PONTIFICIA UNIVERSIDAD CATÓLICA DE VALPARAÍSO
FACULTAD DE INGENIERÍA
ESCUELA DE INGENIERÍA INFORMÁTICA

UTILIZACIÓN DE ALGORITMOS GENÉTICOS EN EL PROBLEMA DE JOB-SHOP

CRISTIAN ALEXIS SEREY AGUILERA

INFORME FINAL DEL PROYECTO
PARA OPTAR AL TÍTULO PROFESIONAL DE
INGENIERO CIVIL EN INFORMÁTICA

(DICIEMBRE 2007)

Pontificia Universidad Católica de Valparaíso
Facultad de Ingeniería
Escuela de Ingeniería Informática

UTILIZACIÓN DE ALGORITMOS GENÉTICOS EN EL PROBLEMA DE JOB-SHOP

CRISTIAN ALEXIS SEREY AGUILERA

Profesor Guía: **Claudio Cubillos Figueroa**

Profesor Co-referente: **Nibaldo Rodríguez Agurto**

Carrera: **Ingeniería Civil en Informática**

(Diciembre 2007)

Agradecimientos

Deseo expresar mi gratitud a todas aquellas personas que hicieron posible que llegase a este punto en mi carrera. En particular deseo agradecer a mis Padres por su apoyo, al Señor Pedro Hidalgo por su gentil ayuda, y a mis amigos por su compañía a lo largo de la carrera.

Resumen

En el problema de planificación Job-Shop Scheduling Problem (JSSP), se tiene un conjunto de máquinas y un conjunto de trabajos con sus respectivas operaciones a ser programadas en las máquinas, bajo relaciones de precedencia y de capacidad de proceso de las operaciones. El nivel de dificultad de este problema ha generado un creciente uso de heurísticas para su resolución. En este contexto, la utilización de Algoritmos Genéticos brinda buenos resultados según diversos estudios.

El objetivo del presente trabajo es el desarrollo y evaluación de una implementación de Algoritmos Genéticos que permita resolver el problema descrito; llevándose a cabo para ello el proceso de recolección de información que aporta la base teórica al trabajo, el desarrollo y construcción de los prototipos y algoritmo final, sometiéndolas a las pruebas correspondientes.

Palabras Claves: Job-Shop, JSP, JSSP, Algoritmos Genéticos

Abstract

In the Job-Shop Scheduling Problem, we have a set of machines and a set of jobs with their respective operations to be scheduled in the machines, under precedence relations and processing capacities of the operations. The level of difficulty of this problem has generated a rising use of heuristics for its resolution. In this context, the utilization of Genetic Algorithms offers decent results according diverse studies.

The objective of the present work is the development and evaluation of an implementation of Genetic Algorithms that allow solving the described problem; being carried out the process of information gathering that constitutes the theoretical basis of this work, the development and construction of the prototypes and final algorithm, subjecting them to the correspondent tests.

Palabras Claves: Job-Shop, JSP, JSSP, Genetic Algorithms

INDICE

CAPITULO 1

PRESENTACIÓN DEL TEMA

1.1.- INTRODUCCIÓN

La planificación es una de las más importantes actividades en un proceso productivo. Una planificación precisa permite un mejor uso de los recursos. De todos los problemas de planificación, que se relacionan con un ambiente productivo, tales como Job-Shop, Flow-Shop y Open-Shop; el de Job-Shop es el que ha sido más estudiado. En investigación de operaciones se han dedicado décadas de esfuerzo con el fin de encontrar métodos eficientes para este problema.

El Job-Shop Scheduling Problem es un problema de optimización combinatoria y consiste, de forma genérica, en la asignación de un número determinado de actividades en un número finito de recursos disponibles. El objetivo al optimizar este problema es encontrar una permutación de dichas actividades que minimice el tiempo de término de todas las actividades en conjunto.

De este problema se pueden encontrar diversas variantes cuya principal diferencia yace en las restricciones propias de cada problema en particular. El tipo de problema sobre el cual se enfoca el proyecto se describe sucintamente a continuación:

El Job-Shop Scheduling Problem provee un conjunto finito de trabajos J, que deberán ser procesados en un conjunto finito M de máquinas. A cada trabajo que se procesa en cada máquina se le denomina operación y esta operación tendrá asignado un tiempo especıfico de procesamiento. Las operaciones correspondientes a un trabajo tendrán una secuencia dada por el problema en particular y esta secuencia será conocida de antemano e inmutable.

El objetivo al optimizar el Job-Shop Scheduling Problem es encontrar un plan de ejecución o programa (permutación de operaciones) donde todas las operaciones hayan sido concluidas en el menor tiempo posible.

Finalmente, el número de planes (soluciones) que pueden ser generados para un problema en particular, sabiendo que cada secuencia de operaciones puede ser permutada independientemente, es de

$(j!)^m$ donde *j* denota el número de trabajos y *m* el número de máquinas. Como ejemplo: si se tiene un total de 24 trabajos a ser procesados en una sola máquina, se tienen entonces: 24! = 6.20×10^{23} posibles soluciones. Esto da una idea de lo complejo que resultaría resolver el problema por enumeración.

Debido a la complejidad del problema del problema de Job-Shop los métodos convencionales de Investigación de Operaciones han resultado útiles para problemas pequeños y se han utilizado múltiples algoritmos para resolverlo en instancias mayores, sin embargo no resulta posible contar con un método totalmente determinista para removerlo en el caso general. De ahí que se hayan desarrollado diversas heurísticas de solución del problema de Job-Shop basadas en Algoritmos Genéticos [1] , Búsqueda Tabú [2], Simulated Annealing [3], entre otras apreciables en la literatura ([4,[5]).

Una de las herramientas señaladas en el párrafo anterior que ha sido utilizada con bastante éxito son los Algoritmos Genéticos. Los Algoritmos Genéticos son introducidos por Holland [[6]] y se utilizan conceptos de genética, población y teoría de la evolución para construir algoritmos que tratan de optimizar la aptitud de los individuos de una población a través del paso de generaciones.

Tras su aparición en los años 70, se han ganado su aceptación por su grado de eficiencia en resolver problemas de distinta complejidad, claro que esta eficiencia depende del problema planteado, en conjunción con la estrategia adoptada por el algoritmo para abordarlo.

Los Algoritmos Genéticos han sido aplicados al problema de Job-Shop, y han presentado buenos resultados, sin embargo, se han presentado variaciones al algoritmo para incluir particularidades de Job-Shop Scheduling Problem y para reducir (o eliminar) el trabajo de reparación de cromosomas, en las etapas del algoritmo. En *"Studies on Metaheuristics for Jobshop and Flowshop Scheduling Problems"* de Yamada [7] se puede apreciar un análisis de variaciones al algoritmo y diversos operadores propuestos para Job-Shop Scheduling Problem.

Este trabajo se encuentra enfocado a la generación de un algoritmo genético que permita resolver el problema de Job-Shop Scheduling, comparando los resultados obtenidos al utilizar dicho algoritmo con los valores óptimos y mejor valor conocido de diversas instancias del problema de Job-Shop Scheduling.

Para ello en lo que resta de capitulo se presentan los objetivos del proyecto y se describe el plan de trabajo. Mientras que en los capítulos siguientes se presenta una revisión teórica del problema de Job-Shop;

una revisión de Algoritmos Genéticos y sus consideraciones teóricas; mientras que en el cuarto capítulo se procede a revisar algunos trabajos relacionados que brindan luz sobre el tema, presentándose en el quinto capítulo el modelo de algoritmo final propuesto y algunas de la decisiones asociadas a su construcción implementación y configuraciones; dando cuenta finalmente en el sexto capítulo de las pruebas realizadas tanto en prototipos como en las configuraciones propuestas para el algoritmo final y sus resultados; para la obtención de conclusiones presentadas en su propio capítulo.

1.2.- OBJETIVOS DEL PROYECTO

1.2.1.- Objetivo General

Desarrollar y evaluar una implementación de Algoritmos Genéticos que permita resolver el problema de minimización del Makespan en Job-Shop.

1.2.2.- Objetivos Específicos

- Estudiar distintas soluciones propuestas para el problema de Job-Shop, poniendo énfasis, en trabajos vinculados con Algoritmos Genéticos. Junto con estudiar las características particulares de los Algoritmos Genéticos aplicados al problema.
- Desarrollar una representación de los planes de ejecución, para obtención del nivel de calidad de una solución, implementación de algoritmos de generación de población inicial, selección, cruzamiento y mutación para aplicarlos al problema, para integración incremental de modo de implementar la estructura general de algoritmo.
- Llevar el resultado anterior a un programa para el proceso de experimentación, utilizando el lenguaje de programación C#.
- Realizar las pruebas del programa mencionado, usando datos experimentales obtenidos de problemas planteados en la literatura sobre el problema de Job-Shop.

1.3.- METODOLOGÍA DE TRABAJO

Para desarrollar el proyecto de titulo, se utilizó una metodología de investigación científica clásica. La misma puede ser descrita bajo las siguientes etapas:

- Se inicia con la concepción de una idea central que permita plantear el problema, por medio de los objetivos, dificultades asociadas y justificación de la viabilidad de afrontar el problema. Esto se refleja en los objetivos del proyecto y en la introducción bibliográfica en que se muestra un resumen del estado del arte del problema.
- Se genera un marco teórico, que contempla la revisión de literatura relacionada con el problema, de modo de obtener y seleccionar la información relevante para su desarrollo. Esto se encuentra directamente asociado con las referencias utilizadas en la investigación y con los contenidos teóricos presentados en el curso del proyecto.
- Se desarrolla una hipótesis de trabajo o planteamiento previo sobre la problemática, se especifican variables del problema de modo conceptual y operacional, además se establece si el problema se abordará experimentalmente. La hipótesis surge del desarrollo del marco teórico.
- Se definen los medios de experimentación y las muestras para realizar los experimentos. En el caso de esta investigación corresponden a datos de entrada analizados en estudios anteriores y a datos de entrada que corresponden a problemas ya definidos y aceptados comúnmente dentro del ámbito de investigación del problema. Mientras que el medio o instrumento de experimentación corresponderá al programa a desarrollar según lo planteado en los objetivos.
- Se realiza el análisis de los resultados obtenidos, a través de un problema planteado convenientemente para estos efectos. El problema debe contemplar los aspectos más relevantes del problema que permitan comparar con otros resultados.
- Se elaboran y presentan los reportes de investigación adecuados a la estructura de presentación de ellos.

1.4.- PLAN DE TRABAJO

Para consecución de los objetivos planteados, se consideró la división en tres etapas generales para la realización del proyecto:

- Recolección de información y análisis investigativo: En esta etapa se realiza una investigación de las fuentes y referencias encontradas. De ellas se obtienen los elementos más relevantes y útiles que tengan relación con las características de Job-Shop Problem, los aspectos teóricos de Algoritmos Genéticos y la aplicación de Algoritmos Genéticos en Job-Shop Scheduling Problem.
- Desarrollo y construcción del modelo de algoritmo: Con la información recolectada y analizada en la etapa anterior, se procede a definir y desarrollar los elementos del modelo de algoritmo a utilizar. Para validar y verificar el modelo, se desarrollan dos prototipos que permiten estudiar el comportamiento y desempeño. Como se ha mencionado, esta etapa iterativa e incremental finaliza con un modelo que presenta un funcionamiento aceptable en relación a los objetivos propuestos.
- Implementación final y pruebas finales: En base a los prototipos y al modelo de algoritmo obtenidos en la etapa anterior, se desarrolla el software para generar soluciones en base a datos de entrada adecuados al problema. Se realizan distintas ejecuciones sobre problemas planteados en la literatura que permiten medir la eficiencia en la implementación del modelo, realizándose comparaciones con resultados encontrados en literatura.

CAPITULO 2

EL PROBLEMA DE PLANIFICACIÓN DE JOB-SHOP

En los términos más amplios, un problema de planificación especifica un conjunto de tareas a ser realizadas utilizando un conjunto finito de recursos. El objetivo es planificar el procesamiento de todas estas tareas de modo que alguna medida de la eficiencia global sea maximizada, por ejemplo el tiempo requerido para completar todas las tareas. La planificación es observable en aplicaciones reales, desde sistemas operativos a problemas logísticos de gran escala. Los detalles de un problema en particular son altamente variables, lo que lleva a los investigadores a desarrollar problemas que representen de manera abstracta una clase particular de problemas reales. A pesar que existen diversos problemas abstractos, el problema más ampliamente estudiado es el problema de planificación estático de Job-Shop (Job-Shop Scheduling Problem, JSSP). En el Job-Shop Scheduling Problem existen trabajos y máquinas donde cada trabajo debe ser procesado una vez en cada máquina, por un tiempo establecido y en un orden previamente establecido; el procesamiento de un trabajo en una máquina se denomina operación. Cada máquina sólo puede procesar una sola operación a la vez, la que una vez iniciada debe ser procesada hasta completarse.

Distintos objetivos de planificación han sido considerados para el Job-Shop Scheduling Problem, empero gran parte de las investigaciones considera el objetivo de minimización del tiempo de completado de todos los trabajos (o *makespan*) o la minimización de la sumatoria de *flowtime* (diferencia entre el tiempo en que termina un trabajo y el tiempo en que comienza este) de de los trabajos.

Aunque el Job-Shop Scheduling Problem es el problema más estudiado, paradójicamente es el menos realista; D.C. Mattfeld [8] apunta las siguientes restricciones, las cuales no necesariamente se encuentran presentes en problemas reales:

- Dos operaciones de un mismo trabajo no pueden ser procesadas simultáneamente.
- No existe derecho preferente de ejecución, la interrupción del procesamiento de operaciones no es permitido.
- Ningún trabajo es procesado dos veces en la misma máquina.
- Cada trabajo debe ser procesado hasta completarse, sin interrupción.

- Los trabajos pueden iniciarse en cualquier instante, no existen fechas de inicio.
- Los trabajos pueden terminarse en cualquier instante, no existen fechas límites de término.
- Los trabajos deben esperar por la siguiente máquina en el orden de proceso hasta que esté disponible.
- Ninguna máquina puede procesar más de un trabajo a la vez.
- Los tiempos de configuración de las máquinas son considerados insignificantes y no se cuentan.
- Existe sólo una máquina de cada tipo.
- Las máquinas pueden estar ociosas dentro del plan de ejecución.
- Las máquinas se encuentran disponibles en todo momento, esto es, no fallan.
- El orden de procesamiento de cada trabajo es conocido de antemano y es inmutable.

A pesar de ello resulta atrayente para los investigadores, debido a ser considerado uno de los de los problemas más difíciles de la categoría NP-Hard; lo que ha llevado a que sea considerado un desafío intelectual el proponer una nueva solución o una nueva instancia para evaluar.

2.1.- DEFINICIÓN DEL PROBLEMA DE PLANIFICACIÓN DE JOB-SHOP

En el problema de planificación de Job-shop de $n \times m$, existen n trabajos, cada uno de los cuales debe ser procesado exactamente una vez en cada una de las m máquinas. Cada trabajo i sigue una secuencia a través de m máquinas en un orden predefinido π_i, donde $\pi_i(j)$ denota la j-esima máquina en la secuencia, como se ve $1 \leq i \leq n$ y $1 \leq j \leq m$. El procesado del trabajo i en la máquina $\pi_i(j)$ es denotado por o_{ij} y es llamado *operación*. Una operación o_{ij} es procesada en la máquina $\pi_i(j)$ por una duración entera $\tau_{ij} \geq 0$. Para $2 \leq j \leq m$, o_{ij} no puede iniciar su procesamiento hasta que o_{ij-1} no haya terminado. Las restricciones impuestas por las secuencias de procesamiento de cada trabajo son llamadas restricciones de precedencia y las impuestas por las unidades de capacidad son llamadas restricciones de recursos.

Cada solución s de una instancia Ω de Job-Shop Scheduling Problem de n x m especifica un orden de procesado para todos los trabajos sobre cada máquina. Existen $n!$ posibles ordenes de procesamiento para cada máquina, resultando en $(n!)^m$ posibles soluciones a Ω, entre las cuales se encuentran muchas soluciones que no son factibles, por no cumplir con las restricciones de precedencia o por crear dependencias cíclicas. El conjunto de soluciones factibles de una instancia Ω se denota por S_Ω.

Cada solución $s \in S_\Omega$ especifica de manera implícita el tiempo más temprano de inicio (est_{ij}) para cada operación o_{ij} tal que todas las restricciones de precedencia y recursos son satisfechas. El tiempo más temprano de completado $\left(ect_{ij}\right)$ es dado entonces por $ect_{ij} = est_{ij} + \tau_{ij}$. El makespan $C_{\max}(s)$ de una solución $s \in S_\Omega$ es el máximo tiempo más temprano de completado de la ultima operación de cualquier trabajo i, esto es $C_{\max}(s) = \max(ect_{1m}, ect_{2m}, ..., ect_{nm})$. El makespan optimo denotado por $C^*_{\max}$, es igual al mínimo makespan entre las soluciones factibles, esto es $C^*_{\max} = \min_{\forall s \in S_\Omega}(C_{\max}(s))$ [9].

2.2.- FORMULACIÓN MATEMÁTICA DEL PROBLEMA

Es reconocido por muchos investigadores que los problemas de planificación pueden ser resueltos de manera óptima usando técnicas de programación matemática; y la formulación matemática más común para el Job-Shop Scheduling Problem es la programación lineal entera mixta, propuesta por Alan Manne en el año 1960. Se compone de un conjunto de restricciones lineales y una función objetivo lineal, pero con la restricción adicional que algunas de las variables de decisión son enteras. En el modelo las variables enteras son binarias y se utilizan para implementar las restricciones disyuntivas. K corresponde a un valor arbitrario grande que ha sido indicado que debe ser mayor que la suma de los tiempos de procesamiento de todas la operaciones menos el tiempo de procesamiento más pequeño.

Minimizar $C_{\max}$

Sujeto a:

Tiempos de inicio $t_{ik} \geq 0$ $\{i, p\} \in J$ $\{k, h\} \in M$

Restricción de Precedencia: $t_{ik} - t_{ih} \geq \tau_{ih}$ si o_{ih} precede a o_{ik}

Restricción Disyuntiva: $t_{pk} - t_{ik} + K(1 - y_{ipk}) \geq \tau_{ik}$ $y_{ipk} = 1$, *si* o_{ik} *precede a* o_{pk}

$t_{ik} - t_{pk} + K(y_{ipk}) \geq \tau_{pk}$ $y_{ipk} = 0$, en otro caso

Donde $K > \left(\sum_{i=1}^{n} \sum_{k=1}^{m} \tau_{ik} - \min(\tau_{ik}) \right)$

Donde:

t_{ik} : Tiempo de inicio de cada operación.

