UNIVERSIDAD AUTÓNOMA DEL
ESTADO DE MORELOS

FACULTAD DE CIENCIAS QUÍMICAS E INGENIERÍA

CENTRO DE INVESTIGACIÓN EN INGENIERÍA
Y CIENCIAS APLICADAS

ESTRUCTURA HÍBRIDA DE VECINDAD PARA EL PROBLEMA DE CALENDARIZACIÓN DE TRABAJOS EN TALLERES DE MANUFACTURA FLEXIBLE

TESIS PROFESIONAL
PARA OBTENER EL GRADO DE:

MAESTRÍA EN INGENIERÍA Y CIENCIAS APLICADAS

P R E S E N T A:
ING. RENÉ LÓPEZ RUIZ

ASESOR: DR. MARCO ANTONIO CRUZ CHÁVEZ
COASESOR: DR. MARTÍN GERARDO MARTÍNEZ RANGEL

CUERNAVACA, MOR. AGOSTO 2014

Resumen

En este trabajo de investigación se desarrolló un algoritmo de búsqueda local iterada, para el cual se propuso una estructura de vecindad híbrida, con el objetivo de mejorar el desempeño de dicho algoritmo para el problema de calendarización de trabajos en un taller de manufactura flexible. Se siguió una metodología de sintonización que permitió realizar el análisis de sensibilidad de los parámetros de control del algoritmo, teniendo como resultado trabajar éste con el mejor rendimiento tanto en eficiencia como en eficacia.

Se tomó el modelo de grafo disyuntivo para representar al problema de calendarización de trabajos en un taller de manufactura flexible. Por su naturaleza, este modelo es considerado NP-Duro, por lo que se justifica el uso de métodos heurísticos para tratar dicho problema.

El algoritmo propuesto cuenta con un método de balanceo de carga, con el objetivo de mantener el equilibrio del número de operaciones que realiza cada máquina. Inicialmente, en este trabajo se realizó un estudio de cada una de las estructuras de vecindad que componen a la estructura de vecindad híbrida por separado, con el fin de conocer la calidad de la solución que dichas estructuras dan a las instancias de prueba y proponer una estructura híbrida con las más eficientes y eficaces. El análisis de resultados se llevó a cabo comparando la búsqueda local con otros resultados dados por heurísticas tomadas de la literatura. Las pruebas experimentales se realizaron tomando cuatro instancias consideradas como pequeñas, medianas y grandes. De acuerdo al análisis experimental realizado, la estructura de vecindad híbrida demostró ser la mejor en eficacia y competitiva en eficiencia.

Abstract

In this research an algorithm of iterated local search, applying a hybrid neighborhood structure were proposed in order to improve the performance of this algorithm to the flexible job shop scheduling problem. Tuning methodology was followed, which allowed for the sensitivity analysis of the control parameters of the algorithm, enabling this to work with the best performance both in efficiency and effectiveness.

Disjunctive graph model was taken to represent the flexible job shop scheduling problem. By its nature, this model is considered NP-Hard, making the use of heuristic methods to treat this problem is warranted.

The proposed algorithm has a method of charge balancing, whit the objective to of number of operations that do every machine. At the beginning, we did an analysis of each one of the neighborhood structures in order to know the solution quality of each structure from instances proof and propose a hybrid structure with the most efficient and effective. The analysis of results was carried out by comparing the local search results given by other heuristics taken from the literature. Experimental tests were made on four instances considered small, medium and large. According to the experimental analysis, the hybrid neighborhood structure proved to be the best in efficiency and competitive effectiveness.

Agradecimientos

A CONACYT por brindarme el apoyo económico para la realización de mis estudios de posgrado.

Al Dr. Marco Antonio Cruz Chávez, director de este trabajo de investigación, por la orientación y consejos para la realización y culminación de este trabajo.

A los integrantes del comité tutoral y revisores de tesis: Dr. Martin G. Martínez Rosales, Dr. Martin H. Martínez Cruz, Dra. Margarita Tecpoyotl Torres, Dr. Fredy Juárez Pérez, por sus comentarios, orientación y consejos para la realización de este trabajo de investigación.

Dedicatorias

Primeramente, a Dios que sin Él no hubiera logrado nada.

A mis padres Luis López Gómez y Paulina Ruiz Vázquez, por su amor, apoyo moral y porque siempre me han impulsado a lograr mis metas. Son mi ejemplo a seguir, si hubiera otra vida desearía volver a ser su hijo.

