UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO

FACULTAD DE INGENIERÍA

Algoritmo genético aplicado a sistemas de manufactura flexible

TESIS

Que para obtener el título de

Ingeniero Mecatrónico

P R E S E N T A

Edilberto Aguilar Ruiz

DIRECTOR DE TESIS

M.I. Armando Sánchez Guzmán

Ciudad Universitaria, Cd. Mx., 2016

AGRADECIMIENTOS

A mi madre Guadalupe, por su apoyo incondicional a lo largo de mi vida; quien siempre ha estado a mi lado en los buenos y malos momentos, y me ha alentado siempre a superarme profesionalmente así como en todos los aspectos de mi vida.

A mi padre Octavio, por su apoyo e interés en hacer de mí un hombre de principios y útil para la sociedad y quien siempre me enseñó a valorar las cosas.

A mis hermanos, por el apoyo y cariño brindado a lo largo de los años así como de los momentos que hemos pasado juntos.

A mis amigos y compañeros durante mi estancia en la universidad, por los buenos momentos, desvelos, estrés y todo lo que nuestra formación como ingenieros demandó.

Al Ingeniero Armando Sánchez por su paciencia, apoyo y consejos brindados durante el desarrollo de este trabajo de tesis.

A todos aquellas personas que contribuyeron de alguna forma en la finalización de este trabajo de tesis.

Agradecimiento a la Dirección General de Asuntos del Personal Académico (DGAPA) de la UNAM, por el financiamiento a través del Programa de Apoyo a Proyectos de Investigación e Innovación Tecnológico (PAPIIT) , en el proyecto IN118314, denominado DESARROLLO DE ALGORITMOS PARA LA OPTIMIZACIÓN DE PROCESOS DE MANUFACTURA.

ÍNDICE

LISTA DE ABREVIATURAS

SMF: Sistema de Manufactura Flexible

JS: Job Shop

JSSP: Job Shop Scheduling Problem

FS: Flow Shop

FSSP: Flow Shop Scheduling Problem

RdP: Redes de Petri

$C_{máx}$: makespan

P_m**:** Porcentaje de Mutación

P_c**:** Porcentaje de Cruce

L_p**:** Tamaño de la población

RS: Recocido Simulado

CdH: Colonia de Hormigas

BT: Búsqueda Tabú

AG: Algoritmo Genético

INTRODUCCIÓN

El aumento de la población trajo consigo un incremento en la demanda de productos ya sean de la industria de alimentos, automotriz, textil, de servicios entre otros. Por lo tanto las empresas se han visto en la necesidad de mejorar sus técnicas de fabricación tanto para incrementar la producción así como para reducir sus costos. Por eso a lo largo de los años muchas técnicas de programación de actividades se han desarrollado; como por ejemplo las reglas de despacho, que son un conjunto de reglas para determinar qué actividades deben ejecutarse en qué momento y en qué orden, y así poder optimizar tiempos de producción, minimizar desperdicios, mano de obra, etc.

El desarrollo de la tecnología ha logrado mejorar los sistemas de producción, visto desde diferentes puntos. Por ejemplo, la introducción de máquinas de control numérico (CNC), máquinas que presentan flexibilidad en las operaciones que pueden ejecutar, han hecho posible la fabricación de productos diferentes que sean de una misma familia en un mismo taller o fábrica, además de hacer posible la automatización de los procesos de manufactura, el cual trae como consecuencia un incremento en la producción y mejora en la calidad de los productos. El desarrollo de software que ayudan en la simulación de sistemas de manufactura hace posible y facilita la evaluación a bajo costo, de estrategias y programas de producción antes de ser implementadas. El desarrollo de técnicas de resolución de problemas para la programación y secuenciación de actividades desarrollado en los últimos años tiene mucho potencial para resolver problemas de optimización y planeación en sistemas de producción en la industria; técnicas desarrolladas en el área de las ciencias de la computación junto al desarrollo de la tecnología en hardware de computadoras prometen incrementar la producción así como los tiempos de entregas.

En esta tesis se abordará el problema de los sistemas de manufactura que presentan flexibilidad en los procesos, modelados con ambientes de manufactura tipos Job Shop y Flow Shop clásico. Se analizará la eficiencia de estos en sistemas de producción reales. En el capítulo 1 se introducen los algoritmos más aplicados para la resolución de problemas de optimización en ambientes de manufactura tipo Job Shop. En el capítulo 2 se plantea la estructura del algoritmo genético utilizado para resolver el problema del Job Shop y lograr minimizar el *makespan*. En el capítulo 3 se aplica el algoritmo desarrollado en el capítulo 2 a tres casos de sistemas de manufactura en cada uno de ellos se idealizan tiempos de procesamiento. Por último en el capítulo 4 se dan las conclusiones y análisis de los resultados obtenidos, así como se plantea los trabajos a futuro y la recomendaciones para mejorar los resultados obtenidos.

JUSTIFICACIÓN

La programación de producción tiene como objetivos incrementar la productividad, mejorar los tiempos de fabricación y reducir los costos. Para lograrlo debe hacer uso de los todos recursos disponibles en el cual se debe decidir sobre como asignar estos a todas las tareas existentes además se debe encontrar una secuencia que optimice alguna variable de interés. Una buena programación puede marcar la diferencia entre poder tener el dominio del mercado y no tenerlo. Sin embargo conseguir una buena programación no es una tarea sencilla, y este dependerá en gran medida de la configuración del sistema de manufactura, del número de recursos disponibles, la cantidad que se desee fabricar, tiempos de procesamiento, tiempos de preparación de máquinas, disponibilidad de recursos, restricciones en los procesos de fabricación, prioridades y tiempos de entrega. Así la programación de producción es una tarea muy compleja y casi imposible para ser realizada por un humano. De ahí la gran importancia del desarrollo de la tecnología y ceder la tarea a las máquinas que son capaces de ejecutar cálculos a gran velocidad para que resuelvan los problemas de programación de actividades.

Existe la necesidad del desarrollo de software para la optimización de sistemas de producción, software como ProModel que ayudan a evaluar estrategias y programas de producción pero que no resuelven en si el problema de optimización sino simplemente ayudan a conocer el resultado de una propuesta y este sigue dependiendo en gran medida de la experiencia y habilidad del programador.

En este trabajo se pretende dar solución al problema de programación de producción, haciendo uso de una técnica desarrollada en el área de la inteligencia artificial para la resolución de problemas complejos computacionalmente hablando. Se utiliza un algoritmo genético clásico para encontrar soluciones óptimas a modelos de sistemas de manufactura idealizados pero que intenta ajustarse en gran medida a la realidad. No se hará un análisis de la complejidad computacional que los problemas de este tipo representan, lo principal es modelar y dar solución a problemas de sistemas de producción que puedan ser aplicados posteriormente en sistemas reales.

OBJETIVO

Aplicar un algoritmo genético a sistemas de manufactura flexible para encontrar una secuencia óptima de operaciones que minimice el *makespan* (tiempo de terminación del último trabajo).

CAPÍTULO 1
Algoritmos aplicados al *Job Shop Scheduling Problem*

1.1 Introducción

Un sistema de manufactura se considera como una fábrica donde los materiales que se manejan están compuestos principalmente de entidades discretas, por ejemplo productos que son maquinados y/o ensamblados en un conjunto de recursos disponibles como los son maquinaria o estaciones de trabajo, por lo tanto los sistemas de manufactura se consideran sistemas de producción discretos [3].

Antiguamente el término "línea de producción" hacía referencia a la producción en masa. La producción en masa concebida por Henry Ford, fue un sistema incapaz de tratar con variaciones en los tipos de productos. Esta rigidez llegó a ser un obstáculo con la llegada de la competitividad en el mundo de la producción. La principal desventaja de este tipo de sistema es que únicamente producían un tipo de producto para una configuración de hardware establecido, es decir, una disposición de máquinas, líneas de transporte de material, etc. En este tipo de producción se tenían obreros y máquinas que realizaban operaciones de manera repetitiva una y otra vez. Pero la demanda en el mercado forzó a la introducción por primera vez de una nueva forma de producción, sistemas más versátiles ante las variaciones en los tipos productos y los tiempos de entrega. Así se introdujeron nuevos sistemas de manufactura que tenía la capacidad de producir diferentes tipos de productos de una misma familia utilizando una misma configuración de hardware, es decir, los sistemas de producción ahora presentaban flexibilidad. Así nacieron los *sistemas de manufactura flexible*.

1.2 Sistemas de Manufactura Flexible (SMF)

Un Sistema de Manufactura Flexible (SMF) es un sistema de producción en el que existe cierta flexibilidad que le permite reaccionar en el caso de cambios previstos e imprevistos. Un SMF intenta conseguir la eficiencia de una línea de producción que es capaz de producir pocos lotes en grandes cantidades y la versatilidad de un sistema que sea capaz de manejar una gran variedad de productos diferentes a bajo costo y en un tiempo dado. En este tipo de sistema distintos productos pueden ser procesados simultáneamente con una única configuración de hardware del sistema. Tal flexibilidad tiene un gran potencial para mejorar e incrementar la producción [18].

Un SMF es un sistema de producción que consta de distintas estaciones de trabajo unidos

a un sistema para manejo de material capaz de habilitar tareas que puedan seguir distintas rutas a través del sistema, monitoreado y controlado por una red enlazada de computadoras, microprocesadores, dispositivos de adquisición de datos y Controladores de Lógica Programable (PLC's). Los principales elementos de un SMF son [3]:

(i) Un conjunto de máquinas flexibles, como son: máquinas CNC, máquinas de medición, máquinas de corte, máquinas de lavado, etc.

(ii) Un sistema de trasporte automático, como por ejemplo: material de trasporte y manejo de equipo, tal como un vehículo automático guiado junto a una estación de carga y descarga, almacén central de material, y almacén local de material dedicado a una máquina de manufactura individual.

(iii) Un sofisticado centro de control computarizado, para decidir en cada instancia que debe ser realizado y en que máquina.

1.2.1 Control de la producción

En una fábrica, muchas actividades deben realizarse para producir productos correctos para el cliente correcto en cantidades correctas en el tiempo correcto [1]. Muchas de estas actividades están enfocadas a la producción física y manipulación de los productos. Otras actividades hacen referencia al manejo y control de las actividades físicas. El manejo de la producción y el control de actividades pueden ser clasificados en tres diferentes jerarquías; estratégica, táctica y actividades operacionales [1], dependiendo de la naturaleza de la tarea.

Manejo estratégica de la producción debe ser capaz de determinar los productos que deben diseñarse y fabricarse según la demanda del mercado y expectativas de los clientes.

Manejo táctica de la producción debe generar los planes detallados según lo demandado por la gerencia de producción. Esto dependerá de los criterios de producción, digamos que se quiere anticipar la demanda del mercado pueden usarse estadísticas de temporadas pasadas para conseguirlo, esto nos lleva a una producción de tipo "fabricar para almacenar". Por otra parte se puede fabricar según la demanda real impuesta por los clientes, es decir una producción de tipo "fabricar por pedido".

Manejo operacional de la producción debe realizarse en tiempo quasi-real del sistema de manufactura. Esto involucra la manipulación de las máquinas apropiadas, trabajadores y otros recursos de una forma coordinada. Para así seleccionar los recursos apropiados para cada tarea y la secuencia apropiada de estas tareas en cada recurso.

Control de la producción se refiere al manejo y control de decisiones (*Scheduling*) a corto plazo únicamente en el nivel operacional.

1.2.2 Tipos de Scheduling en sistemas de manufactura

Se define el problema de *Scheduling* como el proceso de asignar recursos o actividades a lo largo del tiempo [4] o como la determinación de cuándo las operaciones o acciones que componen un proceso deben ser realizados haciendo uso de los recursos. Comúnmente el término *Scheduling* en sistemas de producción hace referencia a la secuenciación de operaciones y tareas aprovechando los recursos disponibles (máquinas o estaciones de trabajo). En otras palabras es un proceso de *optimización*. Y debe garantizar que los tiempos asignados a dichas acciones deben cumplir con una serie de restricciones establecidas en el proceso, así como la optimización de ciertos criterios de interés [5].

Antes de intentar resolver cualquier problema de *Scheduling* es importante primero conocer la distribución o disposición de los recursos disponibles dentro del sistema de manufactura que se desea tratar, además del proceso de fabricación requerido para cada producto el cual definirá el flujo que cada uno de estos deberá seguir a través del sistema. En los problemas de *Scheduling*, la capacidad que tiene cada recurso de poder procesar cierto número de productos es limitada y está definida sobre cierto número de intervalos temporales. Por lo tanto el problema consiste en cubrir la demanda de recursos (sin exceder sus capacidades disponibles) por parte de los productos a lo largo de sus procesos de manufactura. En este tipo de problemas se conoce a priori que recurso va a ser utilizado en cada producto.

Los sistemas de manufactura y subsistemas son comúnmente categorizados según su volumen de producción y flexibilidad. Muchas veces es necesario sacrificar una por la otra, sistemas que presentan poca flexibilidad presentas altos volúmenes de producción y viceversa. Así se da la necesidad de buscar nuevas alternativas que nos eviten el sacrificio de una por la otra, así nuevos enfoques están surgiendo en los sistemas de producción como se verá más adelante.

Dependiendo del uso de los recursos por parte de las actividades y los patrones de flujo permitidos, los sistemas pueden clasificarse principalmente en cinco tipos según su *Scheduling*:

i. Producción con máquina única (*Single Machine Model*)

Es la más simple de los sistemas de producción. Consiste de una única máquina que realiza todas las operaciones de todos los productos una a la vez. En este modelo únicamente se busca en qué orden deben ser procesados cada uno de los productos.

ii. Producción con máquinas en paralelo (*Identical Processors Model*)

Es similar al modelo con máquina única, pero consiste de múltiples e idénticas máquinas en paralelo que realizan procesos idénticos. Cada producto puede ser procesado en

cualquiera de estas máquinas, y una vez procesado se considera como producto terminado.

iii. Flow Shop

Consiste de diferentes máquinas de procesamiento de productos, donde:

- Cada operación debe ser ejecutada en una máquina en específico.
- todas las operaciones de un producto tiene que ser ejecutado en una misma secuencia fija.
- todos los productos tienen que visitar las máquinas en la misma secuencia.

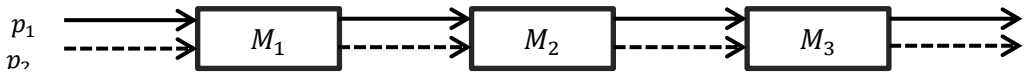


Figura 1.1 Diagrama de bloques de un sistema de manufactura tipo Flow Shop Clásico con 3 máquinas y 2 productos. Fuente [1]

El Flow Shop se puede considerar como un sistema que está estandarizado hasta cierto punto, ya que la línea de producción se mantiene relativamente estable. En este tipo de sistema se producen grandes lotes pero poca variedad de productos diferentes, esto hace que la producción de dichos productos requiera de maquinaria más especializa así como de trabajadores más especializados. En la *Figura 1.1* se muestra un ejemplo del diagrama de bloques así como el flujo que sigue cada producto a lo largo de las estaciones de trabajo en este tipo de sistema.

Figura 1.2 Línea de ensamble automotriz ejemplo práctico de un sistema tipo Flow Shop. Fuente [1].

El Flow Shop clásico es un sistema donde todos sus productos siguen una misma ruta a lo largo de las etapas o estaciones de trabajo y cada una de estas estaciones consta de una sola máquina. A diferencia del Flow Shop clásico el Flow Shop Flexible o Hibrido dispone de múltiples máquinas por estación la cual le da mayor flexibilidad al sistema sin sacrificar volumen de producción.

El principal ejemplo de un sistema tipo Flow Shop es aplicado a la industria automotriz (*ver Figura 1.2*) donde cada modelo a fabricar sigue la misma ruta a lo largo de la línea de ensamble independientemente de que algún modelo se salte alguna operación.

iv. Job Shop

Es igual que el Flow Shop, excepto que aquí cada pedido o producto a fabricar puede visitar cada máquina o estación de trabajo en una secuencia diferente (*ver Figura 1.3*). Este tipo de sistema presenta mayor flexibilidad que el Flow Shop y es capaz de fabricar una gran variedad de productos en pequeños lotes. Como consecuencia no se necesita maquinara muy especializada. En la *Figura 1.3* se muestra un ejemplo de diagrama de bloques así como el flujo que cada producto debe seguir a lo largo de las estaciones de trabajo en este tipo de sistema.

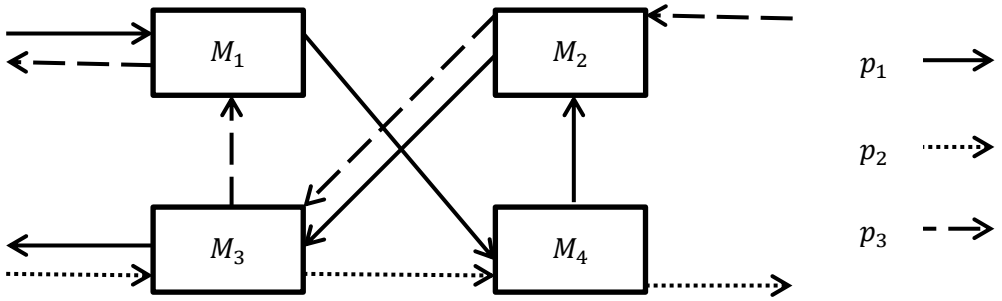


Figura 1.3 Diagrama de bloques de un ejemplo de sistema tipo Job Shop con 4 máquinas y 3 productos. Fuente [1].

