritmo implementado en esta tesis es capaz de superar a otros propuestos en la literatura y desde el punto de vista multi-objetivo cumple con las métricas utilizadas para evaluarlo.

ABSTRACT

In recent decades, the multi-objective algorithms have captured significant attention. Job shop scheduling problems are one of the most complex scheduling problems and are commonly encountered in manufacturing industries. Most of the studies are based on optimizing one objective, excepting a reduced group of researchers that use the Pareto Front approach. Besides, generally, in these approaches, it is only possible to optimize two or three objectives which are predefined with anticipation. In this thesis, we implemented an Multi-objective and Multi-agent Reinforcement Learning algorithm based on the Pareto Front that optimizes the makespan, machines idle times, tardiness and robustness. The algorithm is generic because it allows to add a new objective if you desire, and for that it only necessary to include a new class. The proposed algorithm was applied to a set of instances to optimize different objectives and finally experimental results show that the algorithm implemented in this thesis is able to improve other algorithms proposed in the literature and it gives good results when multi-objective metrics are used to evaluate it.

TABLA DE CONTENIDO

Página

Índice de tablas

Tabla Página

Índice de figuras

Figura Página

INTRODUCCIÓN

Actualmente en nuestro país se busca alcanzar la eficiencia en todos los procesos que se desarrollan. A esto no se encuentran ajenas las industrias manufactureras, incluyéndose en este grupo también los centrales azucareros que han vuelto a funcionar de acuerdo a las necesidades de la nación. Para lograr la eficiencia deseada urge que los procesos que se desarrollan en estas industrias se automaticen.

En la gran mayoría de estas industrias los procesos económicos, de recursos humanos, entre otros ya han sido automatizados, sin embargo, la minoría cuenta con un proceso de producción automatizado. Tanto en los centrales azucareros como en empresas textiles y de manufactura el proceso de producción se desarrolla de acuerdo a un orden y con un conjunto de recursos limitados.

La programación en el tiempo de las tareas en las que se descomponen un conjunto de trabajos teniendo en cuenta que estas deben ser ejecutadas en varios recursos (máquinas) y que cada recurso (máquina) solamente puede ejecutar una tarea en un instante de tiempo se denomina problema de secuenciación de tareas. Cuando el conjunto de máquinas y de trabajos es pequeño se puede encontrar la mejor secuenciación analizando todas las combinaciones posibles, pero cuando estos conjuntos crecen se vuelve una tarea compleja. En estos problemas se busca encontrar aquella solución que optimice una función objetivo dada. Generalmente en la literatura la función objetivo más utilizada resulta minimizar el tiempo total de ejecución (C_{max}).

Los problemas de tipo Job Shop (JSSP, por sus siglas en inglés) constituyen los más conocidos e investigados debido al alto grado de dificultad que presentan y su aplicabilidad a una gran cantidad de situaciones reales. El JSSP tiene la característica de que los trabajos son unidireccionales y básicamente consta de un grupo de trabajos, donde cada uno tiene un conjunto de operaciones a ser procesadas en un conjunto de recursos limitados, a los

que se denomina máquinas. Cada trabajo tiene un orden en el que se deben ejecutar las operaciones, dichas operaciones tienen un tiempo de procesamiento en cada una de las máquinas y este no es modificable. Se trata de uno de los problemas de optimización combinatoria más difíciles de resolver. No solo es del tipo NP-Completo (Garey *et al.*, 1974; Rudy *et al.*, 2014), también es clasificado como uno de los más complejos dentro de los problemas que pertenecen a esta tipología.

Al pertenecer a la clase de los NP-Completos resolverlo mediante un algoritmo exacto no es una opción a medida que crece la dimensión del problema. Existen una gran cantidad de algoritmos que resuelven los JSSP pero la gran mayoría está enfocado hacia un solo objetivo, obviando el carácter multi-objetivo de este tipo de problemas.

Un algoritmo que resuelva problemas de secuenciación de tareas debe encontrar el tiempo de inicio de cada operación ($s_{ij} \geq 0$) para optimizar uno o varios objetivos.

Existen meta heurísticas para resolver los JSSP. Dentro de los algoritmos que mejores resultados brindan para estos problemas se encuentran los que usan la búsqueda tabú (Nowicki and Smutnicki, 2005). También son utilizadas otras meta heurísticas como recocido simulado, optimización mediante colonias de hormigas (ACO), algoritmos evolutivos, optimización mediante enjambre de partículas (PSO), entre otros. En (Bożejko *et al.*, 2009) se introduce un algoritmo de recocido simulado en paralelo, el cual emplea propiedades del problema asociadas a la teoría de bloqueo.

Cuando se trata con la optimización multi-objetivo un enfoque común consiste en combinar los objetivos dentro de una función de escalarización donde se optimiza la suma de estos usando un vector de pesos. Esta combinación no siempre muestra un buen desempeño debido a que se necesitan hacer varias iteraciones para obtener un buen resultado, además las prioridades entre los objetivos pueden variar. Un enfoque multi-objetivo adecuado permitiría encontrar la Frontera de Pareto o un conjunto de soluciones no dominadas.

Uno de los paradigmas más usados para resolver problemas de secuenciación de tareas con un enfoque multi-objetivo es el de los algoritmos evolutivos, principalmente los algoritmos genéticos, de estos se pueden citar varios ejemplos que han mostrado soluciones satisfactorias como son el Multi-objective Genetic Algorithm (MOGA) (Fonseca and Fleming, 1993) y el Modified micro Genetic Algorithm (MmGA) (Tan *et al.*, 2015).

En (Kachitvichyanukul and Sitthitham, 2011) se propone un algoritmo genético en dos etapas, el cual utiliza una función de escalarización con los siguientes objetivos: minimizar el tiempo de finalización de todos los trabajos (makespan), la tardanza y la antelación. Aunque se han obtenido resultados satisfactorios con el uso de algoritmos genéticos la configuración de los operadores resulta casi tan compleja como el problema a resolver .

También se ha usado ACO en la optimización multi-objetivo para este tipo de problemas (Udomsakdigool and Khachitvichyanukul, 2011) y en Lei (2008) y Sha and Lin (2010) se presentan algoritmos multi-objetivo usando PSO. En Suresh and Mohanasundaram (2006) se propone el algoritmo Pareto archived Simulated Annealing (PASA), uno de los algoritmos multi-objetivo que mejores resultados ha mostrado de acuerdo a la literatura. Este tiene como objetivos la optimización del makespan y la media de la suma de los tiempos de completamiento de cada trabajo. PASA utiliza el concepto de dominancia de Pareto.

Recientemente el Aprendizaje Reforzado (RL por sus siglas en inglés) ha recibido considerable atención. En Gabel and Riedmiller (2007) y Gabel and Riedmiller (2012), los autores sugieren y analizan la aplicación de RL para resolver JSSP. En estos trabajos se demuestra que interpretar y resolver este tipo de escenarios a través de Sistemas Multiagente y RL es beneficioso para obtener soluciones cercanas a las óptimas y que este enfoque puede muy bien competir con enfoques de solución alternativos. En Jiménez (2012) se hace uso del RL para resolver problemas Job Shop para un solo objetivo alcanzando buenos resultados. Uno de los algoritmos más utilizado de este paradigma es el Q-Learning el cual trabaja con una función estado-acción que brinda la utilidad esperada de tomar una acción a en el estado s y seguir una política óptima después de esto. Su principal utilidad

radica en la capacidad de comparar la utilidad esperada de las acciones disponibles sin un conocimiento del modelo del ambiente.

Como se ha descrito anteriormente los problemas de secuenciación de tareas han sido tratados en la literatura generalmente desde un punto de vista mono-objetivo obviándose así el carácter multi-objetivo de los mismos. De los algoritmos multi-objetivo que existen para resolver estos problemas una gran parte utilizan funciones de escalarización y solo unos pocos brindan el conjunto de soluciones no dominadas o Frontera de Pareto encontrada. Pero de manera general todos estos algoritmos fueron diseñados para optimizar objetivos específicos.

Por lo que surge como **problema de investigación** de esta tesis la necesidad de contar con un algoritmo multi-objetivo genérico que permita la optimización de los diferentes objetivos que se quieren optimizar en los problemas de secuenciación de tareas tipo Job Shop.

Para dar solución a este problema se plantea como **objetivo general** de este trabajo:

Desarrollar un algoritmo multi-objetivo genérico para la solución de problemas de secuenciación de tareas, específicamente los de tipo Job Shop, basado en la Frontera de Pareto y usando Aprendizaje Reforzado.

Este objetivo general se desglosa en los siguientes **objetivos específicos**:

1. Diseñar una arquitectura genérica que permita resolver problemas de secuenciación de tareas desde un enfoque multi-objetivo.
2. Implementar un algoritmo multi-objetivo basado en la Frontera de Pareto usando Aprendizaje Reforzado y Sistemas Multi-agente.
3. Comparar el algoritmo implementado con otros algoritmos de la literatura.
4. Evaluar el algoritmo usando métricas de la optimización multi-objetivo.

Para dar solución a estos objetivos específicos se plantean las siguientes **preguntas de investigación**:

1. ¿Por qué usar un enfoque basado en la Frontera de Pareto y no otro enfoque de los utilizados en la optimización multi-objetivo?
2. ¿Por qué utilizar el algoritmo Q-Learning y no otro algoritmo del Aprendizaje Reforzado?
3. ¿Cuáles son los principales objetivos a tener en cuenta para este tipo de problemas?

Como **hipotesis de investigación** se plantea que: contar con un algoritmo multi-objetivo genérico para la optimización multi-objetivo de los problemas de secuenciación de tareas tipo Job Shop facilitará la inclusión de nuevos objetivos a optimizar y de nuevos tipos de problemas.

Para lograr los objetivos trazados y demostrar la hipotesis de investigación se proponen las siguientes **tareas de investigación**:

- Análisis de los principales algoritmos multi-objetivo propuestos en la literatura para resolver problemas de secuenciación de tareas tipo Job Shop.
- Definir un método para la combinación del Aprendizaje Reforzado y el conjunto de soluciones no dominadas.
- Diseñar e implementar un algoritmo multi-objetivo para la solución de problemas de secuenciación de tareas tipo Job Shop usando Aprendizaje Reforzado.
- Analizar los resultados de la aplicación del algoritmo implementado a un conjunto de instancias de los problemas de secuenciación de tareas tipo Job Shop.

El **valor práctico** de este trabajo está dado por:

- Disponer de un algoritmo para resolver problemas de secuenciación de tareas tipo Job Shop desde un enfoque multi-objetivo, que permite la optimización de los principales objetivos que se buscan en este tipo de problemas y la inclusión de nuevos objetivos.
- Utilizar los resultados brindados por el algoritmo, el conjunto de soluciones no dominadas para una determinada instancia del problema, hace más fácil y cómoda su aplicación para los investigadores del área.

El **valor social** de este trabajo está dado por:

- Emplear el algoritmo implementado en las industrias manufactureras aumenta la eficiencia de las mismas, disminuye el esfuerzo de los trabajadores y permite implantar jornadas de trabajo más reales y que se adapten a las necesidades del proceso de producción.

La tesis está estructurada en tres capítulos. En el capítulo 1 se hace una breve descripción de los problemas de secuenciación de tareas y su clasificación. Además se introducen tres técnicas para manejar la robustez en este tipo de problemas y se mencionan algunos de los principales algoritmos multi-objetivo que existen para resolver estos problemas. En el capítulo 2 se explica como pueden ser utilizados los Sistemas Multi-agente y el Aprendizaje Reforzado como una vía para dar solución a los problemas de secuenciación de tareas. También se describe el algoritmo implementado así como la interfaz gráfica diseñada para interactuar con el algoritmo. Por último en el capítulo 3 se comparan los resultados obtenidos para tres de los objetivos a optimizar con otros algoritmos propuestos en la literatura. El algoritmo, además, es evaluado a través de dos métricas para la optimización multi-objetivo. La tesis culmina con conclusiones y recomendaciones.

CAPÍTULO 1

Capítulo 1
MARCO TEÓRICO

Este capítulo aborda todo lo referente a la base teórica que fundamenta esta investigación. Se hace una profunda revisión de la bibliografía para determinar las técnicas de optimización que han sido utilizadas para resolver problemas de secuenciación de tareas en configuraciones de tipo Job Shop. Además, se describen las diferentes técnicas que existen para implementar la robustez en los problemas de secuenciación de tareas , se hace una análisis de la optimización multi-objetivo, especialemente de la Frontera de Pareto, y por último se exponen los principales aspectos de los Sistemas Multi-Agente y el Aprendizaje Reforzado.

1.1. Problemas de secuenciación de tareas

Los problemas de secuenciación de tareas están presentes en muchos de los procesos que se llevan a cabo en la rutina diaria, en los cuales es necesario realizar una serie de actividades en un espacio de tiempo determinado y con recursos limitados, tratando siempre de cumplir todos los objetivos propuestos.

Los problemas de secuenciación de tareas son una subclase de problemas de satisfacción de restricciones que aparecen con frecuencia en entornos reales. Podemos encontrar problemas de esta familia en muchas áreas de la industria, la administración y la ciencia; problemas que van desde la organización de la producción hasta la planificación de multiprocesadores, industrias de semiconductores, asignación de puertas de embarque en un aeropuerto, o planificación de horarios ferroviarios (Fernández, 2011).

Los problemas de secuenciación de tareas pertenecen a la clase combinatoria, es decir, que se debe elegir una combinación dentro de un conjunto exponencialmente grande

de combinaciones, por lo que se hace necesario un algoritmo inteligente para encontrar una buena solución en un tiempo razonable. Estos problemas son clasificados como NP-Completos (Garey *et al.*, 1976), lo cual significa que encontrar una solución para este tipo de problemas resulta imposible de no usarse un algoritmo esencialmente enumerativo y el tiempo computacional aumenta exponencialmente según va creciendo el tamaño del problema. Dentro de los problemas de secuenciación de tareas se consideran los problemas de las industrias manufactureras entre los más difíciles de resolver. Para encontrar soluciones a estos se han aplicado diversidad de técnicas de Inteligencia Artificial con mayor o menor éxito.

1.1.1. Clasificación de problemas de secuenciación de tareas

Los problemas de secuenciación de tareas pueden ser caracterizados de acuerdo a tres clasificaciones: el ambiente de las máquinas, las características de los trabajos y la función objetivo (Herrmann *et al.*, 1993). Estas clasificaciones se conocen usualmente como clasificación $\alpha|\beta|\gamma$ (Graham *et al.*, 1979), donde α representa el ambiente de las máquinas (una o múltiples máquinas), β las características de los trabajos (restricciones conocidas) y γ el criterio a optimizar (el makespan es el criterio más optimizado en la literatura).

1. **Ambiente de las máquinas.**

   Cuando el problema cuenta con una sola máquina nos encontramos frente al problema más sencillo de resolver, en este caso todos los trabajos tienen una sola operación y tienen que pasar por esa máquina. Pero cuando los ambientes cuentan con muchas máquinas los problemas se vuelven más complicados, tanto si tienen máquinas de diferentes tipos como si tiene máquinas que trabajan en paralelo, siendo posible además que las velocidades de máquinas de un mismo tipo sean diferentes. Cuando cada trabajo tiene un número fijo de operaciones que requieren pasar por diferentes máquinas se dice que estamos en presencia de un problema shop, y de acuerdo a las restricciones que presentan pueden ser clasificados como:

- Open shop: hay m máquinas y los trabajos deben ser procesados en cada una de ellas. No existen restricciones de orden entre las operaciones de cada trabajo, lo cual significa que los trabajos pueden tomar diferentes rutas.
- Job Shop: hay m máquinas y los trabajos tienen que ser procesados por cada una de ellas pero de acuerdo a un orden específico. Cada trabajo tiene un orden determinado.
- Flow Shop: hay m máquinas y los trabajos deben pasar por todas ellas en el mismo orden.
- Job Shop Flexible: es una generalización del job shop en un ambiente de máquinas paralelas donde cada operación puede ser procesada por un conjunto de posibles máquinas con diferentes velocidades.

2. **Características de los trabajos.**

   Cada problema tiene características que deben cumplir los trabajos. A continuación se listan las características más usadas:

   - *Preempción*: se refiere a los ambientes donde el procesamiento de un trabajo puede ser interrumpido y retomado más tarde, incluso en otra máquina.
   - *Restricciones de precedencia*: los trabajos pueden tener restricciones de precedencia, es decir, que un trabajo no pueda comenzar antes que otro haya terminado.
   - *Fecha de Inicio*: un trabajo no puede comenzar a procesarse hasta una fecha determinada.
   - *Fecha de finalización*: un trabajo debe ser completado antes de una fecha determinada.
   - *Tiempos de configuración*: son los tiempos que necesita una máquina entre la ejecución de dos trabajos para estar lista.