J : Conjunto de n trabajos a ser procesados

M : Conjunto de m recursos o máquinas.

o_{ik} : Operación del trabajo j_i que debe ser procesado en la máquina m_k por un periodo ininterrumpido de tiempo τ_{ik}

A pesar de la elegancia conceptual, el número de variables enteras crece exponencialmente e incluso si se utilizan formulaciones más compactas estas siguen requiriendo un gran número de restricciones. Como resultado las técnicas de programación matemática son capaces de resolver instancias altamente simplificadas, dentro de un tiempo razonable, de modo que no resulta sorprendente que técnicas adecuadas para Job-Shop Scheduling Problem, se encuentren en otras áreas de investigación [5].

2.3.- SOLUCIONES Y PLANES DE EJECUCIÓN

Una solución factible $s \in S_\Omega$ especifica un orden de procesamiento de trabajos para cada máquina en el cual no existen dependencias cíclicas de orden entre pares de operaciones. Toda referencia al tiempo es implícita, pues el tiempo más temprano de inicio puede ser calculado para cada operación. En contraste, un plan de ejecución factible especifica el tiempo de inicio de cada trabajo de modo que las restricciones son satisfechas. En general, existe una relación de uno a muchos entre soluciones y planes de ejecución, pues no existe el requerimiento de iniciar lo más temprano posible; asumiendo un horizonte de tiempo infinito, surgen infinitos planes de ejecución a partir de una solución. Es por ello que con el fin de reducir el conjunto de planes de ejecución, se consideran inadmisibles aquellos planes en que las operaciones comienzan después de su tiempo de inicio más temprano. Para conseguir el makespan $C_{\max}(s)$ es necesario planificar algún subconjunto de las operaciones en su tiempo de inicio más temprano; consecuentemente, las operaciones son asignadas a su tiempo de inicio más temprano. A estos planes resultantes se les conoce como planes admisibles o semi- activos, se puede apreciar que entre estos y las soluciones existe una relación de uno a uno, por lo que generalmente ambos términos se usan indistintamente.

En un plan de ejecución semi–activo, la única manera de reducir el makespan es reordenando la secuencia de procesamiento de trabajos en al menos una máquina. En particular, puede ser posible mover una operación de modo que inicie antes de su tiempo de partida asignado sin que retrase el comienzo de cualquier otra operación, por ejemplo si existe tiempo muerto en medio de la secuencia de una máquina. Ese tipo de movimiento es llamado *left-shift global*, y un plan en que no es posible realizarlo sin que afecte a otra operación es llamado activo. Un plan en el cual ninguna máquina esta ociosa, si existe una operación disponible para procesar, es denominado sin demora (*non-delay)* y por definición también son planes activos.

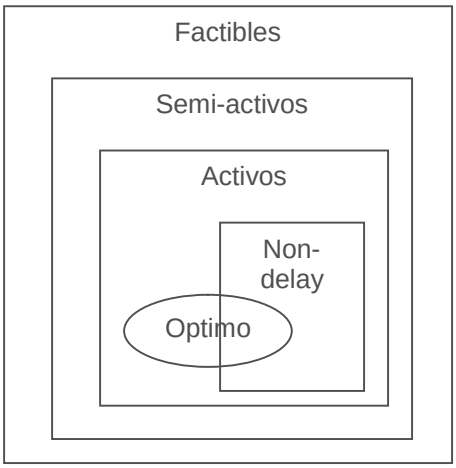

Figura 2-1. – Clasificación de los Planes de Ejecución.

Como se puede observar en la Figura 2-1 un plan de ejecución óptimo puede no ser *non-delay*, pero si activo, por lo que puede parecer ventajoso restringir la búsqueda al conjunto de planes activos, pero no resulta practico en todos los casos, pues el problema de verificar que no existen *left-shifts globales* posibles, es del tipo NP-Completo según lo planteado por Vaessens en su trabajo del año 1995. A raíz de ello muchos algoritmos de búsqueda restringen el espacio de búsqueda a las soluciones semi-activas.

2.4.- REPRESENTACIÓN DE GRAFO DISYUNTIVO

El Job-Shop Scheduling Problem puede ser descrito formalmente por un grafo disyuntivo $G = (V, C \cup D)$, donde:

- V es el conjunto de nodos, representando las operaciones de los trabajos más dos nodos especiales, $source(0)$ y $sink*$ que representan el comienzo y el fin del plan de ejecución, respectivamente.
- C es el conjunto de arcos conjuntivos, representando las secuencias tecnológicas de las operaciones

- D es un conjunto de arcos disyuntivos, representando pares de operaciones que deben realizarse en las mismas máquinas.

El tiempo de proceso de cada operación es el valor del peso unido al nodo correspondiente, en la siguiente figura se puede apreciar un ejemplo de grafo disyuntivo similar al que aparece en *"Genetic algorithms in engineering systems"*de Yamada y Nakano [10].

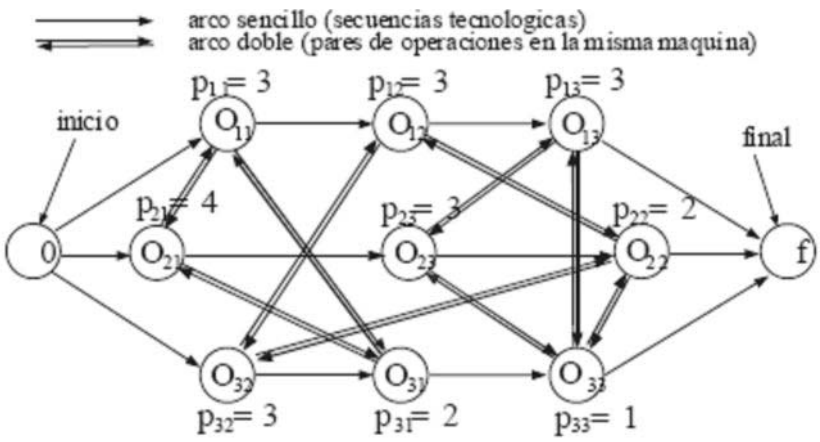

Figura 2-2. – Ejemplo de Grafo Disyuntivo.

La planificación de Job-Shop puede ser vista como definición del orden entre todas las operaciones que deben ser procesadas en la misma máquina, esto es fijar las precedencias entre estas operaciones. En el modelo de grafo disyuntivo, esto se logra convirtiendo todos los arcos no dirigidos en arcos dirigidos. Una *selección* es un conjunto de arcos dirigidos seleccionados de arcos disyuntivos. Por definición [11] una selección es *completa* si todas las disyunciones son seleccionadas. Es *consistente* si el grafo dirigido resultante es acíclico. Un plan obtenido únicamente de una selección completa consistente por secuenciación de operaciones tan tempranas como sea posible es llamado plan semi-activo. Una selección completa consistente y el correspondiente plan semi-activo pueden representarse sin confusión por el mismo símbolo. El *makespan* es dado por el largo de la ruta más larga de pesos, desde el nodo *source* hasta el nodo *sink* en el grafo. Esta ruta es llamada *ruta crítica* y esta compuesta de una secuencia de

operaciones criticas. Una secuencia de operaciones críticas consecutivas en la misma máquina es llamada *bloque crítico*.

2.5.- LOS PROBLEMAS DE BENCHMARK

Para realizar las pruebas de las diversas técnicas y métodos de solución del Job-Shop Scheduling Problem, se utilizan diversos problemas, tales como los tres problemas de tamaños 6 x 6, 10 x 10 y 20 x 5 (conocidos como *mt06*, *mt10* y *mt20*) formulados por Muth y Thompson [12], los que son utilizados comúnmente como base de pruebas para medir la efectividad de un determinado método.

Applegate y Cook propusieron una selección de problemas de benchmark llamados los *"diez problemas difíciles"* como un desafío computacional mayor que el problema *mt10*, por medio de la recolección de problemas difíciles presentes en la literatura, algunos de los cuales aún no han sido resueltos. Este grupo de problemas está compuesto *ab7, abz8, abz9* y *la21, la24, la25, la27, la38, la40.* Los problemas *abz* fueron propuestos originalmente por Adams en el año 1988 [13] y los problemas *la* son parte de los cuarenta problemas *la01 - la40* originales de Lawrence del año 1984 [14].

En el año 1993 Taillard propuso un conjunto de ochenta problemas que cubre un rango variado de tamaños y dificultades [15]. Ellos son generados aleatoriamente por un algoritmo simple, de modo que se ha vuelto común el uso de ellos o el uso de los diez problemas descritos en lugar de los problemas *mt10* y *mt20*, para realizar pruebas y benchmark.

Estos problemas se encuentran disponibles en la OR Library, en la sección de Job-Shop ubicada en: http://people.brunel.ac.uk/~mastjjb/jeb/orlib/jobshopinfo.html.

2.6.- PROBLEMAS SIMILARES Y VARIANTES DE JSSP

En esta sección se describirán de manera breve, tres problemas de características similares a Job-Shop Scheduling Problem y dos variantes de Job-Shop Scheduling Problem. No se entrar en mucho detalle en su descripción, pues realizar un estudio detallado de ellos escapa a la finalidad de este proyecto.

2.6.1. - Open-Shop Scheduling Problem (*OSSP*)

El problema de planificación de Open-Shop, es similar a Job-Shop Scheduling Problem, con la excepción de que no existe un ordenamiento *a priori* de las operaciones dentro de un trabajo. Este problema posee un espacio de búsqueda mayor que el espacio de búsqueda de Job-Shop Scheduling Problem, y es visiblemente menos tratado en la literatura a pesar que resulta ser un problema interesante.

Este problema suele representarse de la misma manera que Job-Shop Scheduling Problem, claro que sin contemplar las relaciones de precedencia.

2.6.2. - Flow-Shop Scheduling Problem (*FSSP*)

En este problema el sistema consta de *m* máquinas dispuestas en serie y cada trabajo debe ser ejecutado por cada una de las máquinas sucesivamente, o sea todos los trabajos poseen la misma secuencia de operaciones (aun cuando pueden variar en duración). A cada trabajo le es asignado un costo basado en su tiempo de completado. Suele tratarse como un sistema con buffer FIFO en cada máquina.

Este problema al igual que el anterior sigue el mismo modelo de representación que Job-Shop Scheduling Problem, con las modificaciones correspondientes a su naturaleza.

2.6.3.- Mixed-Shop Scheduling Problem

El problema de Mixed-Shop consta, al igual que el problema de Job-Shop, de *m* máquinas y *j* trabajos, con la diferencia de que en el conjunto de trabajos se presentan algunos con características de Job-Shop y otros con características de Open-Shop (o Flow-Shop), resultando el problema finalmente una combinación de ambos.

2.6.4. - Job-Shop Scheduling Problem Con Deadlines

Se define este problema de la siguiente manera: Un conjunto de trabajos debe ser procesado en un conjunto de máquinas dentro de un tiempo mínimo de ejecución, sujeto a:

- La secuencia de máquinas para cada trabajo esta definida.
- Cada máquina puede procesar sólo un trabajo a la vez.
- Existen relaciones de precedencia entre operaciones de un mismo trabajo.

Hasta este punto el problema descrito de manera general es idéntico al Job-Shop Scheduling Problem, sin embargo la variación radica en que debe cumplir con la siguiente restricción:

- Los trabajos deben comenzar después de una fecha de puesta en marcha (*release date*) y deben ser completados antes de sus respectivas fechas limites (*deadlines)*.

Esta variante surge por motivos prácticos, el primer motivo es que el Job-Shop Scheduling Problem no captura un aspecto esencial de un departamento de producción. En la práctica no existe una ejecución simple en que todos los trabajos a planificar se encuentren disponibles en el instante inicial y sean conocidos de antemano, si no más bien los trabajos arriban periódicamente por lo que la planificación debe ser revisada y adaptada.

El segundo motivo es que existen ocasiones en que algunas máquinas no están disponibles por diversos motivos (falla o mantención), lo que lleva a asignar prioridad a los trabajos en función de su fecha límite y planificarlos acordemente. Una descripción con mayor detalle del problema se puede encontrar en *"Job Shop Scheduling With Deadlines"* de Balas [16] .

En este problema se introducen y utilizan los siguientes conceptos que no se encuentran contemplados en la definición dada de Job-Shop Scheduling Problem:

- Fecha de puesta en marcha (*Release Date*): Indica el instante de tiempo (a partir del instante cero) antes del cual un trabajo no puede iniciarse.
- Fecha Límite (*Deadline*): Corresponde al plazo máximo dentro del cual el trabajo debe estar terminado.
- Fecha Convenida (*Due Date*): Indica el instante de tiempo convenido (a partir del instante cero) antes del cual la ejecución de un trabajo debería estar terminada.
- Peso (*Weight*): Representa la importancia de un trabajo en relación a otros.
- Tiempo de Llegada (*Arrival* o *Lateness*): Es la diferencia entre el tiempo en que se completó un trabajo y su *due date*. Tomando un valor positivo si existe retraso y un valor negativo en caso de adelanto.

- Tardanza (*Tardiness*): Se define de manera similar al concepto de *lateness*, corresponde a la diferencia entre el tiempo en que se completó un trabajo y su *due date*. A diferencia de *lateness*, toma el valor cero para diferencias negativas.
- Anticipación (*Earliness*): Corresponde al tiempo de adelanto en la finalización de un trabajo, en relación a su fecha convenida.

Dados estos conceptos surgen nuevas posibilidades de optimización, tales como minimizar la suma de los valores de *lateness* o *tardiness*.

2.6.5. - Job-Shop Scheduling Problem Con Derecho Preferente (*Preemptive Job-Shop Problem*)

Este problema se encuentra asociado en general con el problema de Job-Shop con deadlines, aun cuando puede darse con el Job-Shop Scheduling Problem descrito anteriormente. En este problema a los trabajos se les asigna un valor que representa su derecho preferente de ejecución (*preemption*), de este modo se permite la interrupción del procesado de un trabajo en vista de la llegada de un trabajo con un valor mayor de *preemption*. En el caso con *deadlines*, el derecho preferente suele asignarse en función de su fecha de término y de su peso.

2.7.- TÉCNICAS UTILIZADAS EN LA SOLUCIÓN DE JSSP

Un gran número de aproximaciones al modelado y solución de Job-Shop Scheduling Problem han sido señalados en la literatura de Investigación de Operaciones, con distinto grado de éxito. Las aproximaciones giran entorno de una serie de avances tecnológicos que han ocurrido en los últimos treinta y cinco años [4. Estas incluyen programación matemática, reglas de despacho, sistemas expertos, agentes algoritmos genéticos, entre otros [3]. En esta sección se describirán de manera somera algunas técnicas destacadas en cada área separándolas según su naturaleza.

2.7.1.- Técnicas Matemáticas

La programación matemática ha sido aplicada extensamente al problema de Job-Shop Scheduling Problem. El problema ha sido formulado utilizando programación entera, programación entera mixta y programación dinámica. Hasta hace poco el uso de estas aproximaciones se encontraba limitada por el hecho que el problema pertenece a la clase NP-Hard. Para superar esta deficiencia, se ha propuesto la descomposición del problema en sub-problemas, planteando un número de técnicas para resolverlos. En adición, nuevas técnicas de solución, heurísticas más poderosas y el aumento del poder computacional han posibilitado que estas aproximaciones sean utilizadas en problemas de mayores dimensiones. Aún así las dificultades en la formulación de las restricciones de flujo de material como desigualdades matemáticas y el desarrollo de soluciones de software han limitado el uso de estas técnicas.

2.7.1.1.- Estrategia De Descomposición

En el año 1988 Davis y Jones propusieron una metodología basada en la descomposición de problemas de programación matemática que usaban las descomposiciones de tipo Benders y Dantzig/Wolfe. La metodología era parte de un ciclo cerrado, de tiempo real, de sistema de control de shop-floor con dos niveles jerárquicos. El planificador de alto nivel (por ejemplo, el supremo) especificaba el tiempo de inicio más temprano y el tiempo de término más tardío para cada trabajo. Los módulos del planificador de bajo nivel (los ínfimos) refinarían estos tiempos límites para cada trabajo por secuenciación detallada de todas las operaciones. Una función objetivo multicriterio resulta especificada, la que incluye *tardiness, throughput* y costos de utilización de procesos. La descomposición fue conseguida reordenando primero las restricciones del problema original para generar una forma de bloque angular, luego transformando esta en una estructura de árbol jerárquico. De este modo, en general se obtienen N sub-problemas más un conjunto de restricciones que contiene miembros parciales de cada sub-problema, este conjunto es denominado restricciones de *"coupling"*, e incluyen relaciones de precedencia y manejo de materiales. Diversos autores señalan que debido a la naturaleza estocástica de los Job-Shop y la presencia de múltiples, y a veces contrapuestos, objetivos, torna complicado expresar las restricciones de *coupling*

usando relaciones matemáticas exactas. Ello torna casi imposible el desarrollo de una metodología de solución general [4.

2.7.1.2. - Branch And Bound y Relajación Lagrangiana

La técnica de *branch and bound* es una técnica enumerativa cuya idea principal es conceptualizar el problema como un árbol de decisión. Cada punto de decisión –un nodo- corresponde a una solución parcial. De cada nodo surge un número de nuevas ramas, una por cada posible decisión. El proceso continúa hasta que los nodos hojas, que no generan más ramas, son alcanzados [5].

A pesar que métodos eficientes de *branch and bound* han sido desarrollados para mejorar la velocidad de búsqueda, sigue siendo un procedimiento intensivo computacionalmente para problemas de gran envergadura.

La técnica llamada relajación Langrangiana, que se ha utilizado por más de treinta años, considera que si una restricción es considerada un problema para la resolución, entonces por que no removerla. La relajación lagrangiana resuelve problemas de programación entera por medio de la omisión de restricciones de valor entero específicas y agrega los correspondientes costos (debido a las omisiones o relajaciones) a la función objetivo.

Al igual que *branch and bound*, esta técnica es computacionalmente intensiva para problemas de gran tamaño [4.

2.7.2.- Reglas De Despacho

Las reglas de despacho han sido aplicadas de manera consistente a problemas de planificación. Ellas son procedimientos que proveen buenas soluciones a problemas complejos en tiempo real. Han sido clasificadas principalmente de acuerdo al criterio de rendimiento bajo el cual han sido desarrolladas. La clase 1 contiene reglas de prioridad simples, las que se basan en información relacionada con los trabajos. Sus sub-clases se basan en los trozos particulares de información utilizados, por ejemplo las que se basan en el tiempo de procesamiento como *Shortest Processing Time (SPT)*, o en tiempo de arribo como *First In First Out (FIFO)*. La clase 2 consiste en la combinación de reglas de clase 1. La regla resultante puede

entonces depender de la situación existente, por ejemplo SPT hasta que el largo de la cola sea mayor a 5, entonces se cambia a FIFO. De este modo se evita que los trabajos con tiempos grandes de procesamiento estén en cola por periodos de tiempo largos. La clase 3 contiene reglas que son referidas comúnmente como Índices de Prioridad de Peso. La idea es utilizar más de una pieza de información acerca de los trabajos para determinar el plan de ejecución. A la información se le asigna pesos para reflejar su importancia relativa, usualmente es definida una función objetivo f(x) es asignada.