En el Job Shop clásico cada estación de trabajo está compuesta por una sola máquina. Siendo el problema que se abordará en este trabajo. Y es computacionalmente el de mayor complejidad además de que en este tipo de sistema existen soluciones no factibles que deben ser desechadas. A diferencia del Job Shop Clásico el Job Shop Flexible presenta la modificación de que las operaciones de cada producto pueden ser procesados en diferentes máquinas.

v. Producción estática (*Fixed-Position Shop*)

Es un sistema de producción donde los procesos son ejecutados alrededor del producto

que no se mueve dentro del taller. Es un sistema con poca flexibilidad y poco volumen de producción. Una aplicación de este sistema es en la industria aeronáutica como se muestra en la *Figura 1.4.*

Figura 1.4 Un sistema de producción aeronáutico. Ejemplo de un sistema tipo Fixed-Position Shop Fuente [1].

1.2.3 Medidas de desempeño

Las medidas de desempeño son parámetros bajo el cual se puede determinar que tan bien opera nuestro sistema de manufactura, es decir, son métricas de efectividad. Al ser una métrica de efectividad se les consideran como funciones objetivos en nuestro problema de *Scheduling*, es decir, funciones que nos interesan evaluar y optimizar. Así podemos decir que tan bien o mal es nuestro *Scheduling*.

A continuación se enlistas las principales y más comunes medidas de desempeño:

1) ***makespan* (C_{max}):** Tiempo en que se completa la última operación en el proceso de fabricación del último producto que sale del sistema, es decir, es el tiempo para terminar de fabricar todos los productos.
2) **Tardanza Total (*T*):** La tardanza es la diferencia positiva entre el tiempo de terminación y la fecha de entrega de un pedido. La tardanza total es la sumatoria de las tardanzas de todos los pedidos.
3) **Tiempo de flujo ponderado:** Tiempo promedio en completar todos los pedidos en el sistema.
4) **Número de Trabajos tardíos:** es el número de pedidos retrasados a la fecha de entrega.

5) **Número de Trabajos tardíos ponderados:** Se da cuando los pedidos no tiene la misma importancia, por lo tanto, es necesario asignar un peso determinado a cada uno de ellos, así será posible minimizar el peso total de los pedidos tardíos dando tal importancia dependiendo del peso asignado.
6) **Tardanza máxima (T_{max}):** Es la máxima tardanza o peor violación a la restricción de fecha de terminación. Este corresponde al trabajo más tardío.

De las medidas de desempeño mencionadas, el *makespan* es el más abordado por los investigadores debido a su importancia. Y es el que se tratará en este trabajo.

1.3 El Job Shop Scheduling Problem (JSSP)

En un sistema tipo Job Shop no existe la noción de flujo de producción, para cada producto una ruta de producción debe ser definida. Dicha ruta describe una secuencia de operaciones en máquinas que no tienen restricciones físicas de disposición. Tales sistemas pueden manejar cualquier número de familia de productos y están limitadas sólo por el conjunto de operaciones que las máquinas puedan realizar. Es el tipo de sistema más versátil y flexible. Se considera el caso general de los tipos clásicos de *Scheduling* en un sistema de manufactura.

El Job Shop Scheduling Problem es un problema de optimización combinatoria y consiste en asignar un número determinado de actividades en un número finito de recursos disponibles. El objetivo de optimizar este problema es minimizar el tiempo de terminación de todas las tareas en conjunto.

El Job Shop Scheduling Problem consiste en planificar un conjunto de *N* productos o pedidos $\{p_1, \dots, p_j\}$ sobre un conjunto de *M* máquinas o estaciones de trabajo $\{m_1, \dots, m_i\}$. Donde cada producto p_j consta de un conjunto de operaciones $\{\theta_{j1}, \dots, \theta_{ji}\}$ que deben ser ejecutadas de forma secuencial, es decir, la operación $\theta_{j,(i+1)}$ no puede comenzar hasta que la operación θ_{ji} haya terminado completamente. Cada operación θ_{ji} tiene asociado un tiempo de procesamiento sin interrupción de t_{ji} unidades de tiempo durante el cual se requiere del uso exclusivo de una máquina. Cada producto tiene un tiempo de inicio y un tiempo de terminación. En el JSSP clásico existen algunas restricciones en el problema y se enlistan a continuación:

- Un producto no puede visitar una misma máquina dos veces.
- Las operaciones no se pueden interrumpir.
- Cada máquina puede procesar un sólo producto a la vez.
- Cada producto es una entidad, por lo tanto, no pueden procesarse dos

operaciones de un mismo producto simultáneamente.

- Son conocidos y fijos todos los datos que intervienen: número de productos a fabricar, número de máquinas disponibles, tiempos de procesamiento, etc.
- Cada producto incluye una y sólo una operación en cada máquina, por lo que todos los productos contienen una cantidad de operaciones no mayor al número de máquinas.
- Todas las operaciones tendrán la misma prioridad de procesamiento.

1.4 Representación gráfica

Para poder resolver el JSSP, como todo problema, primero hay que tener una forma de representación abstracta. Para el caso del JSSP existen comúnmente dos formas de hacerlo, que son por grafo disyuntivo [15] y redes de Petri [6] y cada solución factible puede hacerse utilizando los diagramas de Gantt.

1.4.1 Grafo Disyuntivo

Es el modelo de representación más utilizado para los problemas de *Scheduling* en ambientes de manufactura, introducido por Roy and Susmann en 1964. Un grafo disyuntivo es un conjunto $G = (V, C \cup D)$, donde:

- V es el conjunto de vértices, con dos nodos principales uno de inicio o fuente del cual salen las primeras operaciones en la secuencia y otro final o destino al cual llegan todas las operaciones terminales. En el JSSP cada nodo de *V* representa la operación de un producto en máquina del sistema (Producto p_j, máquina m_i), es decir, la operación del producto p_i en la máquina m_i. El nodo de inicio se conecta con la primera operación de cada producto, y de igual forma la última operación se conecta con el nodo final.
- C es el conjunto de arcos conjuntivos, en el JSSP representan las rutas que debe seguir cada producto o secuencia de operaciones a lo largo de las máquinas o recursos, a esta secuencia se le conoce como restricción de precedencia o de proceso, es decir, si un arco $(j,i) \rightarrow (j,i+1)$ que pertenecen a *C* entonces el producto p_j tiene que tiene que ser procesado en la máquina m_i antes de ser procesado en la máquina m_{i+1}. Cada arco $(j,i) \rightarrow (j,i+1)$ tiene un tamaño $|(j,i) \rightarrow (j,i+1)| = t_{ji}$ y representa la restricción de que la operación $(j,i+1)$ solo puede iniciar t_{ij} unidades de tiempo después de haber iniciado la operación (j,i)
- D es el conjunto de arcos disyuntivos, los cuales se representan mediante arcos en sentidos opuestos, sirven para conectar a las operaciones de productos diferentes

pero que se realizan en la misma máquina, es decir, nos dictan todas posibles combinaciones secuenciales que los productos pueden seguir en cada máquina.

Por ejemplo para una instancia de 4x3 (4 productos y 3 máquinas) y con las restricciones de proceso y tiempos de procesamiento de la *tabla1.1 y tabla1.2* se tiene el siguiente grafo disyuntivo (*Ver Figura 1.6)*

t_{ji}	*máquina 1*	*máquina 2*	*máquina 3*
producto 1	5	8	2
producto 2	3	9	7
producto 3	1	10	7
producto 4	7	4	11

Tabla1.1 Ejemplo de tiempos de procesamiento para una instancia de 4 productos y 3 máquinas

m_i	*Operación 1*	*Operación 2*	*Operación 3*
producto 1	m_1	m_2	m_3
producto 2	m_3	m_1	m_2
producto 3	m_1	m_3	m_2
producto 4	m_2	m_3	m_1

Tabla1.2 Ejemplo de secuencia de proceso para una instancia de 4 productos y 3 máquinas

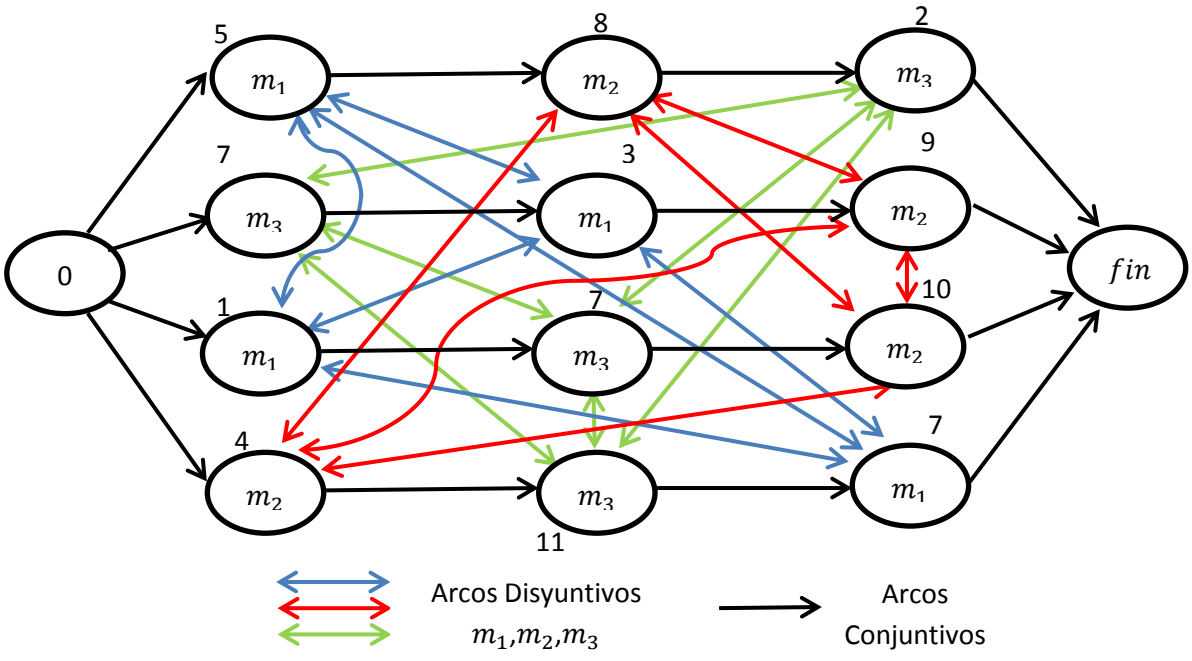


Figura 1.6 Representación por grafo disyuntivo para una instancia de 4x3 del JSSP. Fuente [propia del autor]

1.4.2 Redes de Petri

Las redes de Petri (RdP) fueron introducidas por Carl Adam Petri en 1962 como una nueva herramienta de modelado para sistemas de eventos discretos. Las RdP son un caso especial de grafo dirigido bipartido, compuesto por dos tipos de nodos denominados *places* (plazas o lugares) que son representadas mediante circunferencias y *transitions* (transiciones) que son representados mediante barras. Comúnmente las plazas simbolizan lugares físicos, condiciones o acciones; mientras las transiciones eventos. Las plazas y transiciones se conectan mediantes arcos dirigidos. Dentro de las plazas residen entidades llamadas *tokens*, representadas comúnmente con puntos negros o de colores (según sea el tipo de RdP), donde estos dictan el estado del sistema.

Formalmente las redes de Petri son una estructura [16] $RdP = (P, T, I, O, M_0)$ donde:

- $P = \{p_1, p_2, \dots, p_m\}$ es el conjunto finito de lugares (*places*)
- $T = \{t_1, t_2, \dots, t_n\}$ es el conjunto finito de transiciones (*transitions*), $P \cup T \neq \emptyset$ y $P \cap T = \emptyset$
- $I = (P \times T) \rightarrow N$ es una función de entrada que define los arcos dirigidos que van de los lugares a las transiciones, donde *N* es un conjunto de enteros no negativos.
- $O = (T \times P) \rightarrow N$ es una función de salida que define los arcos dirigidos que van de los transiciones a los lugares.
- $M_0 : P \rightarrow N$ es el marcaje inicial.

Para poder cambiar el estado de la Red, es decir, la distribución de los tokens en las plazas, es necesario que ocurran eventos, es decir, que alguna transición se dispare. Para que una transición pueda dispararse primero debe estar habilitada. Para que esto ocurra es necesario que ciertas condiciones se cumplan.

Representación gráfica de los elementos de una Red de Petri:


Lugares o Plazas (Places)


Transiciones (Transitions)


Arcos (Arcos)


Marcas (Tokens)

- Regla de habilitación de transiciones

Una transición *t* se dice que está habilitado si cada lugar de entrada *p* contenga al menos

el número de tokens igual al peso del arco dirigido que conecta *p* a *t*.

- Reglas de disparo de transiciones

a) Una transición habilitada *t* puede o no dispararse.
b) Un disparo de una transición habilitada *t* remueve de cada lugar de entrada el número de tokens igual al peso del arco dirigido que conecta *p* a *t*. depositando en cada lugar de salida *p* el número de tokens igual al peso del arco dirigido que conecta *t* a *p*.

1.4.3 Diagramas de Gantt

Los diagramas de Gantt son una herramienta en la programación de actividades propuesta por Henry Gantt en 1917. Son un tipo de gráfica de barras que relaciona las actividades o tareas respecto al tiempo, por lo tanto da detalles en la secuencia de cada actividad así como el tiempo que cada uno de estos involucra. En el diagrama de Gantt el tiempo se ubica en el eje horizontal y los recursos disponibles se ubican en el eje vertical. La gran ventaja de estos diagramas es que proporcionan una representación visual que facilita la comprensión de la secuencia de actividades y el cálculo de los tiempos que cada actividad toma en completarse. Para el caso de JSSP facilita la representación de cada solución factible encontrada.

1.5 Complejidad del JSSP

Un algoritmo es un conjunto ordenado y finito de instrucciones, pasos u operaciones que permiten realizar una tarea o resolver un problema. Un algoritmo ejecuta sus operaciones sin ambigüedades, y cuya ejecución ofrece una solución a un problema.

El trabajar con algoritmos implica tener una forma de medir y saber la eficiencia de esté, así es necesario introducir el término complejidad. En un algoritmo la cantidad de recursos computacionales necesarios para su ejecución determina si un algoritmo se puede considerar fácil o difícil desde el punto de vista de complejidad.

La complejidad computacional estudia todos los recursos requeridos durante el procesamiento de un algoritmo. Los parámetros más analizados son:

- Tiempo: relacionado con el número y tipo de pasos para la ejecución del algoritmo; parámetros como la velocidad de procesamiento intervienen.
- Espacio: relacionado con la cantidad de memoria utilizada para la ejecución del algoritmo.

Así según su complejidad un algoritmo puede clasificarse en diferentes clases que se

enlistan a continuación:

- *Clase P:*

  Dado un problema decimos que pertenece a la **clase de complejidad P**, si existe un algoritmo que resuelve cualquier ejemplo de su problema en tiempo polinomial. Los algoritmos de este tipo se dicen que son tratables en el sentido de que se pueden abordar fácilmente en la práctica.

- *Clase NP:*

  Dado un problema decimos que pertenece a la **clase de complejidad NP**, si existe un algoritmo que puede verificar cualquier solución a un ejemplo de esta clase en tiempo polinomial. Este tipo de problemas son intratables, esto conduce a un método de resolución no determinístico, que consiste en aplicar una heurística para obtener soluciones hipotéticas que se van aceptando a ritmo polinómico.

- *Clase NP-Complete:*

  Sea P' un problema en la clase NP. Entonces P' es **NP-complete** si cualquier problema en clase NP se puede reducir a P' en tiempo polinomial. Este tipo de problemas se caracterizan por ser todos iguales en el sentido de que si se descubriera una solución Para alguno de ellos, esta solución sería fácilmente aplicable a todos ellos. Una alternativa para la solución de este tipo de algoritmos es el uso de algoritmos heurísticos que obtiene soluciones óptimas.

- *Clase NP-Hard:*

  Un problema es **NP-Hard** si cualquier problema es clase NP se puede reducir a él en tiempo polinomial. En esta clase no es posible encontrar una solución mediante búsqueda exhaustiva, como los algoritmos heurísticos.

El JSSP es considerado como un problema de complejidad clase ***NP-Hard***, de ahí la dificultad de encontrar una solución óptima. Muchos algoritmos han sido propuestos para la resolución de este tipo de problemas. Para tener una idea de lo complejo que es el JSSP, si se deseará dar todas las soluciones que puedan ser generadas para una instancia en particular, y si se sabe que cada secuencia de operaciones puede ser permutada independientemente, se tendría $(p!)^m$ donde p denota el número de productos y m el número de máquinas. Por ejemplo si se tiene un total de 24 productos y una sola máquina, entonces se tiene $(24!)\ (24!) = 6.20 * 10^{23}$ posibles soluciones. Si la edad del universo es de 20 mil millones de años que corresponde a $6.31 * 10^{17}$ segundos

aproximadamente $6.31 * 10^{23} \mu Seg$. Si procesáramos una solución cada μSeg apenas y se podría resolver este problema por enumeración. De ahí la necesidad de aplicar otras técnicas para resolver este problema y no tener que explorar todo el espacio de soluciones.