A mi esposa Olga L. Cruz, por su apoyo incondicional y por todo lo que directa o indirectamente provoca en mi vida. Te amo.

Al Dr. Marco A. Cruz, por el apoyo y consejos que han sido útiles para la elaboración de este trabajo y para mi crecimiento profesional.

A mi hermanos María, Luis y Karla, por sus consejos y por los momentos familiares que pasamos tan a gusto.

A mis compañeros, por la amistad, apoyo y por los momentos inolvidables que pasamos juntos.

Nomenclatura

BLI Siglas del algoritmo búsqueda local iterada.

BL Siglas del algoritmo búsqueda local.

SA Siglas en inglés para recocido simulado (simulated annealing)

GA Siglas en inglés para algoritmo genético (genetic algorithm).

TS Siglas en inglés para búsqueda tabú (tabu search).

JSSP Siglas en inglés para el problema de calendarización de trabajos en un taller de manufactura (job shop scheduling problema).

FJSSP Siglas en inglés para el problema de calendarización de trabajos en un taller de manufactura flexible (flexible job shop scheduling problem).

Nomenclatura del problema FJSSP y método de solución

N Número total de trabajos

M Número total de máquinas

j Trabajo que necesita ser calendarizado**.**

I Máquina en la que va a ser procesado un trabajo j

k Posición de una máquina *i* en la que va a ser procesado un trabajo *j.*

nop Número de operación del trabajo j.

CP_BL Criterio de paro de la Búsqueda Local.

CP_BLI Criterio de paro de la Búsqueda Local Iterada.

ÍNDICE GENERAL

ÍNDICE DE TABLAS i

ÍNDICE DE FIGURAS ii

CAPÍTULO I. INTRODUCCIÓN 1

1.1 Introducción 2

1.2 Estado del arte 5

1.3 Objetivo general 8

1.4 Objetivos específicos 8

1.5 Alcance de la Investigación 8

1.6 Contribución de la tesis 9

1.7 Organización de la tesis 9

CAPÍTULO II. CALENDARIZACIÓN DE TRABAJOS EN TALLERES DE MANUFACTURA FLEXIBLE 11

2.1 Introducción 12

2.2 Descripción conceptual 13

2.3 El modelo de grafo disyuntivo 14

2.3.1 Tipos de problemas de calendarización flexible 15

2.3.2 Representación de un FJSSP 16

2.4 Modelo matemático 18

CAPÍTULO III. ALGORITMO BÚSQUEDA LOCAL ITERADA 22

3.1 Introducción 23

3.2 Búsqueda local iterada 23

3.3 Estructura de vecindad híbrida 25

3.3.1 Vecindad N1 28

3.3.2 Vecindad N4 29
3.3.3 Vecindad N5 30
3.3.4 Vecindad N6 30
3.3.5 Vecindad híbrida 31
3.4 Algoritmo secuencial para FJSSP 32
3.5 Estrategia de sintonización 38
3.6 Complejidad del algoritmo búsqueda local iterada 41
CAPÍTULO IV. RESULTADOS EXPERIMENTALES 43
4.1 Descripción del equipo utilizado 44
4.2 Análisis de factibilidad de estructuras N4 y N6 44
4.3 Análisis de sensibilidad 48
4.4 Resultados experimentales 51
4.5 Análisis de eficacia y eficiencia del algoritmo 52
4.5.1 Análisis de eficacia 53
4.5.2 Análisis de eficiencia 56
4.6 Análisis de resultados 60
CAPÍTULO V. CONCLUSIONES Y TRABAJOS FUTUROS 63
5.1 Conclusiones 64
5.2 Trabajos futuros 65
Referencias 66
APÉNDICES 76
A. Estructuras de datos utilizadas para la representación simbólica en el algoritmo de búsqueda local iterada. 76
B. Código fuente del algoritmo búsqueda local iterada. 78