3. **Funciones objetivo.**

   La función objetivo a optimizar es el tercer elemento que describe a estos tipos de problemas. Las funciones objetivo típicas optimizan alguno o algunos de estos objetivos:

- Makespan (C_{max}): tiempo total de ejecución ($\max_{i\in\{1,n\}} C_i$, donde C_i representa el tiempo de finalización del trabajo i).
- Inactividad (I_Σ): Suma de los tiempos de inactividad de todas las máquinas $I_\Sigma = \sum_{i\in\{1,\dots,m\}} (TF_i - \sum_{j\in\{1,\dots,n\}} p_{ij})$, donde $TF_i = \max_{j\in\{1,\dots,n\}} (s_{ij} + p_{ij})$ y p_{ij} es el tiempo de procesamiento de la operación i del trabajo j.
- Flow time total (F_Σ): Suma de los tiempos de finalización de todos los trabajos $F_\Sigma = \sum_{i\in\{1,\dots,n\}} C_i$
- Tardanza total (T_Σ): Suma de las tardanzas de todos los trabajos $T_\Sigma = \sum_{i\in\{1,\dots,n\}} T_i$, donde $T_i = \max(0, C_i - K_i)$ y K_i es el tiempo esperado de finalización para el trabajo i.
- Antelación total (E_Σ): Suma de los tiempos de anticipación de todos los trabajos $E_\Sigma = \sum_{i\in\{1,\dots,n\}} E_i$, donde $E_i = \max(0, C_i - K_i)$

4. **Otros criterios.**

   Los problemas de secuenciación de tareas también pueden ser clasificados como deterministas o estocásticos. Cuando todos los parámetros son conocidos y no hay presencia de incertidumbre nos encontramos ante un problema perteneciente a la clase determinista. En los problemas estocásticos generalmente los tiempos de procesamiento de los trabajos son modelados con variables aleatorias.

   También es posible clasificar estos problemas de acuerdo a la forma en que fue construida la secuenciación. Cuando la secuenciación se construye paso a paso, hasta que todas las tareas son secuenciadas, se está en presencia de un método constructivo. Los métodos de reparación iterativa modifican una secuenciación que está completamente construida para eliminar los conflictos que presenta u optimizar la solución. Los métodos constructivos son los más utilizados.

1.1.2. Problemas de secuenciación Job Shop

Uno de los problemas más conocido de la industria manufacturera es el JSSP, el cual involucra a un conjunto de trabajos y máquinas para construir la mejor secuenciación, es

decir, la mejor distribución de las operaciones en las máquinas en intervalos de tiempo determinados de manera que se minimice la duración requerida para completar todos los trabajos (esto es en el caso en que se desee minimizar el makespan). El número total de combinaciones para un problema con n trabajos y m máquinas es $(n!)^m$. Este tipo de problemas es ampliamente reconocido en la literatura como uno de los de mayor dificultad de los pertenecientes a la clase NP-Hard (Garey *et al.*, 1976). Los métodos exactos para resolver este tipo de problemas fallan cuando las dimensiones del problema crecen al no ser capaces de encontrar soluciones debido al tamaño del problema. Por esto se necesita para resolverlos utilizar métodos que devuelvan soluciones adecuadas aunque no necesariamente óptimas.

El problema general de Job Shop constituye un interesante desafío ya que incluso para instancias pequeñas del problema se puede volver extremadamente difícil hacer progresos hacia una solución. Los enfoques clásicos para resolver estos problemas son Ramas y Cotas, Recocido Simulado y Búsqueda Tabú, aunque en los últimos tiempos también se han utilizado las meta heurísticas Algoritmos Genéticos, Optimización mediante Colonia de Hormigas (ACO) y Optimización mediante Enjambre de Partículas (PSO), además del Aprendizaje Reforzado (RL).

La definición de un problema de secuenciación de tareas tipo Job Shop fue dada por French (1982). Dado un conjunto M de m máquinas que solo pueden procesar un trabajo por instante de tiempo y un conjunto J de n trabajos que deben ser procesados por cada una de estas máquinas en un orden determinado y que depende del trabajo, se desea encontrar una solución factible que minimice el tiempo de completamiento de los trabajos. Cada trabajo J_j está compuesto de m operaciones que tienen un orden determinado y que debe ser respetado (restricciones de precedencia) (Zhang and Dietterich, 1996).

Existen $N = n*m$ operaciones, o_{ij}, donde i representa la máquina y j el trabajo, cada operación debe ser procesada en un tiempo ininterrumpido p_{ij}. El flujo de cada trabajo en las máquinas es independiente de los demás trabajos. En un instante dado, cada máquina

solo es capaz de ejecutar un trabajo y cada trabajo solo puede ser ejecutado por una máquina.

El objetivo es determinar para cada operación o_{ij} el instante de tiempo $s_{ij} >= 0$ donde comienza su ejecución, de manera que se minimice el makespan (u otros objetivos) cumpliéndose todas las restricciones del problema.

A continuación se listan las principales restricciones de los JSSP:

- No se puede procesar más de una operación de un trabajo en el mismo instante de tiempo.
- No se permite interrumpir la ejecución de las operaciones (no preempción).
- Ningún trabajo puede ser procesado dos veces en la misma máquina.
- Cada trabajo debe ser procesado hasta completarse.
- Los trabajos pueden comenzar en cualquier instante de tiempo (no existe fecha de inicio).
- Los trabajos pueden terminar en cualquier instante de tiempo (no existe fecha de entrega).
- Las máquinas no pueden procesar más de una operación a la vez.
- Existe solo una máquina de cada tipo.
- Las máquinas pueden estar inactivas en cualquier instante de tiempo durante la secuenciación.
- Los trabajos deben esperar que la próxima máquina donde se tienen que procesar esté disponible.
- El orden por el que los trabajos pasan por las máquinas es conocido con anticipación y es inamovible.

1.1.3. Principales Algoritmos para la resolución de problemas de secuenciación de tareas

La Investigación de Operaciones ofrece diferentes enfoques con el objetivo de resolver problemas de secuenciación de tareas, tanto métodos de Programación Lineal, de Programación Dinámica o de Programación Entera. Estos enfoques solo son posibles utilizarlos cuando el tamaño del problema no es demasiado grande.

La mayoría de los problemas de secuenciación de tareas son NP-Hard y el tamaño no suele ser pequeño, por lo que estos métodos fallan al brindar la solución óptima en un tiempo computacional razonable. Debido a esto, los métodos meta heurísticos reciben toda la atención al ser capaces de obtener buenas soluciones de manera eficiente. La Inteligencia Artificial se convierte desde los años 80 en una poderosa herramienta para resolver problemas de secuenciación de tareas de la vida real (Zhang and Dietterich, 1996). La Búsqueda Tabú como se mencionó anteriormente fue adapatada para aplicarla a este tipo de problemas en Nowicki and Smutnicki (2005). En (Bożejko *et al.*, 2009) se introduce un algoritmo de recocido simulado en paralelo, el cual emplea propiedades del problema asociadas a la teoría de bloqueo. Otras meta heurísticas como ACO (Wang *et al.*, 2004) y PSO (Sha and Hsu, 2006) también han sido utilizadas.

1.2. Incertidumbre y Robustez

Durante el proceso de secuenciación, como se ha mencionado anteriormente, pueden ocurrir diversas interrupciones, las máquinas pueden romperse, las operaciones pueden tardar más de lo esperado, nuevas órdenes pueden llegar o la prioridad de las operaciones puede variar. Todas estas interrupciones eventualmente puede provocar fallas en el sistema, causando retrasos en la entrega de los productos (las fechas de finalización pueden ser violadas), el aumento de los tiempos de espera de la máquinas, entre otras consecuencias.

Por lo tanto, en entornos reales no sería muy útil gastar demasiados recursos ni esfuerzos en producir soluciones óptimas ya que se consigue la optimalidad solo si la solución puede ser ejecutada como estaba previsto. Por el contrario, una solución sub-óptima al

problema que sea capaz de tener cierta flexibilidad para hacer frente a acontecimientos imprevistos, podría proporcionar características útiles. Por lo que, la eficiencia de las técnicas de secuenciación empleadas dependerá de la capacidad de adaptación al grado de incertidumbre de la información proporcionada al comienzo de esta (Policella, 2005).

En muchos procesos de decisión, resulta común que las soluciones tengan un cierto nivel de robustez con el fin de mantener su viabilidad en entornos incompletos o con datos imprecisos. Las soluciones deben tolerar un cierto grado de incertidumbre durante la ejecución, en otras palabras, deben ser capaces de absorber variaciones dinámicas en el problema. A menudo hablamos de problemas de secuenciación robustos, pero hay varias cuestiones que tienen que ser respondidas con el fin de captar realmente el significado de la robustez, por ejemplo:

¿Cómo se define la robustez?

¿Cómo podemos medir la robustez en un problema de secuenciación?

¿Cómo incorporar la robustez en los problemas de secuenciación?

El desarrollo de optimizaciones robustas fue iniciado por Soyster (1973). Este enfoque ha sido ampliamente estudiado y extendido aunque se consideró demasiado conservador, debido a que las soluciones obtenidas sacrificaban demasiado la optimalidad del problema en cuestión con el fin de asegurar la robustez (Bertsimas and Sim, 2004). Después de este trabajo inicial se propusieron algunos otros modelos (Ben-Tal and Nemirovski, 2000; Bertsimas and Sim, 2004; Ma *et al.*, 2008). En todos ellos la principal preocupación radicaba en cómo hacer que los problemas de secuenciación fueran más robustos sin perder demasiado en optimalidad (Jiménez, 2012).

La robustez es fácil de definir, pero compleja de medir de forma cuantitativa. En Sabuncuoglu and Goren (2009) los autores presentan un estudio detallado sobre el tema, el cual muestra que los distintos enfoques se centran en diferentes objetivos por lo que, la forma de medir la robustez no resulta siempre la misma. Las formas más comunes de medir la robustez de acuerdo con la literatura son las siguientes:

- desviación de la planificación original;
- el costo real de la ejecución de la planificación;
- estabilidad, relacionada con el criterio de desempeño;
- número de cambios requeridos para arreglar la planificación.

Las dos primeras medidas fueron propuestas en Gao (1996). Según este autor, la desviación de la planificación original se puede calcular mediante el análisis de la diferencia entre el makespan planificado y el makespan real. En el segundo caso, los costos pueden estar asociados, por ejemplo, a llegadas tardías, lo que significa que hay una penalización por unidad de tiempo programado sobre la fecha de culminación o los costos podrían estar asociados a la ociosidad de los recursos, lo que significa que hay una penalización por cada unidad de tiempo que el recurso se mantuvo inactivo (Jiménez, 2012).

En el caso de la estabilidad (tercera métrica), podemos decir que es muy similar a la del primer indicador. De acuerdo con su definición en Billaut *et al.* (2013), la estabilidad constituye una medida de la diferencia en la secuencia entre las dos planificaciones, que en otras palabras es cómo se desvía el horario real del original.

La esencia de la última métrica, que se introdujo en Gomes (2000), consiste en que: dado un conjunto C de los cambios en la formulación inicial de la instancia del problema, una solución A es más robusta que la solución B en lo que respecta al conjunto C si el número de cambios necesarios para fijar la solución A es menor que el número de cambios necesarios para fijar la solución B (Jiménez, 2012).

Existen diferentes enfoques que se pueden utilizar con el fin de hacer frente a la incertidumbre y la solidez en la planificación , ya sea la búsqueda de soluciones que se adapten dinámicamente a los cambios o que incorporen los conocimientos disponibles sobre los posibles cambios en la solución (Herroelen and Leus, 2005).

1.2.1. Métodos proactivos y métodos reactivos

De acuerdo con la literatura consultada, hay dos maneras de lidiar con la incertidumbre en entornos de secuenciación, la primera es utilizar enfoques proactivos (también

llamados predictivos) y la segunda utilizar enfoques reactivos. El objetivo de la programación proactiva consiste en tomar en cuenta las posibles eventualidades al construir la solución, esto permite que la planificación sea más robusta, pues valora el conocimiento estadístico sobre la incertidumbre (Davenport, 2001). Por otro lado, los métodos reactivos reoptimizan una solución cuando ocurre un evento inesperado, por tanto, las decisiones se toman en tiempo real y se basan en información actual sobre el sistema. Estos enfoques se utilizan cuando el nivel de incertidumbre es significativo o cuando los datos no están disponibles en tiempo, lo cual implica que no es posible calcular una solución de forma predictiva (Jiménez, 2012).

La programación proactiva también se ha definido como las técnicas que tratan de producir una secuencia que es robusta y flexible a los eventos que ocurren en tiempo de ejecución (Beck and Wilson, 2007). La utilidad de este tipo de enfoque depende de si la incertidumbre se puede cuantificar de alguna manera (como saber el tiempo medio entre fallos de las máquinas, etc.). Si la información está disponible, pueden ser utilizadas las técnicas proactivas. Si por el contrario, el grado de incertidumbre es muy alto, se necesita un enfoque más reactivo (Jiménez, 2012).

La programación reactiva implica revisar o reoptimizar una planificación cuando se produce un suceso inesperado. Enfoques completamente reactivos se basan en la información puesta al día sobre el estado del sistema (Davenport and Beck, 2000). Las decisiones se toman en tiempo real, basadas en las reglas de secuenciación prioritarias. Estos enfoques se utilizan cuando el nivel de perturbaciones es siempre significativo o cuando los datos se conocen muy tarde, por lo que predecir los horarios resulta imposible (Aloulou and Portmann, 2005). La figura 1–1 muestra una representación gráfica de las propiedades de ambos tipos de enfoques.

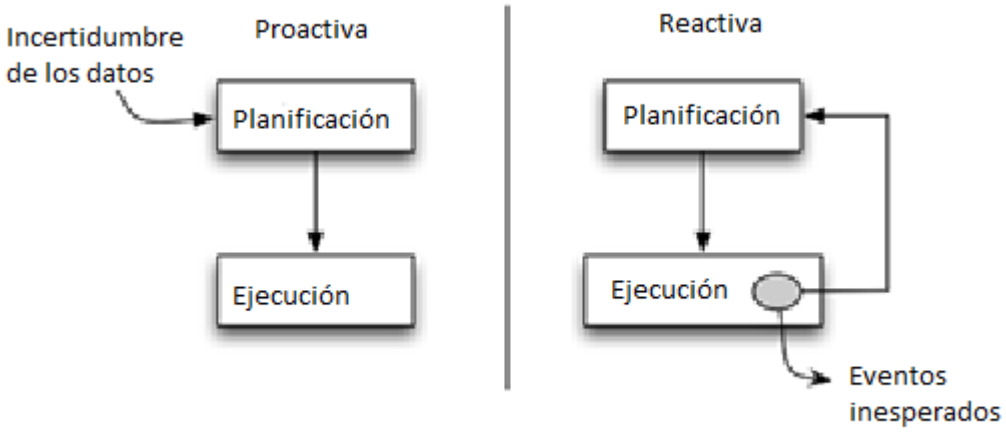

Figura 1–1: Enfoque proactivo vs reactivo

1.2.2. Técnicas para planificaciones robustas

Según la bibliografía revisada las técnicas más usadas para hacer planificaciones robustas son las basadas en reajustes en los tiempos de terminación de las operaciones. Las mismas consisten en darle a cada actividad un tiempo extra para poder manejar en ese tiempo posibles percances sin necesidad de una replanificación. Estas técnicas se basan en el conocimiento que se tiene de las máquinas (cuántas veces se ha roto una máquina, la frecuencia con que se rompe, etc.), por lo que son técnicas proactivas. A continuación se describen tres técnicas para hacer planificaciones más robustas:

- Protección Temporal: fue propuesta por Chiang and Fox (1990) y se basa en la idea de construir la planificación teniendo en cuenta el historial o conocimiento previo de las máquinas. A los recursos cuya probabilidad de romperse es distinta de cero se les llama recursos rompibles. La duración de todas las actividades que involucran recursos rompibles se extiende con el objetivo de tener tiempo extra para tratar las posibles roturas y entonces se resuelve el problema de secuenciación con las técnicas que se utilizan habitualmente (Gao, 1996). Un ejemplo de esta técnica se obtiene de Davenport (2001) y se expone en la figura 1–2. La figura muestra dos actividades, A y B, las cuales tienen que pasar por recursos rompibles. Los rectángulos blancos representan el tiempo de procesamiento original, y los grises el tiempo extra que se le añadió a

cada actividad de acuerdo a la técnica de Protección Temporal. Si el recurso se rompe cuando está ejecutando la actividad A entonces ese tiempo extra puede ser empleado para arreglarlo. Si no se necesita más tiempo que el extra, entonces, la planificación sigue como se había planificado. De necesitarse más tiempo se requiere hacer algunos ajustes con el fin de mantener el resto de las actividades en sus tiempos (replanificar). De no existir ninguna rotura en la ejecución de la actividad A entonces la actividad B puede empezar a ejecutarse antes y el tiempo extra planificado en la A puede ser adicionado a la actividad B. La cuestión más importante para este tipo de técnicas es la cantidad de tiempo extra que se va a adicionar por cada actividad. Si protegemos mucho la planificación (mucho tiempo extra) podríamos tener una planificación de poca calidad, pero altamente robusta. Por el contrario, si la protegemos muy poco resulta una planificación de muy baja calidad en caso de que se rompa un recurso durante la ejecución de una actividad.