Por ejemplo:

$f(x)=peso_1 * tiempo\ de\ procesado\ trabajo(x) + peso_2 * (tiempo\ actual - fecha\ tope\ trabajo(x))$

Entonces cuando se necesita una nueva secuencia de tiempo, la funciones evaluada para cada trabajo en la cola, y son ordenados según la evaluación [3].

2.7.3.- Sistemas Expertos / Basados En Conocimiento

Los sistemas expertos y basados en conocimiento constan de dos partes: una base de conocimiento y un motor de inferencia para operar en esa base. Las formalizaciones del "conocimiento" que los seres humanos expertos utilizan (reglas, procedimientos, heurísticas y otro tipo de abstracciones) son capturadas en la base de conocimiento. Usualmente se incluyen tres tipos de conocimiento; procedural, declarativo, y meta. El procedural es de tipo dominio específico del problema. El conocimiento declarativo provee la entrada de datos definiendo el dominio del problema, mientras que meta conocimiento es aquel acerca de cómo utilizar los dos anteriores para resolver el problema. Variadas estructuras de datos se han utilizado para representar el conocimiento en la base de conocimiento, incluyendo redes semánticas, scripts, cálculo de predicados y reglas de producción. El motor de inferencia selecciona una estrategia a aplicar a la base (o bases) de conocimiento para resolver el problema. Pudiendo utilizar forward chaining (dirigido por los datos) o backward chaining (dirigido por las metas).

El sistema ISIS que data del año 1983 fue el primer sistema experto diseñado para Job-Shop Scheduling Problem, utilizaba un razonamiento dirigido por restricción, con tres categorías de restricción: metas organizacionales, limitaciones físicas y restricciones causales. Utilizaba una búsqueda jerárquica, dirigida por restricciones, de tres niveles, en donde las órdenes se encontraban en primer nivel, el nivel 2

determinaba la disponibilidad de recursos requeridos por la orden; mientras que la planificación detallada se realizaba en el nivel 3.

ISIS brindaba la capacidad de construir y modificar planes de ejecución de manera interactiva, en este ámbito utilizaba su conocimiento de restricciones para mantener la consistencia de los planes e identificar las decisiones de planeación que resultaran en restricciones pobremente satisfechas.

Otros sistemas expertos que han seguido los pasos de ISIS, son MPECS, OPIS (Opportunistic Intelligent Scheduler) y SONIA, detalles al respecto de ellos se pueden observar en el trabajo de Jain y Meeran [5].

2.7.4.- Agentes

El problema de Job-Shop Scheduling Problem se ha enfrentado haciendo uso de un sistema MultiAgente que involucra a dos tipos de agente: *tareas* y *recursos*. Cada agente *tarea* puede ser responsable de planificar una cierta clase de tarea como manejo de material, o inspección sobre los recursos capaces de realizar esta tareas. Esto puede ser hecho utilizando cualquier medida de rendimiento relacionada a las tareas. Cada agente *recurso* puede ser responsable por un solo recurso o una clase de recursos. Los agentes *tarea* deben enviar su solicitud de recurso al agente *recurso* apropiado, junto con el conjunto de operaciones a realizar por el recurso. En la recepción de la solicitud, el agente *recurso* debe generar un nuevo plan de ejecución utilizando sus propias medidas de rendimiento. El agente *recurso* utiliza los resultados para decidir si acepta la solicitud o no. Para evitar situaciones en que ningún agente *recurso* acepte una solicitud, se debe desarrollar un mecanismo de coordinación. Un ejemplo de utilización de MultiAgentes se puede encontrar en *"Job Shop with Multiagents"* de Wellner y Dillger [17].

2.7.5.- Simulated Annealing

La denominación del algoritmo proviene de la analogía con el proceso de templado utilizado en metalurgia. Consistiendo este en el enfriamiento gradual del metal fundido de modo que sus moléculas adoptan poco a poco una configuración de mínima energía. Al iniciarse el proceso, las moléculas vibran y se mueven caóticamente adoptando diverso tipos de configuraciones en la estructura del metal. A medida

que la temperatura decrece el movimiento se ralentiza y las moléculas tienden a adoptar de manera gradual las configuraciones de menor energía, siendo nula en el cero absoluto.

Simulated Annealing intenta realizar un proceso similar de manera numérica, donde el espacio de configuraciones viene dado por la secuencia de tareas que se desea procesar y la energía es un papel asumido por la función objetivo.

2.7.6.- Tabú Search

Tabú Search es una técnica de optimización iterativa, que consiste en un procedimiento que restringe la búsqueda y utiliza una función de memoria de corto plazo que convierte en prohibidos los t movimientos más recientes.

Este algoritmo aplicado a Job-Shop Scheduling Problem genera vecinos a través de la inversión de arcos que unen tareas adyacentes en la ruta crítica. Después que un arco ha sido invertido se introduce su inverso a la lista tabú, de longitud definida que contiene los desplazamientos prohibidos. En cada iteración se evalúan todas las soluciones del vecindario que no se encuentren en la lista tabú, calculando para cada una el valor de $C_{\max}$ y seleccionando la más prometedora.

El algoritmo de tabú search más avanzado es llamado i-TSAB, que fue introducido por Nowicki y Smutnicki el 2001 [18], que se basa en el previamente exitoso TSAB de los mismos autores. El algoritmo se caracteriza por sus componentes: un altamente restrictivo operador de movimiento, re-intensificación de la búsqueda alrededor de soluciones de alta calidad encontradas anteriormente y diversificación de la búsqueda por medio de reconexión de rutas entre soluciones de alta calidad. A pesar de su efectividad, poco y nada se conoce de cómo estos componentes y otros de carácter secundario interactúan para alcanzar el rendimiento que lo pone a la cabeza entre los algoritmos de búsqueda tabú. El análisis y detalles del algoritmo pueden encontrarse en *"On Metaheuristc "Failure Modes": A Case Study in Tabu Search for Job-Shop Scheduling"* [19] y *"Deconstructing Nowicki and Smutnicki's i-TSAB Tabu Search Algorithm for the Job-Shop Scheduling Problem"* [2].

CAPITULO 3

ALGORITMOS GENÉTICOS

En el presente capitulo se aborda el otro aspecto importante en la investigación, el concepto de Algoritmo Genético (GA). Siendo este un tema en extremo extenso como para abordarlo en detalle en este informe, se pretende examinar los Algoritmos Genéticos como herramienta para la solución de problemas complejos, para de este modo acercarse a su aplicación en Job-Shop Scheduling Problem.

Se presentarán los Algoritmos Genéticos en secciones: Primero se verán sus conceptos básicos, estructura y funcionamiento, esto es, una revisión de en que consiste el Algoritmo Genético. Luego se verá una serie de conceptos asociados a la teoría elemental de los Algoritmos Genéticos, que han demostrado ser importantes gracias a la investigación y aplicaciones impulsadas en distintas categorías de problemas y que en las investigaciones recientes tienen un rol importante en la construcción de variantes de Algoritmos Genéticos que sean más eficientes y flexibles.

3.1.- CONCEPTOS BÁSICOS

3.1.1.- Inicios

Los Algoritmos Genéticos fueron creados y desarrollados por John Holland y sus estudiantes de la Universidad de Michigan en los años sesenta y setenta. Más que un tipo de algoritmo que permitiera resolver problemas específicos, la intención de Holland era estudiar el fenómeno de la adaptación natural y desarrollar vías por las cuales se pudieran llevar estos mecanismos a sistemas computacionales. En su trabajo [[6]] presentó los Algoritmos Genéticos como una abstracción de la evolución biológica y entregó un marco de trabajo teórico para la adaptación natural bajo los Algoritmos Genéticos.

El algoritmo propuesto por Holland se puede describir como un método para moverse desde una población de *cromosomas* a una nueva población, usando cierto tipo de selección natural por medio del uso de operadores inspirados en la genética, como son el *cruzamiento* (crossover), *mutación* (mutation) e *inversión* (inversion). Cada *cromosoma* esta compuesto de *genes* (por ejemplo, un bit en el caso de

representación binaria), los que a su vez pueden ser una instancia particular de un *allele* (por ejemplo, un 0 o 1). Un operador de selección escoge en la población los cromosomas considerados más aptos para reproducirse, logrando que en promedio los cromosomas con un valor más alto de *fitness* generen nuevos individuos para la población, en contraste con los menos aptos. El cruzamiento intercambia partes de un cromosoma, imitando de cierto modo la recombinación biológica entre dos organismos cromosómicos simples. La mutación cambia de forma aleatoria el *allele* de ciertas partes de los cromosomas. La inversión revierte el orden de una sección continua de un cromosoma, cambiando el orden de los genes respectivos.

3.1.2.- Algoritmo Genético Canónico

En esta sección se hará una descripción general del funcionamiento del denominado Algoritmo Genético Canónico [20]. La primera etapa del algoritmo consiste en generar una población inicial (generalmente de forma aleatoria) de individuos, que corresponden a los *cromosomas* o *genotipos*. Cada uno de ellos es evaluado y se le asigna una medida de su *calidad* (*fitness*) determinada. En este contexto, la función de evaluación corresponde a una medida de comportamiento que se obtiene en base a una serie de parámetros, mientras que la función de calidad toma dicho valor para transformarlo en una medida que representa posibilidades de reproducción. La función de evaluación es considerada independientemente para cada individuo, no así la de calidad [21].

Un Algoritmo Genético puede verse como un proceso con dos etapas. La primera inicia con la población actual y tras un proceso de selección se genera una población intermedia, con lo que se inicia la segunda etapa en la cual se realizan sobre la población intermedia distintas operaciones de *cruzamiento* (o *recombinación*) y *mutación*, obteniendo de ese modo la siguiente población. Una generación es el paso desde una población a otra tras el proceso descrito. Esto puede apreciarse gráficamente en la Figura 3-1.

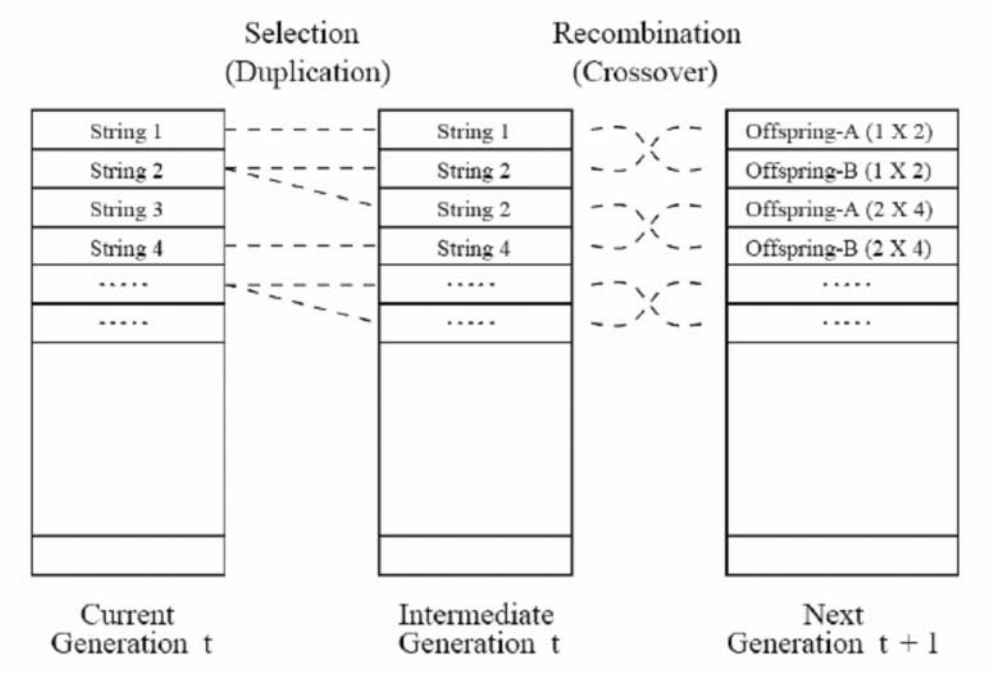

Figura 3-1. – Paso desde una generación a otra. En la población actual se realiza la selección obteniendo una población intermedia, en la que se realiza recombinación para obtener una población nueva.

En el Algoritmo Genético canónico, la calidad de un individuo se calcula con la expresión f_i / F, donde f_i corresponde al resultado de la evaluación y F corresponde al promedio de los resultados de evaluación en la población actual. Con esta calidad calculada, se realiza la selección considerando que la probabilidad de que un individuo pase a la generación intermedia es proporcional a su calidad.

Con la población intermedia mezclada adecuadamente, se procede a realizar la recombinación. La operación de cruzamiento se realiza sobre un par de cromosomas elegidos de forma aleatoria de la población intermedia, con una probabilidad P_c. Supóngase que se tienen los cromosomas:

1101 | 010111

0100 | 011010

Utilizando cruzamiento en un punto (*One-Point Crossover*), para este ejemplo, en la posición 5, se obtienen dos nuevos cromosomas, que constituirán la nueva generación:

Hijo 1: **1101 | 011010**

Hijo 2: **0100 | 010111**

Ya realizada la recombinación, se puede realizar la mutación. Para cada bit en cada elemento de la nueva población, hay una probabilidad P_m de que ocurra mutación, lo que generalmente se traduce en cambiar el bit de 0 a 1 o viceversa.

Con el proceso completo se obtiene la nueva generación, lista para ser evaluada y volver a empezar las etapas de selección, recombinación y mutación.

3.1.3.- Codificación

Un elemento fundamental en la construcción de Algoritmos Genéticos es la codificación que se utilice. El cromosoma, más que un arreglo de bits, funciona como la representación de una solución para el problema que esta abordando el Algoritmo Genético. Así como se formula un mecanismo para llevar la información de parámetros, variables o cualquier elemento que tiene alguna relación con el problema en cuestión, debe existir un mecanismo adecuado para extraer esa información.

En la práctica, la codificación va de la mano con la función de evaluación. Cuando se consideran los parámetros del problema, su codificación dentro del cromosoma debe ser adecuada para asegurar que al ser decodificados, la evaluación pueda realizarse eficientemente. Por ejemplo, si se quiere representar una variable con un rango posible de 1200 valores, se necesitan al menos 11 bits para cubrir este rango, sin embargo hay otros 848 valores que no se usaran para la evaluación, o se usaran incorrectamente.

No sólo una codificación binaria es posible, otros tipos de representación, ya sea con caracteres, enteros, etc., se han estudiado y evaluado. En *"A genetic algorithm tutorial"* de Whitley [20] y *"An Introduction to Genetic Algorithms"* de Mitchell [22] hay breves reseñas respecto a este punto, donde se destaca que existen opiniones tanto a favor como en contra de usar representaciones distintas a la binaria tradicional. Sin embargo, no existen reglas claras para definir cuando una representación distinta puede resultar efectiva o no, o resulta muy dependiente del problema.

3.1.4.- Selección

La selección es el proceso mediante el cual un subconjunto de individuos en una generación es escogido en razón de su calidad para formar parte de otra generación y para engendrar nuevos individuos. Se pueden describir tres mecanismos conocidos:

- *Selección proporcional a la calidad (fitness proportionate selection)*: También conocida como selección de la ruleta (*Roulette-Wheel Selection*), se basa en asignar una probabilidad de selección a cada individuo de acuerdo a su calidad. Esto permite que aunque los individuos con alta calidad tengan altas posibilidades, no necesariamente serán seleccionados. Lo mismo en el caso de los individuos con baja calidad, igualmente tienen posibilidades de ser seleccionados, esto es favorable para la variedad y evitar la convergencia prematura.
- *Selección por torneo (tournament selection)*: se trata de seleccionar un conjunto de individuos en la población y escoger los mejores entre ellos para ser parte de futuras generaciones y procesos de recombinación. En este tipo de selección, el tamaño del conjunto seleccionado es un factor relevante. En el torneo, se va evaluando con una probabilidad p cada individuo en orden de calidad, si es seleccionado o no para recombinación. Una selección por torneo se define determinista si p=1.
- *Muestreo estadístico de resto (remainder stochastic sampling)*: Considerando la expresión de calidad f_i / F , cuando este valor es mayor que 1, la porción entera de este número indica cuantas copias del cromosoma respectivo son copiadas a la población intermedia. Todos los cromosomas pondrán copias en la población intermedia con probabilidad equivalente a la parte fraccionaria de f_i / F . Por ejemplo, si este valor es 1.36, se coloca una copia y hay una probabilidad de 0.36 de que coloque otra.

3.1.5.- Operadores Genéticos

3.1.5.1.- Recombinación o Cruzamiento

En la recombinación, dos individuos generan un tercero mezclando copias de sus genes en el nuevo individuo. El mecanismo más tradicional de cruzamiento es el de un punto. Este mecanismo tiene una desventaja, en el sentido de que dependiendo de cómo se encuentren ordenados los bits en el cromosoma, puede romper la composición de un *schema* presente en él. Como se puede apreciar más adelante, la mezcla en recombinación de esquemas de bajo orden pueden ayudar a encontrar soluciones de forma más efectiva, por lo que romper su estructura en el cromosoma no es algo que resulte conveniente. Bajo este contexto, es que se han desarrollado otros tipos de recombinaciones. A continuación una breve descripción de algunos tipos de recombinación:

- *Cruzamiento en dos puntos (double-point crossover):* De forma análoga al cruzamiento en un punto, en este mecanismo se escogen dos puntos en los cromosomas y los segmentos entre los puntos seleccionados se combinan en los nuevos individuos. Por ejemplo, si se tienen los siguientes cromosomas:

  1101 | 0111 | 00001

  0101 | 0101 | 00011

  Al realizar el cruzamiento en dos puntos, para este caso, en las posiciones 5 y 9, se obtienen los siguientes cromosomas:

  Hijo 1: **1101 | 0101 | 00001**

  Hijo 2: **0101 | 0111 | 00011**

- *Cruzamiento uniforme (Uniform crossover)*: En este mecanismo los genes en cada posición de los cromosomas padres se comparan. Con una probabilidad constante, el *allele* de ambos genes se intercambia. Una variante a este método de cruzamiento es el medio cruzamiento uniforme (*Half Uniform Crossover*), donde exactamente la mitad de los genes distintos se intercambian.