1.6 Algoritmos aplicados la resolución del JSSP

Debido a que el problema de *Scheduling* para el Job Shop es muy complejo (*Complejidad NP-Hard*), muchos algoritmos han sido propuestos para su solución en tiempo de ejecuciones aceptables y soluciones óptimas o casi óptimas. Los enfoques propuestos van desde soluciones exactas, aquellas que exploran todo el espacio de estado pero con el costo de tener tiempos de ejecución elevados; hasta los aproximados, también conocidos como meta-heurísticas que obtienen soluciones óptimas o casi óptimas, con tiempos de ejecución aceptable. El desarrollo y aplicación de algoritmos meta-heurísticos es una rama de la inteligencia artificial que trata de encontrar soluciones a problemas complejos.

Dentro de los algoritmos propuestos, a continuación se describen algunas de las meta-heurísticas más utilizados y los que mejores resultados han ofrecidos.

1.6.1 Ramificación y Poda (*Branch and Bound*)

Las técnicas de ramificación y poda hacen uso de una estructura de árbol construida dinámicamente como forma de representar el espacio de todas las secuencias posibles. El nombre proviene del uso de dicho árbol donde se ramifica para examinar subregiones de la región factible y cotas para mantener el crecimiento del árbol bajo control. La búsqueda comienza en el nodo raíz y se continúa hasta llegar a un nodo hoja. Cada nodo en un nivel *p* de la búsqueda representa una secuencia parcial de *p* operaciones. Desde un nodo seleccionado, la operación de ramificación determina el siguiente conjunto de posibles nodos a partir del cual puede progresar la búsqueda. El procedimiento de poda selecciona la operación con la que continuará la búsqueda y se basa en una estimación de una cota inferior y la mejor cota superior alcanzada hasta el momento. En los métodos de ramificación y poda es esencial afinar las cotas, ya que eso prevé la necesidad de buscar en secciones grandes del espacio de soluciones.

Brucker y colaboradores en 1994 aplican el algoritmo de Braunch and Bound para resolver el JSSP [32] dando como resultado ser uno de los mejores algoritmos para resolver el JSSP utilizando como medida de desempeño el *makespan*. Su característica principal es que emplea un esquema de ramificación que permite fijar arcos en la misma dirección u opuesta a la que tienen en el bloque critico de una solución factible.

Hariri y Potts en 1994 [33] también aplican este algoritmo al Job Shop e intenta minimizar

el máximo tiempo de terminación de tareas.

Arigues and Feillet en 2007 [37] aplican el algoritmo Branch and Bound al problema del JSSP con secuencias dependientes del tiempo preparación. El objetivo fue minimizar el *makespan* de forma óptima.

1.6.2 Recocido Simulado (*Simulated Annealing*)

El algoritmo se basa en principios termodinámicos y en el proceso de recocido del acero. El fenómeno físico de someter un metal a altas temperaturas por un periodo largo para luego enfriarlo lenta y uniformemente. A altas temperaturas se produce un estado de alta energía el cual provoca que las partículas tomen una configuración aleatoriamente, pero al ser enfriado provoca que las partículas tomen una configuración distinta, hasta llegar a una configuración que corresponde al de menor energía remanente. Desde el punto de vista de optimización el estado final del metal constituye la solución óptima al objetivo de minimizar la energía remanente. Con esto Kirkipatrick, Gelatt y Vecchi en 1983 [28] hace la analogía entre el recocido y la búsqueda de una solución óptima.

La idea básica se parte de una solución inicial y se selecciona al azar una solución vecina. Si la solución vecina es mejor, se adopta como nueva solución. Si no lo es, se acota como la nueva solución con una probabilidad que decrece conforme avanza el algoritmo. La idea de aceptar con cierta probabilidad soluciones de menor calidad le permite al algoritmo salir de óptimos locales, así como en el proceso físico donde a medida que desciende la temperatura es más difícil que un átomo se mueva a una posición de menor energía.

Rui Zhang en 2013 [35] utiliza el algoritmo de Recocido Simulado para minimizar el retraso máximo. Usa la teoría de propagación de restricciones para derivar la orientación de una porción de arcos disyuntivos. Después usa el algoritmo SA para encontrar una política de descomposición que satisface el máximo número de arcos disyuntivos orientados.

1.6.3 Búsqueda Tabú (*Taboo Search*)

Es un algoritmo meta-heurístico desarrollado por Glover en 1985 [17] su gran contribución radica en que mientras las otras meta heurísticas guardan información sobre mejor solución encontrada, la búsqueda tabú mantiene solo la información sobre las ultimas soluciones visitadas con el objetivo de servir como guía en la búsqueda y evitar que el algoritmo se mueva a soluciones visitadas recientemente.

En cada iteración el algoritmo explora el vecindario de la mejor solución encontrada. Se elige como mejor solución a la mejor solución encontrada en el vecindario, aún si esta no es mejor solución que la actual, esto se realiza con el objetivo de salir de óptimos locales. Para evitar entrar en ciclos, el algoritmo almacena la información relativa a las soluciones

en una lista llamada *lista tabú*.

La lista tabú es una lista en el cual se almacena sólo la información de los movimientos recientes por un lapso denominado *tiempo de permanencia*. La lista tabú es considerada como una tipo de memoria a corto plazo.

Además de la lista tabú, se consideran dos tipos más de memoria, que son a media y largo plazo. El mecanismo de mediano plazo consiste en favorecer la búsqueda en regiones donde se ha encontrado buenas soluciones. Este proceso es conocido como de *intensificación* y se lleva a cabo dando alta prioridad a soluciones similares a la actual y penalizando las más alejadas a ella. El mecanismo de memoria a largo plazo se ejecuta una vez terminado proceso de intensificación, iniciando un nuevo proceso conocido como diversificación que explora regiones nuevas en la búsqueda que favorece en la búsqueda de soluciones con características diferentes a la solución actual. Los procesos de intensificación y diversificación se incorporan en la función objetivo con pesos que se modifican durante el desarrollo del algoritmo, de tal manera que ambos procesos se alternen durante la búsqueda.

La búsqueda tabú ha sido aplicado al JSSP en Taillard en 1993 [29] el cual es comparado con el algoritmo shifting bottleneck y con uno de Simulated annealing, obteniendo mejores resultados en general que los primeros.

Dell' Amico y Trubia en 1993 [27] aplican la búsqueda tabú para minimizar el *makespan* al JSSP. En el algoritmo que proponen usan reglas de prioridad para generar la solución inicial. La selección de los vecinos se realiza mediante las estimaciones del *makespan*.

Nowicki y Smutnicki en 1996 [8] proponen un algoritmo llamado TSAB, para minimizar la tardanza del *makespan* en el JSSP clásico. Nuevamente Nowicki y Smutnicki en 2005 [9] utilizan su algoritmo haciendo muchas mejoras y renombrándolos i-TSAB. Una de las mejoras es la evolución de los vecinos, para la que detallan y enuncian las propiedades de un algoritmo que es capaz de calcular el *makespan* exacto de cada uno de ellos con un costo computacional relativamente reducido.

Bilge y colaboradores en 2004 [11] proponen un algoritmo de búsqueda tabú para minimizar la tardanza total en un problema de máquinas paralelas y tiempos de configuración dependientes de la secuencia.

Stecco y Cordeau en 2009 [10] proponen un método de búsqueda tabú para la minimización del *makespan* en un problema de una única máquina con tiempos de configuración de dos tipos, los que dependen del tiempo y los que depende de la secuencia. Comparan su método con el Branch and Bound y obtienen mejores resultados en el tiempo de ejecución.

1.6.4 Colonia de Hormigas (Ant Colony Optimizations)

Los algoritmos CdH (*Colonia de Hormigas*) algoritmos meta-heurístico inspirados en el comportamiento de colonias de hormigas reales. Fue propuesta por M. Dorigo en 1992 [19] para la solución de problemas combinatorias como el problema de agente viajero (TSP, por sus siglas en inglés). El proceso imita la forma en como estos insectos encuentran la ruta más corta entre la fuente de alimentos y el hormiguero. En 2008 Lin, Lu, Shy and Tsai [31] explica que cuando las hormigas buscan alimento, dejan cierto compuesto químico, denominado como "*feromona*" en su trayecto. Así cuantas más hormigas caminen a través de un sendero, más feromonas quedaran en el suelo. Debido a que la siguiente hormiga elegirá uno de los caminos con probabilidad proporcional a la mayor cantidad de feromona presente en el camino, al final este proceso de retroalimentación creará una ruta única desde el hormiguero a la fuente de alimentos. Pocos trabajos han aplicado esta meta-heurística para problemas de producción, aunque muestra resultados muy prometedores.

Omkumar y Shahabudeen [34] aplican el algoritmo de colonia de hormigas al Job Shop Scheduling Problem y es comparado con reglas de despacho.

Jun Zhang y colaboradores [36] implementan un sistema de Colonia de Hormigas y los resultados experimentales se compararon con las técnicas de optimización tradicionales además de que fueron implementados en un pequeño Job Shop.

1.6.5 Algoritmos Genéticos (Genetic Algorithm)

Los algoritmos genéticos introducidos por Holland en 1975, son algoritmos meta-heurísticos y están inspirados el proceso biológico de la selección natural. Los algoritmos genéticos imitan el proceso evolutivo, con el supuesto de que los individuos con ciertas características son aptos para sobrevivir y transmitir esas características a sus descendientes. Imitando este proceso se puede encontrar una solución que cada vez mejor que pueda satisfacer una función objetivo. Los algoritmos genéticos operan sobre poblaciones o conjunto de soluciones representadas en cadenas binarias, números enteros o decimales llamadas cromosomas. Durante su ejecución un algoritmo genético, cruza individuos de mejor aptitud para renovar la población y eliminar los de menor aptitud. Con este proceso iterativo se llega a una solución, que está representada por el cromosoma de mejor aptitud.

Algunos elementos de los algoritmos genéticos se enlistan:

- Cromosoma: cadena binaria, entera o decimal que representa un individuo solución, donde cada elemento en la cadena se conoce como gen.
- Población: Conjunto finito de cromosomas.

- Aptitud: Criterio que evalúa la calidad de un cromosoma. A mayor aptitud, mejor solución y mayor probabilidad de supervivencia para poder transmitir sus características a sus descendencia.

Los operadores más comunes en el proceso son:

- Selección: Consiste en seleccionar a ciertos individuos según una función aptitud.
- Cruce: operación por medio del cual se producen nuevos individuos, a partir de dos cromosomas padres elegidos aleatoriamente.
- Mutación: Operación por medio del cual se selecciona uno o más genes del cromosoma para ser cambiados.

Varios autores han propuesto el uso de algoritmos genéticos para resolver tanto el Job Shop Scheduling Problem así como el Flow Shop Scheduling Problem.

Ho y Tay en 2005 [13] estudian el problema del Job Shop Flexible utilizando reglas de despacho para generar la solución inicial como una nueva metodología a la propuesta en 2004 [12], en el cual las poblaciones son influenciadas por reglas heurísticas y en cada generación se guía por el espacio de búsqueda mediante esquemas.

Fattahi y colaboradores en 2007 [26] proponen un modelo matemático para el Job Shop y para probarlo generan diez instancias tanto pequeñas como grandes, aunque solo logran resolver instancias pequeñas.

Sun y colaboradores en 2010 [23] Resuelven el Job Shop para el que proponen un algoritmo genético con función de penalización. Los autores proponen dos operaciones al algoritmo genético clásico, una operación llamada “Clonal Selection” y otra “LifeSpan Extended Strategy”. El objetivo de estas dos operaciones es la evitar la convergencia prematura que ocasiona una solución pseudo-óptima. Y evitar generar alta diversidad en la población inicial para evitar este problema.

Medina y colaboradores en 2011 [9] aplican un algoritmo genético al problema del Job Shop Flexible, generan 8 instancias y determinar los valores óptimos para cada parámetro de entrada para una mejor convergencia y mejores resultados en el makespan.

Balázs y colaboradores en 2012 [10] proponen dos tipos de codificación en diferentes enfoques de algoritmos evolutivos.

Caballero y colaboradores en 2013 [11] aplican un algoritmo genético y redes de Petri para resolver un caso práctico en módulos de inyección de plástico.

Jimenez y colaboradores en 2013 [7] proponen el algoritmo genético de Chu Bealsey para resolver el problema de secuenciación de tareas para un sistema tipo Flow Shop Flexible,

es decir, donde cada etapa cuenta con *m* máquinas, resuelven problemas de baja, media y alta complejidad matemática. Los autores probaron dos metodologías una donde inicializaban la población inicial con una heurística NEH y otras de forma aleatoria. También variaron el tamaño de la población según el tamaño de la instancia resuelta. Los resultados que obtuvieron fue que el tiempo de ejecución del algoritmo para resolver el problema fue mejor cuando se generaba la población inicial con la heurística NEH.

Sureshkummer y colaboradores en 2013 [25] proponen un algoritmo genético con una técnica especial de cruce al que llamaron USXX está técnica básicamente consiste en dado dos progenitores: del primero seleccionar los genes que se heredaran, estos se buscan en el progenitor dos y se marcan, al generar el hijo se toman los seleccionados del progenitor uno se colocan en el hijo conservando el orden seguido de los genes del progenitor dos eliminando los anteriormente marcados. Usando esta técnica de cruce lo aplican al JSSP donde intentan minimizar el *makespan* y un *Schedule* óptimo o casi óptimo en entornos de producción.

Jun Woo Kim en 2014 [22] diseña un juego para resolver problemas de *Scheduling* y optimización, el objetivo es desarrollar habilidades a partir de la experiencia adquirida en dicho juego. Utiliza un algoritmo genético para resolver cada *Schedule* y poder compararlo con el del jugador. La interfaz de usuario cuenta con bloques que el jugador debe acomodar para proponer su *Schedule* hasta encontrar el más adecuado.

CAPÍTULO 2
Algoritmo Genético Aplicado a Sistemas de Manufactura Flexible

2.1 Introducción

En este capítulo se abordará el problema de optimización combinatoria que involucra un problema de sistemas de manufactura flexible, representando a este como un sistema tipo Job Shop. Este es un problema con solución finita que crece de manera exponencial a medida que el tamaño del problema aumenta, generando al final de un espacio de búsqueda demasiado grande, de ahí la necesidad de usar algoritmos de búsqueda eficientes para su resolución, tales como los algoritmos genéticos. Los algoritmos genéticos han demostrado ser métodos sistemáticos para la resolución de problemas de búsqueda y optimización. Estos algoritmos aplican los mismos procesos de la evolución biológica, basadas en una población, selección, reproducción, y mutación, logrando con ellos la evolución del sistema que nos lleva a la solución del problema.

2.2 Algoritmos Genéticos

2.2.1 Inspiración Biológica

La teoría de la Evolución

Charles Darwin, científico británico, sentó las bases de la teoría moderna de la evolución de las especies, al plantear que todas las formas de vida se han desarrollado a lo largo de un proceso de selección natural que involucra millones de años [2].

La teoría de Darwin sostiene que las variaciones entre las especies ocurren al azar y que la supervivencia o extinción de cada organismo está determinado por la capacidad de dicho organismo a adaptarse a su medio ambiente [2], según los siguientes puntos:

- Todas especies tienen gran fertilidad.
- Las poblaciones permanecen aproximadamente del mismo tamaño, con fluctuaciones muy pequeñas.
- Los alimentos son escasos, pero relativamente constantes la mayor parte del tiempo.
- Existirá en ciertas circunstancias una lucha por la supervivencia.
- En la reproducción sexual, los progenitores y descendientes no serán idénticos.

Esto implica que dentro de una población donde existe la lucha por la supervivencia, y los individuos más fuertes son los que tienen mayor probabilidad de sobrevivir. Y al conseguirlo heredaran a sus descendientes aquellos rasgos que los hacen fuertes. Este proceso se repetirá generación tras generación a lo largo del tiempo hasta llegar a tener una población compuesta por individuos predominantes. Este proceso es conocido como *selección natural*.

Selección natural

La selección natural es la que establece las condiciones de un medio ambiente para favorecer o dificultar la reproducción de ciertos individuos con ciertas peculiaridades. Donde los individuos favorecidos son los más aptos para la supervivencia.

La teoría de Charles Darwin de la selección natural se basa en cuatro principios básicos [2]:

1. Los organismos en una población son diferentes, es decir, presentan variaciones en su código genético.
2. Las variaciones genéticas pueden heredarse de una generación a otra, es decir, de padre a hijo.
3. Los organismos tienen más descendientes de los que pueden sobrevivir con los recursos disponibles.
4. Las variaciones que garantizan el éxito reproductivo tendrán más posibilidad de heredarse que las que no la garantizan.

Partiendo de estos principios la selección natural tiene la capacidad de modificar una población lo suficiente como para producir especies nuevas.

Herencia

Darwin dio respuesta a la pregunta de cómo las especies evolucionan a lo largo del tiempo y los factores que la propician. Pero esto no respondía en ¿cómo las especies que sobreviven transmiten sus características a sus descendientes? Fue el monje austriaco Gregorio Mendel quien respondió a esta pregunta a través de sus experimentos. Estos consistieron en cruzar diversas plantas del jardín de su convento para así poder establecer algunas de las leyes más importantes de la genética.