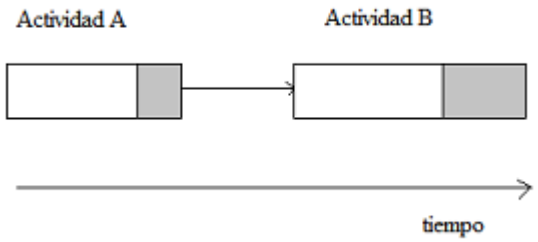

Figura 1–2: Ejemplo de dos operaciones consecutivas ejecutadas sobre un recurso rompible

El enfoque presentado en Gao (1996) propone una fórmula para calcular cuánto tiempo extra debe asignársele a una actividad. La fórmula es la siguiente:

$$p_{ij}ext = p_{ij} + \frac{p_{ij}}{F} * D$$

Donde, p_{ij} es el tiempo de procesamiento original de la actividad, F representa el tiempo esperado entre los fallos de las máquinas y D es la duración de la rotura, $p_{ij}ext$ es el tiempo de procesamiento extendido, que no es más que la adición del tiempo original y el tiempo que se va a agregar, $\frac{p_{ij}}{F}$ da el número de roturas que se

esperan durante la ejecución y $\frac{p_{ij}}{F} * D$ el tiempo extra que se le debe adicionar debido a las roturas de las máquinas.

- Ventana de Tiempos de Inactividad (TWS): hay situaciones en las cuales no es posible aplicar la técnica de Protección Temporal debido a que no es posible compartir el tiempo extra agregado. Para evitar estas situaciones se cambia la forma de agregar el tiempo extra a las operaciones de manera que sea seguro que la planificación tiene el suficiente tiempo extra para cada actividad. El tiempo extra adicionado por esta técnica es mucho mayor que en la Protección Temporal. La cantidad de tiempo extra agregado a cada actividad por este enfoque es igual a la suma de la duración de todas las roturas esperadas en el recurso R, esto se debe a que este enfoque se basa en que el tiempo extra de todas las actividades sobre cada recurso sea compartido.

  A continuación se muestra la fórmula utilizada por este enfoque para calcular el tiempo extra que se va a adicionar.

$$slack_A \geq \frac{\sum_{B \in acts_R} p_{ij}}{\mu_{tbf}(R)} * \mu_{dt}(R)$$

  donde p_{ij} representa la duración de la actividad, $\mu_{tbf}(R)$ es el tiempo medio entre los fallos del recurso R, $\mu_{dt}(R)$ representa el tiempo medio que duró la rotura del recurso R y el conjunto de actividades que pueden ser ejecutadas por el recurso R está dado por $acts_R$.

  La figura 1–3 muestra la manera en que este enfoque adiciona el tiempo extra a cada actividad asumiendo que el tiempo extra de todas las actividades sobre cada recurso debe ser compartido.

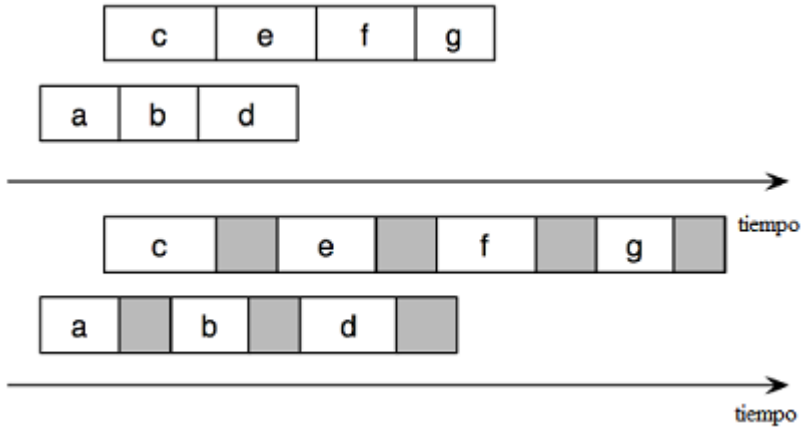

Figura 1–3: Ejemplo de cómo se adiciona tiempo extra con el enfoque TWS

- Ventana de Tiempos de Inactividad Enfocada (FTWS): la manera en que se colocan las actividades en la planificación se toma en cuenta con el objetivo de decidir si se necesitará agregar tiempo extra. Este enfoque tiene en cuenta que si llega una máquina nueva no tiene sentido agregar tiempo extra desde la primera operación que se va a ejecutar porque en ese momento la probabilidad de romperse es muy baja. Es decir, agregar o no tiempo extra se basa en distribuciones conocidas.

  La función utilizada por este enfoque es la probabilidad de que la rotura ocurra antes o durante la ejecución de la operación.

  La figura 1–4 muestra cómo se agrega tiempo extra de acuerdo a este enfoque cuando se trabaja sobre una máquina nueva. Como puede observarse la operación C por ser la primera en ejecutarse no tiene tiempo extra.

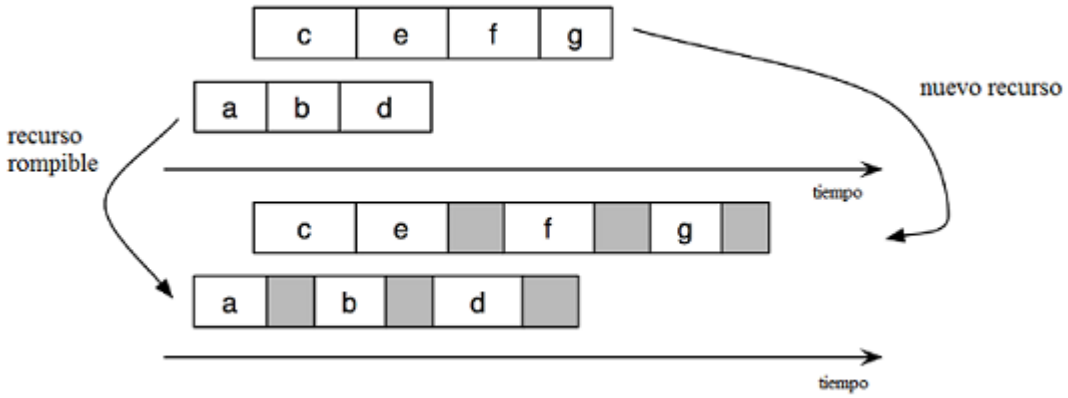

Figura 1–4: Ejemplo de cómo se adiciona tiempo extra en el enfoque FTWS

1.3. Optimización Multi-objetivo

La mayor parte de los problemas de optimización del mundo real son multi-objetivo ya que suelen tener dos o más funciones objetivo que deben satisfacerse simultáneamente y que por lo general entran en conflicto entre sí, es decir, la mejora de un objetivo implica el empeoramiento de otros. Sin embargo, a fin de simplificar su solución, muchos de estos problemas tienden a modelarse como mono-objetivo, combinando los objetivos dentro de una función de escalarización donde se optimiza la suma de estos usando un vector de pesos. Por otro lado, un alto número de meta-heurísticas para resolver problemas multi-objetivo han sido propuestas en los últimos años con el fin de obtener un conjunto de soluciones bastante cercano al óptimo, y sin la necesidad de convertir el problema a uno de un solo objetivo. La mayoría de ellas están limitadas a Frentes de Pareto con ciertas características (p.ej., convexos) y suelen requerir un punto inicial de búsqueda (Coello *et al.*, 2002).

Formalmente en la optimización multi-objetivo se desea encontrar un vector de decisión $x^* = \{x_1^*, x_2^*, \ldots, x_n^*\}^T$ que minimize la función

$$F(x) = \{f_1(x), f_2(x), \ldots, f_n(x)\}$$

$$x \in S$$

donde los f_i son las funciones objetivo a optimizar y S es el conjunto de restricciones.

La optimización multi-objetivo difiere de la optimización de un solo objetivo en muchas formas. Para dos o más objetivos en conflicto, a cada objetivo le corresponde una solución óptima diferente, pero ninguna de estas soluciones es óptima para todos los objetivos a la vez. Por lo que la optimización multi-objetivo no trata de encontrar una solución óptima, sino que busca un equilibrio entre todas las soluciones. La optimización multi-objetivo tiene dos intenciones, la primera es encontrar un conjunto de soluciones lo más cercano posible a la Frontera Óptima de Pareto y la segunda intención, es encontrar un conjunto de soluciones lo más diverso posible (Garen, 2002).

1.3.1. Frontera de Pareto

La primera decisión que debe ser valorada al tratar con un problema de optimización multi-objetivo es precisamente cómo combinar la búsqueda y los procesos de toma de decisiones, de esto se ocupa la Optimalidad de Pareto.

En este enfoque, un vector que contiene todos los valores objetivos representa la aptitud de la solución y el concepto de dominancia se utiliza para establecer la preferencia entre las soluciones. Una solución x se dice que es no dominada si no hay otra solución que sea mejor que x en todos los criterios. Formalmente, la relación de dominancia se describe de la siguiente forma (Coello *et al.*, 2002):

Un vector $u = (u_1, ..., u_k)$ domina a otro $v = (v_1, ..., v_k)$ (denotado mediante $u \preceq v$) si y sólo si u es parcialmente menor a v, i.e., $\forall i \in \{1, ..., k\}, u_i \leq v_i \wedge \exists j \in \{1, ..., k\} : u_j < v_j$.

Dado un problema multi-objetivo con $F(x) = \{f_1(x), f_2(x), \ldots, f_n(x)\}$ el conjunto de óptimos de Pareto P^* se define como:

$$P^* := \{x \in \Omega \mid \nexists\, x' \in \Omega : F(x') \preceq F(x)\}.$$

Para un problema multi-objetivo el Frente de Pareto (PF^*) se define como:

$$PF^* := \{u = (f_1(x), \ldots, f_k(x)) \mid x \in P^*\}.$$

La definición anterior considera el problema de minimización, pero se puede definir de manera análoga para problemas de maximización. Es importante tener en cuenta que el tipo de dominancia puede tener efecto sobre la manera en que se realizará la búsqueda. Esto es porque si una solución está estrictamente dominada significa que fue superada por la otra solución en todos los criterios, mientras que si la solución está dominada vagamente significa que solo fue superada en algunos de los criterios, pero es tan buena como la otra solución en al menos uno de los objetivos.

El Frente Óptimo de Pareto (o Frontera de Pareto) constituye el conjunto de todas las soluciones no dominadas en el espacio multi-objetivo. Sin embargo, cuando las soluciones

en el conjunto obtenido no se encuentran en la Frontera de Pareto, debemos referirnos a ese conjunto como el frente no dominado obtenido o el Frente de Pareto conocido. La optimización de Pareto se refiere a la búsqueda de este frente o un conjunto que representa una buena aproximación a él y usualmente los métodos de optimización multi-objetivo se basan en técnicas para la Optimización de Pareto.

El atractivo de la Optimización de Pareto viene del hecho de que en la mayoría de los problemas de optimización multi-objetivo no existe una única mejor solución o óptimos globales y también de que es muy difícil establecer preferencias entre los criterios antes de la búsqueda. Y aún, si esto es posible, puede que estas preferencias cambien.

Dado que en la Optimización de Pareto el resultado final debe ser un conjunto de soluciones no dominadas, otro aspecto importante a considerar es la forma de evaluar la calidad del frente no dominado obtenido. Existen varios criterios para evaluar que tan bueno fue el frente obtenido entre los cuales se encuentran:

- El número de soluciones no dominadas obtenidas.
- La cercanía entre el frente obtenido y el Frente Óptimo de Pareto.
- El cubrimiento del frente, es decir, la difusión y distribución de las soluciones no dominadas (Silva *et al.*, 2004).
- El hipervolumen (Coello *et al.*, 2002).

1.3.2. Algoritmos multi-objetivo para problemas de secuenciación de tareas

Muchos problemas de la vida real, como el JSSP, persiguen varios objetivos a la vez, por lo que han hecho de la optimización multi-objetivo un área interesante de investigación.

Uno de los paradigmas más usados ha sido el de los algoritmos evolutivos, principalmente los algoritmos genéticos, de estos se pueden citar varios ejemplos que han mostrado soluciones satisfactorias como son el Vector Evaluated Genetic Algorithm (VEGA) (Schaffer, 1985), el Multi-objective Genetic Algorithm (MOGA) (Fonseca *et al.*, 1993), el Niche Pareto Genetic Algorithm (NPGA) (Zitzler, 1999), el Pareto-Archived Evolutionary Strategy (PAES) (Knowles and Corne, 2000), Pareto converging genetic algorithm (PCGA)

(Kumar and Rockett, 2002) y por último el Archive-Based Steady-State Micro Genetic Algorithm (ASMiGA) (Nag *et al.*, 2015).

También se encuentran algunas meta-heurísticas enfocadas a resolver problemas multi-objetivo como son Multi-objective Tabu Search (MOTS) (Hansen, 1997), Pareto Simulated Annealing (PSA) (Czyżżak and Jaszkiewicz, 1998), Memetic Pareto Archived Evolutionary Strategy (M-PAES) (Knowles and Corne, 2000), entre otros (Silva *et al.*, 2004). También la meta heurística ACO ha sido utilizada para la optimización multi-objetivo en problemas Job Shop (Rudy *et al.*, 2014).

1.4. Sistemas Multi-Agente y Aprendizaje Reforzado

El aprendizaje denota cambios en un sistema que permiten hacer la misma tarea o tareas derivadas de manera más eficiente y eficaz la próxima vez (Simon and Lea, 1974). El Aprendizaje Reforzado resulta un área del aprendizaje automático inspirada en la psicología conductista, cuyo objetivo es determinar qué acciones debe escoger un agente en un ambiente dado con el fin de obtener una recompensa que le diga que tan buena fue la acción seleccionada y aprender a resolver una tarea específica mediante continuas interacciones. A pesar de que los agentes se utilizan en una gran variedad de operaciones hoy día, actualmente no existe un acuerdo en la definición de este término. De todas las definiciones encontradas en la literatura citamos en esta tesis la propuesta en Jennings and Wooldridge (1998), donde se describen los requerimientos que un agente necesita satisfacer en el contexto de nuestro trabajo.

"*Un agente es un sistema automatizado situado en algún ambiente, que es capaz de una acción autónoma flexible en este entorno con el fin de cumplir sus objetivos de diseño*".

De acuerdo a esta definición cabe destacar que no es necesario un conocimiento del ambiente por parte del agente, ni objetivos prediseñados, tampoco es necesario conocer la manera de alcanzarlos. Los agentes inteligentes fueron definidos en Wooldridge and Jennings (1999) como agentes que deben operar en ambientes abiertos, impredecibles y con cambios inesperados, donde hay una alta probabilidad de que las acciones fallen.

De acuerdo a Wooldridge and Jennings (1995) existen tres características que un agente inteligente debe poseer para alcanzar sus objetivos:

- reactividad: los agentes inteligentes son capaces de percibir su ambiente, y responder en un tiempo adecuado a los cambios que ocurren en él;
- proactividad: los agentes inteligentes son capaces de tomar la iniciativa y exhibir un comportamiento directo a la meta;
- habilidad social: los agentes inteligentes son capaces de interactuar con otros agentes (incluso humanos).

Si existe la posibilidad de garantizar que el ambiente sea fijo entonces es relativamente fácil diseñar un agente con un comportamiento directo a la meta para operar en él. Sin embargo, el mundo real no es estático y la posibilidad de fallo debe ser tenida en cuenta. Un agente reactivo es el que mantiene una interacción constante con su ambiente y responde a los cambios que ocurren en él.

Un agente ideal debe ser tanto reactivo como proactivo, por lo tanto es necesario encontrar el balance entre estas dos características. También es muy importante que los agentes interactúen con otros agentes situados en el mismo ambiente con el fin de alcanzar los objetivos propuestos, aquí cumple un rol determinante la habilidad social.

Con el objetivo de aprender de la experiencia los agentes deben ser entrenados, por ejemplo, mediante aprendizaje supervisado. La meta del aprendizaje supervisado es inducir una política general mediante los ejemplos del entrenamiento, el cual le permite tratar con ejemplos no vistos.