C. Análisis de la complejidad del algoritmo 80
D. Glosario de términos..... 84

ÍNDICE DE TABLAS

Tabla 4. 1 Equipo de cómputo utilizado 44
Tabla 4. 2 Características de los benchmarks utilizados 44
Tabla 4. 3 Movimientos factibles e infactibles de la estructura de vecindad N4 47
Tabla 4. 4 Movimientos factibles e infactibles de la estructura de vecindad N6 48
Tabla 4. 5 Rangos utilizados en los parámetros de control para el análisis de sensibilidad. 49
Tabla 4. 6 Valores de CP_BL..... 50
Tabla 4. 7 Valores sintonizados correspondientes a los parámetros de control evaluados durante el análisis de sensibilidad 51
Tabla 4. 8 Resultados de las estructuras de vecindad N1 y N4..... 53
Tabla 4. 9 Resultados de las estructuras de vecindad N5 y N6..... 53
Tabla 4. 10 Resultados de la mejor estructura contra la estructura híbrida de vecindad 54
Tabla 4. 11 Error relativo obtenido de la ejecución del algoritmo Búsqueda Local Iterada 55
Tabla 4. 12 Tiempos promedio de ejecución en segundos requeridos por cada estructura de vecindad 59

ÍNDICE DE FIGURAS

Figura 2. 1 Calendarización de flexibilidad-total..... 15
Figura 2. 2 Calendarización de flexibilidad-parcial..... 16
Figura 2. 3 Instancia de un FJSSP de flexibilidad parcial 3 x 3 16
Figura 2. 4 Representación propuesta de un FJSSP de 3 trabajos x 3 máquinas 17

Figura 3. 1 Algoritmo general de Búsqueda Local Iterada..... 24
Figura 3. 2 Representación del espacio de soluciones con una vecindad N(s) = {s'} [Gu y Huang, 1994]..... 26
Figura 3. 3 Algoritmo general de una búsqueda por vecindad..... 27
Figura 3. 4 Ruta crítica de una solución factible [Martínez-Rangel, 2008].... 28
Figura 3. 5 Esquema de movimientos N1 [Laarhoven *et al.,* 1992]..... 29
Figura 3. 6 Esquema de movimientos N4 [Dell'Amico y Trubian, 1993]. 29
Figura 3. 7 Esquema de movimientos N5 [Nowicki y Smutniki, 1996] 30
Figura 3. 8 Esquema de movimientos N6 [Balas y Vazacopoulos, 1998]. 31
Figura 3. 9 Estructura Híbrida de vecindad..... 32
Figura 3. 10 Algoritmo BLI propuesto para el FJSSP 33
Figura 3. 11 Ejemplo del cálculo de la ruta crítica..... 34
Figura 3. 12 Bloques de la ruta crítica 37
Figura 3. 13 Deshacer perturbación..... 38

Figura 4. 1 Solución factible de 3 trabajos y 3 máquinas..... 46
Figura 4. 2 Movimiento N4 factible hacia atrás 46
Figura 4. 3 Movimiento N4 factible hacia adelante 47
Figura 4. 4 Valores de CP_BLI donde el algoritmo converge para las estructuras de vecindad..... 50
Figura 4. 5 Resultados obtenidos de las estructuras de vecindad..... 55
Figura 4. 6 Gráfica comparativa del erro relativo de las estructuras de vecindad e híbrida..... 56

Figura 4. 7 Análisis de eficiencia benchmarking mk01 57
Figura 4. 8 Análisis de eficiencia benchmarking mk02 57
Figura 4. 9 Análisis de eficiencia benchmarking mk03 58
Figura 4. 10 Análisis de eficiencia benchmarking mk07 58
Figura 4. 11 Eficiencia de las estructuras de vecindad vs estructura híbrida 60
Figura 4. 12 Gráfica comparativa del funcionamiento de los algoritmos evaluados con respecto a la solución óptima. 61
Figura 4. 13 Gráfica comparativa del error relativo obtenido por cada uno de los algoritmos evaluados. 62

CAPÍTULO I. INTRODUCCIÓN

En este capítulo se da la introducción del problema de calendarización de tareas en talleres de manufactura flexible, los principales métodos y los algoritmos más eficientes que han abordado este tipo de problemas, así como los objetivos y alcances de la investigación.

1.1 Introducción

El problema de asignación de tareas en un taller de manufactura JSSP (por sus siglas en inglés, Job Shop Scheduling Problem) es uno de los problemas de calendarización más conocidos y difíciles de resolver. JSSP es probablemente el modelo más estudiado y desarrollado de todos los problemas pertenecientes al problema general de calendarización [Pinedo M., 2001]. Esto sirve como una referencia para otras técnicas que tratan de resolver problemas en el mismo campo, por ejemplo: el problema del transporte y el problema de la mochila [Garey y Johnson, 1976].