- *Cruzamiento Parcialmente Coincidente (Partially Matched Crossover)*: En este mecanismo, se seleccionan dos puntos de cruzamiento de forma aleatoria. Los *alleles* de los cromosomas padres, entre estos dos puntos, son intercambiados con los *alleles* correspondientes a aquellos mapeados por el otro padre. Por ejemplo:

  9 8 4 | 5 6 7 | 1 3 2 10

  8 7 1 | 2 3 10 | 9 5 4 6

  Al mirar en el primer padre, el primer gen entre los dos puntos, 5 corresponde al 2 en el otro padre. Por lo que son intercambiados en el primer padre, similarmente se intercambian 6 y 3, 10 y 7; de forma de crear el primer hijo y se aplica al segundo padre. Esto da como resultado:

  Hijo 1: **9 8 4 | 2 3 10 | 1 6 5 7**

  Hijo 2: **8 10 1 | 5 6 7 | 9 2 4 3**

3.1.5.2.- Mutación

Este operador corresponde a la modificación de genes particulares de un cromosoma con una probabilidad muy baja de ocurrencia. La función principal de la mutación es mantener la diversidad de la población y evitar que exista convergencia prematura en la ejecución del algoritmo, además de permitir explorar otros sectores en el espacio de búsqueda.

Pese al mecanismo sencillo con que trabaja la mutación, resulta ser un elemento muy importante en el funcionamiento del Algoritmo Genético. Diversos estudios se han enfocado en comprobar la influencia de este operador en la resolución de problemas complejos, en la ya citada obra de Whitley [20] se habla de casos comparativos respecto a las capacidades de la recombinación. Sin embargo, se destaca que el balance entre mutación, selección y cruzamiento es importante para evitar los problemas frutos de la falta de balance.

3.1.5.3.- Inversión

Este operador revierte el orden de una sección continua en el individuo modificando el orden de los genes en la sección respectiva. A diferencia de la mutación que se puede aplicar de manera individual a

los genes del individuo, la inversión se aplica al individuo como un todo. Por ejemplo, si se tiene el siguiente individuo:

9 8 4 | 5 6 7 | 1 3 2 10

Al aplicar el operador sobre el segmento delimitado, se obtiene:

9 8 4 | 7 6 5 | 1 3 2 10

3.1.6.- Sustitución

Una vez que las nuevas soluciones son creadas usando recombinación y mutación, es necesario incluirlas en la población. Existen diversas maneras de lograrlo, considerando que los cromosomas padres ya han sido seleccionados de a cuerdo a su aptitud, por lo que se espera que la descendencia se encuentre entre los más aptos en la población y que la población en promedio mejore su aptitud. Algunas técnicas de sustitución son descritas a continuación:

- *Supresión de todos*: Esta técnica suprime a todos los miembros de la población actual y los reemplaza con el mismo número de cromosomas que han sido creados recientemente. Esta es probablemente una técnica común y quizás sea la elección de muchos debido a su simpleza de implementación. A diferencia de otros métodos no requiere parámetros.
- *Estado Estable:* Esta técnica suprime *n* elementos antiguos de la población y los reemplaza con *n* nuevos elementos. El número a suprimir y reemplazar, *n*, en cualquier instante de aplicación es un parámetro para esta técnica. Otra consideración es la de decidir que elementos serán suprimidos, pudiendo ser los menos aptos, los padres de la nueva generación o seleccionados al azar, lo que se constituye en otro parámetro para esta técnica.
- *Estado Estable Sin Duplicados:* Es igual que la técnica anterior pero el algoritmo se asegura que no sean añadidos a la población cromosomas duplicados. Esto agrega un overhead pero significa que más del espacio de búsqueda es explorado.

3.1.7.- Criterio de Término

El criterio de término para los Algoritmos Genéticos es el encargado de definir el momento en el cual debe interrumpir el ciclo de evolución y adoptar el individuo más apto como la solución encontrada por el Algoritmo Genético.

A continuación se describen algunos criterios de término comúnmente utilizados:

- *Criterio de Convergencia de Identidad:* Este criterio consiste en detener el algoritmo genético cuando un determinado porcentaje de los individuos representa a la misma solución. Los operadores tienden a preservar y difundir el material genético de los cromosomas más aptos, por lo que es de esperar que luego de un gran número de generaciones, alguna solución con gran valor de aptitud se imponga y domine la población.
- *Criterio de Convergencia de Aptitud:* Puede suceder que existan soluciones equivalentes o casi equivalentes a un problema, que obtengan valores de fitness parecidos. En estos casos, es probable que no haya una solución que se imponga en la población (y el criterio de término por convergencia de identidad nunca se cumpla). Este criterio no espera a que la población se componga mayoritariamente de una sola solución, sino que finaliza la ejecución del algoritmo cuando los valores de fitness de un determinado porcentaje de las soluciones son iguales, o difieren en un porcentaje dado. Por ejemplo, cuando el 90% de las soluciones tenga valores de aptitud que no difieran en más de un 1%.
- *Criterio de Cantidad de Generaciones:* El criterio de término por cantidad de generaciones consiste simplemente en finalizar la ejecución una vez que ha transcurrido un número determinado de generaciones. Los métodos anteriores apuntan a esperar a que la evolución de la población llegue a su fin. Cuando alguno de ellos se cumple, es probable que las soluciones no sigan mejorando mucho más, no importa cuantas generaciones más se ejecuten. Sin embargo, los Algoritmos Genéticos pueden necesitar un número de generaciones elevado para llegar a la convergencia, dependiendo de las tasas de reproducción y mutación. Utilizando cualquiera de los dos criterios anteriores no puede estimarse un número máximo de generaciones, ya que esto dependerá no solamente de los parámetros del algoritmo genético sino también del azar. Esto

puede ser un problema, en el caso que se quisiera comparar los tiempos de resolución de un problema mediante Algoritmos Genéticos con otros métodos. Por lo que este método permite determinar con precisión los tiempos de ejecución del algoritmo a costa de detener la evolución sin la certeza de que las soluciones no seguirán mejorando.

3.2.- CONSIDERACIONES TEÓRICAS

En la literatura relacionada con Algoritmos Genéticos se aprecian variaciones sobre el modelo canónico, con el objetivo de mejorar la efectividad y las prestaciones que los Algoritmos Genéticos brindan en la solución de problemas complejos. Sin embargo, todas estas variaciones se fundamentan en el teorema de esquema (*Schema Theorem*) y la notación de bloques de construcción (*Building Blocks*), que son considerados pilares fundamentales para la comprensión del comportamiento de los Algoritmos Genéticos.

3.2.1.- Hiperplanos y Espacio De Búsqueda

El comportamiento de los Algoritmos Genéticos puede explicarse como una robusta búsqueda de carácter complejo, que trabaja en el espacio de búsqueda vía muestreo implícito de particiones de hiperplanos. Para visualizar y entender que es una partición de hiperplanos, supóngase un espacio de búsqueda de tres dimensiones. Asumiendo que la codificación es de tres bits, se puede representar el espacio de búsqueda como un cubo con origen en el string 000. En cada esquina se tienen strings que varían respecto a sus esquinas adyacentes en un bit. Esto se puede apreciar en la Figura 3-2.

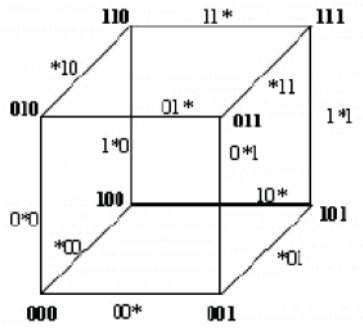

Figura 3-2. – Representación gráfica de los esquemas de longitud tres.

En el plano frontal del cubo se encuentran todos los strings que comienzan con 0; si se considera que "*" es un carácter reemplazable por cualquier bit, el plano queda definido por "0**". Un string que contiene caracteres * se denomina *schema* (esquema) y cada *schema* corresponde a un hiperplano en el espacio de búsqueda donde el orden de este corresponde al número de bits que se encuentran fijos en el *schema*. En el caso de "0**" este tiene orden 1 mientras que un *schema* "1**01***" tiene orden 3.

La Figura 3-3 muestra un espacio de 4 dimensiones, representado por un cubo dentro de otro. Se etiquetan las esquinas de ambos cubos del mismo modo que para el cubo de tres dimensiones; luego se inserta un 1 al inicio de los strings del cubo interior y un 0 al inicio de los del exterior, manteniendo el formato de variación de un solo bit entre esquinas adyacentes. Así el cubo interior corresponde al hiperplano "1***" y el exterior a "0***".

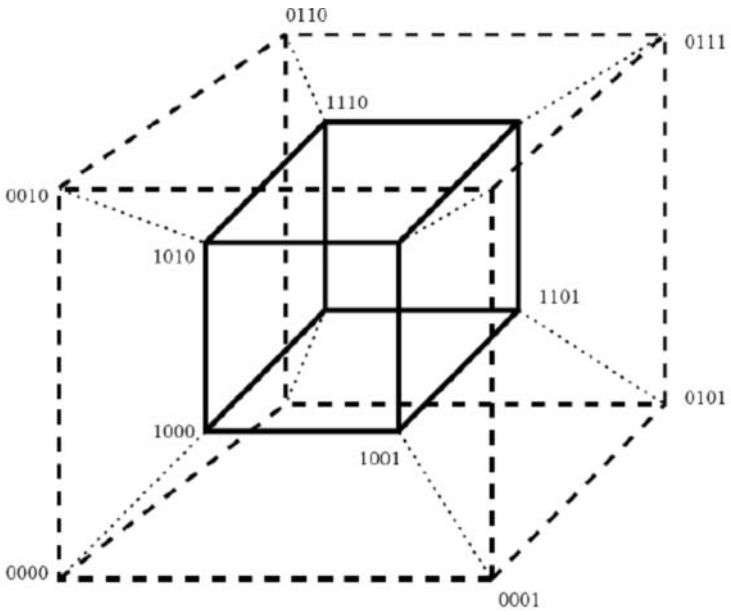

Figura 3-3 – Representación de un espacio de búsqueda de 4 dimensiones.

Se puede apreciar que el *schema* "*0**" representa los planos frontales de cada cubo y el hiperplano de orden 2 "10**" representa la cara frontal del cubo interior.

Un string es parte de un hiperplano, si el mismo puede obtenerse al reemplazar en el *schema* los valores marcados con "*" por los bits que correspondan, por ejemplo el string 1001 pertenece al hiperplano "10**" ya que es posible obtenerlo a partir del *schema*.

Un cromosoma con representación binaria corresponde a una esquina del hipercubo y es parte de $2^L - 1$ hiperplanos distintos, con largo de cromosoma L, sin considerar el *schema* compuesto por sólo *, que representa el espacio de búsqueda. De forma adicional, para un cromosoma con representación binaria, se tienen $3^L - 1$ posibles particiones de hiperplanos, con largo de cromosoma L. Cada elemento del string, puede tomar los valores 0,1,*, de modo que resultan 3^L combinaciones posibles.

El concepto de población en el mecanismo de búsqueda se vuelve importante al considerar poco práctico el examen de un individuo de forma aislada, siendo parte de $2^L - 1$ particiones de hiperplanos.

Una población de distintas muestras de individuos presenta información de diversos hiperplanos, con hiperplanos de bajo orden siendo representados por un gran número de individuos. En este contexto surge el concepto de *paralelismo implícito* (implicit parallelism) considerando que numerosos hiperplanos son muestreados al evaluar una población de individuos, incluso los hiperplanos muestreados son más que los individuos muestreados.

La evaluación de un individuo implica la evaluación implícita y paralela de un conjunto de hiperplanos, pero es el efecto acumulativo de la evaluación de una población el que permite obtener información estadística acerca de algún conjunto de hiperplanos.

El concepto de *paralelismo implícito* sugiere que las competencias entre hiperplanos son resueltas en paralelo. A través de los procesos de reproducción y recombinación, la representación de esquemas de hiperplanos en competencia aumenta o disminuye de acuerdo a la calidad de los individuos pertenecientes a esos hiperplanos. Así en una población se puede ir evaluando la representación proporcional de un *schema* e indicar si esta aumenta o disminuye en el tiempo; cuando se combinan la selección basada en calidad con la recombinación para generar nuevos individuos en la población.

3.2.2.- Schema Theorem

Considérese que *M(H,t)* corresponde al número de strings en una población que son parte del hiperplano *H* en la generación *t*; además *(t+ intermediate)* corresponde a la generación que existe después de la selección, pero antes de las operaciones de crossover y mutación, y *f(H,t)* es la evaluación promedio

de la muestra de strings en *H* para la población actual, mientras que $\bar{f}$ corresponde a la evaluación promedio de toda la población.

Formalmente, el cambio en la representación asociada a los strings obtenidos del hiperplano *H* se expresa como:

$$M(H, t + intermediate) = M(H, t)\frac{f(H, t)}{\bar{f}} \qquad (3-1)$$

A partir de esta formulación, se puede observar que la cantidad de strings en *H*, en el tiempo depende de la razón formada entre el valor promedio de la evaluación de los strings, respecto a la evaluación de todos los strings en la población. Cuando se realiza una copia de strings no se muestrean nuevos hiperplanos pues no existen nuevas muestras.

Si bien teóricamente se pretende obtener muestras nuevas con nuevos individuos, en la práctica no suele ser posible, y es por ello que la recombinación y mutación son mecanismos para obtener nuevas muestras, preservando parcialmente la distribución de strings entre los hiperplanos de la generación intermedia.

Ahora al considerar los efectos de la recombinación en el calculo de *M(H,t)*. Teóricamente este operador posee efectos disruptivos (*losses*) y contractivos (*gains*) respecto a la representación de los individuos asociados a *H*. Un cruzamiento puede eventualmente destruir la estructura de un *schema* asociado a un hiperplano, y se considera que la recombinación es aplicada a parte de la población, y para la otra parte la representación se mantiene. De este modo la formula antes enunciada se convierte en:

$$M(H, t+1) = (1 - p_c)M(H, t)\frac{f(H, t)}{\bar{f}} + p_c\left[M(H, t)\frac{f(H, t)}{\bar{f}}(1 - losses) + gains\right] \qquad (3-2)$$

Donde P_c corresponde a la probabilidad que un individuo sea sometido a recombinación. En este punto se realizan dos supuestos considerados conservadores:

- Un cruzamiento dentro del *largo de definición* (defining length) de un *schema* es siempre disruptivo. El largo de definición (ΔH) corresponde a la diferencia entre la posición de la ultima

instancia de un bit respecto a la primera en el *schema*. Por ejemplo en el *schema* ***11*01*0** ΔH =6. En la práctica, el supuesto no resulta efectivo siempre. Por ejemplo si se considérale *schema* 11*****, y se recombina el string 1110101 entre el primer y el segundo bit con un string como 0100000, no se genera una disrupción en el *schema* pues uno de los individuos resultantes se encuentran dentro de ese hiperplano.

- Los efectos constructivos se asumen nulos. El supuesto al igual que el anterior es impreciso. Si se toma el mismo *schema* anterior y se recombinan los strings 1000000 y 0100000, entre el primer y el segundo bit se obtiene un nuevo individuo perteneciente al *schema*. De este modo se tiene la siguiente desigualdad:

$$M(H,t+1) \geq (1-p_c)M(H,t)\frac{f(H,t)}{\overline{f}} + p_c\left[M(H,t)\frac{f(H,t)}{\overline{f}}(1-disruptions)\right] \quad (3-3)$$

En esta desigualdad el efecto de la disrupción se sobreestima. Se considera una excepción en este contexto: dos strings pertenecientes al mismo hiperplano *H*, en recombinación no generan disrupción. Sea *P(H,t)* la representación proporcional de *H* obtenida al dividir *M(H,t)* por el tamaño de población, la probabilidad que un individuo elegido aleatoriamente pertenezca a *H* es *P(H,t)*. Para el cruzamiento en un punto, $\Delta H / L-1$ es una medida directa de cómo el punto puede encontrarse dentro del largo de definición de un *schema*, donde L es el largo del cromosoma.

Considerando esto, la disrupción se puede obtener por:

$$\frac{\Delta(H)}{L-1}(1-P(H,t)) \quad (3-4)$$

En este punto se puede simplificar la desigualdad, dividiendo ambos lados por el tamaño de la población para convertirla en una expresión para *P(H,t+1),* la representación proporcional de *H* en la generación *t+1*. Incluso más la expresión puede ser reordenada con respecto a p_c.

$$P(H,t+1) \geq P(H,t)\frac{f(H,t)}{\overline{f}}\left[1-p_c\frac{\Delta(H)}{L-1}(1-P(H,t))\right] \qquad (3-5)$$

Con lo que se tiene una versión del teorema de schema, que aún no considera la mutación; pero no es la única que existe en la literatura. Por ejemplo esta versión es consistente con el hecho que la selección del primer padre esta basada en aptitud y el segundo es elegido aleatoriamente. Pero también puede contemplar el que ambos padres sean seleccionados en basa a su aptitud. Ello se logra indicando que el otro padre es elegido de la población intermedia después de la selección.

$$P(H,t+1) \geq P(H,t)\frac{f(H,t)}{\overline{f}}\left[1-p_c\frac{\Delta(H)}{L-1}(1-P(H,t)\frac{f(H,t)}{\overline{f}})\right] \qquad (3-6)$$

Finalmente la mutación es incluida. Sea *o(H)* una función que retorna el orden del hiperplano *H* y la probabilidad de mutación p_m donde la mutación siempre cambia el bit. De este modo la probabilidad que la mutación afecte al *schema* que representa a *H* es $(1-p_m)^{o(H)}$, lo que lleva a la siguiente expresión del teorema [20].

$$P(H,t+1) \geq P(H,t)\frac{f(H,t)}{\overline{f}}\left[1-p_c\frac{\Delta(H)}{L-1}(1-P(H,t)\frac{f(H,t)}{\overline{f}})\right](1-p_m)^{o(H)} \qquad (3-7)$$

3.2.3.- Bloques De Construcción (Building Blocks)

Un bloque de construcción es la sub-estructura del cromosoma que le permite ser asociado a un *schema*, o formar parte del hiperplano correspondiente. Un algoritmo genético opera en torno a los bloques, permitiendo que su número aumente y mezclando los unos con otros, para así obtener soluciones del problema abordado.