El Aprendizaje Reforzado puede ser aplicado a problemas en los cuales no se tiene conocimiento del dominio o es demasiado costoso de obtener Moriarty *et al.* (1999). Tampoco esta técnica requiere conocimiento a priori de cuáles son las decisiones correctas o incorrectas, consecuentemente, un agente debe explorar activamente su ambiente con el objetivo de observar los efectos de sus acciones, donde por cada acción tomada se recibe una señal numérica que representa cuán buena es la acción. Esta interacción, prueba y

error con el ambiente, es más adecuada para aplicarla a los problemas de secuenciación de tareas.

1.4.1. Sistemas Multi-Agente

En un Sistema Multi-Agente varios agentes actúan en el mismo ambiente para cumplir con una tarea específica. El creciente interés en el uso de este tipo de sistemas en problemas del mundo real viene dado por la habilidad de resolver problemas muy grandes por un agente centralizado, y también por la habilidad de obtener soluciones donde la experiencia es distribuida, como por ejemplo, problemas de secuenciación en ambientes de manufactura (Sycara, 1998). El campo de los sistemas multi-agente se centra en procesos descentralizados (sistemas distribuidos), ya que cada agente del sistema tiene su propia percepción (pueden estar en ubicaciones distintas y ser responsables de diferentes partes del sistema), control (diferente experiencia) y forma de actuar (diferentes acciones potenciales) (Kalech and Kaminka, 2005). Todo esto se resume en la siguiente definición:

"*Un Sistema Multi-Agente es una red de agentes que trabajan juntos para resolver problemas que están más allá de las capacidades individuales o el conocimiento de cada agente*"(Jennings and Wooldridge, 1998)

Los Sistemas Multi-Agente se caracterizan porque cada agente tiene un punto de vista limitado, no existe un control global del sistema, los datos pueden ser descentralizados y el cálculo puede ser asíncrono (Jiménez, 2012). Existen dos posibles escenarios cuando se trabaja con múltiples agentes, ellos pueden trabajar juntos tratando de alcanzar un objetivo común, o pueden tener sus propios objetivos e intereses que entran en conflicto, lo que significa que puede darse que sean aprendices independientes o en conjunto. Cuando se utilizan aprendices en conjunto se asume que las acciones que toman los otros agentes pueden ser observadas. Los aprendices independientes no necesitan observar las acciones tomadas por los otros agentes.

1.4.2. Aprendizaje Reforzado

RL es aprender qué hacer (cómo mapear situaciones a acciones) para maximizar una recompensa. El aprendiz debe descubrir qué acciones conllevan una mayor recompensa y tomarlas. En los problemas de secuenciación de tareas las acciones no solo pueden afectar la recompensa inmediata sino que también pueden tener consecuencias sobre la próxima acción. En el modelo estándar del Aprendizaje Reforzado, un agente está conectado a su ambiente vía percepción y acción, como se muestra en la figura 1–5. En cada paso el agente percibe del ambiente el estado actual s y entonces selecciona una acción a para cambiar este estado. Este cambio genera una señal de recompensa r, la cual es recibida por el agente. La tarea del agente consiste en aprender una determinada política para la elección de las acciones en cada estado y así recibir la mayor acumulación de recompensa.

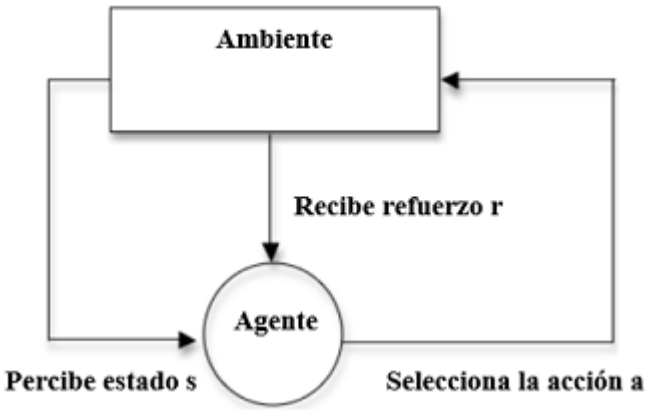

Figura 1–5: Modelo del Aprendizaje Reforzado

Formalmente el modelo de Aprendizaje Reforzado está compuesto por:

- un conjunto de estados S
- un conjunto de acciones A
- un conjunto de recompensas escalares en $\Re$
- una función de transición T

En un instante de tiempo t, el agente percibe el estado s_t que pertenece a S y el conjunto de posibles acciones asociadas a ese estado $A(s_t)$. El agente elige una acción a que pertenece a $A(s_t)$ y recibe una recompensa r_{t+1}, esto significa que el agente implementa

una correspondencia entre los estados y las probabilidades de seleccionar cada posible acción. Esta correspondencia es la política del agente que se denota π_t , donde $\pi_t(s, a)$ es la probabilidad de que $a_t = a$ si $s_t = s$.

La función de recompensa define el objetivo en un problema de Aprendizaje Reforzado. Esta función mapea cada estado percibido (o par estado-acción) del ambiente a un solo valor numérico, una recompensa, indicando que tan bueno es tomar esa acción en ese estado. La función de recompensa define cuáles eventos son buenos y cuáles malos para el agente.

De la misma forma en que la función de recompensa indica cuál acción es buena inmediatamente, existe una función valor que especifica cuál acción es buena a largo plazo.

El Aprendizaje Reforzado es un enfoque flexible para el diseño de agentes inteligentes en situaciones para las cuales, por ejemplo, no es práctico aplicar aprendizaje supervisado. Un ejemplo de la aplicación del Aprendizaje Reforzado se encuentra en la robótica. Los diseñadores de robots autónomos donde por lo general no se tiene todo el conocimiento del ambiente operacional no pueden usar aprendizaje supervisado para diseñar una política de control para el robot. En este caso, la meta del RL sería permitir al robot generar políticas de decisión efectivas que le permitan explorar su entorno (Moriarty *et al.*, 1999).

1.4.3. QLearning

Uno de los algoritmos más usados del Aprendizaje Reforzado es el algoritmo Q-Learning, este se basa en aprender de una función acción-valor que devuelve la utilidad esperada de tomar una determinada acción en un estado dado. El centro del algoritmo es una actualización del valor de la función en cada iteración, cada par (s, a) tiene un Q-valor asociado. Cuando la acción a es seleccionada por el agente que está en el estado s, el Q-valor de ese par estado-acción es actualizado basado en la recompensa inmediata recibida cuando seleccionó esa acción y el mejor Q-valor para el subsecuente estado s'. La regla mediante la cual el algoritmo Q-Learning realiza la actualización de los Q-valores se muestra a continuación.

$$Q(s,a) \leftarrow Q(s,a) + \alpha[r + \gamma \max_{a'} Q(s',a') - Q(s,a)]$$

En esta expresión $\alpha \in [0,1]$ representa la velocidad de aprendizaje del agente y determina en qué medida va a ser tomado en cuenta el valor anterior, si α es cercano a 1 entonces este valor es reemplazado por el nuevo estimado, para valores cercanos a 0 los Q-valores sufren solo pequeñas variaciones en cada actualización, generalmente se utilizan valores pequeños para la velocidad de aprendizaje, por ejemplo, $\alpha = 0{,}1$. El factor de descuento, γ, representa la importancia que se le dará a la acción futura y también toma valores entre 0 y 1. Si $\gamma = 0$ el agente solo tendrá en cuenta la recompensa inmediata, sin embargo si γ tiende a 1 se dará importancia a la recompensa futura. La recompensa dada al agente por el ambiente después de haber seleccionado una acción está representada por r.

El Algoritmo 1 se usa por los agentes para aprender de la experiencia o el entrenamiento. Cada episodio equivale a una sesión de entrenamiento, y en cada sesión el agente explora el entorno y consigue recompensas hasta alcanzar el objetivo deseado. El propósito del entrenamiento consiste en incrementar el conocimiento del agente, el cual está representado por los Q-valores. Mayor cantidad de entrenamiento trae consigo mejores valores que pueden ser usados por el agente para moverse de manera más óptima.

Algorithm 1 Q-Learnig

Inicializar los Q-valores de forma arbitraria

for cada iteración **do**

Inicializar s

for cada paso de la iteración **do**

Escoger la acción a en el estado s

Tomar la acción a, observar el estado s' y obtener recompensa r

Actualizar los Q-Valores,

$Q(s,a) \leftarrow Q(s,a) + \alpha[r + \gamma * \text{máx}_{a'} Q(s',a') - Q(s,a)]$

$s \leftarrow s'$

end for

end for

1.5. Conclusiones del capítulo

Después de realizada una profunda revisión de la bibliografía podemos arribar a las siguientes conclusiones:

- Los problemas de secuenciación de tareas tipo Job Shop se encuentran clasificados dentro de los más complejos.
- Existen diferentes objetivos a optimizar en este tipo de problemas, sin embargo, la literatura para enfoques multi-objetivo es escasa, siendo este problema tratado como mono-objetivo (makespan) en gran parte de la bibliografía.
- Es muy importante en problemas de manufactura tratar con la incertidumbre que puede ocurrir en este tipo de ambiente por lo que utilizar una técnica para manejar la robustez a la hora de optimizar los problemas Job Shop resulta imprescindible.
- Existen diferentes enfoques para tratar con la optimizacion multi-objetivo pero la utilización de la Frontera de Pareto emerge como el más adecuado.
- Aunque para la optimización multi-objetivo en problemas Job Shop se han utilizado algoritmos evolutivos la configuración de los mismos es muy compleja (casi tanto como

el algoritmo), sin embargo, los Sistemas Multi-agente y el Aprendizaje Reforzado pueden ser aplicados de manera natural a estos problemas, y al menos en el caso mono-objetivo de manera eficiente.

CAPÍTULO 2

Capítulo 2
ALGORITMO MULTI-OBJETIVO PARA JSSP

En este capítulo se describe cómo es la configuración del Aprendizaje Reforzado, especialmente el algoritmo Q-learning, para su aplicación a los problemas de secuenciación de tareas tipo Job Shop, en este caso desde un enfoque multi-objetivo. También se definen las estrategias utilizadas para la selección de la acción por parte de los agentes. Además, se explica el formato de un problema (instancia) tipo Job Shop. Se describen y analizan las principales clases y métodos utilizados en la creación del algoritmo multi-objetivo y cómo se realiza la construcción del Frente no dominado de Pareto. Por último, se muestra la interfaz gráfica creada para la interacción del usuario con el algoritmo.

2.1. Aprendizaje Reforzado para problemas tipo Job Shop

Este epígrafe está enfocado a cómo se configura un ambiente en Aprendizaje Reforzado haciendo énfasis en el algoritmo Q-Learning para problemas de secuenciación de tareas tipo Job Shop. Un agente en el ambiente estará representado por uno de los objetivos a optimizar, y para este agente un estado será el conjunto de operaciones que esperan a ser ejecutadas en una máquina dada. El agente se encarga en un momento dado de realizar una acción, una acción no es más que la selección de la próxima operación que será ejecutada por un recurso (de las disponibles en ese instante de tiempo).

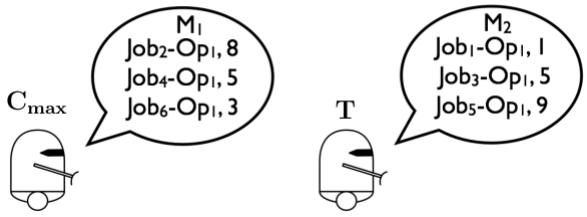

Figura 2–1: Dos agentes con su conjunto de acciones (estado actual).

En la figura 2–6 se muestran dos agentes (makespan y tardanza), cada uno tiene un conjunto de operaciones disponibles pertenecientes a distintos trabajos que esperan ser ejecutadas en una máquina, además del tiempo que toma ejecutar cada una de estas operaciones. La recompensa obtenida por seleccionar una acción vendrá dada por la calidad de la solución y el objetivo a optimizar.

El ambiente del problema de secuenciación define el número de agentes y la relación entre ellos. Los agentes pueden no conocer el estado global del sistema y para obtener un mejor desempeño se comunican entre ellos para determinar sus acciones, basadas en información limitada. Claramente, el enfoque centralizado se aplica a problemas en los cuales la información global está disponible y los agentes son cooperativos (Jiménez, 2012).

Las principales características que deben ser consideradas cuando se modela un problema de secuenciación de tareas se resumen en Gabel (2009):

- **Estado Global Factorizado**: El estado global de un problema de secuenciación de tareas tipo Job Shop puede ser factorizado. Se asume que cada objetivo tiene un agente i asociado que observa el estado local de su objetivo y controla su comportamiento. Por lo tanto se cuenta con tantos agentes como objetivos a optimizar.
- **Observabilidad Local Completa**: El estado s_i del agente i, y por tanto la situación del objetivo obj_i, es completamente observable. Además, la composición de todos los objetivos determina el estado global del problema de secuenciación.
- **Acciones factorizadas**: Las acciones corresponden al inicio de las operaciones de los trabajos (expedición de los trabajos). Por tanto, una acción local del agente i refleja la decisión de procesar un trabajo particular (más específicamente, la próxima operación de ese trabajo) entre un conjunto A_i de operaciones que se encuentran esperando por el recurso r_i.
- **Conjuntos variables de acciones**: Si las acciones denotan la expedición de las operaciones que esperan por ser procesadas, entonces el conjunto de acciones disponibles para un agente varía en el tiempo, ya que el conjunto de operaciones esperando en un recurso cambia a medida que dichas operaciones se van ejecutando. A_i^r corresponde

al conjunto de operaciones $o_{j,k}$ que debe ser procesado en el recurso r_i, donde j es el trabajo al que pertenece la operación y k es el identificador de la operación. Además, el estado local s_i del agente i está completamente descrito por el conjunto cambiante de acciones esperando por ser procesadas en el recurso r_i, por tanto, $s_i = A_i$.

2.2. Modelación del Q-Learning para resolver problemas de secuenciación de tareas tipo Job Shop

Q-Learning es de los algoritmos de RL de los más utilizados, además de brindar, de acuerdo a la literatura, excelentes resultados para problemas de secuenciación de tareas tipo Job Shop. Para la utilización del algoritmo existen elementos importantes que deben ser definidos, para la presente investigación estos elementos pueden ser descritos de la siguiente manera:

Estados y Acciones: existe un agente por cada objetivo, y este agente tomará las decisiones sobre las acciones futuras buscando optimizar a este objetivo. Que cada agente elija una acción significa decidir cuál operación será la próxima a procesar del conjunto de operaciones disponibles que están en espera de un recurso determinado, por lo que un estado será un conjunto de operaciones que esperan en cola (Ω) a ser procesadas por un recurso m. Nótese que dos agentes distintos pueden alcanzar el mismo estado y tomar soluciones completamente diferentes ya que la seleción de la acción para cada uno corresponde a los intereses de su objetivo a optimizar.

Estrategia de selección de la acción: en el algoritmo presentado las estrategias usadas para seleccionar una acción (próxima operación a ser ejecutada por un recurso) son ϵ-greedy y Regla de Borda. La primera ha sido empleada con éxito en entornos multi-agentes (Rodrigues Gomes and Kowalczyk, 2009), en esta estrategia el agente solo mira su objetivo a la hora de tomar su decisión. Mientras que la segunda es usada para tener interacción entre agentes responsables de distintos objetivos, las soluciones generadas por esta interacción son conocidas por soluciones compromiso. Ambas estrategias serán descritas más adelante.

Q-Valores: Sean Ag, R, J el número de agentes, recursos y trabajos respectivamente del sistema. De acuerdo a las restricciones de los JSSP, cada recurso ejecutará J operaciones. De acuerdo a Gabel (2009), el conjunto de estados para el agente i se denota como: $S_i = P(A_i^r)$, la cantidad de estados locales posibles para el agente i será $|Si| = R{*}2^J$, en la cola de espera del recurso m pueden haber a lo sumo J operaciones y la cantidad de subconjuntos posibles de operaciones es 2^J y como tenemos R recursos la cantidad de posibles estados para un agente dado es $R{*}2^J$, por lo que la cantidad de posibles estados del sistema está acotado superiormente por $|S| = (R * 2^J)^{Ag}$. Debido a las restricciones de orden del problema, muchos de estos estados puede que nunca sean alcanzados. Además, el nuevo algoritmo solo almacena los estados por los que transita. Estos estados son almacenados en el orden en que aparecen. Por ejemplo, si el algoritmo se ejecuta solo una vez entonces solo se almacenarán $R * J$ estados, que son los estados donde se encontraba el agente cuando se eligieron las acciones. Otras ejecuciones del algoritmo conllevan a la creación de nuevos estados.

Recompensa: Para las señales de retroalimentación se usaron diferentes alternativas para cada uno de los objetivos a optimizar. Cada recompensa depende del agente y las recompensas utilizadas serán explicadas en el epígrafe 2.6.1.