La genética moderna, es la ciencia de la herencia, y ha podido establecer que los genes, portadores de los rasgos hereditarios, se encuentran en los cromosomas del núcleo celular.

En especies de plantas y animales superiores, los cromosomas de las células se agrupan en parejas. Al juntarse las dos células sexuales, cada cromosoma se reúne con su pareja, y cada gen con su homólogo del otro cromosoma [2]. Las cualidades no pueden heredarse

directamente únicamente lo hacen los genes que lo determina, es decir, el genotipo. El término genotipo se refiere al conjunto de genes de un individuo y fenotipo como el conjunto de características física determinadas por el genotipo. A continuación se enlistas algunas definiciones comunes en la genética moderna.

1. *Genotipo:* expresión genética de un organismo o estructura genética del organismo. Es la información contenida en el genoma.
2. *Fenotipo:* características físicas de un organismo, atribuidas a la expresión de su genotipo.
3. *Cromosoma:* molécula única de ADN unida a histonas (Proteínas básicas) y otras proteínas que se condensa durante la mitosis y la meiosis formada por una estructura compacta.
4. *Gen:* especifica la herencia de un carácter.
5. *Alelo:* valor de un gen.

2.2.2 Generalidades

La primera idea de un algoritmo genético surgió en la tesis de J. D. Bagley: *"El funcionamiento de los sistemas adaptables empleando algoritmos genéticos y correlativos"* en 1967. Esta tesis fue la que influyó decisivamente en J. H. Holland, quien es considerado como el pionero de los algoritmos genéticos [2].

Un algoritmo genético (AG) es una técnica de búsqueda iterativa inspirada en los principios de la selección natural. Los algoritmos genéticos artificiales buscan encontrar un mínimo o un máximo de una función establecida, por lo tanto, el objetivo de una AG es la optimización a partir de la derivación de estrategias de búsqueda. En un AG los organismos mejor aptos sobreviven y los menos aptos tenderán a desaparecer.

Partiendo del proceso evolutivo biológico se puede hacer la siguiente analogía con un algoritmo genético artificial (*ver Tabla 2.1*) [2].

Evolución Biológica	Algoritmo Genético
Genotipo	Código de la cadena
Fenotipo	Elementos sin codificar
Cromosoma	Cadena codificada
Gen	Posición en la cadena
Alelo	Valor en una posición determinada de la cadena

Tabla 2.1 Analogía entre la evolución biológico y un algoritmo genético

En general un AG se puede plantear como sigue [2]:

$$x_0 \in X \; tal \; que \; f \; es \; un \; máximo \; o \; un \; mínimo \; en \; x_0,$$
$$donde \; f: X \to \mathbb{R} \; por \; lo \; tanto$$
$$f(x_0) = máx_{x \in X} f \text{ ó } f(x_0) = mín_{x \in X} f(x)$$

En el sentido estricto es casi imposible obtener una solución óptima, dependerá del problema planteado si es suficiente encontrar el valor óptimo o basta con encontrar uno cercano al óptimo.

La transición de una generación a otra en un algoritmo genético se da de la siguiente manera:

- ***Evaluación:*** Determina la aptitud de cada individuo según cierto criterio o medida, es decir, qué individuos son los más aptos para sobrevivir.
- ***Selección:*** Selecciona a los individuos más fuertes de una generación que deben pasar a la siguiente generación o deban reproducirse.
- ***Cruzamiento:*** Toma dos individuos para reproducirse y cruzar sus informaciones genéticas para generar nuevos individuos que formarán una nueva población.
- ***Mutación:*** Altera de forma aleatoria los genes de un individuos como consecuencia de errores en la reproducción o deformación de los genes.
- ***Remplazo:*** procedimiento para crear una nueva generación de individuos.

Un AG parte de una población inicial con individuos generados aleatoriamente o a partir de ciertas reglas que consigan mayor diversidad en la población. Generación tras generación se ejecutan los procesos enlistados, hasta llegar a una generación preestablecida o el algoritmo converja a una solución.

2.3 Solución al Job Shop Scheduling Problem

El Job Shop Scheduling Problem es uno de los más famosos problemas de planificación de tareas que ha atraído la atención de muchos investigadores debido a su importancia práctica así como por su complejidad computacional.

En este trabajo se aplicará un Algoritmo Genético Clásico para intentar resolver problemas prácticos en sistemas de manufactura, haciendo a un lado el problema de complejidad que este representa. También se intenta encontrar las ventajas o desventajas de este modelo para representar sistemas reales. Por lo tanto se resolverán casos particulares de un sistema tipo Job Shop clásico, siendo este el más general dentro de los ambientes de sistemas de manufactura clásicos. Este problema es de tipo permutacional, es decir, que los elementos de cada solución pueden intercambiar posiciones, por lo tanto para cada instancia del problema existirán una gran cantidad de posibles soluciones aunque muchas

de ellas serán soluciones no factibles, es decir, que no se puede asignar un tiempo de producción.

A continuación se explica el planteamiento del problema y su resolución con algoritmos genéticos según la definición y respetando las restricciones para el JSSP enlistadas en la *sección 1.3.*

2.3.1 Parámetros de entrada

Los parámetros de entrada son los que definen una instancia del problema, por lo tanto son indispensables para su resolución. Estos pueden ser ingresados por el usuario o pueden ser preestablecidos previamente.

Los parámetros de entrada para resolver el problema se describen a continuación:

1. Conjunto de máquinas o estaciones de trabajo ***"M"*** disponibles en el sistema con *i* elementos

Donde $\boldsymbol{M} = \{m_1, m_2, m_3, \dots, m_i\} \ \forall \ m_i \in \mathbb{Z}^+$

2. Conjunto productos o pedidos a ser fabricados ***"P"*** con *j* elementos

Donde $\boldsymbol{P} = \{p_1, p_2, p_3, \dots, p_j\} \ \forall \ p_j \in \mathbb{Z}^+$

3. Tiempos de procesamiento "$\boldsymbol{T}$"

Matriz de tiempos de procesamiento. Cuyos elementos t_{ji} representa el tiempo de procesamiento del $j - esimo$ producto en la $i - esima$ máquina. Si una máquina no procesa un producto se asumirá que su tiempo de procesamiento en dicha máquina es $t_{j,i} = 0 \ [unidades \ de \ tiempo]$ de esta forma se simplifica más la programación del algoritmo.

$$T = \begin{bmatrix} t_{1,1} & t_{1,2} & \dots & t_{1,i} \\ t_{2,1} & t_{2,2} & \dots & t_{2,j} \\ \vdots & \vdots & \ddots & \vdots \\ t_{j,1} & t_{i,2} & \dots & t_{j,i} \end{bmatrix} \quad \forall \ t_{j,i} \in \mathbb{R}^+$$

4. Restricción por Proceso **"$\boldsymbol{Rp}$"**

Matriz donde cada fila representa la trayectoria que cada producto sigue a través de las diferentes máquinas disponibles en el sistema, es decir, representa el flujo del proceso de fabricación de cada producto a través del sistema. Cada elemento $m_{j,o}$ representa la $o - ésima$ operación del $j - ésimo$ producto procesado en una máquina que pertenece a ***M***. Para fines de este trabajo, suponemos que cada máquina procesará cada producto una

sola vez. Por lo tanto el número de operaciones es exactamente igual al número de máquinas disponibles. Matricialmente se tiene:

$$\boldsymbol{Rp} = \begin{bmatrix} m_{1,1} & m_{1,2} & \dots & m_{1,o} \\ m_{2,1} & m_{2,2} & \dots & m_{2,o} \\ \vdots & \vdots & \ddots & \vdots \\ m_{j,1} & m_{j,2} & \dots & m_{j,o} \end{bmatrix} \quad \forall\, m_{j,o} \in \boldsymbol{M}$$

5. Porcentaje de Cruce "P_c"

Representa la probabilidad que dos individuos al ser seleccionados sean cruzados para generar nuevos individuos.

$$0 \leq P_c \leq 100$$

Se recomienda tener porcentajes de cruce mayor a un 90%. Para que el algoritmo evolucione y pueda generar nuevos individuos que sean más aptos que sus predecesores.

6. Porcentaje de Mutación "P_m"

Representa la probabilidad que los nuevos individuos generados a partir de cruce sean mutados.

$$0 \leq P_m \leq 100$$

Se recomienda tener porcentajes mutación menor al 10% para evitar una divergencia del algoritmo o una convergencia prematura.

7. Tamaño de la población "L_p"

Representa el número de individuos de cada generación.

$$L_p > 0$$

El número de individuos puede depender del tamaño de la instancia a resolver aunque no se recomienda tamaños de población mayor a 200 debido a que este genera un mayor costo computacional además de provocar que el algoritmo converja en tiempos de ejecución elevados.

2.3.2 Representación del Problema

Para representar gráficamente el problema del Job Shop y tener una visualización más clara del problema, se recurre al grafo disyuntivo *(Ver Sección 1.4.1.)*. El grafo disyuntivo provee de representación fácil de visualizar y de un planteamiento matemático más sencillo, comprado con redes de Petri, que nos proporcionan una descripción más detallada del sistema, pero un modelo matemático más amplio y que nos incrementa el

costo computacional. Aunque el grafo disyuntivo es una representación que puede despreciar detalles del sistema simplificándolo a algo más general y práctico, aun así proporciona una poderosa herramienta para la resolución del Job Shop siempre y cuando se plantee correctamente y se tomen las consideraciones suficientes y adecuadas.

El objetivo de buscar una solución es dar dirección a los arcos disyuntivos, así cada solución determina el flujo o ruta diferente de los productos a fabricar a lo largo de las máquinas, una vez establecido el flujo de los productos en cada máquina, se puede calcular el tiempo de producción, aunque hay que aclarar que también se pueden generar soluciones no factibles, es decir, que matemáticamente representan un flujo pero que en la práctica es imposible de implementar.

Se toma el ejemplo propuesto en la *Sección 1.4.1* que es con el que se trabajará en este capítulo.

Matricialmente se tiene:

t_{ji}	*máquina 1*	*máquina 2*	*máquina 3*
producto 1	5	8	2
producto 2	3	9	7
producto 3	1	10	7
producto 4	7	4	11

m_i	*Operación 1*	*Operación 2*	*Operación 3*
producto 1	m_1	m_2	m_3
producto 2	m_3	m_1	m_2
producto 3	m_1	m_3	m_2
producto 4	m_2	m_3	m_1

A modo de ejemplo una solución factible de esta instancia sería:

Máquina 1	1	3	4	2
Máquina 2	4	1	3	2
Máquina 3	1	4	3	2

Por lo tanto el grafo disyuntivo del ejemplo quedaría como se muestra en la *Figura 2.1*. Ahora los arcos disyuntivos dejan de ser bidireccionales para pasar a ser unidireccionales, y así generar una secuencia de procesamiento de los productos en cada máquina.

2.3.3 Codificación

Para resolver el problema es indispensable codificar las soluciones de una manera que nos permita calcular el tiempo de producción, y además que sea capaz de efectuar las operaciones de cruce y mutación de una forma sencilla. La codificación utilizada en este

trabajo es la de máquina-producto, es decir, cada cromosoma está representado por una secuencia de productos [21]. A continuación se describe la representación utilizada.

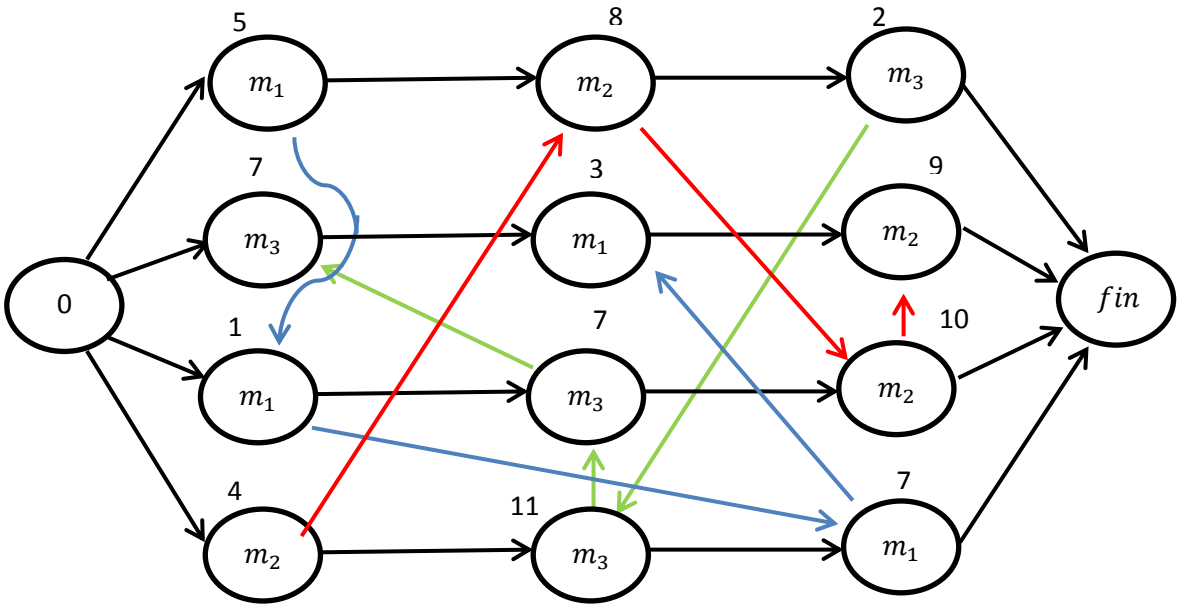


Figura 2.1 Solución factible a una instancia de 4 productos con 3 máquinas. Fuente [Propia de autor]

Un cromosoma representa a un individuo dentro de una población, es decir, una solución posible del problema. Y cada elemento de este está conformado por máquinas, que son genes que representan la secuencia en que los productos serán procesados en cada máquina.

Por lo tanto cada elemento de un Gen, es decir un alelo, representa a un producto. Siendo más explícito un Cromosoma es una solución del problema, este cromosoma posee el conjunto de máquinas que dispone el sistema, donde cada máquina cuenta con un conjunto de productos que serán procesados en dicha máquina en el orden dado. Entonces un cromosoma tiene la siguiente estructura:

$$\underbrace{\underbrace{p_1 \quad p_2 \quad \dots \quad p_j}_{Gen = m_1} \mid \underbrace{p_1 \quad p_2 \quad \dots \quad p_j}_{Gen = m_2} \mid \dots \mid \underbrace{p_1 \quad p_2 \quad \dots \quad p_j}_{Gen = m_i}}_{Cromosoma}$$

Dónde:

$$p_j \in \boldsymbol{P}, m_i \in \boldsymbol{M},$$

Matricialmente cada individuo se representa como:

$$\boldsymbol{Cr} = \begin{bmatrix} p_{1,1} & p_{1,2} & \cdots & p_{1,j} \\ p_{2,1} & p_{2,2} & \cdots & p_{2,j} \\ \vdots & \vdots & \ddots & \vdots \\ p_{i,1} & p_{i,2} & \cdots & p_{i,j} \end{bmatrix} \quad \forall\, p_{i,j} \in \boldsymbol{P}$$

Donde cada fila representa a un gen y cada elemento es un alelo. La principal ventaja de esta codificación es que permite una fácil comprensión y facilita el cálculo del tiempo de producción.

Según el ejemplo de la sección anterior (*ver Figura 2.1*), el cromosoma correspondiente tendrá la siguiente estructura:

$$\underset{1}{1\ 3\ 4\ 2} \mid \underset{2}{4\ 1\ 3\ 2} \mid \underset{3}{1\ 4\ 3\ 2}$$
$$Solución$$

2.3.4 Población inicial

La población inicial es el conjunto de individuos del cual parte el algoritmo, se puede decir que es la materia prima como material genético, así que mientras mayor diversidad de material genético se tenga mejor será el desarrollo o evolución del algoritmo por cada generación que transcurra, por lo tanto menos probabilidad de caer en óptimos locales.

La población inicial tendrá exactamente el mismo número de individuos como cualquier otra generación establecido por el parámetro de entrada "L_p", dicho parámetro tendrá un valor dependiendo del caso de estudio, es decir, dependerá del número de productos y máquinas de cada instancia del problema. Aunque no se recomienda exceder un tamaño de población mayor a 200 pero tampoco un tamaño muy pequeño para evitar encontrar una solución lejana a la óptima y evitar una convergencia prematura. En general se recomiendan poblaciones de entre 100 a 200 individuos.

Dada una población:

$$\underset{Población}{Cromosoma\ 1 \quad Cromosoma\ 2 \quad \ldots \quad Cromosoma\ L_p}$$

Se define un conjunto P_0 como población inicial

$$P_0 = \{cr_1, cr_2, cr_3, \ldots, cr_{i=Lp}\}$$

La población inicial se genera de forma aleatoria, intentando garantizar tener la mayor diversidad en los individuos, por ejemplo, no se admiten individuos con el mismo código genético o con el mismo tiempo de producción (*makespan*). También se descartan soluciones no factibles provocando que este proceso del algoritmo sea muy lento. En el

algoritmo genético la población inicial representa la generación 0.