La estructura de un bloque de construcción esta asociada directamente con un *schema*, por ello es posible que en el proceso de recombinación sea destruido el bloque. Esta probabilidad esta influenciada por el tipo de operador que se utilice en la recombinación, como se puede ver en este ejemplo extraído de Whitley [20]: Sean dos *schemata* de orden 2 con estructuras del siguiente tipo: 11********** y

1**********1. En caso de cruzamiento en un punto, para el primer *schema* existe una probabilidad de 1/*L*-1 de romper su estructura, con largo de string *L*. Para el segundo string, sin embargo, existe una probabilidad de *L*-1/*L*-1 =1.0. Como se observa la posición de los bits fijos en el *schema* afecta el estado del cruzamiento en un punto sobre su estructura. El crossover en dos puntos funciona mejor en este aspecto; el *schema* se puede representar como un anillo:

Las posiciones de los bits en el string van de b1 hasta b12; el crossover en un punto se puede ver de este modo como un caso particular del crossover de dos puntos, cuando uno de los puntos escogidos esta entre b1 y b12. Las mayores probabilidades de romper el esquema ocurren cuando los bits fijos en el *schema* se encuentran en posiciones contrarias del anillo. En el caso planteado se puede ver que la representación de anillo es la misma, al contrario del crossover en un punto. Se puede apreciar que la posición de los bits en los *schemata* tiene alta influencia en sus probabilidades de ser destruidos en la recombinación. Se dice que poseen una *representación compacta* (*compact representation*), en el *schema* si están cerca unos de otros, lo que disminuye la disrupción en la recombinación.

En este contexto surge el concepto de *vinculación* o *acoplamiento* (*linkage*), que corresponde al fenómeno en que un conjunto de bits actúan como *alleles* coadaptados y tienden a ser heredados en conjunto. El *linkage* es una generalización de la representación compacta de los bits, en el sentido que esta última depende de la cercanía que ellos posean. Pero no debe pensarse que sólo se produce entre bits cercanos; un *building block* puede estar distribuido en el cromosoma y estar sometido a *linkage*, dependiendo de la representación. Por ejemplo, en varias aplicaciones de Algoritmos Genéticos se puede encontrar la idea de representación independiente de la posición. En esta se tiene que los genes se representan por un vector (posición, *allele*). Por ejemplo se tiene el siguiente cromosoma bajo esta representación:

$$((9\ 0)\ (6\ 0)\ (2\ 1)\ (7\ 1)\ (5\ 1)\ (8\ 1)\ (3\ 0)\ (1\ 0)\ (4\ 0))$$

El cromosoma representa al string 010010110. En él se pueden aplicar operadores para cambiar el grado de acoplamiento observable en los distintos bloques que lo conforman. Un operador clásico en este contexto es la *inversión* que consiste en tomar un par de genes en intercambiarlos. En este caso el nivel de acoplamiento puede cambiar, pero el cromosoma mantiene la misma codificación. En este tipo de operaciones la dificultad natural que surge es que en la recombinación pueden resultar cromosomas con posiciones repetidas u omitidas.

De manera adicional, es importante considerar como se ve afectada la estructura de un *building block* respecto al orden del *schema* al que se encuentre asociado, cuando es manipulado en operaciones de recombinación. Por lo visto anteriormente, es natural pensar que los bloques de mayor orden, tienen mayores probabilidades de que su estructura se vea alterada. Analizando esto de un punto de vista del *schema theorem*, el grado de disrupción aumenta directamente pues aumentan las perdidas que se generan en el proceso de recombinación, lo que en consecuencia disminuye la representación en la población de hiperplanos cuyos *schemata* presentan niveles de aptitud altos en su evaluación. De esto se puede obtener la conclusión que se repite en la literatura: El diseño debe facilitar la mezcla de bloques de construcción de bajo orden y que presenten altos niveles de aptitud en su evaluación. Así mismo, el grado de acoplamiento en la estructura de estos bloques debe ser asegurado.

CAPITULO 4

TRABAJOS RELACIONADOS

En esta sección se pasará revista a una seria de trabajos realizados por investigadores, en los cuales se han utilizado los Algoritmos Genéticos como herramientas de solución, y en los que se han presentado representaciones y operadores para Job-Shop Scheduling Problem.

Al considerar los objetivos de este proyecto, en esta sección no se pasará revista a aproximaciones distintas a loa Algoritmos Genéticos para la resolución de Job-Shop Scheduling Problem, sin embargo en algunos de los trabajos señalados existen referencias a otras técnicas para abordar el problema. Los trabajos en esta sección serán presentados por autor (o autores) y de manera cronológica.

4.1.- TAKESHI YAMADA Y RYOHEI NAKANO

4.1.1.- A Genetic Algorithm Applicable To Large-Scale Job-Shop Problems (1992)

En este trabajo se propone un nuevo método para resolver Job-Shop Scheduling Problem basado en algoritmos genéticos, en que cada individuo representa una solución derivada directamente de los tiempos de finalización de las operaciones en lugar de ser codificados en strings binarios. Aplicando un operador de crossover, basado en el algoritmo de Giffler y Thompson para generación de *schedules* activos, en el que ambos progenitores son *schedules* activos de modo que resultan como descendencia nuevos *schedules* activos que heredan las características de los progenitores [23].

El algoritmo GT es delineado de la siguiente forma, se considera la planificación de cada operación en un orden temporal. Un conjunto *C* de todas las operaciones más tempranas en la secuencia entre operaciones que no han sido planificadas aún es definido y se le denomina *corte* (*cut*). El tiempo de completado más temprano $EC_{j,i}$, es calculado para cada operación $O_{j,i} \in C$. Luego se tiene los siguientes pasos:

(A1) Encontrar O_{j^*,i^*,r^*} que tenga un mínimo *EC* en *C*: $EC_{j^*,i^*} = \min\{EC_{j,i} \mid O_{j,i} \in C\}$

Especificar *G*: un conjunto de operaciones que consiste de $O_{j,i.r^*} \in C$ que comparten la misma máquina M_{r^*} con O_{j^*,i^*,r^*} y el procesamiento de $O_{j,i.r^*} \in C$ y O_{j^*,i^*,r^*} se sobreponen. *G* es llamado un *conflict set*.

(A2) Seleccionar una de las operaciones O_{j_s,i_s} desde *G*, y planificar O_{j_s,i_s} de acuerdo a EC_{j_s,i_s}

(A3) Actualizar *C* y *EC s*.

Se repiten los pasos hasta que todas las operaciones están planificadas, y entonces se obtiene un plan activo. Además, si existe "empate" en (A1) este se rompe aleatoriamente. Y en el paso (A2), si todas las opciones posibles son consideradas, planes activos son generados para todas.

En el método propuesto cada individuo *psn* representa un plan activo que se obtiene de forma directa usando elementos $\{psn_{j,i,r}\}$ de tiempos de completado. El makespan S_{psn} del plan de ejecución representado por *psn* es calculado como sigue:

$$S_{psn} = \max\{psn_{j,i} \mid 1 \le j \le n, i = m\} \qquad (4-1)$$

En el crossover propuesto se tiene a los progenitores, uno de ellos llamado "papá" (*dad*) y el otro "mamá" (*mom*) y el nuevo individuo llamado "niño" (*kid*).

El algoritmo es apreciable a continuación:

(C1) Realizar el paso (A1) del algoritmo GT, obteniendo *C, EC s* y *G.*

(C2) Elegir una de las operaciones desde *G* a ser planificada a continuación, tal como sigue:

(a) Generar un número aleatorio $\varepsilon \in [0,1)$ y compararlo con $R_\mu \in [0,1)$ que es una constante predefinida denominada *tasa de mutación.* Si ($\varepsilon < R_\mu$) entonces elegir cualquier operación O_{j_s,i_s} desde *G* (ocurre mutación)

(b) De otra manera seleccionar cualquiera de los dos, *mamá* o *papá* con probabilidad igual a ½.

Se asume que *Mamá* es seleccionada.

Encontrar una operación O_{j_s,i_s} que haya sido planificada más temprano en *mamá* entre todas las operaciones en *G*.

$$mamá_{j_s,i_s} = \min\{mamá_{j,i} \mid O_{j,i} \in G\}$$

(c) Planificar O_{j_s,i_s} de acuerdo a EC_{j_s,i_s}, entonces asignar $kid_{j_s,i_s} = EC_{j_s,i_s}$

(C3) Actualizar *C* y *EC s*.

Al repetir los pasos hasta que las operaciones son planificadas, el nuevo individuo es obtenido, en la ejecución del algoritmo, se aplica 2 veces a un par de "papá" y "mamá" para obtener dos nuevos individuos "niño1" y "niño2". En orden de mantener al individuo más apto de la población actual, los nuevos "papá" y "mamá" son elegidos de la siguiente manera:

La nueva "mamá", es el mejor entre los progenitores y los hijos. Mientras que el nuevo "papá" es el "niño" no seleccionado como "mamá", o el mejor de los "niños" si ninguno fue seleccionado previamente.

Como se puede apreciar con este enfoque se evita la necesidad de un mecanismo de reparación ya que los *schedules* generados son todos validos.

4.1.2. - A Genetic Algorithm With Multi-Step Crossover For Job-Shop Scheduling Problems (1995)

Se propone en este trabajo un nuevo operador de crossover llamado *Multi-Step Crossover (MSX)* que utiliza una estructura de vecindad y distancia en el espacio del problema. Dados ambos padres, MSX genera de manera exitosa los descendientes a lo largo de la ruta que conecta ambos padres [24].

El MSX es definido usando la distancia y la estructura de vecindad; la distancia es utilizada para medir las similitudes entre las soluciones y para determinar la dirección de búsqueda en el crossover. Y la vecindad de un punto *x* puede ser interpretada como el conjunto de puntos cercanos a *x*.

La idea básica es tomar un punto *x* perteneciente a la vecindad de uno de los padres p_1, y evaluarlo en distancia respecto al otro padre p_2. Partiendo del primer padre, el punto *x* seleccionado es modificado paso a paso y de manera unidireccional acercándolo al segundo padre. En el proceso *x* va perdiendo características de p_1 y obtiene gradualmente aquellas de p_2. Después de algunas iteraciones, las nuevas soluciones resultantes deberían contener partes tanto de p_2 como de p_1, aunque en distintas proporciones.

También en este trabajo se propone un operador de mutación (*Multi-Step Mutation MSM*) definido por las mismas ideas de MSX, en que partiendo de un punto *p*, un individuo *x* es modificado repetidamente alejándose de *p*.

El algoritmo propuesto en este trabajo sigue los siguientes pasos:

1- Inicializar la población: Se genera un conjunto de planes generados aleatoriamente, y se busca la vecindad para cada miembro en el conjunto.

2- Se seleccionan 2 panes *S,T* desde la población de manera aleatoria con sesgo dependiente en sus valores de makespan.

3- Si su distancia es menor que un valor pequeño predefinido, se aplica MSM a *S* y se genera *U*, entonces se pasa al quinto paso.

4- Se aplica MSX a *S,T* utilizando la vecindad de *S* y la distancia y se genera un nuevo plan *U*.

5- Se aplica búsqueda en la vecindad de *U* y se genera *U'*.

6- Si el makespan de *U'* es menor que el del peor individuo de la población, se reemplaza este por *U*.

7- Se repiten desde el segundo al sexto paso hasta que no haya mejoras observables después de un número determinado de evaluaciones.

Este algoritmo propuesto fue testeado utilizando problemas de la clase *mt*, con tamaño de población 100 y 500 generadas aleatoriamente. Los resultados que en su momento permitieron la validación del algoritmo se muestran en la tabla a continuación.

Tabla 4-1. – Comparación de rendimiento utilizando problemas *mt*.

Problema	Método	Mejor	Promedio	Var	Pob	runs
MT 10x10	CBSA	930	930.8	2.4	-	10
(*Opt.*=930)	GA/GT+ECO	930	963	14	2025	200
	PGA+SBP	930	947	8.2	100	200
	GVOT	949	977	?	500	?
	GA/MSX	930	934.5	5.1	500	10
MT 20x5	CBSA	1178	1178	0	-	5
(*Opt.* = 1165)	GA/GT+ECO	1181	1213	16	5041	200
	PGA+SBP	1165	1188	10.3	100	200
	GVOT	1189	1215	?	500	?
	GA/MSX	1165	1177.3	4.2	100	10

4.2.- LEE HUI PENG Y SUTINAH SALIM

4.2.1. - A Modified Giffler And Thompson Genetic Algorithm On The Job Shop Scheduling Problem (2006)

En este trabajo se propone una modificación al algoritmo propuesto por Yamada y Nakano en [23] y revisado en [7], en orden de obtener soluciones “mejores que” o “tan buenas como” en términos de velocidad de convergencia en comparación con los algoritmos GT/GA existentes con menores tiempos computacionales; de modo de resultar una alternativa entre los algoritmos GT/GA.

En este algoritmo modificado el método de apareo es distinto al del GT/GA original, utilizándose selección por torneo en lugar de selección aleatoria.

Siendo el algoritmo modificado el que se aprecia a continuación:

<u>*Nota:*</u> La matriz de secuencia tecnológica $\{T_{ik}\}$ y la matriz de tiempos de procesado $\{p_{ik}\}$ son dados como entrada. Los siguientes parámetros de GA también son dados: tamaño de población *N*, tasa de crossover R_c y la tasa de mutación R_m.

1- Un población inicial aleatoria *P (t=0)* de tamaño *N* es construida, en la que cada individuo es generado utilizando el algoritmo GT con selecciona aleatoria de operaciones. El makespan de cada individuo es calculado automáticamente como uno de los resultados del algoritmo GT.

2- Seleccionar aleatoriamente *N x* R_c individuos de *P (t)* y emparejarlos utilizando selección por torneo. Aplicar el crossover GT (con mutación incorporada de probabilidad R_m) a cada par y generar nuevos *N x* R_c individuos que son insertados en *P' (t)*. El resto de los miembros de *P (t)* son sólo copiados a *P' (t)*. Como resultado del crossover GT, el makespan de cada individuo es calculado automáticamente

3- Si el mejor makespan en *P' (t)* no es tan bueno como en *P (t)* , entonces el peor individuo de *P' (t)* es reemplazado por el mejor individuo de *P (t)* (estrategia elitista)

4- Reproducir *P (t+1)* desde *P' (t)* por medio del uso de selección de ruleta, en que cada individuo en *P' (t)* es ordenado en orden descendiente de su makespan de modo tal que el peor individuo es numerado como x_1 y el mejor como x_N. Entonces la selección por ruleta es aplicada con el fitness f de un individuo x_i definido como $f(x_i) = i$ para obtener *P (t+1).*

5- Asignar *t = t +1*.

6- Repetir pasos 2 al 5 hasta que alguna de las condiciones de termino se alcance.

7- El mejor individuo en *P (t)* se considera la mejor solución obtenida.

Para comparar con respecto al algoritmo original se utilizaron los problemas *ft06* y *ft10* con los siguientes parámetros [1] :

- Tamaño Población: 100
- Tasa de Crossover: 0.9
- Tasa de mutación: 0.001

Los resultados de la comparación con diez intentos para *ft06* se muestran en la Tabla 4-2.

Tabla 4-2. – Comparación para el GT/GA existente y modificado (*ft06*).

	Makespan	
Prueba	GT/GA Existente	GT/GA Modificado
1	79 (103.39)	62 (104.062)
2	65 (90.578)	71 (72484)
3	71 (87.735)	66 (72.484)
4	60 (104.093)	76 (84.438)
5	83 (95.359)	77 (111.000)
6	78 (102.328)	71 (94.656)
7	77 (88.594)	69 (88.078)
8	76 (81.468)	74 (93.187)
9	72 (100.797)	70 (97.860)
10	60 (110.344)	73 (111.250)
Promedio	72.1 (96.469)	70.9 (93.487)

4.3. - TAKESHI YAMADA

4.3.1. - Studies On Metaheuristics For Jobshop And Flowshop Scheduling Problems (2003)

En este trabajo, Yamada revisita el algoritmo GA/GT y realiza un análisis de operadores de crossover basados en representaciones basadas en codificaciones no binarias, como *Subsequence Exchange Crossover* (*SXX*) y *Precedence Preservative Crossover* (*PPX*).

Para la utilización del SXX se utiliza la siguiente representación de permutación particionada donde el cromosoma es una permutación de trabajos en cada máquina donde la permutación se lee de izquierda a derecha, como se puede ver a continuación:

M1	M2	M3
1 2 3	3 1 2	2 1 3

Ahora, sean dos permutaciones particionadas p_0 y p_1, que corresponden a dos soluciones factibles. Un par de sub-secuencias, una de p_0 y otra de p_1 sobre la misma máquina, son llamadas *intercambiables* si y sólo si ellas consisten en el mismo conjunto de números de trabajos.

El SXX primero identifica los pares de sub-secuencias intercambiables en p_0 y p_1 en cada máquina e intercambia cada par para producir nuevas permutaciones particionadas C_0 y C_1, como se puede apreciar a continuación:

	M1	M2	M3
P0	1 2 3 6 4 5	3 2 1 5 6 4	2 3 5 6 1 4
P1	6 2 1 3 4 5	3 2 6 4 5 1	6 3 5 4 2 1
C0	2 1 3 4 6 5	3 2 5 1 6 4	2 6 3 5 1 4
C1	6 1 2 3 4 5	3 2 6 4 1 5	3 5 6 4 2 1

El SXX asegura que C_0 y C_1 son siempre permutaciones particionadas validas por lo tanto no existen inconsistencias dentro de cada máquina que deban ser corregidas. Sin embargo pueden surgir inconsistencias entre máquinas que requieran corrección a nivel global.

Ahora, para utilizar PPX se utiliza una permutación no particionada de *n* números de trabajo con *m* repeticiones. En esta representación se considera la permutación de *n* números de trabajo pero cada número de trabajo idéntico se repite *m* veces. Cuando se da una permutación así, es decodificada leyendo la permutación de izquierda a derecha y refiriéndose a la *k*-esima ocurrencia de un número de trabajo a la *k*-esima operación en la secuencia de ese trabajo. Una decodificación puede ser observada a continuación:

	1	**2**	**2**	**3**	**1**	**3**	**2**	**2**	**1**	**3**
M1	**1**						**3**	**2**		
M2			**2**		**3**				**1**	
M3				**2**		**1**				**3**

La ventaja de esta representación es que una permutación arbitraria con repeticiones puede ser decodificada en un plan factible. A consecuencia de ello no es necesario un proceso de reparación.

El PPX respeta perfectamente el orden absoluto de los genes en los cromosomas progenitores. Supónganse dos padres p_0 y p_1 codificados con permutación con repetición y la generación de un nuevo individuo *k* también representado por permutación con repetición. Primero, una plantilla de bits *h* de largo *mn* es dada de modo de determinar de que padre, p_0 o p_1, serán obtenidos los genes para generar el nuevo individuo. Un valor de bit 0 significa que el gen correspondiente será copiado de p_0 y un valor de 1 que será copiado de p_1. Cuando un gen es obtenido de un padre y agregado a la descendencia, es borrado del padre y el gen correspondiente es borrado en el otro padre también. Este paso se repite hasta que ambos cromosomas padres están vacíos y la descendencia tiene todos los genes supervivientes.