Después de construida una secuenciación en una iteración el ambiente da una recompensa global a todos los estados por los que el sistema transitó durante este proceso. Esta recompensa está dada por $\frac{1}{F_i(X)}$, donde X es la secuenciación obtenida en la iteración y F_i es la función objetivo para agente i. Estas actualizaciones dan una medida más real de cuánto contribuyó realmente la selección de una acción en un estado dado al valor final del objetivo.

2.3. Instancias JSSP

Para medir el rendimiento del programa que se introduce en esta tesis, se utilizarán un conjunto de instancias para problemas de tipo JSSP disponibles en OR-Library (Beasley, 1990) Esta biblioteca de investigación de operaciones está disponible en Internet.

El formato de la instancia ft06, disponible en esta biblioteca, se muestra en la figura 2–2.

- La primera línea contiene dos números enteros (nmachine, njob) que son el número de trabajos y el número de máquinas del problema respectivamente.
- Cada una de las siguientes $njob$ líneas contiene una lista de pares donde el primer elemento del par representa la operación que tiene que ser procesada para ese trabajo (ID de la máquina) y el segundo elemento es el tiempo que demora procesar dicha operación para ese trabajo (la segunda línea corresponde al trabajo 0, la tercera línea al trabajo 1 y así sucesivamente). Las máquinas son numeradas comenzando por 0.

Figura 2–2: Instancia ft06 de OR-library

En el ejemplo mostrado en la figura 2–2 se tienen 6 trabajos y 6 máquinas. Como puede verse el trabajo 0 debe de ser procesado por la máquina con ID 2 en 1 unidad de tiempo, luego debe ir a la máquina con ID 0 en 3 unidades de tiempo y así hasta llegar a la máquina con ID 4, después de esta terminar de procesar la operación (6 unidades de tiempo) se dará por concluido el trabajo. Es importante resaltar que el orden de procesamiento descrito en la instancia para cada trabajo no puede ser violado.

2.4. Estrategia de selección de una acción

Uno de los retos que se plantean en el Aprendizaje Reforzado, resulta el equilibrio entre la exploración y la explotación. Para obtener buenas recompensas, un agente debe preferir acciones que ya se hayan seleccionado y para las cuales se hayan obtenido buenas recompensas. Pero para descubrir este tipo de acciones, es necesario escoger acciones que no hayan sido seleccionadas, es decir, explorar. El agente tiene que explotar lo que ya sabe con el fin de obtener buenas recompensas, pero también tiene que explorar con el objetivo de tratar de encontrar acciones que mejoren las recompensas obtenidas por las ya exploradas. El dilema es que ni la exploración ni la explotación pueden ser perseguidas exclusivamente sin fallar en la tarea (Sutton and Barto, 1998). El agente debe tratar de favorecer a una variedad de acciones, y progresivamente favorecer a las que parecen ser mejores. Un correcto control del equilibrio entre la exploración y la explotación es importante para la construcción de un método de aprendizaje eficiente. A continuación se explican brevemente dos métodos comunmente utilizados para la selección de la acción, greedy y ϵ-greedy.

Si el agente siempre decide elegir la mejor entre las posibles acciones, entonces se dice que está siguiendo una estrategia de selección greedy. Sin embargo, siempre elegir la mejor acción puede provocar un rendimiento que no sea óptimo, o se puede caer también en óptimos locales, dependiendo de la variación de las recompensas de acción. Una alternativa a este comportamiento greedy consiste en seguir una estrategia de selección ϵ-greedy. En este método el agente elige su mejor acción con una probabilidad $1 - \epsilon$, y con una probabilidad ϵ escogerá al azar (de manera aleatoria) la próxima acción a realizar, siendo ϵ un número entre 0 y 1. Por lo tanto, para valores de ϵ cercanos a uno, la acción seleccionada en la mayoría de los casos seguirá una estrategia greedy, y para valores cercanos a cero se escogerán de manera aleatoria la mayoría de las acciones. Elegir el valor del parámetro ϵ en una estrategia ϵ-greedy resulta sumamente importante para su buen funcionamiento y la obtención de buenas soluciones. En los problemas de Aprendizaje Reforzado el valor de ϵ se recomienda que se encuentre en el intervalo [0.1, 0.4] dándole de esta forma peso

a los Q-Valores, pero sin obviar la exploración. En el presente trabajo el valor de ϵ se fue calculando dinámicamente de acuerdo al número de la iteración mediante la fórmula $\epsilon = 1 - \frac{i}{n}$, donde i es el número de la iteración actual y n la cantidad de iteraciones. Nótese que los valores de ϵ calculados de esta forma proporcionan una mayor exploración al inicio, mientras que a medida que aumenta el número de iteraciones se le da una mayor prioridad a la explotación.

Usando la estrategia ϵ-greedy cada agente se centra solamente en escoger buenas acciones teniendo en cuenta su objetivo, usando estas soluciones solo se tendrán puntos extremos para cada uno de los objetivos. Para obtener una mayor diversificación del Frente de Pareto es necesario establecer algún tipo de interacción entre los agentes responsables de los diferentes objetivos. Las soluciones generadas por esta interacción son conocidas como soluciones compromiso, porque para tomar una decisión el agente no mira solamente cuán buena es para él esa acción, sino también para el grupo.

Para hallar las soluciones compromiso se usó la regla de Borda descrita en Young (1995), en la cual se asume que hay n votantes, m candidatos y un vector $\{s_1, s_2, \ldots, s_m\} \in N^m$ tal que $s_1 \geq s_2 \geq \ldots \geq s_m$. Cada uno de los m votantes da su ranking personal asignando una puntuación s_1 a la alternativa que ocupa la primera posición, s_2 a la alternativa que ocupa la segunda posición y así sucesivamente. En el caso de la Regla de Borda se tiene que $s_i = m - i$. La puntuación para una alternativa o candidato $(S(x))$ es la suma de las puntuciones que cada votante dio a esa alternativa, $\sum_{i=1}^{n}(S_i(x))$. La alternativa selecionada será el candidato de mayor puntuación general.

En nuestro problema las votaciones se harán en un estado dado para seleccionar la próxima operación a ser ejecutada, como se explicó anteriormente, en un estado hay un conjunto de operaciones que esperan por ser procesadas. Estas operaciones representan las alternativas y cada agente será un votante. La figura 2–3 muestra el proceso de selección de una acción mediante esta regla. Como se puede apreciar se tienen tres agentes $\{Ag_1, Ag_2, Ag_3\}$ y tres posibles operaciones $\Omega = \{op_1, op_2, op_3\}$ a ejecutarse en una máquina M. La celda a_{ij} de la matriz 2–3(a) almacena el Q-value del agente Ag_i al seleccionar

la operación op_j ($a_{ij} = Q(S, op_j)$ con $S = (\Omega, M)$). La mejor opción de seleción para cada agente es la operación con mayor Q-valor. La matriz 2–3(b) muestra las operaciones después de rankeadas. La solución compromiso selecionada es la operación op_2 que es la de mayor puntuación general.

	Op_1	Op_2	Op_3
Ag_1	5.56	2.10	3.02
Ag_2	1.36	4.22	1.81
Ag_3	4.72	5.66	4.62

(a) Q-value por acción

	Op_1	Op_2	Op_3
Ag_1	3	1	2
Ag_2	1	3	2
Ag_3	2	3	1
	6	(7)	5

(b) Luego de los votos

Figura 2–3: Regla de Borda para selecionar una acción compromiso

2.5. Construcción del Frente de Pareto

La solución obtenida por el algoritmo implementado tiene como resultado un conjunto de vectores que conforman el Frente no dominado de Pareto. Este conjunto se irá formando con cada solución obtenida por los agentes, cada vez que un agente obtiene una nueva solución se genera un vector $F(x) = (f_1(x), f_2(x), \ldots, f_n(x))$, donde cada $f_i(x)$ representa la solución para el objetivo i. La cantidad de elementos del vector solución está dada por la cantidad de objetivos que se desea optimizar, después de obtener el vector se analiza si constituye una solución no dominada para agregarla al Frente de Pareto, este procedimiento se describe en el algoritmo 2.

Lema 1. *La relación de dominancia entre vectores reales es una relación transitiva, si tenemos tres vectores* V_1, V_2, V_3 *entonces* $(V_1 \preceq V_2) \bigwedge (V_2 \preceq V_3) \rightarrow (V_1 \preceq V_3)$.

En efecto si V_{1i}, V_{2i}, V_{3i} son las componentes i-ésimas de los vectores V_1, V_2, V_3, si $(V_1 \preceq V_2)$ y $(V_2 \preceq V_3)$ entonces $V_{1i} > V_{2i}$ y $V_{2i} > V_{3i}$, por tanto como estos tres valores son reales $V_{1i} > V_{3i}$, y esto se cumple para cada componente de los vectores V_1 y V_3, entonces se puede concluir que V_1 domina a V_3.

Algorithm 2 Agregar una nueva solución al Frente de Pareto

Entrada: Frente de Pareto(PF), Vector solución (v)
Salida: Nuevo Frente de Pareto
for cada vector w en el Frente de Pareto **do**
  if $w \preceq v$ **then**
    Salir del ciclo
  end if
  if $v \preceq w$ **then**
    $PF = PF \setminus w$
  end if
end for
if $\nexists w \in PF \ : \ w \preceq v$ **then**
  $PF = PF \cup v$
end if

En el algoritmo 2 se puede ver que cada vez que se tiene una nueva solución dada por un agente se recorre el frente existente, si la nueva solución es dominada por alguna de las existentes en el frente, no es necesario seguir analizando las soluciones pertenecientes a este; la solución obtenida no estará en el frente (por ser dominada) y, además, no dominará a ninguna de las soluciones existentes ya que por el Lema 1 existirían dos soluciones en el frente que se dominen y esto no es posible. Si la solución actual domina a alguna de las soluciones existentes en el frente, estas serán eliminadas automáticamente (nótese que si la solución actual domina al menos a una de las soluciones pertenecientes al frente esta automáticamente pertenecerá ya que no será dominada por ninguna, si no existirían nuevamente por el Lema 1 dos soluciones en el frente que se dominen). Finalmente, si la nueva solución no es dominada por ninguna otra, esta se agregará al Frente de Pareto.

2.6. Descripción del algoritmo

En este epígrafe se describe el funcionamiento general del algoritmo propuesto. Se explican las principales clases y métodos implementados y en la Figura 2–4 se muestra el diagrama de clases de las principales clases utilizadas.

- ***Operations:*** Representa una operación de un trabajo específico, sus principales atributos son: un Id que la identifica unívocamente, trabajo al que pertenece (JOB), el Id

de la máquina donde debe de ser ejecutada (Maq), el tiempo en que demora su ejecución (Proc_Time), los instantes de tiempo en que empieza (initial_time) y termina su ejecución (end-time).

- ***Job:*** Representa un trabajo del problema, posee una lista de operaciones y un Id que lo identifica, dependiendo del problema puede tener además una fecha de culminación (due time).
- ***Machine:*** Representa una máquina donde se ejecutarán los trabajos, posee un Id que la identifica, contiene dos colas de operaciones, una cola de operaciones que esperan a ser ejecutadas por esta máquina en un instante de tiempo, y la cola de las operaciones que han sido ejecutadas por esta máquina hasta el momento.
- ***State:*** Representa un par estado acción en la configuración del Q-Learning para nuestro problema, posee una Lista de Id de las operaciones que esperaban en un instante dado ser ejecutadas por una máquina, en alguna toma de decisión de un agente, y el Id selecionado por este dentro de esta lista de operaciones.
- ***Agente:*** Es una clase abstracta que rige el comportamiento de un Agente (objetivo a optimizar). Cada agente posee un Id y una tabla hash que mapea un *State* con una recompensa, cada entrada de la tabla va a ser un *State* (par estado acción), y el valor asociado a esa entrada es el Q-valor asociado a este par. Cada agente en el sistema debe de implementar los siguientes métodos
  - **select_next_action**. Cada agente debe ser capaz dado un estado de seleccionar la próxima acción a ejecutarse, con el fin de optimizar su objetivo.
  - **get_reward**. Una vez seleccionada una acción el agente debe dar una recompensa, en dependencia del objetivo a optimizar.
  - **update_qvalues**. Luego de seleccionar una acción un agente actualiza el Q-valor del estado en que se encuentra, de acuerdo a la regla de actualización del Q-Learning.

- **update_global**. Luego de construida una secuenciación el agente debe actualizar con una recompensa global todos los estados por los que transitó, esta recompensa está en dependencia del valor de su objetivo para la secuenciación construida.
- **evaluate_scheduling**. Dada una secuenciación el agente debe ser capaz de obtener el valor de su objetivo para esta secuenciación.

- ***Multi_Agent_System:*** Esta clase contiene todos los agentes del sistema, y es la que permite la interacción entre ellos. Su principal método es **select_next_action_vectorial**, el cual para seleccionar una acción encuentra una solución compromiso usando la Regla de Borda.
- ***Environment:*** Representa el Ambiente en un problema de secuenciación de tareas tipo Job Shop, sus principales atributos son la cantidad de iteraciones a realizar por el algoritmo, una lista de trabajos, una lista de máquinas, el Frente de Pareto en construcción y un sistema multiagente. Esta clase es la encargada del funcionamiento general del algoritmo y sus principales métodos son:
  - **read_data**. Se encarga de leer y almacenar una instancia
  - **pareto_add**. Se encarga de incluir los vectores calculados en cada iteración en el Frente de Pareto.
  - **dynamic_epsilon**. Calcula dinámicamente el paramétro ϵ usado en la estrategia ϵ-gredy para la selección de una acción.
  - **execute**. Es el método principal del algoritmo, para cada una de las iteraciones del problema, cada agente calcula una secuenciación, y se construye el vector solución asociado a cada secuenciación construida, el cual de ser no dominado se incluye en el Frente de Pareto.
  - **build_scheduling**. Es el encargado en cada iteración de construir para un agente dado una secuenciación factible.

  En la siguiente figura se muestra el diagrama de clases de las principales clases del sistema.

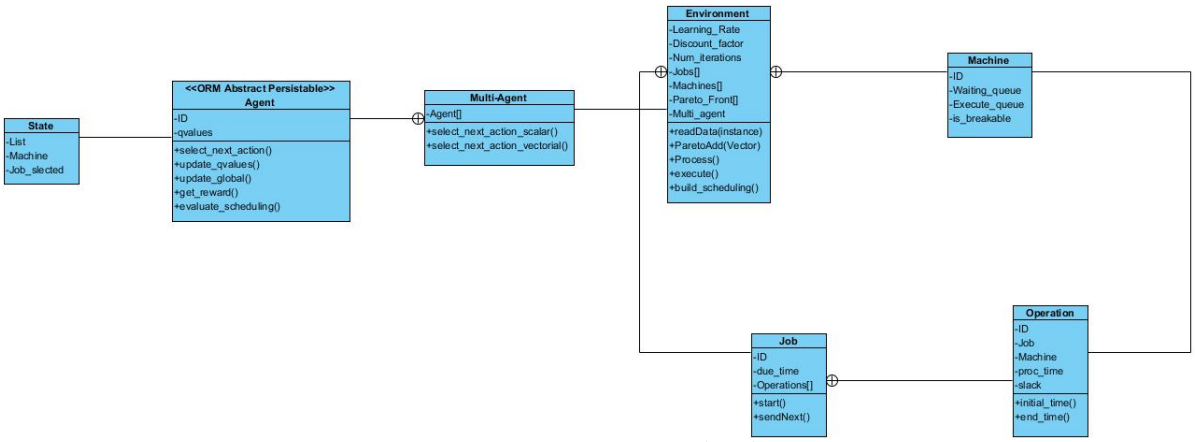

Figura 2–4: Diagrama de clases

Los seudocódigos de los dos últimos métodos mencionados se muestran en los algoritmos 3 y 4 respectivamente.

Notación:

$nagent$ es el número de agentes (objetivos a optimizar),

$nmach$ es el número de máquinas,

$pcompromise$ es la iteración donde el algoritmo comienza a buscar soluciones compromiso,

S es el conjunto de secuenciaciones factibles,

o_{ij} es la i-ésima operación del j-ésimo trabajo,

$S_{a_k}^{(i)}$ es la secuenciación obtenida por el agente k en la i-ésima iteración,

$V_{a_k}^{(i)} \in \Re^{nagent}$ es el vector solución asociado a la secuenciación obtenida por el agente k en la i-ésima iteración,

Ω_m es el conjunto de operaciones que esperan ser procesadas por la máquina m,

$f_{a_i} : S \to \Re$ es la función objetivo a optimizar por el agente i,

Pf es el conjunto de soluciones no dominadas (Frente de Pareto conocido) y

$mach(o_{ij})$ es la máquina donde la operación o_{ij} será ejecutada.