2.3.5 Evaluación

Cada individuo de la población debe ser evaluado, es decir, para cada posible solución calcular el tiempo de producción o *makespan*. Este proceso del algoritmo se realiza según se explica a continuación:

Para evaluar a un individuo se recurre a las matrices "$\boldsymbol{Rp}$" y renombrando a "$\boldsymbol{Cr}$" entonces se tiene que:

$$\boldsymbol{Rp} = \begin{bmatrix} m_{1,1} & m_{1,2} & \dots & m_{1,o} \\ m_{2,1} & m_{2,2} & \dots & m_{2,o} \\ \vdots & \vdots & \ddots & \vdots \\ m_{j,1} & m_{j,2} & \dots & m_{j,o} \end{bmatrix} \quad representa\ las\ restricciones\ de\ proceso$$

Donde para cada producto existe una restricción de proceso:

$$para\ cada\ p_j\ \exists\ \ m_{j,o-1} \rightarrow m_{j,o}\ \forall\ m_{j,o} \in \boldsymbol{M}$$

Es decir el producto p_j no puede pasar a la máquina $m_{j,o}$ sin antes pasar por la máquina $m_{j,o-1}$, dicho de otra manera el proceso de p_j no puede seguir a la operación "o" sin antes terminar la operación anterior "o-1".

$$\boldsymbol{Cr} = \begin{bmatrix} p_{1,1} & p_{1,2} & \dots & p_{1,j} \\ p_{2,1} & p_{2,2} & \dots & p_{2,j} \\ \vdots & \vdots & \ddots & \vdots \\ p_{i,1} & p_{i,2} & \dots & p_{i,j} \end{bmatrix} = \boldsymbol{Rm} \quad representa\ las\ restricciones\ de\ máquina$$

Donde para cada máquina existe una restricción de máquina:

$$para\ cada\ m_i\ \exists\ \ p_{i,j-1} \rightarrow p_{i,j}\ \forall\ p_{i,j} \in \boldsymbol{P}$$

Es decir, la máquina m_i no puede procesar al producto $p_{i,j}$ sin antes procesar a $p_{i,j-1}$.

Se definen los tiempos:

$$ts_{ji} = tiempo\ de\ inicio\ de\ una\ operación\ del\ producto\ j\ en\ la\ máquina\ i$$

$$tf_{ji} = tiempo\ al\ final\ de\ una\ operación\ del\ producto\ j\ en\ la\ máquina\ i$$

$$tp_{ji} = tiempo\ de\ procesamiento\ del\ producto\ j\ en\ la\ máquina\ i$$

Por lo tanto

$$tf_{ji} = ts_{ji} + tp_{ji}$$

Donde:

$$ts_{ji} = max(tf_{j-1,i}, tf_{j,i-1})$$

Es decir, el tiempo al final de una operación del $producto\ j$ en la $máquina\ i$ será igual al tiempo de inicio de dicha operación más su tiempo de procesamiento, donde el tiempo de inicio será el tiempo mayor entre la última operación procesada por la $máquina\ i$ y la operación anterior del $producto\ j$.

Se define un tiempo de terminación para el $producto\ j$

$$C_j = tf_{j,i=ultima\ operación}$$

Entonces el tiempo de producción será:

$$makespan = C_{máx} = max(C_j)$$

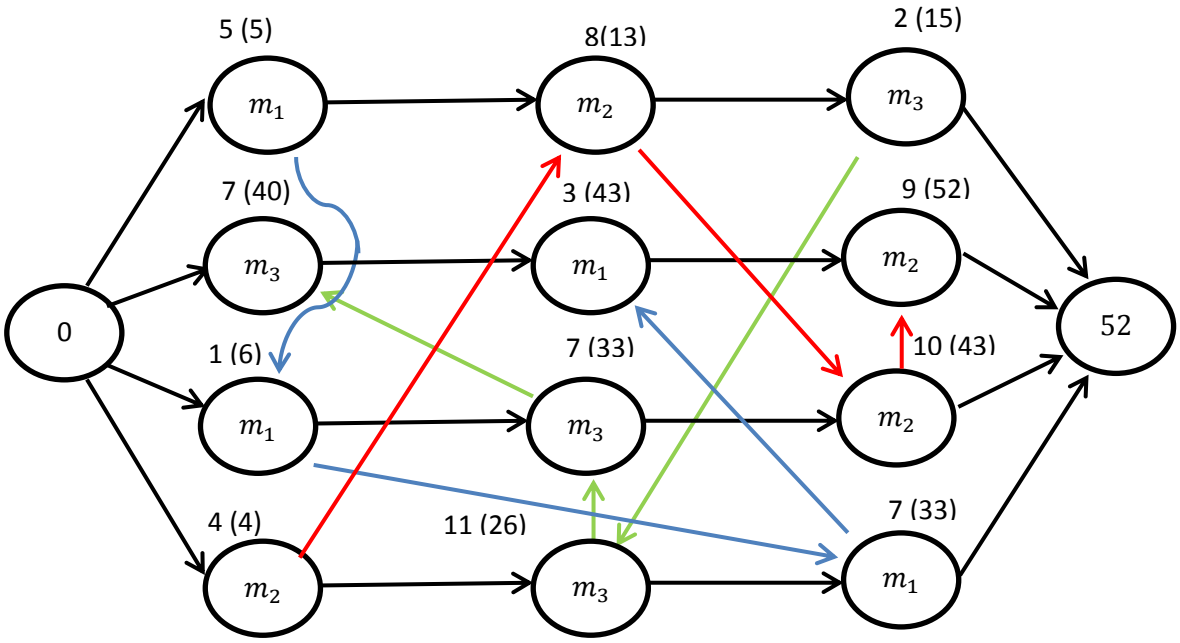


Figura 2.2 Ejemplo del cálculo del makespan, se muestra el tiempo de procesamiento junto a cada nodo y el tiempo acumulado hasta ese instante entre paréntesis. Fuente [Propia del autor]

Según la solución factible de la *sección 3.3.2* el *makespan* calculado gráficamente consiste simplemente en sumar los tiempos respetando las restricciones del proceso y restricciones por disponibilidad de máquina a lo largo de cada arco dirigido y al llegar a cada nodo verificar cual es tiempo es el mayor que será el que se sumará al tiempo de procesamiento actual, estos tiempos predecesores representan si el producto debe esperar a que se procese otro producto o si la máquina espera a que el producto en turno

finalice su operación anterior. Así al final cada máquina tendrá asignado un tiempo de terminación y el *makespan* será el mayor de todos los tiempos de terminación (*ver Figura 2.2*). El correspondiente diagrama de Gantt se observa en la *Figura 2.3*.

Por lo tanto la función objetivo, la que se desea optimizar, en este caso minimizar es el *makespan*. Al minimizar el tiempo de producción o *makespan* este a su vez minimiza el tiempo de máquinas en paro (Idle) y maximiza el tiempo de trabajo de cada máquina aprovechando al máximo los recursos. El valor del *makespan* es la que guiará la búsqueda a una solución óptima a través del algoritmo. Existen otras funciones objetivos mencionados en la *sección 1.2.4* que en este trabajo no se abordarán.

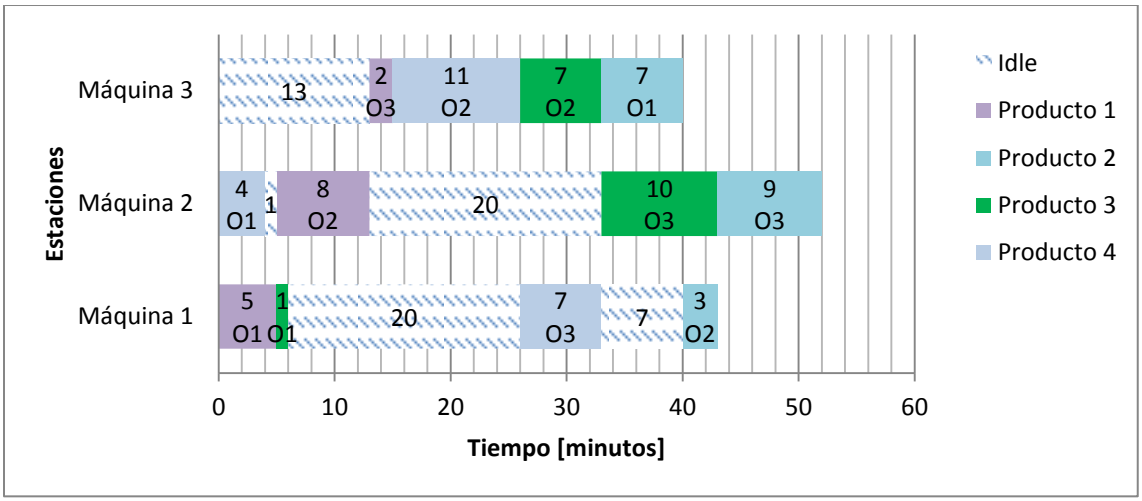


Figura 2.3 Diagrama de Gantt para una instancia de 4 productos y 3 máquinas. Fuente [propia del autor]

2.3.6 Selección

La selección es una de las principales componentes que permite la convergencia del algoritmo. Elige a los individuos más aptos de una población previamente evaluada con una función de aptitud y desprecia a los menos aptos. Dicha selección se realiza de manera aleatoria, la probabilidad de escoger a un individuo apto es directamente proporcional a su aptitud. Así a lo largo de las generaciones los individuos más aptos serán los predominantes en la población dando como consecuencia la convergencia del algoritmo.

En el algoritmo planteado el proceso de selección se realiza como sigue:

Una vez evaluado cada individuo de una población se procede a seleccionar a los más aptos que serán los que pasarán a la siguiente generación o los que se cruzarán para generar nuevos individuos. Este procedimiento se basa en el cálculo de aptitudes.

Dada una población con L_p individuos

$$P_i = \{cr_1, cr_2, cr_3, \dots, cr_{Lp}\}$$

Serán seleccionados L_p individuos mediante un proceso aleatorio conocido como método de la ruleta. El método de la ruleta consiste en lanzar un dardo a una ruleta que se encuentra girando para determinar el individuo elegido. Matemáticamente el método puede ejecutarse de la siguiente manera.

Para el método es necesario crear una matriz de aptitudes $\boldsymbol{A}$:

$$\boldsymbol{A} = \begin{bmatrix} cr_1 & C_{máx_1} & A_1 & pA_1 \\ cr_2 & C_{máx_2} & A_2 & pA_1 \\ \vdots & \vdots & \vdots & \vdots \\ cr_{Lp} & C_{máx_{Lp}} & A_{Lp} & pA_{Lp} \end{bmatrix}$$

Dónde:

$$C_{máx_n} = makespan\ del\ n - esimo\ individuo$$

$$A_n = Aptitud\ del\ n - esimo\ individuo = max\left(C_{máx_{1 \to Lp}}\right) - C_{máx_n}$$

$$pA_n = Porcentaje\ de\ Aptitud = \frac{A_n}{\sum_{n=1}^{Lp} A_n}$$

Una vez que se tiene $\boldsymbol{A}$ se procede a crear la ruleta que puede ser representado por una matriz

$$\boldsymbol{r} = \begin{bmatrix} cr_1 & 0 & pA_1 \\ cr_2 & pA_1 & pA_2 \\ \vdots & \vdots & \vdots \\ cr_{Lp} & pA_{Lp-1} & pA_{Lp} \end{bmatrix}$$

Una vez que se tiene la ruleta se genera un numero aleatorio que este entre 0-100 para simular el lanzamiento de un dardo a una ruleta que se encuentra girando, según el valor de este número aleatorio se determina el individuo elegido. Los individuos que tengan un mayor intervalo de $[pA_{Lp-1}, pA_{Lp}]$ tendrán una probabilidad mucho mayor de ser elegidos.

2.3.7 Cruce

La reproducción sexual como sucede en la naturaleza consiste en mezclar el material genético de los progenitores. Por lo general los cromosomas son aleatoriamente divididos y mezclados, por lo tanto ciertos genes de los descendientes provienen de un progenitor, mientras otros provienen del otro, los descendientes tendrán información en sus genotipo

de ambos progenitores.

El cruce representa la posibilidad de heredar las características que hacen a un individuo más apto para sobrevivir, así al cruzar dos individuos aptos se tiene una alta probabilidad de poder generar hijos con igual o mayor aptitud que los progenitores. Así el cruce garantiza la conservación de las características que hacen a un individuo fuerte para sobrevivir y la eliminación de las que la hacen débiles.

Para realizar el cruce se recurre a los individuos que previamente han sido seleccionados, de donde se tomarán pares de individuos aleatoriamente para realizar este proceso.

Entonces, dada una población de individuos seleccionados con L_p individuos

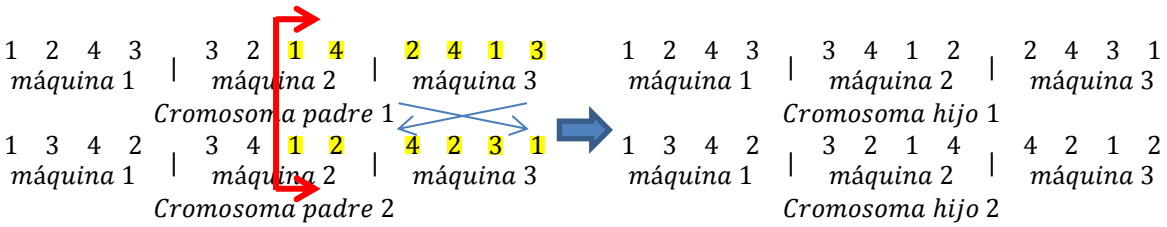
$$P_{seleccionados} = \{cr_{s1}, cr_{s2}, cr_{s3}, \dots, cr_{sLp}\}$$

Se eligen aleatoriamente dos individuos de esta población. Luego se genera un número aleatorio entre 0-100 y si es menor al parámetro de entrada "P_c" entonces el cruce se ejecutará. Supón que se eligieron dos individuos de $P_{seleccionados}$ con 4 productos y 3 máquinas:

$$\underset{máquina\ 1}{1\ 2\ 4\ 3} \mid \underset{máquina\ 2}{3\ 2\ 1\ 4} \mid \underset{máquina\ 3}{2\ 4\ 1\ 3}$$
$$Cromosoma\ 1$$

$$\underset{máquina\ 1}{1\ 3\ 4\ 2} \mid \underset{máquina\ 2}{3\ 4\ 1\ 2} \mid \underset{máquina\ 3}{4\ 2\ 3\ 1}$$
$$Cromosoma\ 2$$

Se generan dos números aleatorios para elegir la máquina y el producto como punto de cruce, supón que se eligió la máquina 2 y el tercer producto de esta máquina:



Se intercambiarán todos los alelos que se encuentren hacia la derecha a partir del punto de cruce hasta el final de la cadena, evitando que los números se repitan dentro de un mismo gen. Si lo hacen se sustituye por el valor sustituido.

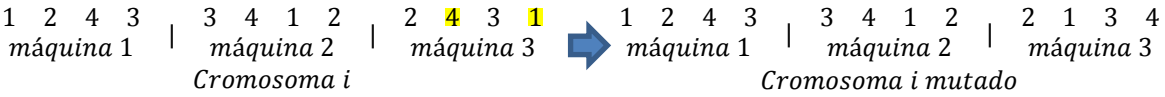
2.3.8 Mutación

El último ingrediente de los algoritmos genéticos es la mutación, este se refiere a la deformación aleatoria de la información genética en un individuo. En la reproducción

natural, la probabilidad de que determinado gen sea mutado es casi igual para todos los genes. La mutación es un proceso que da mayor diversidad a una población, es claro que sus efectos pueden ser tanto positivos como negativos pero sucede con una probabilidad muy baja, así pocos individuos sufrirán este proceso.

Para determinar al individuo que sufrirá mutación primero debe surgir por cruce, es decir, únicamente los hijos de dos individuos cruzados pueden ser mutados. Una vez que dos individuos fueron cruzados se genera un numero aleatorio entre 0-100 y si este es menor o igual al parámetro de entrada "P_m" entonces los individuos generados, es decir los hijos, serán mutados de lo contrario los hijos pasan a la siguiente generación sin ser alterados. Una vez que se determina que se realizará mutación, se debe determinar el gen que se mutará y la posición de los alelos que se intercambiarán. Se recomienda establecer P_m valores pequeños para evitar resultados indeseados o que quizás el algoritmo nunca converja o lo haga a valores indeseados.

A modo de ejemplo, se toma uno de los hijos generados en ejemplo anterior de cruce de la *sección2.3.7*. Supón como punto de mutación la máquina 3 y el segundo producto de esta máquina. Después se elige aleatoriamente un alelo del mismo gen con el que se intercambiará el alelo seleccionado previamente, en este caso el cuarto producto de la máquina 3. Al final se intercambian los alelos elegidos.



2.3.9 Algoritmo Genético

Con lo planteado anteriormente se genera el pseudocódigo para el algoritmo genético *(Ver Figura 2.4)*. Donde el criterio de paro es la convergencia del algoritmo, es decir, cuando el individuo más fuerte y el individuo más débil de la población $\boldsymbol{P_i}$ son iguales entonces se rompe el ciclo.

Para finalizar, se aplica el algoritmo genético implementado para resolver el problema del ejemplo de la *sección 2.3.2*. Al ejecutarlo se encuentra que la solución óptima es la siguiente

Máquina 1	1	3	2	4
Máquina 2	4	1	2	3
Máquina 3	2	3	4	1

y que tiene un valor mínimo de *makespan* de 32[minutos]. El correspondiente diagrama de Gantt se muestra en la *Figura 2.5* en él se observa como el *makespan* es minimizado si

se compara con la solución factible planteado en la *sección 2.3.2* que tenía un valor para el *makespan* de 52[minutos]. También se obtiene que el tiempo de máquinas en paro se reduce drásticamente aprovechando al máximo el uso de los recursos.