La Figura 4-1, muestra un ejemplo de este crossover. Comenzando por la imagen superior izquierda, los dos primeros bits de *h* son cero, así que el primer y el segundo trabajos de números 3 y 2 son

copiados a *k* desde p_0 como se muestra. Las ocurrencias más a la izquierda de los trabajos 3 y 2 son borradas de ambos padres en la imagen superior derecha y se borran también los dos primero bits de *h*. Entonces los bits no eliminados más a la izquierda en *h* son cuatro unos consecutivos lo que significa que los siguientes cuatro números de trabajos serán copiados de p_1; los cuatro números de trabajo no eliminados y más a la izquierda son 1121 que se copian en *k*. En la imagen inferior, las ocurrencias más a la izquierda de los números de trabajo 1121 son borrados tanto de p_0 y p_1 como de *h*. Los bits no eliminados restantes en *h* son tres ceros, lo que significa que los números de trabajo restantes serán copiados de p_0 (sin embargo, en el ejemplo los bits no eliminados de p_1 coinciden con los de p_0).

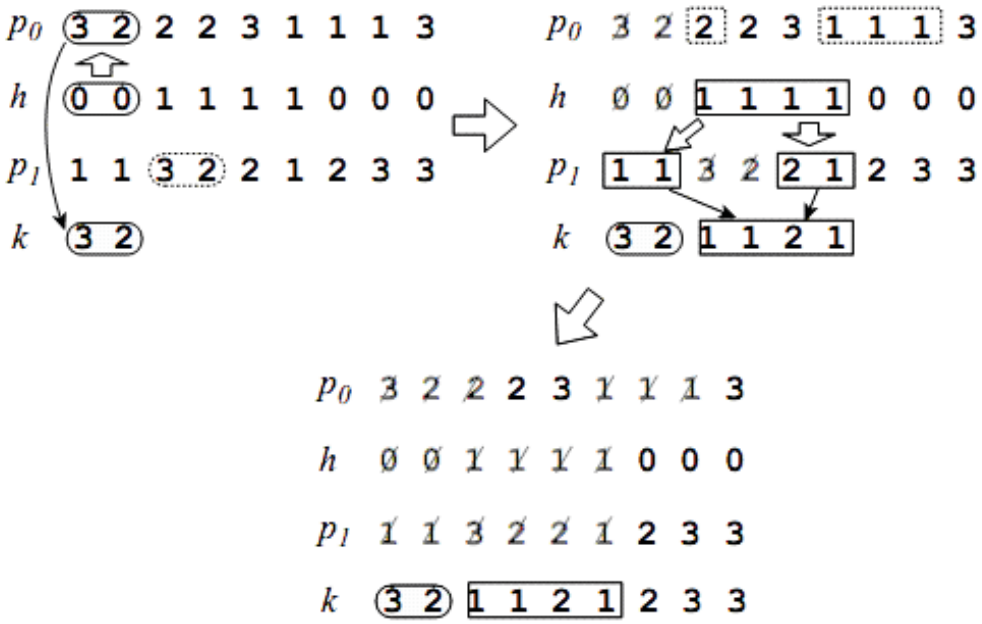

Figura 4-1. – Ejemplo de PPX, donde *k* es generado a partir de p_0 y p_1 utilizando *h*.

CAPITULO 5

MODELOS PROPUESTOS

En este capítulo se revisaran los detalles referentes al modelo de GA que corresponde a lo implementado, que posee una estructura que se puede apreciar a continuación:

Algoritmo Genético

Generar una población inicial aleatoriamente.

Evaluar la población actual;

mientras no se satisfaga una de las condiciones de término

hacer

1. Aplicar el criterio de Selección a los cromosomas de la población actual;
2. Aplicar los operadores genéticos de crossover y mutación a los cromosomas seleccionados en el paso 1.-
3. Evaluar los cromosomas generados en el paso 2.-
4. Aplicar el criterio de sustitución al conjunto de cromosomas formado por los cromosomas seleccionados en el paso 1.- junto con los generados en el paso 2.

fin mientras;

Devuelve el mejor de los cromosomas evaluados;

fin.

En la sección que se encuentra a continuación se llevará a cabo una descripción de los elementos considerados para el modelo.

5.1.- SELECCIÓN DE PROBLEMAS

Para seleccionar los problemas que se utilizan en el proceso de prueba, se procedió a la obtención de los problemas desde el sitio web de la OR Library. Encontrándose todos en un solo archivo de texto plano, separados en secciones señaladas por sus nombres de instancia.

Debido a ello se procedió a la separación de los problemas, en archivos individuales, conservándose el formato original de presentación de los datos, que fueron identificados por su nombre de instancia. Cada uno de estos archivos presenta el siguiente formato:

- La primera línea corresponde a la dimensión del problema, señalada en número de trabajos y número de máquinas separados por espacio.
- Las líneas siguientes corresponden cada una a un trabajo con la lista de operaciones señaladas como número de máquinas y el tiempo de procesado, separados por espacio.
- Finalmente se debe señalar que las máquinas son numeradas partiendo de cero.

Como ejemplo de formato se presenta el correspondiente al problema *mt06*, el que se puede apreciar a continuación:

Tabla 5-1 – Ejemplo de formato de presentación de datos

```
6 6
2 1 0 3 1 6 3 7 5 3 4 6
1 8 2 5 4 10 5 10 0 10 3 4
2 5 3 4 5 8 0 9 1 1 4 7
1 5 0 5 2 5 3 3 4 8 5 9
2 9 1 3 4 5 5 4 0 3 3 1
1 3 3 3 5 9 0 10 4 4 2 1
```

5.2.- REPRESENTACIÓN

La representación del problema en Algoritmos genéticos presenta los siguientes temas a considerar:

- Como se representa un plan de ejecución (que será el fenotipo) por medio de un cromosoma (genotipo).
- Efecto en el algoritmo de la representación utilizada. Permite una búsqueda variada en el espacio de búsqueda o la limita a un trozo.
- Como construir los planes en forma de cromosoma.
- Como pasar de la representación genética al plan de ejecución representado.
- La representación se encuentra relacionada con los operadores; puede que existan algunos que no sean compatibles con la representación.

En vista de ellos y luego de revisar los trabajos de Biewirth, referentes a representaciones por medio de permutaciones [25][[26]], se optó por una permutación con repetición como la representación a utilizar.

En esta representación entera no particionada para un problema de *m* máquinas y *n* trabajos, se tiene una permutación con *m* repeticiones por cada trabajo. De modo que cada número de trabajo aparece exactamente *m* veces en la permutación, asegurando así que al leer la permutación todas sus operaciones se encuentran planificadas. Al leer la permutación de izquierda a derecha, la *i*-esima ocurrencia de un número de trabajo se refiere a la *i*-esima operación en la secuencia tecnológica del trabajo. De este modo se evita el planificar operaciones sin que sus predecesores en el trabajo hayan sido planificados.

Por ejemplo, una permutación representada por la carta Gantt de la Figura 5-1 corresponde a:

Permutación de trabajos: **1 3 2 1 2 1 3 2 3**

Indice de ocurrencia: 1 1 1 2 2 3 2 3 3

Máquina uitilizada: 1 2 1 3 2 2 3 3 1

La línea índice de ocurrencia se refiere a la *i*-esima ocurrencia del número de trabajo y es utilizada para apuntar a la correspondiente operación al leer la permutación. Una permutación con repetición expresa meramente el orden en el que las operaciones son planificadas y se puede apreciar que las secuencias de máquinas no se encuentran presentes directamente en la permutación y sólo aparecen al leerla.

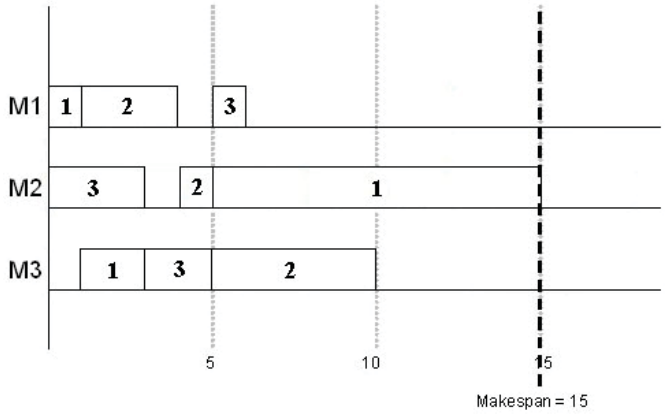

Figura 5-1. – Carta Gantt que surge de la interpretación de una permutación con repetición.

5.3.- FUNCIÓN DE EVALUACIÓN

La función de evaluación calcula el costo de los planes, en términos del tiempo consumido en total por los trabajos planificados, esto es su *makespan*. Como al utilizar la representación señalada en la sección anterior se elimina la existencia de elementos infactibles al decodificar una permutación, la función de evaluación no requiere asignar una penalización al cromosoma por no respetar las restricciones de precedencia. Además como se busca minimizar el tiempo que consumen los planes, un valor fitness menor será considerado mejor que otro mayor.

5.4.- GENERACIÓN DE POBLACIÓN INICIAL

La construcción de los cromosomas pertenecientes a la población inicial, comienza formando las permutaciones con repetición de trabajos y para ello se cuenta con dos posibles implementaciones:

- **Asignación Aleatoria De Trabajos**

En este tipo de asignación de trabajos se seleccionan aleatoriamente las posiciones de las *m* repeticiones de cada trabajo, hasta completar la permutación en el cromosoma, para luego utilizar el algoritmo de decodificación en un plan activo con el fin de determinar su makespan. Al utilizar este método, cada plan se encuentra totalmente determinado por el azar.

- **Asignación De Trabajos Por Utilización De Algoritmo GT**

En este tipo de asignación se selecciona aleatoriamente un número de trabajo, el que se utiliza de punto de partida para el algoritmo de Giffler y Thompson, cuya descripción se puede apreciar en la sección 4.1.1, para generar la secuencia de la permutación en el cromosoma. En este tipo de asignación el valor de makespan se calcula a medida que el algoritmo avanza, a diferencia del método anterior que lo calcula una vez obtenida la permutación. Aun cuando en este caso también es posible aplicar el algoritmo de decodificación en un plan activo, a la permutación resultante.

A diferencia de la asignación aleatoria, la asignación por utilización de algoritmo GT tiende a generar el mismo individuo cuando parte de un mismo trabajo inicial, esto implica que su utilización requiere que se permita la existencia de elementos duplicados en la población inicial. Mientras que en el caso de la utilización de la asignación aleatoria se puede forzar libremente la no existencia de elementos duplicados, pudiendo llegarse a forzar la no existencia de elementos que generen el mismo plan al ser decodificadas sus permutaciones.

5.5.- SELECCIÓN

El mecanismo de selección utilizado es el del torneo, explicado en la sección 3.1.4 de este trabajo. Como se ha mencionado se busca minimizar el tiempo total consumido por los trabajos al ser planificadas sus operaciones, por lo que el ganador del torneo resulta ser el que posea el menor valor de fitness.

Como el tamaño del torneo es fijado en dos, sólo compiten dos cromosomas en cada torneo, seleccionándose en total *n/2* elementos de la población al finalizar.

5.6.- OPERADORES GENÉTICOS

5.6.1.- Cruzamiento

Se cuenta con tres operadores de cruzamiento diseñados para permutaciones con repetición, los cuales trabajan contemplando que se transmitan de manera uniforme las características de ambos progenitores.

Los operadores implementados son: *PPXCrossover*, *GOXCrossover* y *GPMXCrossover*, de los cuales el PPX ya fue descrito en la que fue descrito en la sección 4.3.1, por lo cual sólo el GOX y el GPMX serán explicados a continuación.

- ***GOXCrossover***

El *GOXCrossover* corresponde a una generalización del *Order Crossover* (*OX*) para permutaciones, para adecuarlo a la representación por permutación con repetición.

En el *GOX* existe un padre *donador* (*donor*) y un padre *receptor* (*receiver*), el *donador* contribuye con un substring de largo correspondiente al rango de un tercio a la mitad del largo del cromosoma. Eligiendo un largo dentro de este rango asegura que la descendencia herede una cantidad similar de información de ambos padres. Como se aprecia a continuación, los genes en el substring seleccionado reciben también un índice que representa el número de operación del trabajo correspondiente.

Cromosoma: 1 2 2 1 3 1 2 3 3

Índice de genes en el substring: 2 1 3 3

Para determinar la posición donde implantar el string, se utiliza un método proveniente del OX en el problema de TSP. Los genes en el cromosoma receptor que son representativos del substring en términos de de número de índice son marcados y el padre obtiene un corte en el gen que es igual al primer gen (y su índice) en el string de crossover del donante. Todos los genes marcados son entonces removidos. Como se aprecia a continuación, el padre receptor es mostrado con los genes marcados en negritas.

2 1 2 **2 3** ***1*** 3 3 **1**

El descendiente obtiene el substring 1 3 1 2 después del ***1*** y el resultado es el siguiente:

2 1 2 1 3 1 2 3 3

- ***GPMXCrossover***

Tal como en el *GOXCrossover*, existe un padre *donador* (*donor*) que contribuye con un substring de largo correspondiente al rango de un tercio a la mitad del largo del cromosoma y un padre *receptor* (*receiver*). Como se aprecia a continuación, los genes en el substring seleccionado reciben también un índice que representa el número de operación del trabajo correspondiente.

Cromosoma: 3 2 2 2 3 1 1 1 3

Índice de genes en el substring: 2 3 2 1

Para determinar la posición donde implantar el string, los genes en el cromosoma receptor que son representativos del substring en términos de de número de índice son marcados y el padre obtiene un corte en la posición en que el substring ocurre en el donante. Todos los genes marcados son entonces removidos. Como se aprecia a continuación, el padre receptor es mostrado con los genes marcados en negritas.

1 1 3 2 **2** 1 **2 3** 3

El descendiente obtiene el substring 2 2 3 1 en la posición en que ocurre en el donante y el resultado es el siguiente:

1 3 2 2 3 1 2 1 3

Tal como plantea Biewirth [[26]], se espera que el GOXCrossover traspase el orden relativo de los genes a los descendientes, mientras que el GPMXCrossover traspase las posiciones de los genes respetando hasta cierto punto el orden y el PPXCrossover respete el orden absoluto de los genes, resultando en una preservación de las relaciones de precedencia entre genes.

Los tres operadores implementados terminan con descendientes factibles, por lo que no resulta necesaria una función de reparación luego de aplicarlos.

Tomando estos tres operadores, se implemento un operador que aplica los tres operadores y luego selecciona el mejor descendiente como resultado del operador.

5.6.2.- Mutación

Se cuenta con cuatro operadores de mutación diseñados para permutaciones con repetición. Los operadores son aplicados, según una probabilidad dada, a cada hijo después del cruzamiento.

- **Mutación por Inserción**

El operador es aplicado al cromosoma como un todo. Si la mutación ocurre, entonces se selecciona un gen de forma aleatoria y se le inserta en una posición arbitraria. Por ejemplo; si se considera el cromosoma:

C: 2 1 2 *1* 3 1 2 3 3

Donde el índice del gen seleccionado es 3, que corresponde al valor *1* en cursiva y la posición de inserción es 7. Se inserta el gen en la posición indicada resultando el nuevo cromosoma:

C': 2 1 2 3 1 2 3 *1* 3

- **Mutación por Desplazamiento**

Es la generalización de la mutación por inserción, en lugar de desplazar un solo gen, se mueven varios a la vez. Pudiendo establecerse si los genes desplazados son a su vez invertidos en su ordenamiento, lo que aumenta el efecto del operador. Por ejemplo; si se considera el cromosoma:

C: 2 1 2 1 *3 1 2* 3 3

Donde los genes a desplazar corresponden a los ubicados en las posiciones 4, 5 y 6; y la posición de desplazamiento es 2. Se insertan los genes en la posición indicada resultando el nuevo cromosoma:

C': 2 1 2 *3 1 2* 1 3 3

Ahora si el operador incluye la inversión de los genes seleccionados, el nuevo gen resulta ser:

C'': 2 1 2 *2 1 3* 1 3 3

- **Mutación por Intercambio Recíproco**

El operador es aplicado al cromosoma como un todo. Si la mutación ocurre, entonces se seleccionan dos puntos al azar y se intercambian estos genes de posición. Por ejemplo; si se considera el cromosoma:

C: 2 *1* 2 1 3 1 2 *3* 3

Donde los índices de los genes seleccionados son 1 y 7, que corresponden a los valores 1 y *3* en cursiva. Se intercambian los genes de posición resultando el nuevo cromosoma:

C' 2 *3* 2 1 3 1 2 *1* 3

- **Mutación por Inversión de Permutación**

El operador es aplicado al cromosoma como un todo. Si la mutación ocurre, entonces se seleccionan dos puntos al azar y se invierte la permutación entre estos puntos. Por ejemplo; si se considera el cromosoma:

C: 2 1 2 1 3 1 2 3 3

Donde el segmento a invertir abarca desde la posición 2 hasta la posición 5. Se invierte el segmento indicado resultando el nuevo cromosoma:

C': 2 1 1 3 1 2 2 3 3

5.7.- SUSTITUCIÓN

Una vez que se han producido nuevos cromosomas por selección, crossover y mutación de cromosomas de la población anterior, se calcula la calidad de los cromosomas producidos. Si se producen menos cromosomas que el tamaño de población original, los nuevos cromosomas deben ser insertados en la población. De manera similar, si se producen más cromosomas que los necesitados o no todos son utilizados; se vuelve necesario un esquema de sustitución para determinar que cromosomas serán insertados en la nueva población.

Existen diversos esquemas de sustitución tales como:

- Producir tantos descendientes como el número de progenitores y reemplazar todos los progenitores por los descendientes (Sustitución Pura)
- Producir menos descendientes que el número de progenitores y reemplazar los progenitores aleatoriamente de manera uniforme (Sustitución Uniforme).
- Producir menos descendientes que el número de progenitores y reemplazar los peores progenitores (Sustitución Elitista)
- Producir más descendientes que los necesarios para sustitución y se insertan sólo los mejores descendientes (Sustitución basada en calidad).

La sustitución pura es el esquema más simple, donde cada individuo “vive” tan sólo una generación. Sin embargo, se suele dar que buenos individuos son reemplazados sin producir mejores descendientes y de ese modo se pierde información que pudiese llevar a mejores soluciones. La sustitución elitista previene la pérdida de información, en cada generación los elementos de menor calidad de la población son reemplazados por el mismo número de descendientes. La sustitución basada en calidad produce un esquema de selección truncada entre descendientes antes de insertarlos en la población (antes que puedan participar de un proceso reproductivo). Por otra parte los mejores individuos pueden vivir muchas generaciones, aun cuando se inserten nuevos individuos cada generación.

En la sustitución uniforme, no se verifica si los progenitores son reemplazados por mejores o peores descendientes. Es por ello que si son reemplazados por cromosomas con menor calidad, la calidad promedio de la población puede descender [27].