Algorithm 3 execute

Entrada: instancia JSSP, parámetros del QLearning
Salida: conjunto de solucoiones no dominadas (Pf)
repeat
  for $k = 1$ hasta nagent **do**
    $S_{a_k}^{(i)} \leftarrow$ build_scheduling
    for $j = 1$ hasta nagent **do**
      $V_{a_k}^{(i)} \leftarrow f_{a_j}(S_{a_k}^{(i)})$
    end for
    a_k.update_global
    if $V_{a_k}^{(i)}$ es no dominada **then** $Pf \leftarrow V_{a_k}^{(i)}$
    end if
  end for
until max_iteration_number
return Pf

Algorithm 4 build_scheduling

while $\bigcup_{i=1}^{nmach} \Omega_i \neq \emptyset$ **do**

Seleccionar aleatoriamente m tal que $\Omega_m \neq \emptyset$

if current_iteration $<$ pcompromise **then**

Seleccionar $o_{ij} \in \Omega_m$ usando la estrategia ϵ-greedy

else

Seleccionar $o_{ij} \in \Omega_m$ usando la Regla de Borda

end if

$S_{a_k}^{(i)} \leftarrow o_{ij}$

Calcular la recompensa asociada a o_{ij}

a_k.update_qvalues

$\Omega_m \leftarrow \Omega_m \setminus o_{ij}$

$m^* \leftarrow mach(o_{(i+1)j})$

$\Omega_{m^*} \leftarrow o_{(i+1)j}$

end while

2.6.1. Agentes Implementados

Aunque el algoritmo permite la implementación de todos los objetivos que se deseen optimizar en un problema de secuenciación tipo Job Shop, en esta tesis solo fueron implementados seis objetivos, es decir, el sistema cuenta con seis agentes. Las principales funcionalidades de un agente son: el cálculo de la función de evaluación (epígrafe 1.1.1) y de las recompensas local y global. A continuación se describen las recompensas utilizadas para cada agente.

- **El agente Makespan** trata de minimizar el tiempo de completitud de todas las tareas. El agente seleciona la acción, y el ambiente calcula la recompensa local para este agente de acuerdo a la fórmula $r = \frac{1}{s_{ij} + p_{ij}}$.
- **El agente Inactividad** busca minimizar el tiempo de inactividad total de todas las máquinas. La recompensa local dada al agente luego de seleccionar una acción está basada en la siguiente fórmula: $r = \frac{1}{(s_{ij} - (s_{ik} + p_{ik}))}$, donde k representa el número de la operación previamente ejecutada en la máquina i.
- **El agente Tardanza Total** busca minimizar la tardanza de todos los trabajos. La recompensa local es dada por: $r = T_j$, donde T_j es la diferencia entre el tiempo esperado de finalización del trabajo j y el tiempo final de la operación selecionada. Nótese que si la operación finaliza después del tiempo esperado la recompensa puede ser negativa.

Las recompensas globales para el makespan, inactividad y tardanza son, $\frac{1}{C_{max}}$, $\frac{1}{I_\Sigma}$ y $\frac{1}{T_\Sigma}$ respectivamente.

Los agentes implementados para manejar la robustez tienen una funcionalidad adicional que los distingue de los demás agentes. Esta funcionalidad es el cálculo del tiempo de inactividad que se agrega de acuerdo a la técnica implementada. Estos agentes adicionan el tiempo de inactividad de acuerdo a las técnicas descritas en el epígrafe 1.2.2. Las técnicas implementadas deben estar asociadas a la optimización de un objetivo, para esto

solo es necesario usar como función de evaluación y funciones de recompensa las mismas del objetivo a optimizar.

2.7. Interfaz gráfica

Para una mejor interacción con el algoritmo se creó una aplicación desktop, usando el lenguaje Java y el entorno de desarrollo NetBeans, que posibilita al usuario seleccionar la instancia y objetivos que desea optimizar.

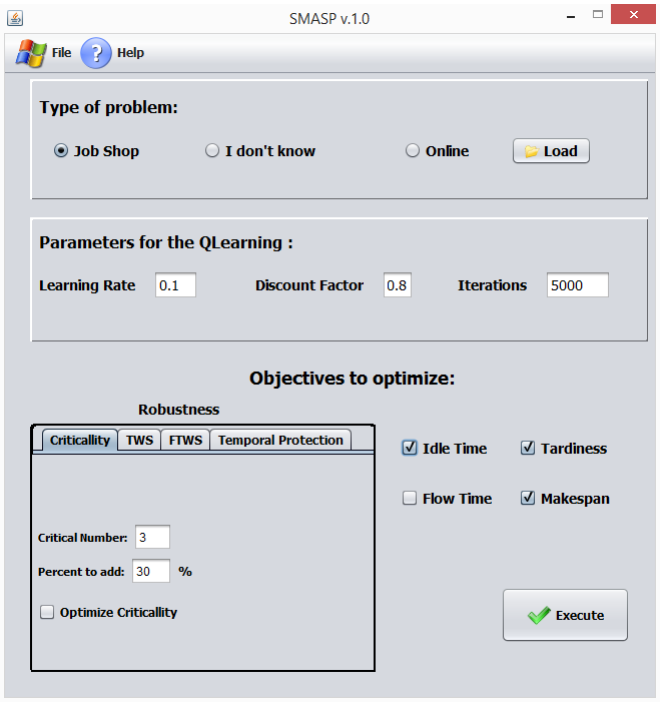

Figura 2–5: Ventana principal de la aplicación.

Para la instancia y objetivos seleccionados se devuelve la Frontera de soluciones no dominadas de Pareto. La interfaz permite además visualizar dichas soluciones.

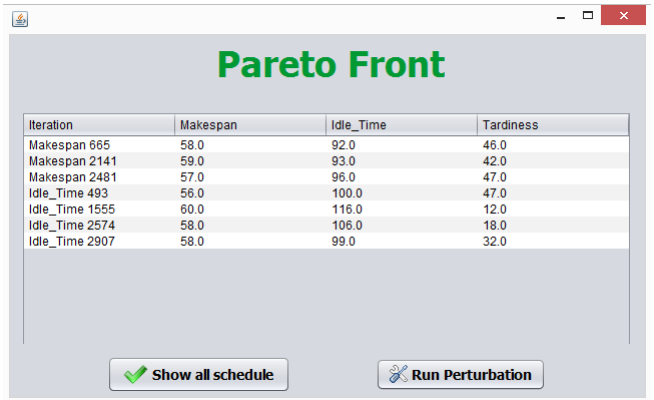

Figura 2–6: Frontera de Pareto obtenida para la instancia ft06, para los objetivos, Makespan, Tardanza e Inactividad.

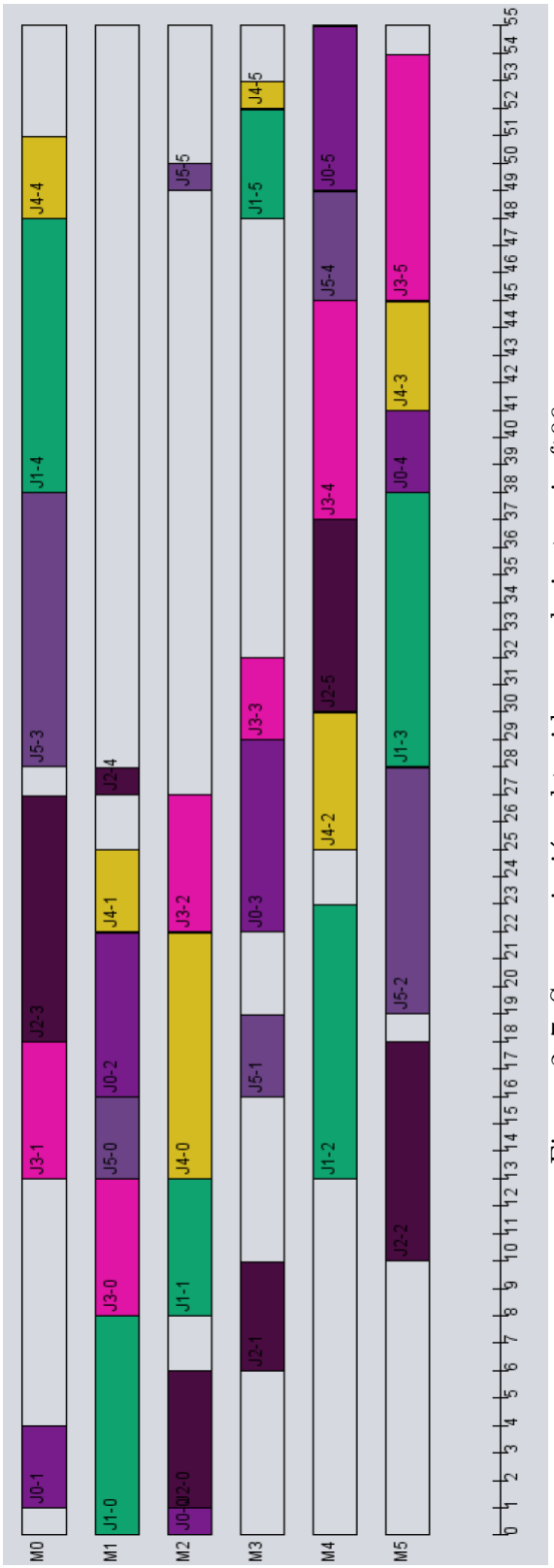

Figura 2–7: Secuenciación obtenida para la instancia ft06

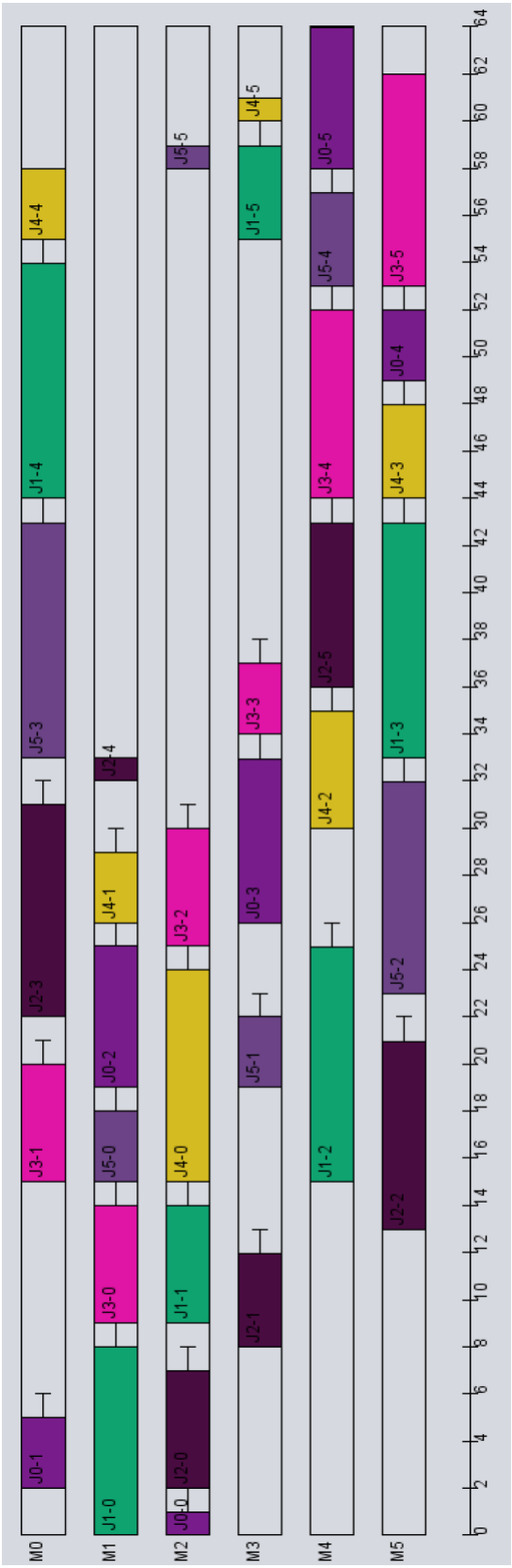

Figura 2–8: Secuenciación robusta obtenida para la instancia ft06 usando Protección Temporal

2.8. Conclusiones del capítulo

En este capítulo se realizó un análisis de la configuración del Aprendizaje Reforzado, específicamente el algoritmo Q-Learning, lográndose con esto la aplicación de este algoritmo a los problemas de secuenciación de tareas tipo Job Shop desde un enfoque multi-objetivo. Se expone el algoritmo general así como el algoritmo seguido para la construcción del Frente de Pareto y por último, se desarrolló una aplicación desktop que permite a los usuarios obtener el Frente de Pareto asociado a una instancia y a un conjunto de objetivos seleccionado por ellos. Esta aplicación también permite visualizar la secuenciación obtenida, y en caso de haber seleccionado una técnica de robustez permite visualizar la secuenciación robusta asociada a la secuenciación obtenida.

CAPÍTULO 3

Capítulo 3
VALIDACIÓN DE LOS RESULTADOS

En este capítulo se realizan una serie de pruebas estadísticas para evaluar el desempeño del algoritmo propuesto en el capítulo anterior. Se comparan los resultados obtenidos por el algoritmo implementado con los resultados de otros dos algoritmos de la literatura. Sin embargo, para comprobar los resultados obtenidos al utilizar las técnicas basadas en tiempos de inactividad que fueron implementadas, fueron necesarias otro tipo de pruebas que también se exponen en este capítulo. Para finalizar se mide el espaciamiento y el hipervolumen del Frente de Pareto obtenido para evaluar de esta manera el algoritmo desde un enfoque multi-objetivo que fue el fin con que se creó.

3.1. Resultados obtenidos

Se hace necesario después de la implementación del algoritmo usar un conjunto de instancias para probar el desempeño del mismo. Para realizar los experimentos se tomaron un conjunto de 22 instancias de la biblioteca OR-Library (Beasley, 1990). Para cada una de las instancias se hicieron 10 corridas y la configuración de parámetros para el algoritmo Q-Learning que mejor resultados brindó fue: $\alpha = 0{,}1$ y $\gamma = 0{,}8$.

3.2. Makespan y tiempo de inactividad de las máquinas

Los resultados obtenidos por el algoritmo implementado para el tiempo de completitud de todas las operaciones (makespan) y el tiempo de inactividad de las máquinas (idle-time) fueron comparados con los resultados alcanzados con los algoritmos MOGA (Fonseca and Fleming, 1993) y MOPSO (Sha and Lin, 2010).

El algoritmo MOGA está basado en machine-wise priority dispatching rule y su función objetivo es la suma pesada del makespan, la tardanza total y el tiempo total de inactividad de las máquinas, los pesos utilizados fueron generados de manera aleatoria.

El otro algoritmo, MOPSO, adapta el enfoque tradicional de PSO a espacios discretos. Para esto modifican la representación de la posición de la partícula, su movimiento y su velocidad.

La Tabla 3–6 muestra los resultados obtenidos para los objetivos makespan y tiempo de inactividad de las máquinas al aplicar los algoritmos a las instancias seleccionadas. Los resultados marcados en negrita resaltan donde el algoritmo implementado en esta tesis fue capaz de mejorar los resultados obtenidos por los dos algoritmos con los que está siendo comparado. La comparación entre los resultados obtenidos por los tres algoritmos se evidencia mediante el error relativo.

Tabla 3–1: Comparación entre MOGA, MOPSO y MORLA de acuerdo al makespan y el idle time

Inst	n	m	MOGA	MOPSO	MORLA	MOGA	MOPSO	MORLA	Relative Error
abz5	10	10	1587	1338	**1234**	8097	3987	3767	0.0
abz6	10	10	1369	1046	**943**	7744	2937	2446	0.0
la16	10	10	1452	1040	**945**	9169	2718	3174	0.16777
la17	10	10	1172	889	**784**	7044	3365	2798	0.0
la19	10	10	1251	938	**842**	7164	2796	2593	0.0
la20	10	10	1419	985	952	8745	2883	3098	0.0746
orb01	10	10	1704	1181	1181	11631	3909	3858	0.0
orb02	10	10	1284	1029	**889**	7585	3539	3102	0.0
orb03	10	10	1643	1114	1114	11138	3788	3617	0.0
orb04	10	10	1543	1122	**1024**	9802	3921	3754	0.0
orb05	10	10	1323	1013	944	8322	3727	3429	0.0
orb06	10	10	1645	1144	1125	10836	3478	3194	0.0
orb07	10	10	583	302	**397**	3423	1381	1324	0.0
orb08	10	10	1340	1000	1000	8840	3542	3497	0.0
orb09	10	10	1462	1044	1017	9439	4224	4080	0.0
orb10	10	10	1382	1077	1027	8271	4177	3460	0.0
la01	10	5	1256	709	**666**	3431	571	283	0.0
la02	10	5	1066	713	686	2687	573	278	0.0
la03	10	5	821	671	**597**	1722	633	525	0.0
la04	10	5	861	631	**590**	1798	557	367	0.0
la05	10	5	813	593	**593**	2182	473	363	0.0
ft06	6	6	76	56	**55**	259	100	91	0.0

Como puede verse en la tabla para el makespan el algoritmo propuesto es capaz de superar o al menos igualar los resultados de MOGA y MOPSO para todas las instancias utilizadas. En el caso del tiempo de inactividad de las máquinas, nuestro algoritmo es capaz de superar al algoritmo MOGA en todos los casos y al algoritmo MOPSO en 20 de las 22 instancias utilizadas.