Sean los parámetros de entrada: $\boldsymbol{M}, \boldsymbol{P}, \boldsymbol{T}, \boldsymbol{Rp}, Lp, Pc, Pm$
Sea $\boldsymbol{P_i} = \{cr_1, cr_2, \dots, cr_{Lp}\}$ *una población*
Sea rand: un número aletorio $\in [0,100]$
Generar $\boldsymbol{P_0}$: *Población incial y hacer* $\boldsymbol{P_i} = \boldsymbol{P_0}$
WHILE *no se cumpla el criterio de paro*
|*Evaluar* $\boldsymbol{P_i}$
|*seleccionar aleatoriamente como padres* $\{cr_a, cr_b\} \in \boldsymbol{P_i}$
| ***WHILE*** *no se rellene* $\boldsymbol{P_{i+1}}$
| | ***IF*** *al generar rand;* $rand < Pc$
| | | *generar hijos a partir de operación de cruce*
| | | ***IF*** *al generar rand;* $rand < Pm$
| | | |*mutar hijos generados e ingresar a* $\boldsymbol{P_{i+1}}$
| | | ***ELSE***
| | | |*ingresar hijos generados a* $\boldsymbol{P_{i+1}}$
| | ***ELSE***
| | |*ingresar padres seleccionados a* $\boldsymbol{P_{i+1}}$
| ***FIN***
|*hacer* $\boldsymbol{P_i} = \boldsymbol{P_{i+1}}$
FIN

Figura 2.4 Pseudocódigo del algoritmo genético planteado. Fuente [propia del autor]

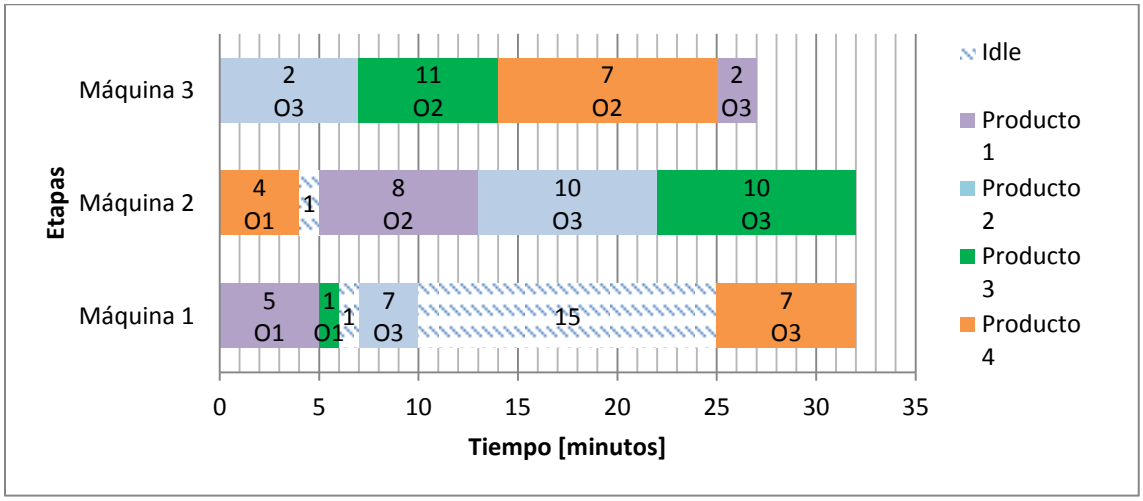


Figura 2.5 Diagrama de Gantt para solución encontrada con el algoritmo genético desarrollado para una instancia de 4x3. Fuente [propia del autor]

CAPÍTULO 3
Aplicación de un Algoritmo Genético para minimizar el *makespan*

3.1 Introducción

En este capítulo se resolverán instancias de un Job Shop, principalmente problemas de tipo Flow Shop, que como se vio en la *sección 1.2.3* es un caso particular del Job Shop. Primero se abordará un caso de tres máquinas, que es el tipo de sistema con el que se cuenta en los laboratorios de Manufactura Avanzada en la Facultad de Ingeniería de la UNAM. Como segundo ejemplo se abordará un caso muy superficial de la industria automotriz en la producción y ensamble de automóviles en la Wolksvagen. Como tercero y último se abordará un caso genérico que involucre mayor flexibilidad en el proceso, al permitir que cada pedido tenga un proceso de fabricación diferente.

En todos los casos resueltos los tiempos de procesamientos representan idealizaciones de tal manera que puedan aproximarse más al valor real. Por ejemplo el tiempo de procesamiento de un producto será igual al tiempo de procesamiento de la operación en turno más el tiempo de preparación del producto para iniciar su procesamiento y más el tiempo que el producto viajó desde la última estación.

Para resolver este problema se programó un algoritmo genético clásico en código C#.net que es un lenguaje de programación orientado a objetos de alto nivel, desarrollado por Microsoft Corporation. El algoritmo se programó siguiendo la metodología explicada en el capítulo 2.

Finalmente todos los casos fueron simulados en *Promodel7* versión estudiantil para comprobar los resultados obtenidos en el algoritmo genético. Para la simulación se eligió un solo resultado de todas las corridas ejecutas en cada caso, dicho resultado fue el de menor *makespan* de todas las corridas. La limitación de este software en su versión estudiantil limitó los casos prueba, debido a que este software sólo admite hasta 8 entidades.

3.2 Casos de estudio

Caso 1: Laboratorio de Manufactura Avanzada, Faculta Ingeniería UNAM

Se tiene una celda de manufactura flexible, dicho sistema cuenta con 3 estaciones de trabajo principales y cada uno de ellos cuenta con una única máquina. La primera estación cuenta con un torno, la segunda con una fresadora y la última etapa es la de inspección. En este sistema se tiene una banda trasportadora que obliga a todos los productos a seguir una misma ruta de fabricación. Por lo tanto el sistema es de tipo Flow

Shop, donde todos los productos empiezan en el almacén, siguen al torno, luego con la fresadora y finalizan como producto terminado en la etapa de inspección. En la *Tabla 3.1* Se enlistan las 3 estaciones, cada una con un identificador (ID), un nombre y una descripción general de cada estación. En la *Tabla 3.2* se enlistan los productos a fabricar cada una con su correspondiente proceso de fabricación y los tiempos de procesamiento en cada estación. El resto de los parámetros de entradas se enlistan continuación:

$$P_c = 98\%, \qquad P_m = 10\%, \qquad L_p = 100\ individuos$$

Lista de Estaciones de trabajo o de Etapas de producción		
ID	**Nombre**	**Descripción**
1	Torno	Se realizan operaciones de torneado
2	Fresa	Se realizan operaciones de Fresado
3	Inspección	Se realiza la operación para validar el producto final

Tabla 3.1 Lista de estaciones de trabajo con la que cuenta el sistema de manufactura del laboratorio de la Facultad de ingeniería de la UNAM

Lista de Productos a fabricar				
ID	**Nombre**	**Proceso de Fabricación [ID Etapa(tiempo de procesamiento)]**		
1	Producto 1	1 (20 min)	2 (30 min)	3 (6 min)
2	Producto 2	1 (17 min)	2 (35 min)	3 (8 min)
3	Producto 3	1 (32 min)	2 (30 min)	3 (6 min)
4	Producto 4	1 (20 min)	2 (39 min)	3 (8 min)
5	Producto 5	1 (30 min)	2 (35 min)	3 (7 min)
6	Producto 6	1 (50 min)	2 (13 min)	3 (8 min)
7	Producto 7	1 (60 min)	2 (86 min)	3 (6 min)
8	Producto 8	1 (60 min)	2 (50 min)	3 (9 min)

Tabla 3.2 Lista de Productos a fabricar para el caso 1 y cada uno con su respectivo proceso de fabricación y tiempos de procesamiento dado en minutos

	Resultados de Ejecutar AG									
Corrida	1	2	3	4	5	6	7	8	9	10
Iteraciones	92	91	59	97	92	52	53	67	42	75
makespan	362 [min]	384 [min]	377 [min]	365 [min]	370 [min]	382 [min]	375 [min]	349 [min]	380 [min]	370 [min]

Tabla 3.3 Resultados obtenidos de ejecutar 10 corridas para el caso 1

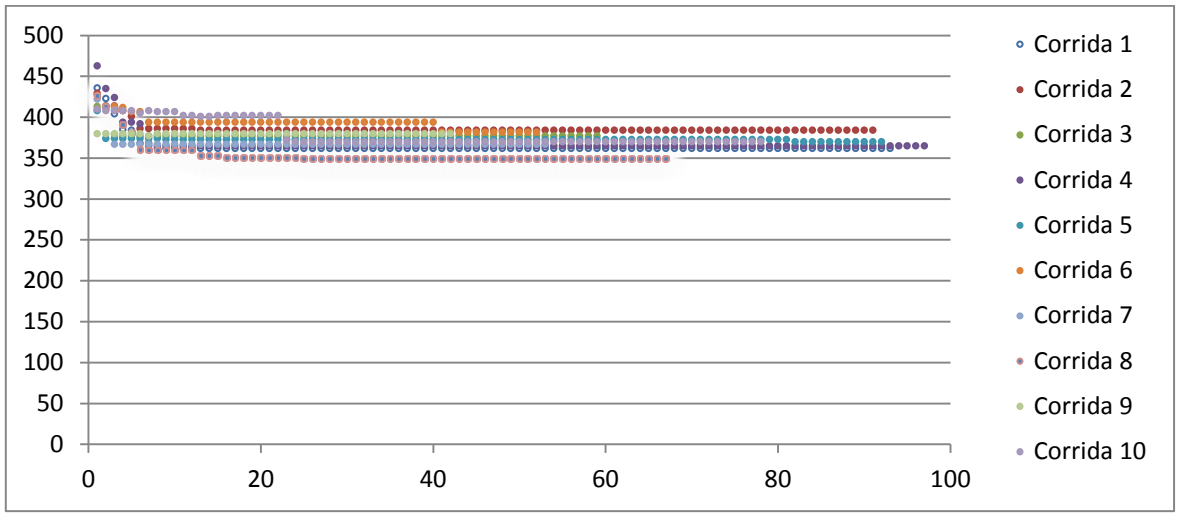


Figura 3.1 Evolución del makespan en las corridas ejecutadas para el caso 1. Fuente [propia del autor]

Con los resultados obtenidos en la *Tabla3.3* se tiene un porcentaje de error de **9.11%**, en la *Figura 3.1* se observa la evolución del algoritmo hasta llegar a la convergencia en todas las corridas. La mejor solución de todas las corridas ejecutadas es la corrida 8 con un valor de 349[min]. A continuación se muestra la secuencia de la mejor solución encontrada y su correspondiente diagrama de Gantt se muestra en la *Figura* 3.3.

Máquina 1	4	2	7	8	3	1	5	6
Máquina 2	4	2	7	8	1	3	5	6
Máquina 3	4	2	8	3	7	5	1	6

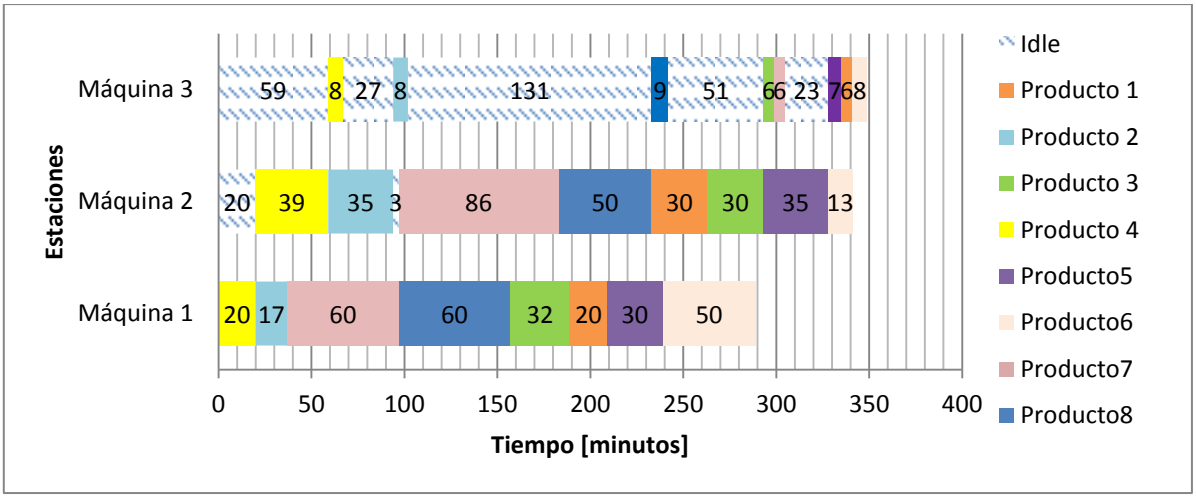


Figura 3.2 Diagrama de Gantt de la mejor solución de todas las corridas ejecutadas para el caso 1

En la *Figura 3.3* se muestra la disposición física de las máquinas y el flujo que siguen todos los productos en el proceso de fabricación, parten del almacén de materia prima, para luego procesar la primera operación en el Torno, seguir con el proceso de Fresado, y finalizar con el proceso de Inspección.

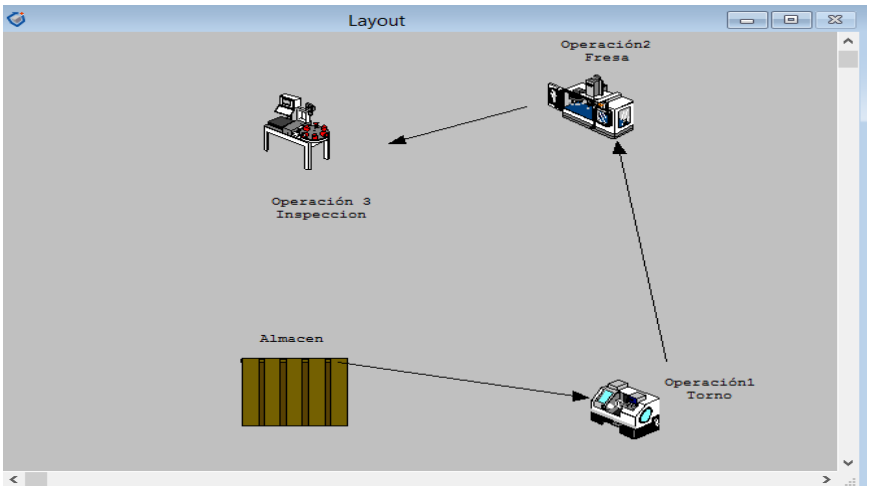


Figura 3.3 Disposición de máquinas para el caso1, imagen tomada de Promodel7. Fuente [propia del autor]

En la *Figura 3.4* se observa que el producto6 que en la solución es el último producto en salir del sistema, finaliza a un tiempo 349 [min], lo cual comprueba la validez de la solución encontrada.

General Report (Normal Run - Rep. 1)

General | Locations | Location States Multi | Failed Arrivals | Entity Activity | Entity States | Variables

Caso1_LabFI.MOD (Normal Run - Rep. 1)

Name	Total Exits	Current Qty In System	Avg Time In System (MIN)	Avg Time In Move Logic (MIN)	Avg Time Waiting (MIN)	Avg Time In Operation (MIN)
Pieza1	1.00	0.00	341.00	0.00	285.00	56.00
Pieza2	1.00	0.00	102.00	0.00	42.00	60.00
Pieza3	1.00	0.00	299.00	0.00	231.00	68.00
Pieza4	1.00	0.00	67.00	0.00	0.00	67.00
Pieza5	1.00	0.00	335.00	0.00	263.00	72.00
Pieza6	1.00	0.00	349.00	0.00	278.00	71.00
Pieza7	1.00	0.00	305.00	0.00	153.00	152.00
Pieza8	1.00	0.00	242.00	0.00	123.00	119.00

Figura 3.4 Estadísticas de cada producto en su proceso de fabricación. Obtenidas de la simulación en ProModel7 para el caso 1. Fuente [propia del autor]

Caso 2: Industria Automotriz, Línea de producción Wolksvagen

Se presenta un ejemplo en una línea de ensamble de Wolksvagen, se intentará resolver de manera superficial y se simplificará cada estación de trabajo como si se contará con una

única máquina.

$$P_c = 98\%, \qquad P_m = 10\%, \qquad L_p = 150\ individuos$$

Lista Estaciones de trabajo o Etapas de producción		
ID	**Nombre**	**Descripción**
1	Estampado	Se fabrican los productos en diferentes líneas de producción como por ejemplo pisos, costados, puertas, tapas, toldos que forman las carrocerías de cada uno de los modelos.
2	Construcción de Carrocerías	Los productos son ensamblados hasta formar una carrocería completa.
3	Pintura de Carrocerías	Consta de 6 etapas: 1. Pre-tratamiento de carrocerías 2. Aplicación de primer catódico 3. Sellado 4. Filler 5. Esmalte 6. Barniz
4	Pintura Partes Plástica	Consta de 5 etapas: 1. Cabina de flama 2. Filler 3. Curado 4. Esmalte 5. barniz
5	Montaje	Es el proceso de ensamble final, en el cual las carrocerías ya pintadas se les integran el resto de los elementos que componen un automóvil.