CAPITULO 6

PRUEBAS Y RESULTADOS

6.1.- DATOS DE BENCHMARK

Para realizar las distintas pruebas, se utilizan como datos de benchmark las instancias *mt06, mt10, mt20, abz07, abz08* y *abz09* (de los "*diez problemas difíciles*") y *ta11, ta13* y *ta*41 utilizadas en diversos trabajos sobre Job-Shop Scheduling Problem.

Las dimensiones de las instancias seleccionadas son:

- *abz07, abz08 y abz09*: 15 x 20
- *mrt06:* 6 x 6
- *mt10:* 10x 10
- *mt20*: 20 x5
- *ta11* y *ta13*: 20 x 15
- *ta41*: 30 x 20

Quedando fuera las instancias *ta76 y ta79*, a pesar de ser consideradas en principio por su dificultad y tamaño (100 x20), debido a que los resultados en ambos casos no eran aceptables en cuanto a tiempo requerido para procesarlos.

6.2.- PRUEBAS DE PROTOTIPOS

En la etapa precedente de este proyecto se llevo a cabo la prueba de dos prototipos que llevaron al modelo final y configuraciones de prueba del algoritmo final, es por ello que a continuación se señalan sus características particulares y un resumen de resultados obtenidos en sus pruebas

6.2.1.- Configuración de Prototipos

6.2.1.1.- Selección

La selección utilizada en el prototipo I corresponde a la selección por torneo, ya que las ventajas que presenta este medio de selección son por una parte el manejo de los factores como tamaño del torneo o la probabilidad de selección entre los participantes; y el mapeo entre la prioridad de selección de los participantes del torneo es directa; aquellos con función de evaluación menor tienen mayores posibilidades que si fueran seleccionados de forma elitista

Su implementación se determinó realizarla con tamaño de torneo 2 por simplificación, aun cuando puede ser extendido a tamaño 3 o 4, de resultar necesario.

Mientras que los métodos de selección implementados en el prototipo II corresponden a elección por Roulette-Wheel descrita en la sección 3.1.4 y a selección por ranking lineal de individuos, la que se describe a continuación de manera somera, ya que en la etapa final no fue utilizada.

La selección por ranking es un método en el cual la población es ordenada de acuerdo a su fitness y el valor esperado de cada individuo depende de su ranking en lugar de su fitness.

En la selección la población es ordenada de mayor a menor calidad para luego realizar una selección proporcional, en la cual los rangos de la ruleta se establecen como siguen; el peor elemento consigue 1, el segundo peor 2 y así hasta llegar al mejor que obtiene N (donde N es el número de cromosomas en la población). Como se puede apreciar el número total de rangos corresponde a $N(N+1)/2$.

6.2.1.2.- Crossover y Mutación

Los operadores de crossover y mutación a utilizar en el prototipo I corresponden a *Precedence Preservative Crossover (PPX)* y a *mutación por inserción*, con valores de probabilidad 0.60 y 0.75, para el crossover; mientras que para la mutación los valores son 0.05 y 0.1.

Los operadores de crossover y mutación implementados en el prototipo II corresponden a *Generalized Order Crossover (GOX)* y a *mutación por desplazamiento*, *mutación por intercambio recíproco y mutación por inversión de permutación*; con valores de probabilidad 0.60 y 0.75, para el crossover; mientras que para la mutación los valores son 0.05 y 0.1.

6.2.1.3.- Sustitución

Considerando las posibilidades de sustitución, se seleccionó la sustitución elitista (de un 20 % de la población) para ser implementada en el prototipo I, a pesar que la sustitución pura resulta más sencilla de implementar, para evitar la perdida de información que lleve a buenas soluciones.

Mientras que para el prototipo II, se seleccionó la sustitución basada en calidad para ser implementada en el prototipo, con una tasa de reemplazo de un 20% de la población.

6.2.1.4.- Condición De Término

La condición de termino para ambos prototipos se determino que sería en base al número de generaciones que se pase cómo parámetro al algoritmo.

6.2.2.- Resumen De Pruebas De Prototipos y Resultados

En esta sección se mostrarán algunos resultados obtenidos con los dos prototipos generados, en base a distintos parámetros que se utilizaron para la ejecución de los mismos.

6.2.2.1.- Detalles De Ejecución Y Resultados de Prototipo I

Para este prototipo, se realizaron 180 ejecuciones considerando las diferentes configuraciones que se aprecian a continuación:

- Instancias de los conjuntos de datos mencionados en 6.1: m*t06, mt10, mt20, abz07, abz08 y abz09*.
- Tamaño de la población: 100.
- Cantidad de generaciones:
  - *mt06* : 100 y 500
  - *mt10*: 1000, 5000 y 10000.
  - *mt20*: 1000, 5000 y 10000.
  - *abz07*,*abz08* y *abz09*: 1000

- Tamaño del torneo en la selección: 2.
- Probabilidad de crossover: 0.60, 0.75.
- Probabilidad de mutación: 0.05, 0.1.

Para cada combinación se realizaron 5 ejecuciones y los resultados están basados en el promedio de las mismas. Los mejores resultados de estas ejecuciones se muestran en la Tabla 6-1.

Tabla 6-1 – Mejores resultados en ejecución de prototipo I

Set de Datos	Tiempo (min.)	Crossover	Mutación	Nº Generaciones	Makespan	Optimo / BKS
abz07 *(20 x 15)*	3	0.6	0.05	1000	735	656
abz08 *(20 x 15)*	2	0.75	0.1	1000	759	645/669
abz09 *(20 x 15)*	3	0.6	0.1	1000	777	669/679
mt06 *(6 x 6)*	1	0.6-0.75	0.05-0.1	100	55	55
mt10 *(10 x 10)*	4	0.75	0.1	5000	951	930
mt20 *(20 x 5)*	5	0.75	0.1	10000	1185	1165

Como se puede apreciar en la tabla anterior, sólo en el caso del problema *mt06*, el mejor resultado fue obtenido indistintamente con cualquiera de las combinaciones de los valores de crossover y mutación; y con una cantidad de generaciones baja en comparación a los otros problemas. Pudiendo apreciarse que en relación al resto de los problemas, el prototipo se acercó a los valores óptimos sólo en los problemas *mt*,

quedando como desafío el mejorar en la resolución de estos y en otros de mayor envergadura como los problemas *abz.*

6.2.2.2.- Detalles De Ejecución Y Resultados De Prototipo II

Para este prototipo, se realizaron 180 ejecuciones considerando las diferentes configuraciones que se aprecian a continuación:

- Instancias de los conjuntos de datos mencionados en 6.1: m*t06, mt10, mt20, abz07, abz08 y abz09*.
- Tamaño de la población: 100.
- Cantidad de generaciones:
  - *mt06* : 100 y 500
  - *mt10*: 1000, 5000 y 10000.
  - *mt20*: 1000, 5000 y 10000.
  - *abz07*,*abz08* y *abz09*: 1000
- Selección por ranking lineal de individuos.
- Probabilidad de crossover: 0.60, 0.75.
- Probabilidad de mutación: 0.05, 0.1.

Para cada combinación se realizaron 5 ejecuciones y los resultados están basados en el promedio de las mismas. Los mejores resultados de estas ejecuciones se muestran en la Tabla 6-2.

Tabla 6-2 – Mejores resultados en ejecución de prototipo II

Set de Datos	Tiempo (min.)	Crossover	Mutación	Nº Generaciones	Makespan	Optimo / BKS
abz07 (20 x 15)	3	0.75	0.1	1000	790	656
abz08 (20 x 15)	2	0.75	0.1	1000	804	645/669

Tabla 6-2 Cont. – Mejores resultados en ejecución de prototipo II

Set de Datos	Tiempo (min.)	Crossover	Mutación	Nº Generaciones	Makespan	Optimo / BKS
abz09 *(20 x 15)*	3	0.75	0.1	1000	838	669/679
mt06 *(6 x 6)*	1	0.6-0.75	0.05-0.1	100	55	55
mt10 *(10 x 10)*	6	0.75	0.1	5000	1043	930
mt20 *(20 x 5)*	6	0.75	0.1	10000	1257	1165

Como se puede apreciar en la tabla anterior, sólo en el caso del problema *mt06*, el mejor resultado fue obtenido indistintamente con cualquiera de las combinaciones de los valores de crossover y mutación; y con una cantidad de generaciones baja en comparación a los otros problemas. Para el resto de los problemas el prototipo encuentra el valor más cercano al óptimo (o al mejor valor conocido según el problema), al utilizarse la combinación de los mayores valores de crossover y mutación.

6.2.3.- Conclusiones De Las Pruebas De Los Prototipos

Al comparar los resultados obtenidos, se pueden observar los siguientes aspectos importantes en los resultados:

- Se observa que el prototipo I tiene un mejor desempeño en comparación al prototipo II, tanto en tiempo como en calidad de las soluciones, aun cuando la diferencia en los valores no es de magnitud como para desechar el modelo del prototipo II.
- Para ambos modelos los mejores resultados se obtuvieron en general con los mayores valores de las tasas de crossover y mutación.

6.3.- CONFIGURACIONES DEL ALGORITMO FINAL

Con la variedad de operadores disponibles e implementados, se utilizan las siguientes configuraciones del algoritmo final, para comparar:

- **Configuración 1**
  - Generación de la población inicial utilizando asignación aleatoria de trabajos, forzando la no existencia de elementos duplicados.
  - Selección por torneo
  - Utilización como operador de cruzamiento, un operador que aplique los tres operadores implementados para luego seleccionar el mejor descendiente producido como resultado del operador.
  - Utilización como operador de mutación, un operador que aplique los operadores implementados y seleccione el mejor elemento mutado, como resultado del operador.
  - Sustitución elitista

- **Configuración 2**
  - Generación de la población inicial utilizando asignación aleatoria de trabajos, forzando la no existencia de elementos duplicados.
  - Selección por torneo
  - Utilización como operadores de cruzamiento en base a resultados de configuración 1, de PPXCrossover y GPMXCrossover, en intervalos de generaciones fijados de antemano, para comparar en relación al operador de configuración 1.
  - Utilización como operador de mutación, un operador que aplique los operadores implementados y seleccione el mejor elemento mutado, como resultado del operador. Esto se mantiene sin variación ya que no existe una relación apreciable en el uso de los diversos operadores de mutación a lo largo de las generaciones.
  - Sustitución elitista

6.4.- CALIBRACIÓN DE PARÁMETROS

Dentro del funcionamiento del algoritmo existe una diversidad de parámetros que afectan a los distintos procedimientos y operadores. Estos parámetros normalmente interactúan entre sí de forma no lineal, por lo que no pueden optimizarse de manera independiente.

La calibración de estos parámetros es una tarea de alta importancia, dado que influye directamente con el rendimiento final que se desea obtener. La forma adecuada de definir los parámetros de un algoritmo genético ha sido motivo de investigación desde los orígenes mismos de la técnica, y no existe hasta la fecha una solución satisfactoria a este problema.

Considerando la cantidad de operadores y la gran cantidad de posibilidades de combinación de parámetros existentes, se ha decido la construcción de un programa que automatice la ejecución del algoritmo con distintas configuraciones y parámetros, entregando los resultados estadísticos de cada ejecución.

Dentro de la variedad de parámetros que deben ser contemplados y analizados se pueden mencionar los siguientes:

- Número de individuos en la población inicial
- Probabilidad de cruzamiento
- Probabilidad de mutación.
- Tasa de sustitución

6.4.1.- Número De Individuos En La Población Inicial

El número de individuos presentes en la población inicial fue determinado en base a pruebas cortas de 10 repeticiones en la ejecución con los valores de crossover y mutación utilizados en los prototipos mostrados en la sección 6.2.; estas pruebas contemplaron los valores de 100, 200 y 500 individuos. Los resultados obtenidos con estos valores mostraron que la influencia en la calidad de las soluciones encontradas

6.4.2.- Probabilidad De Cruzamiento

La probabilidad de cruzamiento, fue determinada al igual que el número de individuos en la población inicial, por medio de pruebas cortas con 10 repeticiones, evaluándose los valores 0.6, 0.70, 0.75, 0.80 y 0.85; presentándose fluctuaciones en la calidad de los resultados; obteniéndose los mejores resultados, en conjunto con los valores de mutación utilizados para las pruebas (0.05, 0.08, 0.1, 0.15), en los valores de 0.75 y 0.85.

6.4.3.- Probabilidad De Mutación

La probabilidad de mutación, al igual que los valores mencionados en las secciones precedentes, se determinó por medio de pruebas cortas en las cuales se utilizaron los valores 0.08, 0.1, 0.15 y 0.20 en conjunción con los valores obtenidos para la probabilidad de cruzamiento. Siendo seleccionados finalmente los valores 0.1 y 0.15

6.4.4.- Tasa De Sustitución

La tasa de sustitución o reemplazo, por simplificación, fue fijada utilizando en un 30 por ciento de la población. Valor que no produce una variación exagerada de elementos entre generaciones y tampoco permite que se produzca un estancamiento por pocos reemplazos.

6.5.- RESUMEN DE PRUEBAS DE ALGORITMO FINAL

6.5.1.- Detalles De Ejecución Y Resultados De Configuración I

Para esta configuración, se realizaron 960 ejecuciones considerando las diferentes combinaciones que se aprecian a continuación:

- Instancias de los conjuntos de datos mencionados en 6.1: *mt10, mt20, abz07, abz08, abz09* y *ta11, ta13 y ta41*.
- Tamaño de la población: 100.
- Cantidad de generaciones: 1000

- Tamaño del torneo en la selección: 2.
- Probabilidad de crossover: 0.75, 0.85.
- Probabilidad de mutación: 0.1, 0.15.

Para cada combinación se realizaron 30 ejecuciones y los resultados están basados en el promedio de las mismas. Los mejores resultados de estas ejecuciones se muestran en la Tabla 6-3.

Tabla 6-3 – Mejores resultados en ejecución con configuración I

Set de Datos	Tiempo (min.)	Crossover	Mutación	Optimo / BKS	Makespan	% ERs
abz07 *(20 x 15)*	3	0.75	0.1	656	742	13,10
abz08 *(20 x 15)*	2	0.85	0.1	645/669	762	18,13/13,90
abz09 *(20 x 15)*	3	0.75	0.1	669/679	802	19,88/18,11
mt10 *(10 x 10)*	1	0.75	0.1	930	989	6,34
mt20 *(20 x 5)*	1	0.85	0.1	1165	1207	3,60
ta11 *(20 x15)*	3	0.75	0.1	1323/1357	1598	20,78/17,75
ta13 *(20 x15)*	3	0.75	0.1	1282/1342	1575	22,85/17,36
ta41 *(30 x 20)*	13	0.85	0.1	1859/2014	2503	34,64/ 24,28

Tal como se señaló en la sección 6.3 para determinar los valores de los intervalos de utilización de los operadores de crossover, se debió analizar el comportamiento de los porcentajes de cada uno de los operadores de crossover por generación.

Al finalizar las distintas ejecuciones de un mismo problema y comparar los gráficos que representan el promedio calculado a finalizar las 30 repeticiones de cada combinación, un ejemplo de estos gráficos se puede apreciar en la Figura 6.1, que muestra el grafico correspondiente al problema *ta13*, se pudo apreciar el mismo comportamiento; en el inicio del gráfico el porcentaje de utilización del PPX crossover disminuye en beneficio de la utilización del GPMX crossover.

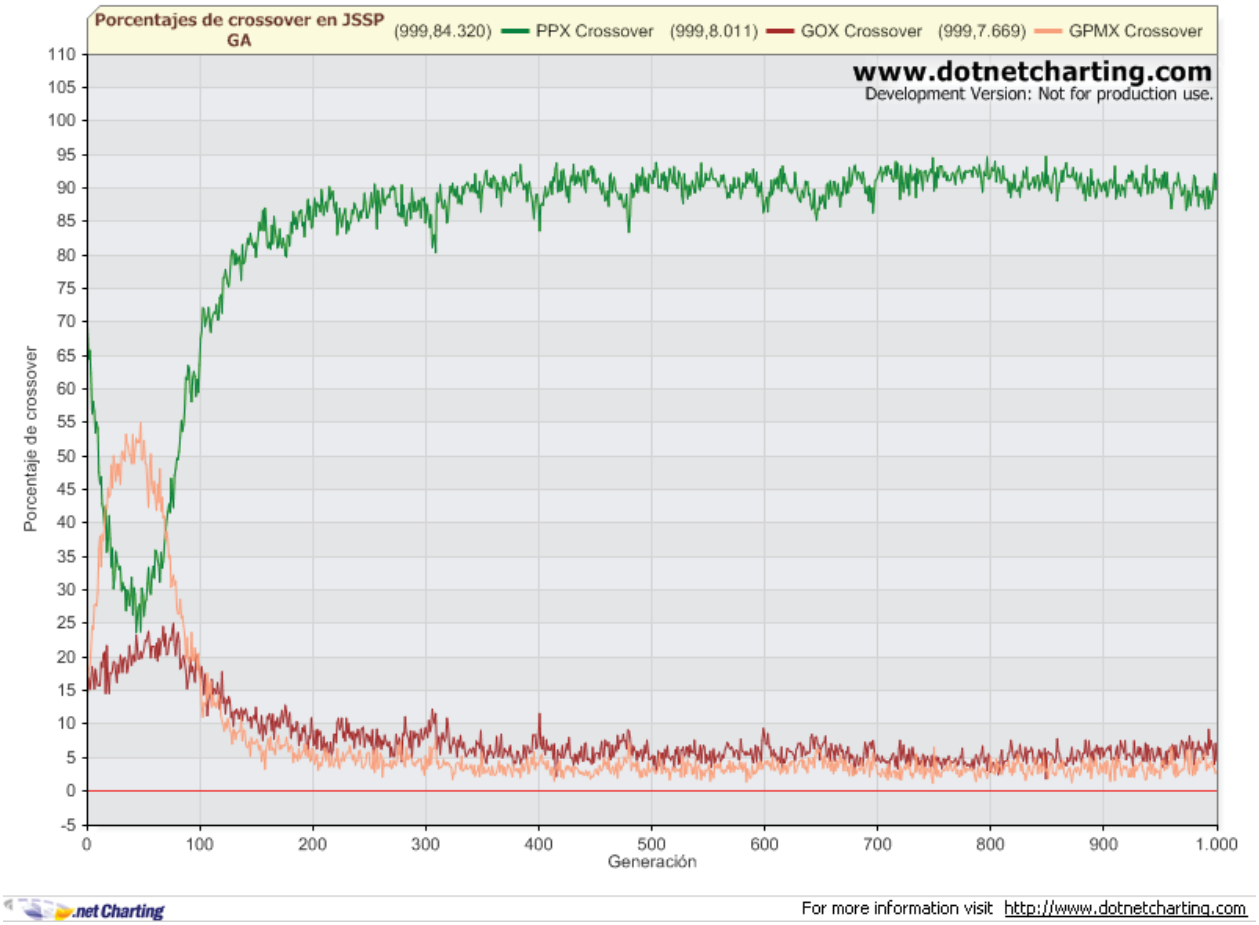

Figura 6-1.- Grafico de promedios de porcentajes de utilización de operadores de crossover para el problema *ta13*.