3.3. Tardanza

En el caso de la minimización de la tardanza no se cuenta con los datos (fechas de entrega) necesarios para comparar con los algoritmos anteriores. Estos datos fueron generados de acuerdo a Vilcot and Billaut (2008) y Demirkol *et al.* (1998). Los fechas de entregas (d_j) usadas fueron generadas usando una distribución uniforme de acuerdo a la

fórmula:

$$d_j = U\left[\mu_j * \left(1 - \frac{R}{2}\right); \mu_j * \left(1 + \frac{R}{2}\right)\right]$$

donde $\mu_j = \left(1 + \frac{T*n}{m}\right) * \sum_{i=1}^{m} p_{ij}$, T y R dos parámetros fijos con valores 0.3 y 0.5 respectivamente . Tanto las fechas de entrega generadas mediante la fórmula como la tardanza encontrada mediante la aplicación del algoritmo propuesto se muestran en la Tabla 3–2 para futuras comparaciones.

Tabla 3–2: Fechas de entrega generadas y Tardanza para las instancias de prueba

Jobs / Insts	Job 1	Job 2	Job 3	Job 4	Job 5	Job 6	Job 7	Job 8	Job 9	Job 10	Tardiness
abz5	1208.04	1016.91	795.86	1099.49	1098.9	988.03	1061.88	910.74	906.48	726.01	1283
abz6	694.74	931.56	614.51	1050.13	646.91	760.85	579.26	697.07	849.94	733.31	758
la16	685.74	657.34	668.49	873.23	755.26	450.94	677.56	482.48	672.91	809.82	1236
la17	456.62	736.32	917.23	428.53	470.51	876.9	738.54	589.36	531.78	617.28	1066
la19	948.78	739.18	811.21	758.38	719.33	979.53	764.42	642.76	641.99	682.61	322
la20	628.52	704.22	603.26	560.81	657.13	421.79	925.16	756.94	645.68	585.85	1593
orb01	741.22	709.47	734.43	514.70	565.47	454.67	591.28	728.76	840.92	803.9	2404
orb02	841.56	580.46	562.33	566.9	668	592.45	923.5	649.35	500.08	866.12	1241
orb03	544.76	696.16	879.14	576.67	821.21	669.85	471.2	651.3	675.69	637.53	2640
orb04	485.03	884.53	678.6	692.67	740.78	726.1	592.29	811.61	839.52	545.71	1863
orb05	661.09	798.88	457.21	586.84	792.27	507.99	452.96	812.12	732.67	487.13	1445
orb06	684.39	889.6	930.46	573.24	607.3	856.81	441.47	500.73	892.79	840.92	1877
orb07	309.96	294.82	333.22	330.38	405.65	307.51	329.36	271.57	378.85	312.29	378
orb08	652.37	529.52	768.6	539.25	682.49	726.15	740.28	435.52	400.4	453.37	2312
orb09	583.49	846.11	797.45	627.56	714.53	419.54	882.5	367.91	778.12	692.24	1818
orb10	918.54	708.92	731.09	673.28	874.95	719.91	522.64	697.99	613.64	750.35	1458
la01	494.9	239.1	410.08	581.19	462.96	558.63	822.8	423.8	400.13	640.3	482
la02	307.28	266.56	444.43	642.02	253.16	451.26	309.5	611.93	387.14	336.02	1090
la03	457.53	204.5	409.95	364.38	388.13	489.94	368.84	289.8	375.23	290.46	756
la04	376.96	358.84	270	339.31	336.57	537.32	455.64	613.32	407.87	275.04	906
la05	679.82	264.66	452.04	264.86	287.52	388.12	379.7	277.35	307.4	432.1	891

3.4. Robustez

3.4.1. Perturbaciones

Las pruebas realizadas para este objetivo son diferentes. Lo primero fue perturbar las secuenciaciones obtenidas. Para simular distintas interrupciones que pueden ocurrir

en tiempo real (roturas, falta de fluido eléctrico, etc.) y medir cómo las soluciones obtenidas con las diferentes técnicas implementadas reaccionan a estos eventos inesperados, a cada una de las instancias que se muestran en la Tabla 3–3 se le aplican una lista de perturbaciones, donde cada perturbación se define de la siguiente manera:

- ID de la máquina en la que se producirá la perturbación
- Tiempo de inicio de la perturbación
- Duración de la perturbación

Tabla 3–3: Lista de perturbaciones para la instancia ft06

Id de la máquina	Tiempo de inicio	Duración
0	24	2
1	29	8
1	31	3
2	25	1
3	50	4
4	28	3
5	22	4
5	50	2

En la Figura 3–1 se muestra un ejemplo de dos perturbaciones; la primera se desarrolla en la máquina uno, empezando en el instante de tiempo 23 y dura 4 unidades de tiempo, la segunda en la máquina dos y empezando en el instante de tiempo 45 con 8 unidades de duración.

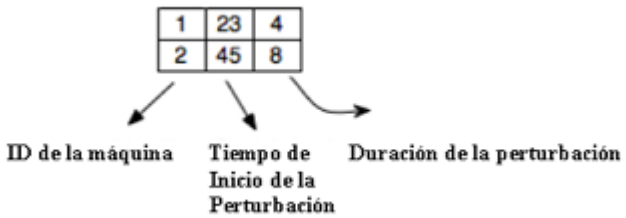

Figura 3–1: Formato de una Perturbación

Como en problemas tipo Job Shop cada operación puede ser ejecutada solamente en una máquina, durante una perturbación la máquina estará ese tiempo inactiva, por lo

que si sucede en medio de la ejecución de una operación esta debe detenerse y retomar su ejecución en el punto en que fue detenida una vez que termine esta interrupción, las operaciones que esperan en cola tienen que aguardar a la vez por esta operación para ser ejecutadas. La distribución de Poisson se usó para generar el tiempo inter arribo entre las perturbaciones y para la duración de cada una de estas. El tiempo inter arribo fue generado utilizando Poisson con media $\frac{\mu_{Br}}{\Theta}$ que es una cota superior para el tiempo de terminación de todas las tareas. En el algoritmo presentado, como trata las perturbaciones para una secuenciación específica (después de que esta ya es construida), $\frac{\mu_{Br}}{\Theta}$ puede ser igual al makespan de la secuenciación obtenida. A mayores valores de Θ se obtendrán menores valores promedios de tiempos inter arribos y por tanto mayor cantidad de perturbaciones. El promedio usado para generar los tiempos de duración de las perturbaciones fue de 3 unidades de tiempo, luego a estos tiempos se les agregó un dos por ciento del tiempo estimado de duración del proceso ($\frac{\mu_{Br}}{\Theta}$) para adaptar la duración al tamaño y complejidad de las instancias correspondientes.

Si a la solución óptima obtenida para la instancia ft06 (Figura 2–7) le ocurrieran las perturbaciones mostradas en la Tabla 3–3 la secuenciación resultante será la mostrada en la figura 3–2.

Cuando se inserta una perturbación en un instante de tiempo pueden suceder tres casos.

1. La perturbación no interfiere en la ejecución de ninguna operación.
2. El tiempo de inicio de la ejecución de una operación queda dentro del tiempo de ocurrencia de la perturbación
3. El tiempo de ejecución de la operación se intercepta con el de ocurrencia de la perturbación, pero el tiempo de inicio de la operación es inferior al de la perturbación.

De ocurrir el primer caso la secuenciación no se afectaría, a diferencia de los casos dos y tres, en el segundo caso habría que esperar a que termine la perturbación para empezar a ejecutar la operación en su máquina correspondiente y en el tercer caso esperar a que

termine la perturbación para reanudar la ejecución de la operación en el punto que se quedó antes de la perturbación en la máquina afectada debido a que los JSSP no permiten la preempción.

Las perturbaciones están representadas por bloques con líneas horizontales amarillas y rojas. Supongamos que el makespan se refiere al número de días que va a tomar para terminar la ejecución de todo el programa. Si se comparan las dos soluciones que se indican anteriormente (Figura 2–7 y Figura 3–2), se verá que el usuario tendrá que esperar 5 días adicionales con el fin de conseguir su orden.

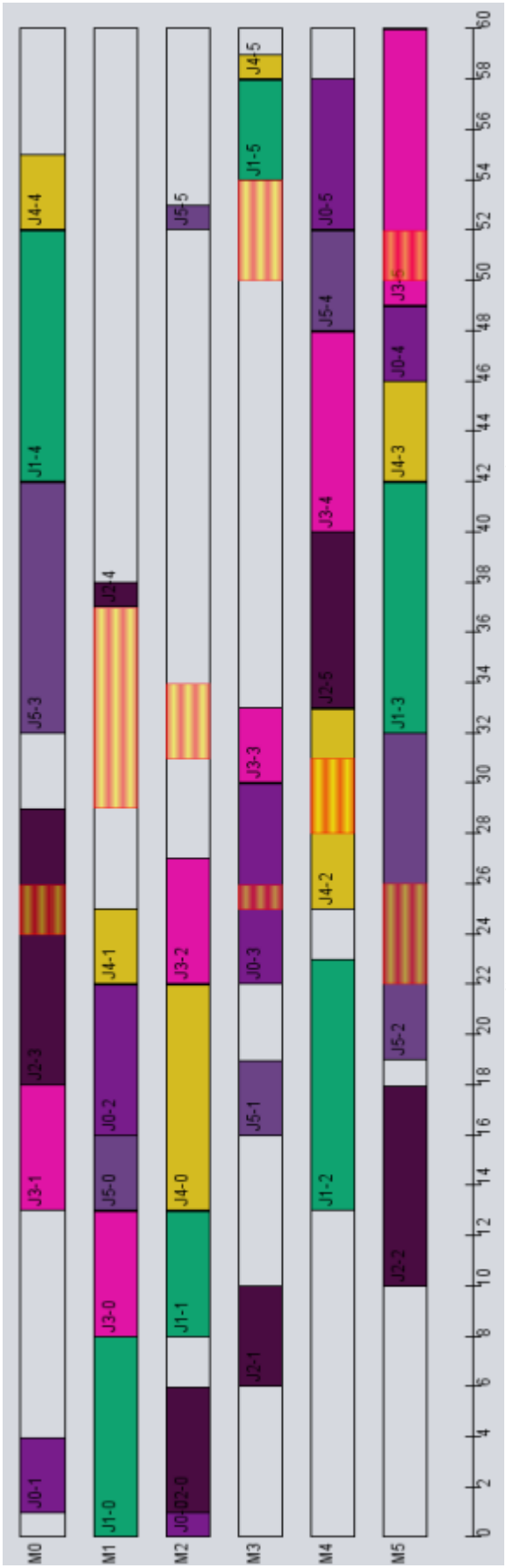

Figura 3–2: Secuenciación perturbada para la instancia ft06

3.4.2. Simulación

Para el diseño de la simulación se utilizaron los experimentos propuestos en Davenport (2001). En este trabajo los autores proponen utilizar como media del tiempo entre fallos o roturas de las máquinas $\mu_{tbf} = \frac{lb}{2^{(mr-1)}}$, donde mr es el número de máquinas rompibles, y como media de la duración de estas roturas $\mu_{dt} = \sum_{A \in opers_R} \frac{dur_A}{njobs}$ donde $njobs$ es la cantidad de trabajos de la instancia, $opers_R$ es el conjunto de todas las operaciones que se ejecutan sobre la máquina R y lb puede ser calculada mediante la ecuación:

$$lb = \text{máx}(\min_{A \in opers_R}(est_A) + \sum_{A \in opers_R} dur_A, \max_{A \in opers_R}(lft_A))$$

donde est_A y lft_A son los tiempos de inicio y de fin de la operación A respectivamente.

A continuación en la Figura 3–3 se muestran los resultados obtenidos para algunas de las instancias analizadas al aplicar el makespan como función de recompensa para la robustez utilizando las tres técnicas descritas anteriormente.

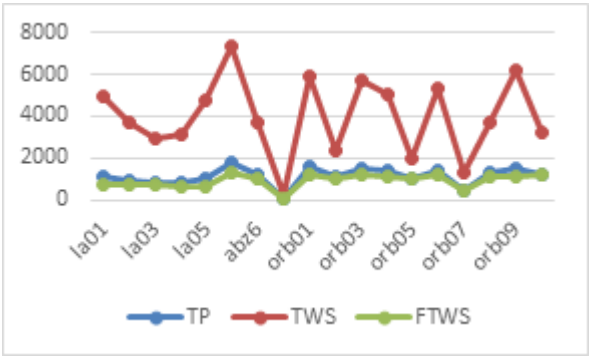

Figura 3–3: Resultados de las técnicas para algunas instancias

En la figura puede corroborarse que TWS es la técnica que más tiempo de inactividad agrega a las operaciones, mientras que PT y FTWS tienen aproximadamente el mismo comportamiento. Sin embargo, la cantidad de tiempo que agrega una técnica no es suficiente para evaluar la efectividad de la misma.

Para saber cuál técnica o técnicas de las implementadas son más eficientes es necesario saber si son capaces de absorber todas las incertidumbres, y en caso afirmativo, que la desviación entre lo esperado y lo real sea lo más pequeña posible. Para esto se simularon

perturbaciones (entre 25 y 50) de acuerdo a las medias para los tiempos entre roturas y la duración de las mismas para cada instancia siguiendo una distribución de Poisson. Se normalizaron los resultados obtenidos en dos categorías, 1 si la secuenciación perturbada absorbía las roturas simuladas y 0 en otro caso. Luego se aplicó la prueba de McNemar para comprobar si existían diferencias entre las diferentes técnicas implementadas. En la Tabla 3–4 se muestran los resultados.

Tabla 3–4: Resultados de la Prueba de McNemar

	TP-TWS	**TP-FTWS**	**TWS-FTWS**
Sig.	0.250	0.000	0.000

Como puede verse en la tabla entre la TP y TWS no existen diferencias significativas, no siendo así entre ellas con la técnica FTWS. De acuerdo con los resultados obtenidos en las simulaciones FTWS solo fue capaz de absorber las perturbaciones en 1 de las 18 instancias evaluadas.

Aunque mediante McNemar se demostró que no existen diferencias entre TP y TWS para absorber las perturbaciones es necesario determinar cuál técnica logra una menor desviación entre lo esperado y la secuenciación después de ser perturbada, para un análisis estadístico de esta desviación se utiliza la prueba de signos de Wilcoxon. En la tabla se muestran los resultados obtenidos.

Tabla 3–5: Resultados de la Prueba de Wilcoxon

Técnicas	R^+	R^-	**Sig.**
TWS - TP	171.000	0.000	0.000

En la tabla se ratifica el resultado que podía ser inferido por el gráfico de la figura 3–3. De acuerdo a esta prueba estadística existen diferencias significativas entre las dos técnicas y al ser los rasgos positivos mayores que los negativos entonces la protección temporal muestra una menor desviación entre la solución real y la estimada.

3.5. Métricas multi-objetivo

En esta sección se analizan las métricas que se utilizan en optimización multi-objetivo para así medir el desempeño de nuestro algoritmo. Del gran conjunto de métricas existentes para las cuales no es necesario conocer el Frente de Pareto verdadero se van a utilizar solamente dos, debido a que para el problema tratado no se cuenta con este dato.

3.5.1. Espaciamiento e Hipervolumen

El espaciamiento (Coello *et al.*, 2002) es uno de los indicadores más populares para observar la diversidad entre las soluciones no dominadas, describe la extensión de los vectores en el Frente de Pareto conocido y se le conoce como la métrica $\boldsymbol{S}$. Esta métrica calcula la distancia relativa entre las soluciones consecutivas. Dicha métrica está definida por la siguiente ecuación:

$$\boldsymbol{S} \triangleq \sqrt{\frac{1}{|PF_{conocido}| - 1} * \sum_{i=1}^{|PF_{conocido}|} (\overline{d} - d_i)^2}$$

donde $d_i = min_j(\sum_{k=1}^{n} |f_k^i(x) - f_k^j(x)|)$, $i, j = 1, \ldots, n$, $\overline{d}$ es la media de todas las d_i y n es el número de vectores en el Frente de Pareto conocido. Cuando $\boldsymbol{S} = 0$ todos los miembros están espaciados de manera uniforme.