El JSSP es un problema de secuenciar operaciones en máquinas; la secuencia de operaciones factibles se denominada un *Schedule.* El JSSP se presenta por lo general en la industria de manufactura y se asocia a la problemática de mejorar la eficiencia de los recursos disponibles en un sistema para la producción; en dicho sistema los recursos corresponden a un conjunto *M* de máquinas. En el sistema, se ejecuta un conjunto de trabajos y por cada trabajo se requiere efectuar un cierto número de operaciones, en un orden específico de acuerdo a los requerimientos de los trabajos. Una vez definido dicho orden, el JSSP consiste en encontrar la secuencia de ejecución de operaciones para cada máquina, que permita hacer eficiente el desempeño total del sistema, además, posteriormente todas las operaciones se deben calendarizar asignándoles un tiempo de inicio de acuerdo al *Schedule* encontrado. Es decir, en el JSSP lo que se busca es minimizar los costos de producción a través de una buena planificación horaria de actividades o *Scheduling* [Cruz-Chávez, 2005].

En la actualidad los procesos de manufactura se han vuelto muy complicados debido a que las máquinas que intervienen en estos trabajos, pueden ejecutar un mayor número de operaciones en lugar de una operación en cada máquina como previamente se realizaba [Sule D., 1997]. En la

programación de la producción de las industrias manufactureras, se debe decidir sobre la asignación de los recursos o trabajos para optimizar uno o más objetivos en el corto plazo. Para apoyar estas decisiones tradicionalmente se utiliza el modelo del *Job Shop*, el cual se considera un conjunto de máquinas y trabajos, compuestos por una secuencia ordenada de operaciones que se deben procesar en las máquinas con el objetivo de minimizar; entre otros, como es el caso de ésta tesis que aborda el problema de determinar el tiempo de finalización de la última operación conocido como: *makespan*. Debido a la alta competencia en los mercados de productos manufacturados, en la actualidad se ha dado una mayor atención a un problema de calendarización [Pinedo M., 2001], debido a que se presenta frecuentemente en la industria y cuya solución permite hacer más eficientes los procesos de manufactura [Adams *et al*, 1988].

El problema del *Flexible Job Shop* (FJSP) es una generalización del problema clásico del *Job Shop* (JSP) que *c*onsidera que las operaciones pueden ser procesadas por un grupo de máquinas, por tal motivo se apega más a los sistemas de manufactura reales. Debido a la necesidad adicional de asignar las operaciones a las máquinas, el FJSP es más complejo que el JSP e incorpora todas las restricciones y complejidad del JSP [Zhang *et al.*, 2009].

Para instancias grandes no existe un algoritmo determinístico que pueda resolver los problemas de calendarización. Por esta razón, se utilizan metaheurísticas para la búsqueda del óptimo global [Applegate y Cook, 1991], a fin de generar algoritmos que puedan dar buenas soluciones acotados en tiempo polinomial. En muchos ambientes de manufactura que tienen que ver con el control de la producción y/o la planeación, el problema de calendarizar trabajos en un taller de manufactura flexible FJSSP (Flexible Job Shop Scheduling Problem) reviste gran importancia dentro de los procesos de fabricación. Hoy en día, las máquinas utilizadas son de

multipropósito, y en consecuencia pueden realizar múltiples operaciones, lo que permite que los procesos se puedan realizar en más de una máquina [Martínez-Oropeza, 2010]. El FJSSP por ser una generalización del JSSP clásico y en consecuencia, al ser una extensión de este, también es NP-Duro ya que incorpora todas las dificultades y complejidades de su antecesor JSSP. Tanto JSSP como FJSSP por su naturaleza misma, son uno de los problemas más difíciles de resolver dentro del grupo de problemas encontrados en el área de optimización discreta [Garey *et al.*, 1976; Applegate y Cook, 1991; Laarhoven *et al.*, 1992; Amico y Turbian, 1993; Nowicki y Smutnicki, 1993; Burke y Smith, 1997; Mastrolilli y Gambardella, 1998; Kacem *et al.*, 2002; Ho y Tay, 2004; González, *et al.*, 2005].

El FJSSP puede ser agrupado de acuerdo a las características de los problemas. Un grupo integra a los problemas con flexibilidad total, en el cual, cualquier operación puede disponer del total de máquinas que forman del proceso y debe seleccionarse una de ellas que la pueda procesar. Por otra parte, está el grupo de problemas de flexibilidad parcial, donde no todas las máquinas están disponibles para todas las operaciones, este último tipo de problemas es el más duro de tratar dentro de los FJSP [Yuan *et al.,* 2012].