Tabla 3.4 Lista de etapas en el proceso de producción de automóviles. Fuente [38]

Lista de Productos a Fabricar						
ID	**Nombre**	**Proceso de Fabricación [ID Etapa(tiempo de procesamiento)]**				
1	Modelo 1	1 (202 min)	2 (309 min)	3 (633 min)	4 (838 min)	5 (638 min)
2	Modelo 2	1 (172 min)	2 (355 min)	3 (893 min)	4 (858 min)	5 (558 min)
3	Modelo 3	1 (322 min)	2 (306 min)	3 (636 min)	4 (636 min)	5 (736 min)
4	Modelo 4	1 (203 min)	2 (396 min)	3 (826 min)	4 (323 min)	5 (323 min)
5	Modelo 5	1 (303 min)	2 (351 min)	3 (766 min)	4 (563 min)	5 (963 min)
6	Modelo 6	1 (506 min)	2 (139 min)	3 (866 min)	4 (653 min)	5 (753 min)
7	Modelo 7	1 (606 min)	2 (869 min)	3 (636 min)	4 (983 min)	5 (883 min)
8	Modelo 8	1 (607 min)	2 (503 min)	3 (968 min)	4 (989 min)	5 (993 min)

Tabla 3.5 Lista de Modelos a fabricar para el caso 2 y cada uno con su respectivo proceso de fabricación y tiempos de procesamiento en minutos

	Resultados de Ejecutar AG									
Corrida	1	2	3	4	5	6	7	8	9	10
Iteraciones	320	522	213	470	117	408	169	253	190	317
makespan	8579 [min]	8802 [min]	9071 [min]	8529 [min]	8717 [min]	8851 [min]	8923 [min]	8926 [min]	8320 [min]	8706 [min]

Tabla 3.6 Resultados obtenidos de ejecutar 10 corridas para el caso 2

Con los resultados obtenidos en la *Tabla 3.6* se tiene un porcentaje de error de **8.28%**, la evolución del algoritmo de todas las corridas se muestra en la *Figura3.5*.

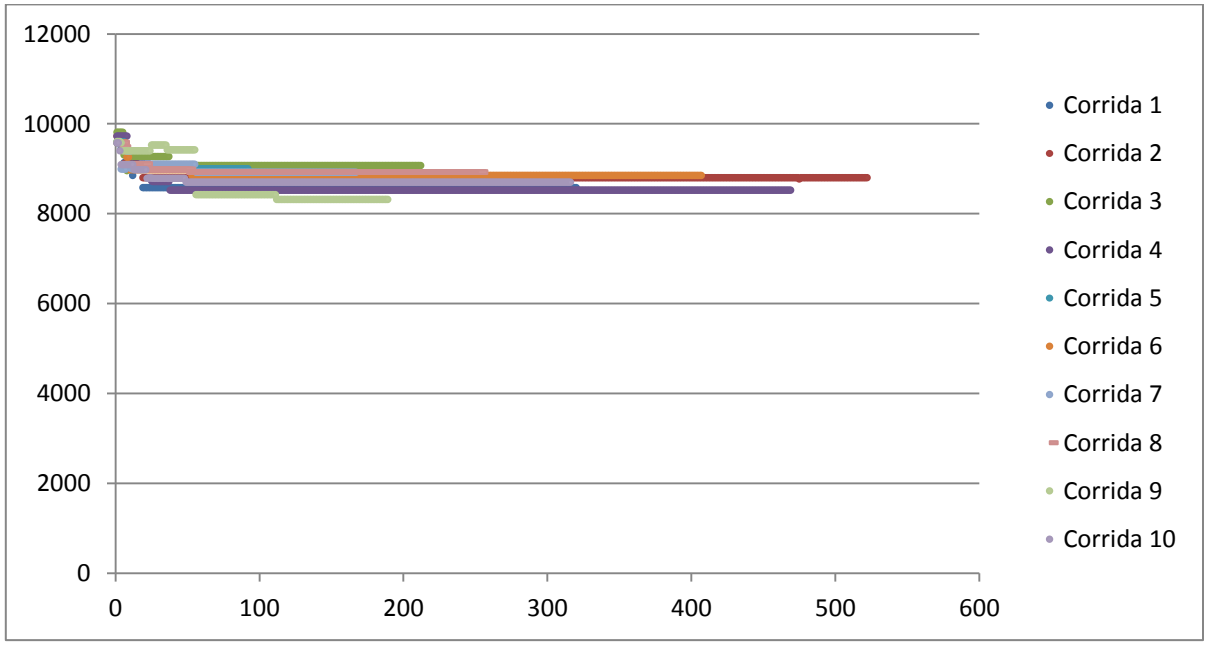


Figura 3.5 Evolución de makespan en las 10 corridas ejecutadas para el caso 2

La mejor solución encontrada fue la corrida 9 con un valor de 8230[min]. La secuencia de esta solución se da a continuación y su correspondiente diagrama de Gantt se muestra en la *Figura 3.6*.

Estación 1	1	3	7	2	5	8	6	4
Estación 2	1	3	7	2	8	6	5	4
Estación 3	1	3	7	2	5	8	6	4
Estación 4	1	3	7	2	5	8	6	4
Estación 5	1	3	7	2	5	8	6	4

En la *Figura 3.7* se muestra la disposición física de las estaciones de trabajo y el flujo que siguen todos los productos en el proceso de fabricación, es decir, todos productos parten

del almacén de materia prima, para luego procesar la primera operación en el área de estampado, para seguir con la construcción de carrocerías, después el área de pintura de carrocerías, seguir con la pintura de partes plásticas, y finalizar con el proceso de montaje.

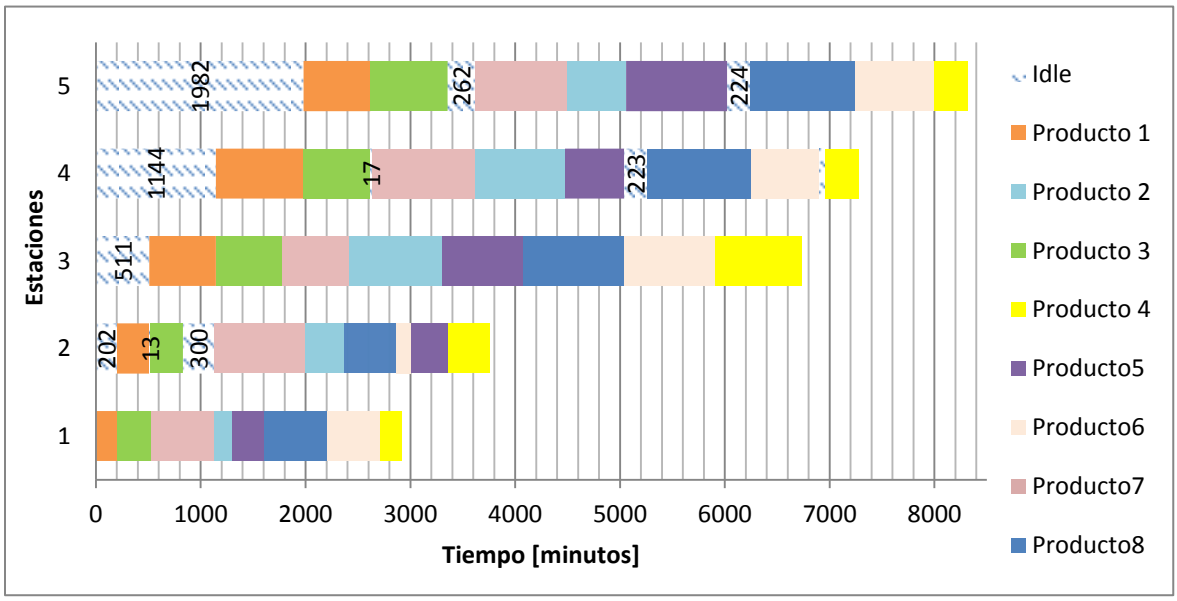


Figura 3.6 Diagrama de Gantt de la mejor solución de todas las corridas ejecutadas para el caso 2. Fuente [propia del autor]

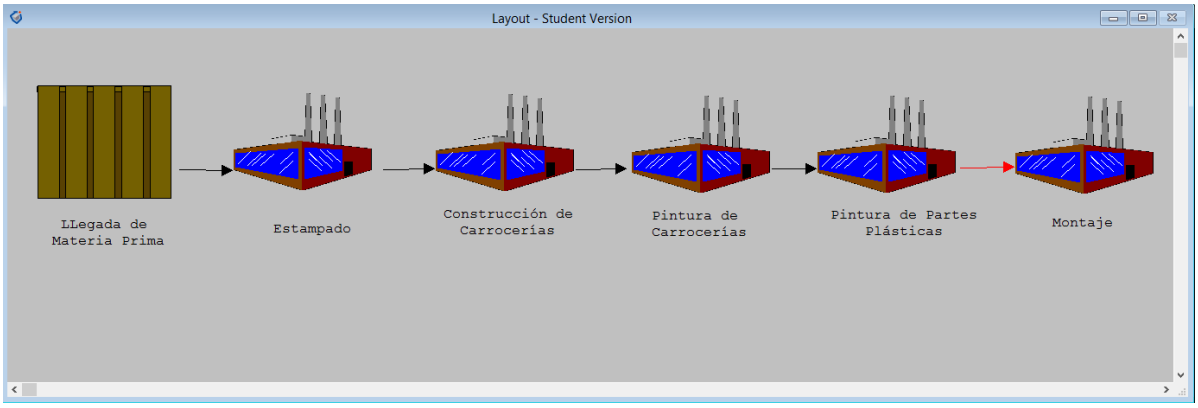
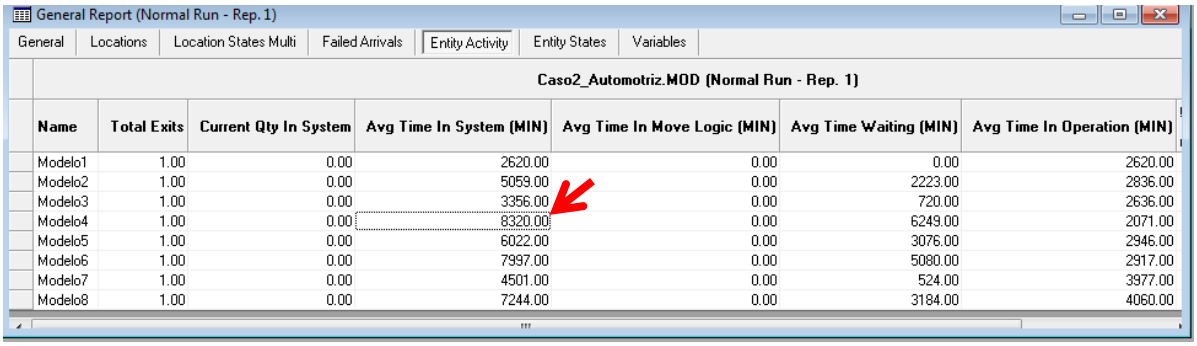


Figura 3.7 Disposición de máquinas para el caso, imagen tomada de Promodel. Fuente [propia del autor]

En la *Figura 3.8* se observa que el modelo 4 que en la solución es el último producto en salir del sistema, finaliza a un tiempo 8230 [min], lo cual comprueba la validez de la dicha solución.



Name	Total Exits	Current Qty In System	Avg Time In System (MIN)	Avg Time In Move Logic (MIN)	Avg Time Waiting (MIN)	Avg Time In Operation (MIN)
Modelo1	1.00	0.00	2620.00	0.00	0.00	2620.00
Modelo2	1.00	0.00	5059.00	0.00	2223.00	2836.00
Modelo3	1.00	0.00	3356.00	0.00	720.00	2636.00
Modelo4	1.00	0.00	8320.00	0.00	6249.00	2071.00
Modelo5	1.00	0.00	6022.00	0.00	3076.00	2946.00
Modelo6	1.00	0.00	7997.00	0.00	5080.00	2917.00
Modelo7	1.00	0.00	4501.00	0.00	524.00	3977.00
Modelo8	1.00	0.00	7244.00	0.00	3184.00	4060.00

Figura 3.8 Estadísticas de cada producto en su proceso de fabricación. Obtenidas de la simulación en ProModel7 para el caso 2. Fuente [propia del autor]

Caso 3: Industria Metalmecánica, Taller tipo Job Shop

Se presenta un caso genérico en la industria metalmecánica, donde cada operación debe procesarse en una única máquina y cada producto tiene un proceso de maquinado diferente.

Lista Estaciones de trabajo o Etapas de producción		
ID	**Nombre**	**Descripción**
1	Máquina 1	
2	Máquina 2	
3	Máquina 3	
4	Máquina 4	
5	Máquina 5	

Tabla 3.7 Lista de las etapas en el proceso de producción

Lista de Productos a Fabricar						
ID	**Nombre**	**Proceso de Fabricación [ID Etapa(tiempo de procesamiento)]**				
1	Producto 1	1 (202 min)	2 (309 min)	3 (633 min)	4 (838 min)	5 (638 min)
2	Producto 2	1 (172 min)	3 (893 min)	2 (355 min)	5 (558 min)	4 (458 min)
3	Producto 3	5 (736 min)	3 (636 min)	2 (306 min)	4 (636 min)	1 (322 min)
4	Producto 4	1 (203 min)	2 (396 min)	3 (826 min)	5 (323 min)	4 (323 min)
5	Producto 5	2 (351 min)	4 (563 min)	1 (303 min)	3 (766 min)	5 (963 min)
6	Producto 6	1 (506 min)	2 (139 min)	3 (866 min)	4 (653 min)	5(753 min)
7	Producto 7	1 (606 min)	2 (869 min)	3 (636 min)	5 (883 min)	4 (983 min)
8	Producto 8	3 (968 min)	1 (607 min)	4 (989 min)	5 (993 min)	2 (503 min)

Tabla 3.8 Lista de productos a fabricar y sus respectivos procesos, los tiempos son dados en minutos.

	Resultados de Ejecutar AG									
Corrida	1	2	3	4	5	6	7	8	9	10
Iteraciones	27	46	39	17	17	22	46	25	44	30
makespan	8034 [min]	8034 [min]	8798 [min]	9040 [min]	9830 [min]	9281 [min]	8990 [min]	8369 [min]	8088 [min]	7966 [min]

Tabla 3.9 Resultados obtenidos de ejecutar 10 corridas para el caso 3

Con los resultados en la *Tabla3.9* obtenidos se tiene un porcentaje de error de **18.96%**, en la *Figura 3.9* se observa la evolución de cada una de estas corridas a lo largo de cada generación hasta llegar a la convergencia.

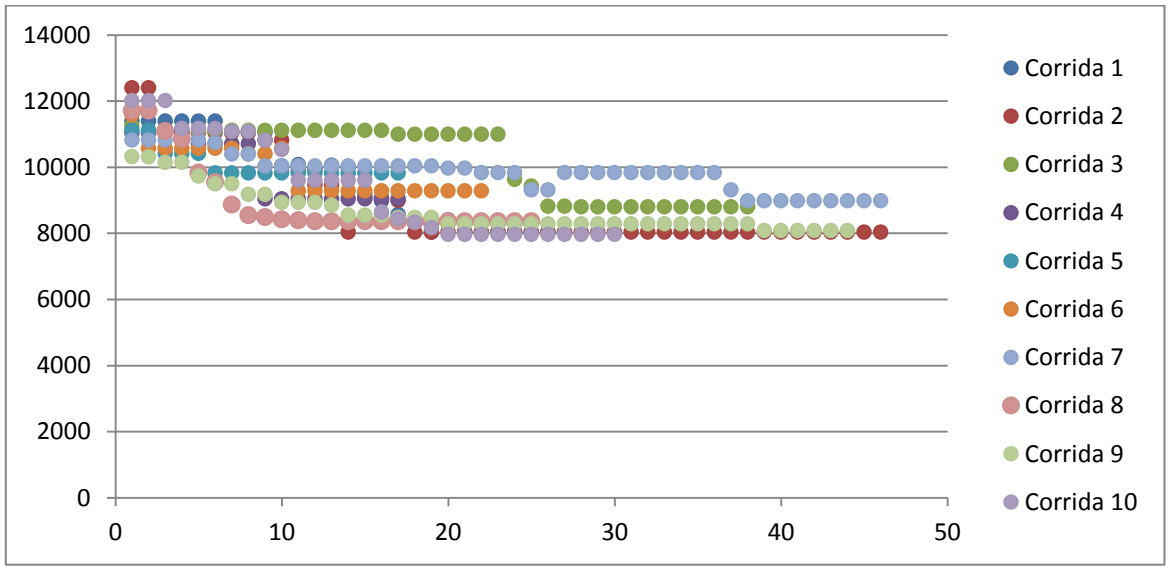


Figura 3.9 Evolución de makespan en las 10 corridas ejecutadas para el caso 3. Fuente [propia del autor]

La mejor solución encontrada es la corrida 10 con un valor de 7996[min]. Y la secuencia de dicha solución se da a continuación y su correspondiente diagrama de Gantt se muestra en la *Figura 3.10.*

Estación 1	1	7	4	6	8	5	2	3
Estación 2	5	1	7	4	6	3	8	2
Estación 3	8	1	7	4	6	3	5	2
Estación 4	1	5	8	6	3	7	4	2
Estación 5	3	1	7	4	8	5	2	6

En la *Figura 3.11* se muestra la disposición física de las estaciones de trabajo y el flujo que

sigue cada uno de los productos en su proceso de fabricación. Todos productos parten de un almacén de materia prima, y debido a cada producto sigue un proceso de fabricación diferente se tienen diferentes rutas a lo largo de las máquinas.