En relación a un mismo problema el comportamiento apreciable en los gráficos, puede ser un comportamiento que resulte exclusivo del problema, por lo que verlo reflejado en los gráficos correspondientes a otros problemas, tal como se puede apreciar en la Figura 6.2, que corresponde al problema *mt10*, plantea la necesidad de determinar los valores del intervalo en el cual el PPX crossover cede ante el GPMX crossover. Resultando los valores promedio de los límites del intervalo, la generación 16 y la generación 78, siendo estos valores traspasados a la configuración II del algoritmo para comparar los resultados y verificar si se presentan mejoras al utilizar esta combinación de uso de operadores; eliminándose para efectos de esta comparación la utilización del GOX Crossover.

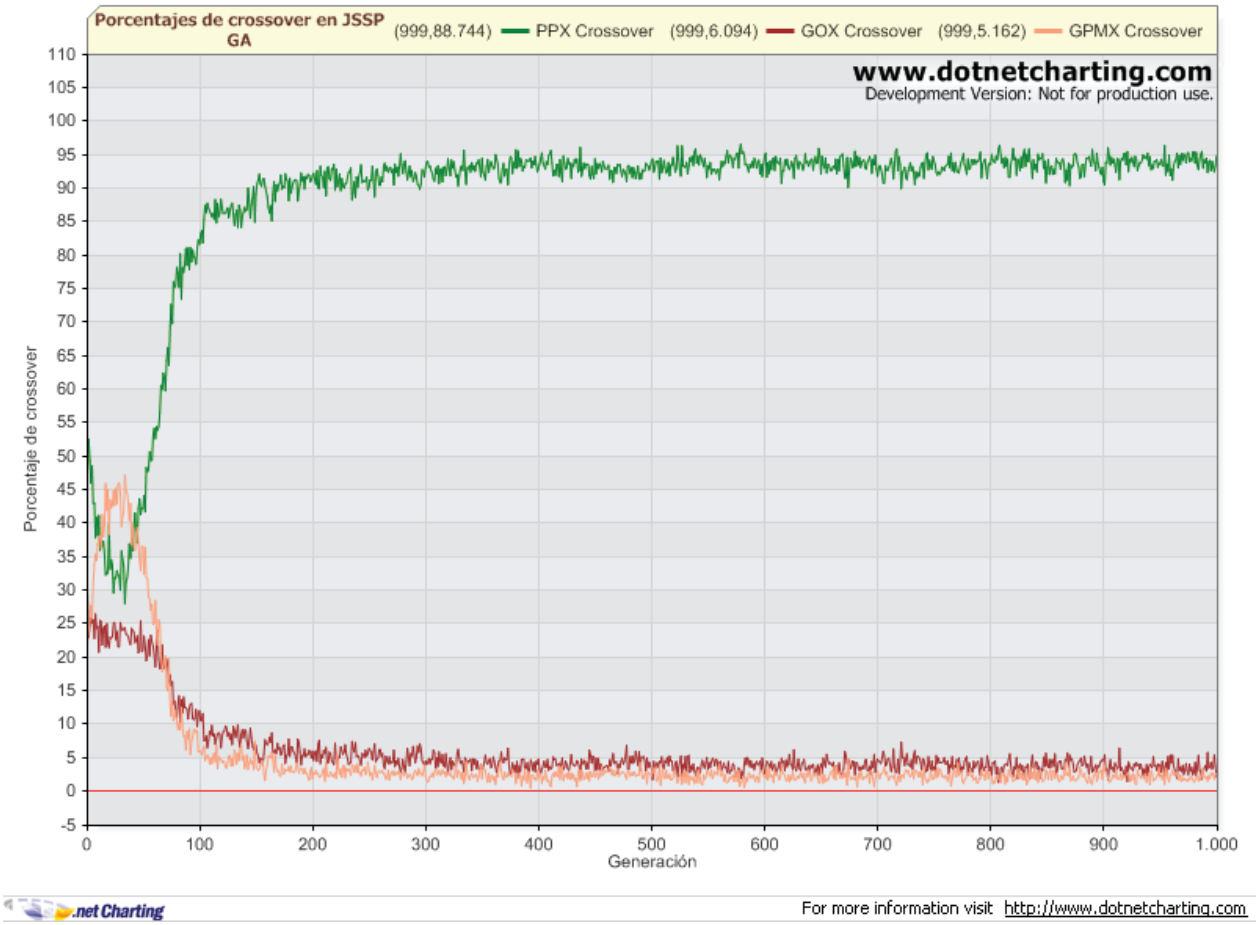

Figura 6-2.- Grafico de promedios de porcentajes de utilización de operadores de crossover para el problema *mt10*.

6.5.2.- Detalles De Ejecución Y Resultados De Configuración II

Para esta configuración, se realizaron 960 ejecuciones considerando las diferentes combinaciones que se aprecian a continuación:

- Instancias de los conjuntos de datos mencionados en 6.1: *mt10, mt20, abz07, abz08, abz09* y *ta11, ta13 y ta41*.
- Tamaño de la población: 100.
- Cantidad de generaciones: 1000.
- Tamaño del torneo en la selección: 2.
- Probabilidad de crossover: 0.75, 0.85.
- Probabilidad de mutación: 0.1, 0.15.
- Utilización de:
    - GPMX Crossover : Entre generaciones 16 y 78
    - PPX Crossover: Desde el inicio hasta generación 15 y luego desde generación 79 hasta generación 1000.

Para cada combinación se realizaron 5 ejecuciones y los resultados están basados en el promedio de las mismas. Los mejores resultados de estas ejecuciones se muestran en la Tabla 6-4.

Tabla 6-4 – Mejores resultados en ejecución con configuración II

Set de Datos	Tiempo (min.)	Crossover	Mutación	Optimo / BKS	Makespan	% ER
abz07 (20 x 15)	2	0.85	0.1	656	738	12,5
abz08 (20 x 15)	2	0.75	0.1	645/669	774	20/15,69
abz09 (20 x 15)	2	0.75	0.1	669/679	797	19,13/17,37

Tabla 6-4 Cont. – Mejores resultados en ejecución con configuración II

Set de Datos	Tiempo (min.)	Crossover	Mutación	Optimo / BKS	Makespan	% ER
mt10 (10 x 10)	1	0.75	0.1	930	980	5,37
mt20 (20 x 5)	1	0.75	0.1	1165	1242	6,60
ta11 (20 x15)	3	0.85	0.1	1323/1357	1614	21,99/18,93
ta13 (20 x15)	3	0.85	0.1	1282/1342	1580	23,24/17,73
ta41 (30 x 20)	11	0.85	0.1	1859/2014	2566	38,03/27,40

Como se puede apreciar al comparar los valores obtenidos, no existe una variación cualitativa respecto a la configuración I como para suponer que se produce algún tipo de interferencia al aplicar los operadores de crossover en determinadas generaciones de manera selectiva, quedando sólo la interrogante de cómo se habría de comportar el algoritmo con una estructura de utilización probabilística de operadores de crossover.

CAPITULO 7

CONCLUSIONES

En este trabajo se ha estudiado el problema de Job-Shop. Resultando el mayor interés en este trabajo el generar y utilizar un algoritmo genético en la resolución del problema. Desarrollándose un algoritmo para resolver el problema, para lo cual se ha presentado el desarrollo del marco teórico del proyecto, por medio de la definición de objetivos, plan de trabajo y estudio conceptual de Job-Shop Scheduling Problem y Algoritmos Genéticos; el desarrollo de dos modelos para la estructura y operadores de un algoritmo genético aplicado a la resolución del problema de Job-Shop.

La revisión bibliográfica del tema se encuentra directamente enfocada por una parte a conocer en que consiste el problema de planificación de Job-Shop (JSSP), sus principales características y restricciones que lo definen como problema complejo; mientras que por la otra parte lo fue a conocer los aspectos teóricos de Algoritmos Genéticos, con su funcionamiento básico y posibles variantes que surgen en relación con su utilización en Job-Shop Scheduling Problem. Para lo último señalado se produjo una revisión de diversos trabajos previos que guardan relación con el propósito de este trabajo.

En base a ello, se plantean los siguientes puntos:

- La representación, que corresponde a un factor fundamental en el uso de GA en Job-Shop Scheduling Problem no es trivial, pues dependiendo si es binaria o no, es posible utilizar determinados operadores de crossover y mutación. Ante ello se puede ver que la representación utilizada en este fue trabajo corresponde a entera del tipo permutación con repetición de trabajos, que resulta una representación simple de fácil lectura e interpretación para entender la secuencia de operaciones planificadas, a diferencia de otras representaciones en las cuales la interpretación resulta más compleja y aumenta la probabilidad de errar en ella. La utilización de la representación señalada, conlleva una restricción en los operadores de Crossover y Mutación, aun cuando se cuenta con varios de ellos adecuados a la representación.
- El uso del algoritmo genético tradicional no parece una buena alternativa de solución del problema que atañe al proyecto, aun cuando su estructura básica se mantenga, es necesario contemplar las

modificaciones que han sido propuestas para una utilización adecuada en Job-Shop Scheduling Problem.

- Los parámetros probabilidad de mutación, tasa de crossover y población inicial, son importantes para evitar tanto la convergencia temprana, como la tardía del algoritmo y que debían ser ajustados para las pruebas del algoritmo propuesto.

En base a los puntos detallados y al uso de la representación escogida, se definieron dos modelos de algoritmo a implementar como prototipos, después de considerar las posibilidades que se presentaron para cada parte constituyente de los modelos.

Pudiendo afirmarse que el uso de prototipos cumple la función de encauzar las mejoras en el desarrollo del algoritmo final, al permitir resolver problemas antes de que se convirtiesen en un lastre que retrase el resultado final; y por medio de los resultados obtenidos en las pruebas preliminares se permitió la corrección de errores de implementación e integración de los diversos módulos.

El valor de los parámetros, los que fueron determinados por medio de la experimentación con diversos valores, encontrándose de esta manera los valores utilizados en la evaluación final; aún pueden ser mejorados con un tunning más fino y preciso.

En base a los resultados finales, se observa que si bien estos resultan aceptables, se puede mejorar el desempeño de los algoritmos planteados, de modo de encontrar soluciones en menos tiempo y con menor consumo de recursos.

El trabajo futuro se encuentra orientado a diversas direcciones no cubiertas en este trabajo. A continuación se señalan algunas que no resultan ser todas las posibles:

- Planteamiento de un esquema de generación de población inicial, por medio de una heurística constructiva, que permita reducir los valores promedio de makespan de los individuos generados.
- Generación de un algoritmo que utilice de manera probabilística los operadores de crossover, pudiendo compararse esos resultados con los obtenidos en el desarrollo de este proyecto.
- Generación de un algoritmo hibrido, partiendo del algoritmo final planteado este proyecto, por medio de la utilización de estrategias complejas de búsqueda local como Simulated Annealing o Tabú Search, para mejora de las soluciones encontradas.

- La posibilidad de incorporar el componente Cultural para generar un Algoritmo Genético Cultural, que permita resolver el problema de Job-Shop, ofreciendo mejores valores en las soluciones encontradas.

REFERENCIAS BIBLIOGRÁFICAS

[1] **Hui Peng, Lee. Salim, Sutinah**. *"A modified Giffler and Thompson Genetic Algorithm on the Job Shop Scheduling Problem (2006)"*. MATEMATIKA, 2006, Vol. 22, Number 2, Año 2006, PP 91-107.

[2] **Watson, Jean Paul. Howe, Adele. Whitley, Darrel**. *"Deconstructing Nowicki and Smutnicki's i-TSAB Tabu Search Algorithm for the Job-Shop Scheduling Problem"*. Computers and Operations Research Journal vol. 33, Elsevier Science. Año 2006, PP 2623-2644.

[3] **Jain, Anant Singh. Meeran, Sheik**. *"Deterministic Job-Shop Scheduling: Past, Present And Future"*, European Journal Of Operational Research vol. 113, Año 1999, PP 390-434.

[4] **Jones, Albert. Rabelo, Luis.** *"Survey of Job Shop Scheduling Techniques"*. NISTIR, National Institute of Standards and Technology, Gaithersburg, MD, Online Publication, Año 1998.

[5] **Jain, Anant Singh. Meeran, Sheik**. *"A State-Of-The-Art Review Of Job-Shop Scheduling Techniques"*, Department of Applied Physics, Electronics and Mechanical Engineering University of Dundee, Dundee, Scotland. Año 1998.

[6] **Holland J.H..** *"Adaptation in Natural and Artificial Systems: An Introductory Analysis with Applications to biology, control and artificial intelligence"*, MIT Press, ISBN 0-262-58111-6. Año 1998 (NB original printing 1975).

[7] **Yamada, Takeshi.** *"Studies on Metaheuristics for Jobshop and Flowshop Scheduling Problems"*, PhD Thesis, Kyoto University. Kyoto, Japan. Año 2003.

[8] **Mattfeld, D.C..** *"Evolutionary Search and the Job Shop – Investigations on Genetic Algorithms for Production Scheduling"*, Physica-Verlag, Heidelberg. Año 1996.

[9] **Watson, Jean Paul**. *"The Job-Shop Scheduling Problem"*, Empirical Modeling and Analysis of Local Search Algorithms for the Job-Shop Scheduling Problem, Colorado State University, Año 2003.

[10] **Yamada, Takeshi. Nakano, Ryohei**. *"Genetic algorithms in engineering systems"*, Chapter 7: Job-Shop Scheduling. IEE control engineering series 55, The Institution of Electrical Engineers, ISBN: 0 85296 902. Año 1997, PP. 134-160.

[11] **Yamada, Takeshi. Nakano, Ryohei**. *"Genetic Algorithms for Job-Shop Scheduling Problems"*. Preceedings of Modern Heuristic for Decision Support, UNICOM seminar, 18-19, March 1997, London, PP. 67-81.

[12] **Muth, J.F.. Thompson**, **G.L..** *"Industrial Scheduling"*, Prentice Hall, Englewood Cliffs, New Jersey, Año 1963, PP 225-251.

[13] **Adams, J.. Balas, E.. Zawack, D..** *"The shifting bottleneck procedure for job shop scheduling"*, Management Science vol. 34, Año 1988, PP. 391-401.

[14] **Lawrence, S..** *"Resource constrained project scheduling: an experimental investigation of heuristic scheduling techniques"*, Unpublished Working Paper, Graduate School of Industrial Administration, Carnegie-Mellon University, Pittsburgh, Pennsylvania. Año 1984.

[15] **Taillard, E. D..** *"Benchmarks for basic scheduling problems"*, European Journal of Operational Research vol. 64, Año 1993, PP. 278-285.

[16] **Balas, Egon. Lancia, Giuseppe. Serafín, Paolo. Vazacopoulos, Alkiviadis.** *"Job Shop Scheduling With Deadlines"*, Journal of Combinatorial Optimization 1, Año 1998, PP 329-353.

[17] **Wellner, Jörg. Dilger, Werner**. *"Job Shop with Multiagents"*. 13th Workshop "Planen und Konfigurieren (PuK)", 03-05 März 1999

[18] **E. Nowicki, E.. Smutnicki, C..** *"Some new ideas in TS for job-shop scheduling"*. Metaheuristic Optimization via Memory and Evolution, Tabu Search and Scatter Search, Springer US, Año 2001, PP 165-190.

[19] **Watson, Jean Paul**. *"On Metaheuristic "Failure Modes": A Case Study in Tabu Search for Job-Shop Scheduling"*. MIC2005: The Sixth Metaheuristics International Conference. Año 2005.

[20] **Whitley, Darrel**. *"A genetic algorithm tutorial"*. Statistics and Computing 4, Año 1994, PP 65-85.

[21] **Urra Coloma, Enrique.** *"Estudio de Algoritmos Genéticos para el problema de transporte de pasajeros (DARPTW)"*. Informe Final Proyecto I. Año 2006.

[22] **Mitchell, M..** *"An Introduction to Genetic Algorithms"*. The MIT press, Cambridge, MA, ISBN: 978-0262631853, Año 1996.

[23] **Yamada, Takeshi. Nakano, Ryohei**. *"A Genetic Algorithm Applicable To Large-Scale Job-Shop Problems"*. Parallel Problem Solving from Nature, 2, Elsevier Science Publishers B.V.. Año 1992, PP 281-290.

[24] **Yamada, Takeshi. Nakano, Ryohei**. *"A Genetic Algorithm with Multi-Step Crossover for Jo-Shop Scheduling Problems"*. First IEE/IEEE International Conference on Genetic Algorithms in Engineering Systems: Innovations and Applications (GALESIA '95), 12-14 September 1995, Sheffield, UK, Año 1995, PP. 146-151.

[25] **Bierwirth, Christian**. *"A Generalized Permutation Approach to Job Shop Scheduling with Genetic Algorithms"*, OR Spektrum, vol 17, Año 1995, PP 87-92.

[26] **Bierwirth, Christian; Mattfeld, Dirk C.; Kopfer, Herbert**. *"On Permutation Representations for Scheduling Problems"*. Lecture Notes in Computer Science vol. 1141, Proceedings of the 4th International Conference on Parallel Problem Solving from Nature. Año 1996. PP 310-318

[27] **Moraglio, Alberto.** *"Genetic Local Search for Job Shop Scheduling Problem"*. Tesi di Laurea (Master Thesis), Politecnico Di Torino, Facoltà di Ingegneria, Torino, It. Año 2000.

[28] **Pezzella, Ferdinando. Merelli, Emanuela.** *"A tabu search method guided by shifting bottleneck for the job shop scheduling problem"*, European Journal of Operational Research vol.120, Año 2000, PP 297-310.

[29] **Geyik, Faruk. Hakki, Ismail.** *"The strategies and parameters of tabu search for job-shop scheduling"*, Journal of Intelligent Manufacturing vol. 15, PP. 439-448, 2004.

[30] **Schultz, Scott. Hodgson, Thom. King, Russell.** *"On solving the classic job shop makespan problem by minimizing L_{max}"*, Proceedings of the 2002 Industrial Engineering Research Conference, Año 2002.

[31] **Binato, S.. Hery, W.., Loewenstern, D.. Resende, M..**, *"A Grasp For Job Shop Scheduling"*, Essays and Surveys on Metaheuristics, Kluwer Academic Publishers, Año 2001, PP 59-79.

[32] **Ombuki, B.. Ventresca, M..** *"Local Search Genetic Algorithms for the Job Shop Scheduling Problem"*, Applied Intelligence vol. 21, Año 2004, PP 99-109.