En este trabajo de investigación se abordan problemas del tipo flexibilidad parcial por la gran similitud que existe en ambientes de sistemas de manufactura reales conocidos como sistemas de manufactura flexible (SMF). Un SMF incorpora el proceso distribuido de información y el flujo automatizado de materiales a través de maquinaria controlada por computadora, celdas de ensamble, robots industriales, máquinas de inspección, sistemas de manejo y almacenamiento de materiales e integrados por sistemas inteligentes [Rankyn P., 1983; Klahorts, 1981]. Al incrementarse el volumen de producción se recomienda un SMF, en este sistema también se realizan varias operaciones sobre el material antes de salir del mismo. Sin embargo, aquí no todas las operaciones se realizan en

un solo centro de maquinado, por lo que se requiere que la pieza o material se transfiera en forma automática de un centro de maquina a otro [Martínez-Rangel, 2008].

1.2 Estado del arte

Inicialmente el estudio del FJSSP se remonta a 1990, Bruker y Schlie [1990] propusieron un algoritmo en tiempo polinomial para resolver el FJSSP con dos trabajos. Pero de manera general, está considerado como un problema NP-Hard. Debido a esta propiedad, los algoritmos exactos no son efectivos para resolver FJSSP, especialmente los problemas considerados como grandes. Durante las dos últimas décadas, las metaheurísticas se han vuelto muy populares para tratar este tipo de problemas. Como recocido simulado (AS) [Najid *et al.*, 2002], búsqueda tabú (TS) [Hurink *et al.*, 1994; Dauzère-Pérès y Paulli, 1997; Mastrolilli y Gambardella, 2000], algoritmos genéticos (GA) [Chen *et al*. 1999; Jia *et al.*, 2003; Zhang y Gen, 2005; Pezzella, 2008; Zhang *et al.*, 2011], optimización por enjambre de partículas (PSO) [Girish y Jawahar, 2009], optimización basada en biogeografía (BBO) [Rahmati y Zandieh, 2012] y técnicas hibrídas [Xia y Wu, 2005; Fattahi *et al.*, 2007; Gao *et al.*, 2008; Li *et al.*, 2011; Raeesi y Kobti, 2012]. Estos métodos han sido aplicados con el objetivo de encontrar una calendarización en tiempo computacional aceptable y se pueden clasificar en dos principales categorías: enfoque jerárquico y enfoque integrado.

El enfoque jerárquico reduce la dificultad del FJSSP descomponiendo en dos sub-problemas: ruteo y planificación, en donde ambos problemas son tratados por separado. La idea es natural para resolver el FJSSP, porque cuando se fija la asignación de la máquina para cada operación, ingresa el sub-problema de planificación en el FJSSP. Brandimarte [1993] fue el primero en adoptar el enfoque jerárquico para el FJSSP. Resolvió el sub-problema del ruteo usando algunas reglas de despacho y después resolvió el

JSP resultante usando la heurística TS. Tung *et al*., [1999] presentaron un algoritmo similar para el sistema de calendarización en talleres de manufactura flexible. Kacem *et al*. [2002] propusieron un algoritmo genético controlado por el modelo asignado que es generado por el enfoque de la localización. Xia y Wu [2005] presentaron un método hibrido efectivo para el multi-objetivo FJSSP. Usaron el método hibrido de partículas (PSO) para asignar operaciones en máquinas y un algoritmo recocido simulado para secuenciar las operaciones en cada máquina. Fattahi *et al*. [2007] desarrollaron cuatro tipos de enfoques, jerárquicos basados en recocido simulado y búsqueda tabú para resolver el FJSSP.

El enfoque integrado es usado considerando la asignación y planificación al mismo tiempo y se ha demostrado que tiene un mejor rendimiento pero es más difícil de diseñar. Hurink *et al*. [1994] presentaron una heurística TS en donde el reasignamiento y recalendarización son considerados como dos diferentes tipos de movimientos. Dauzere-Péres y Paulli [1997] desarrollaron un procedimiento TS basado sobre una nueva estructura de vecindad que no hace distinción entre reasignación y recalendarización. Mastrolli y Gambardella [2000] mejoraron su técnica de TS y propusieron dos funciones de vecindad para el FJSP. Chen *et al.* [1999] propusieron un GA efectivo para resolver el FJSP. En su método, la representación del cromosoma es dividido en dos partes, la primera denota la asignación concreta de las operaciones para cada máquina y el segundo describe la secuencia de las operaciones dentro de la máquina. Jia *et al*. [2003] propusieron una modificación al GA capaz de resolver problemas de calendarización distribuida y FSJSP. Zhang y Gen [2011] propusieron un GA basado en múltiples etapas para tratar el problema desde el punto de vista de la programación dinámica. Pezzella *et al*. [2008] presentaron un GA donde mezcla diferentes estrategias para generar la población inicial, seleccionando los individuos para reproducción, e integrando la reproducción con nuevos individuos. Gao *et al*. [2008] combinaron un algoritmo genético con un