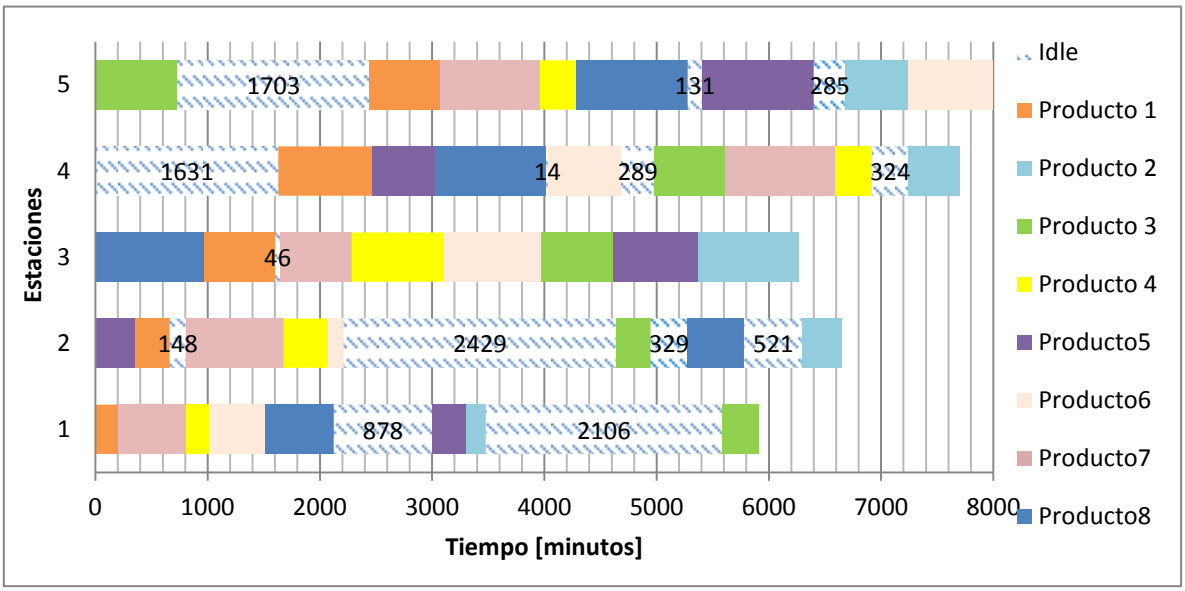


Figura 3.10 Diagrama de Gantt para la mejor solución encontrada en las 10 corridas para el caso 3. Fuente [propia del autor]

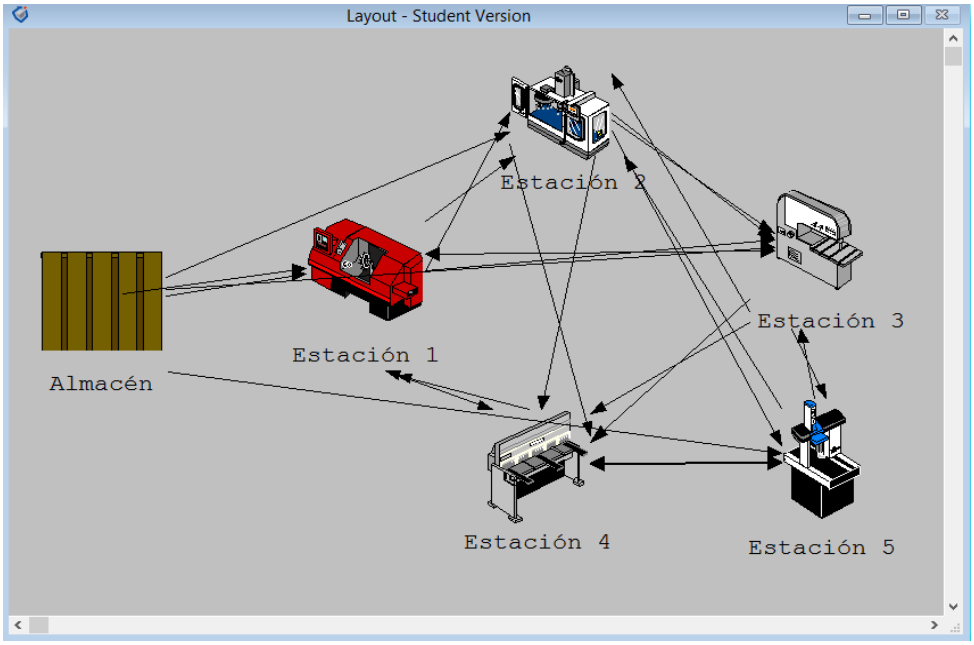


Figura 3.11 Disposición de máquinas para el caso3, imagen tomada de Promodel7. Fuente [propia del autor]

En la *Figura 3.12* se observa que el Producto6 que en la solución es el último producto en terminar su fabricación, finaliza en un tiempo 7996 [min], lo cual comprueba la validez de la solución encontrada.

General | Locations | Location States Multi | Failed Arrivals | Entity Activity | Entity States | Variables

Caso3_metalmecanica.MOD (Normal Run - Rep. 1)

Name	Total Exits	Current Qty In System	Avg Time In System (MIN)	Avg Time In Move Logic (MIN)	Avg Time Waiting (MIN)	Avg Time In Operation (MIN)
Producto1	1.00	0.00	3107.00	0.00	457.00	2650.00
Producto2	1.00	0.00	7671.00	0.00	5235.00	2436.00
Producto3	1.00	0.00	5905.00	0.00	3269.00	2636.00
Producto4	1.00	0.00	6889.00	0.00	4818.00	2071.00
Producto5	1.00	0.00	6370.00	0.00	3424.00	2946.00
Producto6	1.00	0.00	7966.00	0.00	5049.00	2917.00
Producto7	1.00	0.00	6566.00	0.00	2589.00	3977.00
Producto8	1.00	0.00	5809.00	0.00	1749.00	4060.00

Figura 3.12 Resultados obtenidos al simular la mejor solución para el caso 3 en Promodel7. Fuente [propia del autor]

CAPÍTULO 4

Análisis de resultados y conclusiones

En el capítulo 3 se aplicó el algoritmo genético implementado a sistemas de manufactura idealizados, algunos de ellos son sistemas reales (primeros dos casos), para el caso del último es un sistema que puede ser cualquiera que tuviese un comportamiento similar. Es claro que nuestro planteamiento aún desprecia muchos detalles del sistema de manufactura real, por ejemplo cuando se fabrica por pedido, se deben tener fechas de entregas, pedidos que tienen mayor prioridad que otros, también existen restricciones de fabricación entre productos, por ejemplo para algunos productos puede ser más fácil la preparación de las máquinas si se fabrican después de la finalización de algún otro proceso y dificultando su fabricación que si se procesa después de otros. El algoritmo solo es capaz de resolver sistemas que cuentan con una sola máquina o una sola sub-línea de producción en cada estación de trabajo, es incapaz de tratar problemas donde cada estación de trabajo cuente con máquinas en paralelo o máquinas diferentes, algo que en los sistemas reales es muy común. Por ejemplo en el caso prueba 2 de un sistema de producción automotriz donde la estaciones de pintura de carrocerías y de partes plásticas cuentan con 6 y 5 sub-etapas respectivamente no puede ser abordado con el enfoque clásico si se desea una solución precisa, para este caso no existe mucho problema para resolverla de forma ideal porque se hace la suposición de que solo hay una maquina por cada una de estas estaciones de trabajo y que cada modelo a ser procesado en esta estación debe esperar la finalización del modelo procesado actualmente en dicha estación, pero en el sentido estricto la solución arrojada no nos sirve de mucho. Dado este problema se plantea la necesidad de extender el problema de *Scheduling* a un ambiente de manufactura tipo Flow Shop Flexible o Job Shop Flexible con la finalidad de tener una representación más detallada y precisa del sistema y que pueda aplicarse a problemas reales de la industria.

Respecto al algoritmo genético este fue incapaz de resolver instancias muy grandes del problema, se corrió a modo de prueba una instancia con 20 productos y 15 máquinas pero fue imposible su resolución a tal grado que ni se pudo generar una población inicial debido a la enorme cantidad de posibles soluciones no factibles que deben descartarse. Por lo tanto es necesario para el caso de Job Shop determinar reglas que nos permitan descartar soluciones no factibles antes de ser generadas para así poder mejorar los tiempos de ejecución. Para un caso tipo Flow Shop es obvio que el primer producto en ingresar al sistema debe ser el primero en ingresar a todas las máquinas respetando la restricción de proceso, según los primeros dos casos, en las mejores soluciones

encontradas todas las máquinas tenían el mismo producto de inicio así como el de finalización. Para el último caso no se pudo detectar alguna tendencia en la solución que nos ayuden a eliminar soluciones no factibles.

Trabajos a futuro y recomendaciones

Al analizar los resultados obtenidos, como trabajos a futuro se debe extender el modelo utilizado para representar a los sistemas de manufactura de un Job shop a un Flexible Flow Shop o un Flexible Job Shop con el objetivo de tener una representación que pueda ajustarse mejor a los sistemas reales. También el utilizar las redes de Petri si se desea abordar un sistema de manufactura en especial, ya que este proporciona mayor información del sistema aunque con un costo computacional mayor. Optimizar junto al *makespan* otras medidas de desempeños y considerar factores como prioridades, tiempos de entrega y limitaciones físicas.

Para mejorar el algoritmo se recomienda combinar el algoritmo genético con una técnica de aprendizaje por ejemplo las redes neuronales, y así lograr en cada ejecución una mejor solución. Y posiblemente eliminar o al menos reducir el problema de la generación de soluciones no factibles.

BIBLIOGRAFÍA

[1] Caramia Massimiliano and Dell'Olmo Paolo. Effective Resource management in manufacturing Systems: Optimization algorithms for production planning-(Springer series advanced manufacturing), Springer, 2006.

[2] Ponce Cruz Pedro, *Inteligencia Artificial con aplicaciones a la ingeniería*. Primera edición, México D.F., México, Alfaomega, 2010.

[3] Silva Manuel, Valette Robert. *Petri Nets and Flexible Manufacturing*, 2005.

[4] Baker KR. *Introduction to Sequencing and Scheduling*, New York: John Wiley & Sons, 1974.

[5] Fox M.S. and Sadeh N. *Why is scheduling difficult? A CSP perspective*. In proceedings of the 9th European Conference on Artificial Intelligence, ECAI-90, pp. 754-767, 1990.

[6] Petri C.A. 1962. *Kommunikation mit Automaten*. Schriften des IIM Nr. 3. Institute für Instrumentelle Mathematik, Bonn. English translation: *Communication with Automata*. Tech Rep. RADC-TR-65-377, vol 1, 1966. Griffiss Air Force Base, New York.

[7] Jiménez A.P., Muñoz C.A. y Toro E.M.. *Solución del problema de Flow Shop Flexible Aplicando el Algoritmo Genético de Chu Bealsey.* Entre Ciencia e Ingeniería, Año 7, pp. 24 40, 2013

[8] Nowicki E. and Smutnicki C. *A fast taboo search algorithm for the job shop scheduling problem*. Management Science, pp. 797-813, 1996.

[9] Nowicki E. and Smutnicki C. An advanced tabu search algorithm for the job shop problem. Journal of Scheduling, pp. 145-159, 2005.

[10] Stecco G. and Cordeau J.-F. A tabu search heuristic for a sequence-dependent and time-dependent scheduling problem on a single machine. Journal of Scheduling, pp. 3-16, 2009.

[11] Bilge U., Kiraç F., Kurtulan M., and Pekgün P. *A tabu search algorithm for parallel machine total tardiness problem*. Computers & Operations Research, pp. 397- 414, 2004.

[12] Ho N.B. and Tay J.C. *GENACE: An efficient cultural algorithm for solving the flexible job-shop problem*. Congress on Evolutionary Computation, pp. 1759-1766, 2004.

[13] Ho N.B. and Tay J.C. *Evolving dispatching rules for solving the flexible job-shop problem*. IEEE Congress on Evolutionary Computation. Vol. 3, pp. 2848-2855, 2005.

[14] M. Dell' Amico and M. Trubian. *Applying tabu search to the job-shop scheduling problem*. Annals of Operational Research, pp. 231-252, 1993.

[15] Roy B. and Sussmann B. *"Les problèmes d'ordonnancement avec constraintes dijonctives"*. Note D.S. no.9 bis, SEMA, Paris, France, Décembre (1964)

[16] Murata T. *"Petri Nets: Properties, Analysis and Applications"*, Processings of the IEEE, vol. 77, no. 4, 1989.

[17] Glover F., C. McMilan and B. Novick. *"Iteractive Desición Software and Computer Graphics for Architectural and Space Planinning".* Annal Operations Resource, Vol. 5, pp. 557-573, 1985.

[18] Archetti F., Lucertini M. and Serafini P. *Operations Research Models: in Flexible Manufacturing Systems* (New York: Springer Verlag),1989.

[19] Dorigo M. *Optimization, Learning and Natural Algorithms (in Italian).* PhD thesis, Dipartimento di Elettronica e Informazione, Politecnico di Milano, IT, 1992.

[20] ElMaraghy H. A. *Flexible and reconfigurable manufacturing systems paradigms.* Reconfigurable Manufacturing Systems, Kluwer Academic Publishiers (2005).

[21] Holsapple C., V. Jacob, R. Pakath and J. Zavari. *"A Genetics-Base d Hybrid Scheduler for generating Static Schedules in Schedules in Flexible Manufacturing Contexts".*IEEE Transactions on System, Mans and Cybrnetics, 23(4), pp.953-971, 1993.

[22] Jun Woo Kim. *A Job Shop Shceduling Game with GA-based Evaluation,* Applied Mathematics & Information Sciences: An international Journal, No.5, pp. 2627-2634, 2014.

[23] Liang Sun, Xiachun Cheng, Yanchun Liang. *Solving Job Shop Problem Using Genetic Algorithm with Penalty Function*. International Journal of Intelligent Information Processing, Volume 1, No. 2, 2010.

[24] Amit Kummar and Rajnesh Singh. *Performance Genetic Algorithm For Solving Flexible Job-Shop Scheduling Problem.* Internacional Journal of Information Technology and Knowledge Management, Volume 4, No. 1, pp.105-108, 2011.

[25] Sureshkummer S., Saravanan. G., S. Thiruvenkadam. *Optimizing Makespan in JSSP Using Unordered Exchange Crossover in GA.* IOSR Journal Computer Engineering, Volume 8, pp. 41-46, 2013.

[26] Fattani P., M.S. Mehrabad and F. Jolai. *Mathematical Modeling and Heuristic Approaches to Flexible Job Shop Scheduling Problem*. Journal of intellgence Manufacturing, Volumen 18, pp. 331-342.

[27] Dell' Amico M. and Trubian M. *Applying tabu-Search to the Job Shop Scheduling Problem.* Annals of Operational Resource, 41 pp. 231-252, 2013.

[28] Kirkpatrick S., Gelatt D., Vecchi M., *Optimazation by simulated annealing*. Science, vo.

220, pp. 671-680,1983.

[29] Taillard E.D. *Benchmarks for basic shceduling problems.* European Journal of Operational Research, pp.278-285,1993.

[30] Taillard E.D. Parallel taboo search techniques for the job shop shceduling problema. RSA J. Compute, vol. 6, pp. 108-117, 1994.

[31] B. M. T. Lin, C. Y. Lu, S. J. Shyu, and C. Y. Tsai, "Development of new features of ant colony optimization for flowshop scheduling", *International Journal of Production Economics*, vol. 112, pp. 742-755, 2008.

[32] Brucker Peter, Bernd Jurisch, Bernd Sievers. *"A branch and Bound algorithm for the Job Shop Scheduling Problem"*. Discrete Applied Mathematics, Vol. 49, pp. 107-127, 1994.

[33] Hariri A.M.A. and Potts C.N. *"A Branch and Bound Algorithm for Job Shop´Scheduling"*. J.K.A.U. Sci, Vol.3, pp.201-209, 1991.

[34] M. Omkumar and Shahabudeen. *"Ant Colony Optimazation for Multilevel Assembly Job Shop Scheduling"*. The International Journal of Applied Management and Technology, Vol 6.

[35] Ruin Zhang. "A simulated Annealing-based Heuristic Algorithm for Job Shop SCheduling to Minimi Lateness". International Journal of Advance Robotic Systems, Vol. 10, 2013.

[36] Zhang Jun, Xiaomin Hu, X. Tan, J.H. Zhong and Q. Huang. *"Implementation of an Ant Colony Optimization technique for Job Shop SCheduling Problem"*. Transacions of the Institute of Measurement and Control, Vol. 28, pp. 93-108, 2006.

[37] Christian Artigues and Dominique Feillet. *"A Branch and Bound method for the Job Shop Problem with Sequence-dependent setup times"*. Annals of Operations Research, Springer Verlag 2007.

[38] www.stps.gob.mx.mx/DGIFT_STPS/PDF Consulta: 9/Diciembre/2015