El hipervolumen (**HV**) (Coello *et al.*, 2002) se ha vuelto extremadamente popular hoy día en el área de la optimización multi-objetivo. Debido a sus buenas propiedades matemáticas, se ha utilizado no solo como indicador de desempeño para comparar los resultados finales de los algoritmos multi-objetivo (AMOs), sino también como criterio de selección de los optimizadores multi-objetivo. Dado que el hipervolumen es el único indicador de desempeño unario que es compatible con la dominancia de Pareto y que se ha podido demostrar que su maximización es equivalente a lograr convergencia al verdadero Frente de Pareto, su uso ha cobrado gran importancia en años recientes. Este indicador está definido por el área que cubre el Frente de Pareto conocido con respecto al espacio objetivo. La ecuación de esta métrica se muestra a continuación:

$$\textbf{HV} \triangleq \{\cup_i vol_i | vec_i \in PF_{conocido}\}$$

Todas las comparaciones hechas hasta el momento demostraron que el algoritmo propuesto contiene en la Frontera de Pareto obtenida las soluciones óptimas para cada objetivo analizado o al menos soluciones bien cercanas a las óptimas. Sin embargo, un algoritmo multi-objetivo no se mide solo por encontrar el óptimo para cada objetivo, también es necesario que el frente encontrado cumpla con una serie de característica, destacándose entre ellas, de acuerdo a la literatura, la diversidad.

Con el objetivo de determinar si los Frentes de Pareto obtenidos cumplen con algunas de las métricas utilizadas para la optimización multi-objetivo se utilizan 3 objetivos de los implementados (makespan, tardanza y tiempo de inactividad) y el algoritmo se aplica al mismo conjunto de instancias que fueron utilizadas para las pruebas anteriores. Los resultados obtenidos para estas métricas se muestran a continuación.

Tabla 3–6: Métricas para las instancias utilizadas

Inst	Espaciamiento	Hipervolumen
abz5	0.1135	0.2257
abz6	0.1014	0.2156
la16	0.1612	0.2000
la17	0.1079	0.1898
la19	0.1246	0.1370
la20	0.1451	0.2090
orb01	0.1203	0.3048
orb02	0.1055	0.1947
orb03	0.1942	0.3565
orb04	0.0000	0.1859
orb05	0.1163	0.1835
orb06	0.1416	0.3601
orb07	0.1641	0.1230
orb08	0.3124	0.3020
orb09	0.0882	0.1751
orb10	0.1726	0.1787
la01	0.2458	0.9317
la02	0.1615	0.3833
la03	0.1261	0.5963
la04	0.1646	0.5784
la05	0.1704	0.3648
ft06	0.1266	0.3064

3.6. Conclusiones parciales del capítulo

En este capítulo se realizaron las pruebas necesarias para evaluar el desempeño del algoritmo propuesto.Primeramente se compararon los resultados obtenidos por el algoritmo con otros dos algoritmos multi-objetivo propuestos en la literatura en cuanto a makespan y tiempo de inactividad. El algoritmo propuesto fue capaz de mejorar para el makespan a los otros en todas las instancias utilizadas y en cuanto a los tiempos de inactividad lo logra en 20 de las 22 instancias tratadas. Las pruebas utilizadas para medir las técnicas que manejan la robustez demostraron que la técnica Protección Temporal es mejor que las otras dos debido a que es capaz de absorber las posibles interrupciones que puedan

ocurrir y esto lo hace con una menor desviación entre lo real y lo planificado. Por último se utilizaron dos métricas usadas en la optimización multi-objetivo obteniendo excelentes resultados en ambas.

CONCLUSIONES Y RECOMENDACIONES

CONCLUSIONES

- Se adaptaron los principales elementos del algoritmo Q-Learning a los problemas de secuenciación de tareas tipo Job Shop que nos permitieron diseñar una arquitectura genérica para resolver estos problemas desde un enfoque multi-objetivo.
- Se implementó un algoritmo multi-objetivo basado en la Frontera de Pareto utilizando Aprendizaje Reforzado y Sistemas Multi-agente que permite resolver de forma óptima los problemas de secuenciación de tareas tipo Job Shop.
- El algoritmo implementado fue comparado con otros algoritmos de la literatura para dos de los objetivos implementados, makespan y tiempos de inactividad. Para estos objetivos nuestro algoritmo resultó superior llegando incluso a encontrar el óptimo para la 20 de las 22 instancias analizadas.
- Para el objetivo tardanza se generaron las fechas de entrega para las instancias utilizadas de acuerdo a una fórmula propuesta en la literatura. Los resultados obtenidos se muestran en la tesis para futuras comparaciones.
- En cuanto a los resultados obtenidos para las técnicas que manejan la robustez se comprobó que la técnica Protección Temporal es capaz de absorber las perturbaciones que fueron simuladas y presenta la menor desviación entre lo real y lo planificado.
- También se utilizaron dos métricas para medir el desempeño del algoritmo con un enfoque multi-objetivo, espaciamiento e hipervolumen. Estos resultados se muestran en la tesis para futuras comparaciones, aunque cabe destacar que un 86 % de los casos analizados el espaciamiento es un valor pequeño, llegando a alcanzar el valor 0 para una de las instancias.

RECOMENDACIONES

- Implementar nuevos objetivos adicionando nuevos agentes.
- Extender el algoritmo a otros tipos de problemas de secuenciación de tareas.
- Aplicar un método de reducción del Frente de Pareto al obtenido por el algoritmo implementado.

REFERENCIAS BIBLIOGRÁFICAS

Aloulou, Mohamed Ali and Marie-Claude Portmann (2005). An efficient proactive-reactive scheduling approach to hedge against shop floor disturbances. In: *Multidisciplinary scheduling: theory and applications*. pp. 223–246. Springer.

Beasley, John E (1990). Or-library: distributing test problems by electronic mail. *Journal of the operational research society* **41**(11), 1069–1072.

Beck, J Christopher and Nic Wilson (2007). Proactive algorithms for job shop scheduling with probabilistic durations. *Journal of Artificial Intelligence Research* pp. 183–232.

Ben-Tal, Aharon and Arkadi Nemirovski (2000). Robust solutions of linear programming problems contaminated with uncertain data. *Mathematical programming* **88**(3), 411–424.

Bertsimas, Dimitris and Melvyn Sim (2004). The price of robustness. *Operations research* **52**(1), 35–53.

Billaut, Jean-Charles, Aziz Moukrim and Eric Sanlaville (2013). *Flexibility and robustness in scheduling*. John Wiley & Sons.

Bożejko, Wojciech, Jarosław Pempera and Czesław Smutnicki (2009). Parallel simulated annealing for the job shop scheduling problem. In: *Computational Science–ICCS 2009*. pp. 631–640. Springer.

Chiang, Whay-Yu and Mark S Fox (1990). Protection against uncertainty in a deterministic schedule. In: *Fourth International Conference on Expert Systems in Production and Operations Management, South California, USA*. Vol. 17.

Coello, Carlos A Coello, David A Van Veldhuizen and Gary B Lamont (2002). *Evolutionary algorithms for solving multi-objective problems*. Vol. 242. Springer.

Czyżżak, Piotr and Adrezej Jaszkiewicz (1998). Pareto simulated annealing?a metaheuristic technique for multiple-objective combinatorial optimization. *Journal of Multi-Criteria Decision Analysis* **7**(1), 34–47.

Davenport, AJ (2001). C. ge ot & jc beck. slack-based techniques for robust schedules. In: *Six European Conference on Planning (ECP-2001), Toledo, Spain*.

Davenport, Andrew J and J Christopher Beck (2000). A survey of techniques for scheduling with uncertainty. *Unpublished manuscript. Available from http://tidel. mie. utoronto. ca/publications. php*.

Demirkol, Ebru, Sanjay Mehta and Reha Uzsoy (1998). Benchmarks for shop scheduling problems. *European Journal of Operational Research* **109**(1), 137–141.

Fernández, Miguel Angel González (2011). Soluciones metaheurísticas al"job-shop scheduling problem with sequence-dependent setup times". PhD thesis. Universidad de Oviedo.

Fonseca, Carlos M and Peter J Fleming (1993). Multiobjective genetic algorithms. In: *Genetic algorithms for control systems engineering, IEE colloquium on*. IET. pp. 6–1.

Fonseca, Carlos M, Peter J Fleming et al. (1993). Genetic algorithms for multiobjective optimization: Formulationdiscussion and generalization.. In: *ICGA*. Vol. 93. Citeseer. pp. 416–423.

French, Simon (1982). *Sequencing and scheduling: an introduction to the mathematics of the job-shop*. Vol. 683. Ellis Horwood Chichester.

Gabel, Thomas (2009). Multi-agent reinforcement learning approaches for distributed job-shop scheduling problems.

Gabel, Thomas and Martin Riedmiller (2007). On a successful application of multi-agent reinforcement learning to operations research benchmarks. In: *Approximate Dynamic Programming and Reinforcement Learning, 2007. ADPRL 2007. IEEE International Symposium on*. IEEE. pp. 68–75.

Gabel, Thomas and Martin Riedmiller (2012). Distributed policy search reinforcement learning for job-shop scheduling tasks. *International Journal of Production Research* **50**(1), 41–61.

Gao, Hong (1996). *Building robust schedules using temporal protection: an empirical study of constraint based scheduling under machine failure uncertainty.*. University of Toronto.

Garen, J (2002). Multiobjective job-shop scheduling with genetic algorithms using a new representation and standard uniform crossover. In: *Workshop on Multiple Objective Metaheuristics, Paris, Nov.*

Garey, Michael R, David S Johnson and Larry Stockmeyer (1974). Some simplified np-complete problems. In: *Proceedings of the sixth annual ACM symposium on Theory of computing*. ACM. pp. 47–63.

Garey, Michael R, David S Johnson and Ravi Sethi (1976). The complexity of flowshop and jobshop scheduling. *Mathematics of operations research* **1**(2), 117–129.

Gomes, Carla P (2000). Artificial intelligence and operations research: challenges and opportunities in planning and scheduling. *The Knowledge Engineering Review* **15**(01), 1–10.

Graham, Ronald L, Eugene L Lawler, Jan Karel Lenstra and AHG Rinnooy Kan (1979). Optimization and approximation in deterministic sequencing and scheduling: a survey. *Annals of discrete mathematics* **5**, 287–326.

Hansen, Michael Pilegaard (1997). Tabu search for multiobjective optimization: Mots. In: *Proceedings of the 13th International Conference on Multiple Criteria Decision Making*. Citeseer. pp. 574–586.

Herrmann, J, Chung-Yee Lee and J Snowdon (1993). A classification of static scheduling problems. *Complexity in numerical optimization* pp. 203–253.

Herroelen, Willy and Roel Leus (2005). Project scheduling under uncertainty: Survey and research potentials. *European journal of operational research* **165**(2), 289–306.

Jennings, Nicholas R and Michael Wooldridge (1998). Applications of intelligent agents. In: *Agent technology*. pp. 3–28. Springer.

Jiménez, Yailen Martínez (2012). A generic multi-agent reinforcement learning approach for scheduling problems.

Kachitvichyanukul, Voratas and Siriwan Sitthitham (2011). A two-stage genetic algorithm for multi-objective job shop scheduling problems. *Journal of Intelligent Manufacturing* **22**(3), 355–365.

Kalech, Meir and Gal A Kaminka (2005). Towards model-based diagnosis of coordination failures. In: *AAAI*. Vol. 5. pp. 102–107.

Knowles, Joshua D and David W Corne (2000). M-paes: A memetic algorithm for multiobjective optimization. In: *Evolutionary Computation, 2000. Proceedings of the 2000 Congress on*. Vol. 1. IEEE. pp. 325–332.

Kumar, Rajeev and Peter Rockett (2002). Improved sampling of the pareto-front in multiobjective genetic optimizations by steady-state evolution: a pareto converging genetic algorithm. *Evolutionary computation* **10**(3), 283–314.

Lei, Deming (2008). A pareto archive particle swarm optimization for multi-objective job shop scheduling. *Computers & Industrial Engineering* **54**(4), 960–971.

Ma, Shuping, Chenghui Zhang and Zhaolin Cheng (2008). Delay-dependent robust h? control for uncertain discrete-time singular systems with time-delays. *Journal of Computational and Applied Mathematics* **217**(1), 194–211.

Moriarty, David E, Alan C Schultz and John J Grefenstette (1999). Evolutionary algorithms for reinforcement learning. *J. Artif. Intell. Res.(JAIR)* **11**, 241–276.

Nag, Kaustuv, Tandra Pal and Nikhil R Pal (2015). Asmiga: An archive-based steady-state micro genetic algorithm. *Cybernetics, IEEE Transactions on* **45**(1), 40–52.

Nowicki, Eugeniusz and Czesław Smutnicki (2005). An advanced tabu search algorithm for the job shop problem. *Journal of Scheduling* **8**(2), 145–159.

Policella, Nicola (2005). Scheduling with uncertainty: a proactive approach using partial order schedules. *Artificial Intelligence Communications* **18**(2), 165–168.

Rodrigues Gomes, Eduardo and Ryszard Kowalczyk (2009). Dynamic analysis of multiagent q-learning with ε-greedy exploration. In: *Proceedings of the 26th Annual International Conference on Machine Learning*. ACM. pp. 369–376.

Rudy, JarosÅ et al. (2014). Solving multi-objective job shop problem using nature-based algorithms: new pareto approximation features. *An International Journal of Optimization and Control: Theories & Applications (IJOCTA)* **5**(1), 1–11.

Sabuncuoglu, Ihsan and Selcuk Goren (2009). Hedging production schedules against uncertainty in manufacturing environment with a review of robustness and stability research. *International Journal of Computer Integrated Manufacturing* **22**(2), 138–157.

Schaffer, J David (1985). Multiple objective optimization with vector evaluated genetic algorithms. In: *Proceedings of the 1st international Conference on Genetic Algorithms*. L. Erlbaum Associates Inc.. pp. 93–100.

Sha, DY and Cheng-Yu Hsu (2006). A hybrid particle swarm optimization for job shop scheduling problem. *Computers & Industrial Engineering* **51**(4), 791–808.

Sha, DY and Hsing-Hung Lin (2010). A multi-objective pso for job-shop scheduling problems. *Expert Systems with Applications* **37**(2), 1065–1070.

Silva, J Dario Landa, Edmund K Burke and Sanja Petrovic (2004). An introduction to multiobjective metaheuristics for scheduling and timetabling. In: *Metaheuristics for multiobjective optimisation*. pp. 91–129. Springer.

Simon, Herbert A and Glenn Lea (1974). Problem solving and rule induction: A unified view.

Soyster, Allen L (1973). Technical note?convex programming with set-inclusive constraints and applications to inexact linear programming. *Operations research* **21**(5), 1154–1157.

Suresh, RK and KM Mohanasundaram (2006). Pareto archived simulated annealing for job shop scheduling with multiple objectives. *The International Journal of Advanced Manufacturing Technology* **29**(1), 184–196.

Sutton, Richard S and Andrew G Barto (1998). *Reinforcement learning: An introduction*. Vol. 1. MIT press Cambridge.

Sycara, Katia P (1998). Multiagent systems. *AI magazine* **19**(2), 79.

Tan, Choo Jun, Samer Hanoun and Chee Peng Lim (2015). A multi-objective evolutionary algorithm-based decision support system: A case study on job-shop scheduling in manufacturing. In: *Systems Conference (SysCon), 2015 9th Annual IEEE International*. IEEE. pp. 170–174.

Udomsakdigool, Apinanthana and Voratas Khachitvichyanukul (2011). Ant colony algorithm for multi-criteria job shop scheduling to minimize makespan, mean flow time and mean tardiness. *International Journal of Management Science and Engineering Management* **6**(2), 116–122.

Vilcot, Geoffrey and Jean-Charles Billaut (2008). A tabu search and a genetic algorithm for solving a bicriteria general job shop scheduling problem. *European Journal of Operational Research* **190**(2), 398–411.

Wang, Chang-qing, Yun-fu Cao and Guo-zhong Dai (2004). Bi-directional convergence aco for job-shop scheduling. *COMPUTER INTEGRATED MANUFACTURING SYSTEMS-BEIJING-* **10**, 820–824.

Wooldridge, Michael and Nicholas R Jennings (1995). Intelligent agents: Theory and practice. *The knowledge engineering review* **10**(02), 115–152.

Wooldridge, Michael J and Nicholas R Jennings (1999). Software engineering with agents: Pitfalls and pratfalls. *IEEE Internet Computing* **3**(3), 20–27.

Young, Peyton (1995). Optimal voting rules. *The Journal of Economic Perspectives* pp. 51–64.

Zhang, Wei and Tom G Dietterich (1996). High-performance job-shop scheduling with a time–delay td network. *Advances in neural information processing systems* **8**, 1024–1030.

Zitzler, Eckart (1999). *Evolutionary algorithms for multiobjective optimization: Methods and applications*. Vol. 63. Citeseer.