procedimiento descendente de vecindario variable (VND) para resolver un multi-objetivo FJSSP. Recientemente, se desarrollaron más metaheurísticas basadas en el enfoque integrado para resolver el FJSSP. Bagheri *et al*. [2010] presentaron un algoritmo inmune artificial (AIA) usando algunas reglas útiles. Yazdani *et al.* [2010] propusieron un algoritmo de búsqueda de vecindad variables paralela basado en la aplicación de búsquedas independientes múltiples. Xing *et al.* [2010] desarrollaron un algoritmo basado en la optimización de la colonia de hormigas que provee una efectiva integración entre colonia de hormiga y un modelo de optimización conocido. Li *et al*. [2011] propusieron un TS híbrido con una eficiente estructura de vecindad para el FJSSP. Wang *et al*. [2011] y Wang et al. [2012] aplicaron un algoritmo de colonia de hormigas artificial y un algoritmo de estimación distribuida (EDA) para el FJSP, ambos con el balance de estrés entre la exploración global y la explotación local.

Las metaheurísticas utilizadas para problemas de calendarización son caracterizadas por búsquedas a través de vecindades. Por esta razón, es importante el desarrollo de mecanismos más eficientes y efectivos que aceleren las búsquedas para mejorar estas metaheurísticas. Un gran número de metaheurísticas han sido propuestas para encontrar buenas soluciones para el FJSSP en tiempo polinomial. En la literatura se encuentran diversos enfoques que han sido propuestos y que permiten encontrar buenas soluciones para el FJSSP. Todos estos enfoques requieren evaluar la calidad de las soluciones para encontrar el makespan (valor de la función objetivo del problema) durante cada paso del algoritmo de la metaheurística. Para lograr esto, el algoritmo de calendarización generalmente utilizado es el propuesto por [Nakano y Yamada, 1995]. Cada vez que la metaheurística obtiene una nueva solución (calendarización), este algoritmo es aplicado para la solución a fin de asignar un tiempo de inicio para cada una de las operaciones que son parte del FJSSP y obtener un valor de makespan [Martínez-Rangel, 2008].

1.3 Objetivo general

Desarrollar un algoritmo de búsqueda local iterada utilizando una estructura de vecindad híbrida propuesta para resolver el problema de calendarización de tareas en un taller de manufactura flexible.

1.4 Objetivos específicos

- Proponer una estructura híbrida de vecindad que mejore el desempeño del algoritmo de búsqueda local iterada, utilizando funciones de vecindad que estén catalogadas como las de mejor resultado en la literatura aplicadas a FJSSP.
- Implementar en forma secuencial el algoritmo de búsqueda local iterada utilizando la estructura de vecindad híbrida
- Analizar la eficiencia y la eficacia del algoritmo de búsqueda local iterada propuesto, respecto a los obtenidos con otros resultados encontrados en la literatura.

1.5 Alcance de la Investigación

- El algoritmo de búsqueda local iterada propuesto contará con dos partes fundamentales, la primera es la asignación de las máquinas y la planificación de los trabajos, tomando en cuenta el sobrecargo de trabajo en las máquinas y la segunda comprende la implementación de la estructura híbrida, que permita una mejor explotación y exploración del espacio de soluciones.
- Se aplicará un estudio de sensibilidad a los parámetros de control del algoritmo propuesto basado en la metodología de sintonización propuesta por [Martínez-Oropeza, 2010], que se explica en el capítulo 4.

- Los resultados obtenidos durante las pruebas realizadas al algoritmo de búsqueda local iterada se comparan con los obtenidos por otros algoritmos, bajo las mismas instancias de pruebas.
- Este trabajo de investigación abordó problemas del tipo flexibilidad parcial, ya que tienen mayor similitud a los sistemas de manufactura actuales.
- En la investigación no se trabajó con tiempos de preparación, limpieza y espera en las máquinas, conocidos como set-up times, se considera que estos tiempos van implícitos en los tiempos de procesamiento.

1.6 Contribución de la tesis

Las aportaciones de la presente investigación son:

- Una estructura de vecindad hibrida basada en el estudio e implementación de cuatro estructuras de vecindad (N1, N4, N5 y N6) aplicadas al FJSSP reportadas con los mejores resultados, con el objetivo de evaluar la eficacia y eficiencia.
- Un análisis de factibilidad de las estructuras de vecindad N4 y N6, para conocer el porcentaje de movimientos permitidos y no permitidos.
- La implementación de la estructura de vecindad híbrida propuesta abordando el problema del FJSSP aplicando Búsqueda Local Iterada.
- La propuesta de una representación gráfica del modelo de grafo disyuntivo, aplicado al FJSSP, basándose en el modelo planteado por [Martínez-Rangel, 2008] del mismo problema.

1.7 Organización de la tesis

En el capítulo 2 se da una introducción al problema de calendarización de trabajos en talleres de manufactura flexible, la descripción conceptual del problema y los tipos de FJSSP existentes. Además se muestra

detalladamente la representación del problema tratado con un grafo disyuntivo y la formulación matemática.

En el capítulo 3 se presenta el algoritmo utilizado para el tratamiento del problema, búsqueda local iterada, se explica su funcionamiento y los pasos que lo conforman. Además se presenta el mecanismo de calendarización utilizado, así como la estructura híbrida de vecindad propuesta aplicada al algoritmo; Se muestra la función correspondiente a la complejidad del algoritmo propuesto.

En el capítulo 4 se describe las características del equipo donde se realizaron las pruebas además de una explicación del proceso realizado para la sintonización de parámetros. Se muestran los resultados experimentales obtenidos en las pruebas realizadas al algoritmo propuesto, además de presentar la comparación de los resultados obtenidos de eficiencia y eficacia entre el algoritmo propuesto y los algoritmos reportados en la literatura para el mismo problema y las mismas instancias.

Capítulo 5 se presentan las conclusiones finales del trabajo de investigación, en función a los resultados experimentales, así como los trabajos derivados a éste, para ser tratados en el futuro.

CAPÍTULO II.
CALENDARIZACIÓN DE TRABAJOS EN TALLERES DE MANUFACTURA FLEXIBLE

Este capítulo describe el problema de asignación de tareas en un taller de manufactura flexible, iniciando con la descripción conceptual del problema, siguiendo con la descripción detallada del modelo del grafo disyuntivo propuesto por [Martínez-Rangel, 2008]. Para finalizar el capítulo, se describe el modelo matemático y los tipos de problemas de calendarización que existen.

2.1 Introducción

La programación como un proceso de toma de decisiones juega un papel importante en la mayoría de los sistemas de fabricación y producción, así como en la mayoría de entornos de procesamiento de información [Pinedo M., 2010]. El problema clásico de JSSP es uno de los más importantes y difíciles en este ámbito y ha recibido una enorme cantidad de atención en la literatura de investigación [Gonçalves *et al*., 2005]. En el JSSP un conjunto de trabajos debe ser procesado en un conjunto de máquinas. Cada trabajo se compone de una serie de operaciones consecutivas que tienen el orden de precedencia, cada operación debe ser realizada en exactamente una máquina específica. Las máquinas están siempre disponibles en el tiempo cero y se puede procesar una operación a la vez y sin interrupciones. La decisión se refiere a la secuencia de las operaciones en cada máquina, de modo que típicamente en el JSP se busca reducir el makespan, que es el tiempo requerido para completar todas las tareas.

El FJSSP es una generalización del problema clásico JSP [Mastrolilli y Gambardella, 1998], en el que se permite a las operaciones ser procesadas por cualquier máquina a partir de un conjunto de máquinas candidatas, en lugar de una específica. Cada operación tiene asociada una duración *t* (O_i), en una asignación de tareas se fija un tiempo de inicio *st* (O_i) para cada operación, tal que:

- Una máquina no pueda procesar más de una operación en un mismo instante (restricción de capacidad de recursos).
- El orden de las operaciones en cada trabajo es respetado (orden tecnológico o restricción de precedencia).

El tiempo total de la calendarización (makespan) es igual al tiempo máximo de terminación (el tiempo de inicio más la duración) de la última operación asignada. El objetivo es encontrar una asignación que minimice el

makespan. Estas mismas características son heredadas al FJSSP, pero además se le agrega una serie de restricciones que su antecesor no tiene. Para cada una de las operaciones de un trabajo existe más de una máquina que puede ser elegida para ser procesada, por lo cual se tiene que considerar un grupo de tiempos (costos) dependiendo la máquina asignada.

En comparación con el clásico JSSP, el FJSSP está más cerca de un entorno de producción real y tiene aplicabilidad más práctica. Pero, también es más difícil, debido a su decisión adicional de asignar a cada operación la máquina apropiada. JSP es conocido por ser NP-duro y de igual manera FJSP es NP-duro [Yuan *et al.* 2012].

2.2 Descripción conceptual

El FJSSP en general consiste en organizar la ejecución de N trabajos sobre M máquinas. En este problema, hay un conjunto de *M* máquinas y un *N* conjunto de trabajos, cada trabajo consta de un subconjunto *O* que es una secuencia de operaciones. Cada operación tiene duración $\mu\ (O_i)$ y en una asignación de trabajos, un tiempo de inicio st (O_i) es definido para cada operación de tal manera que:

- Una máquina no puede procesar más de una operación en un mismo tiempo (restricción de capacidad de recursos).
- El orden de ejecución de las operaciones en cada trabajo es respetado (restricción de precedencia).
- Cada una de las operaciones tiene más de una máquina que la puede procesar.

Usualmente uno de los objetivos de las búsquedas de FJSSP es encontrar una calendarización que minimice el makespan [Martínez-Rangel, 2008].

2.3 El modelo de grafo disyuntivo

En este trabajo se cita un modelo matemático que representa el FJSSP, el cual tiene como base el modelo del grafo disyuntivo *G = (V, A, E, F, μ)* [Martínez-Rangel, 2008] como una extensión al modelo introducido por [Roy y Sussmann, 1964] para el problema clásico JSSP.

$$\mathrm{V} = O \cup \{I, D\}$$

$$\mathrm{A} = \left\{\left[I, O_{1j}\right], \left[O_{ij}\right], \left[O_{(i+1)j}\right] \mid \forall i, j: O_{ij} \in O\right\}$$

$$\mathrm{E} = \left\{\left\{O_{ij}, O_{i'j'}\right\} \mid \forall i, j, i', j', j \neq j': O_{ij}, O_{i'j'} \in \mathrm{O} \wedge M_k\left(O_{ij}\right) = M_k\left(O_{i'j'}\right)\right\}$$

$$\mathrm{F} = \left\{\left\{O_{ij}\right\} \mid \forall i, j: O_{ij} \subseteq O\ M_{ij}\left(O_{ij}\right) \subseteq M\right\}$$

$$\mathrm{P} = \left\{\left\{p_{ijk}\right\} \mid \forall i, j, k: O_{ij} \in O \wedge M_k\left(O_{ij}\right) \in M \wedge \left(p_{ijk}\right) > 0\right\}$$

$$\mu: V \rightarrow IN$$

Dónde:

Los vértices en V representan las tareas. Adicionalmente se agregan dos vértices que no tienen tiempo de procesamiento, estos son el origen I y el destino D. El peso de un vértice es *μ(O)* dado por el tiempo de procesamiento *p*(O) por lo que $\mu(O) := p(O) + s(O)$, $(\mu(I) = \mu(F) = 0)$, dado que I y D no consumen recursos. La fuente I es un nodo del cual salen todas las primeras tareas de cada trabajo y el destino D es el nodo en el cual llegan todas las tareas finales de cada trabajo. La restricción de precedencia entre O_{ij}, $O_{(i+1)j}$ es representado por un arco dirigido $[O_{ij}, O_{(i+1)j}] \in A$.

Similarmente, para cada par de tareas O_{ij}, $O_{i'j'} \in O$ existe una máquina que las puede procesar por lo que $M(O_{ij}) = M(O_{i'j'})$, la restricción disyuntiva es representada por un arco disyuntivo $\{O_{ij}, O_{i'j'}\} \in \mathrm{E}$ y las dos formas de dirigir la disyunción corresponden a las dos posibles orientaciones de $\{O_{ij}, O_{i'j'}\}$

(restricciones de capacidad de recursos). El conjunto de elementos F, corresponde a la propiedad que tienen los sistemas flexibles, donde una operación que forma parte del conjunto O, puede hacer uso de más de una máquina del sistema para ser procesada. El conjunto P corresponde a los tiempos de procesamiento de cada operación [Martínez-Rangel, 2008].

Este trabajo aborda problemas de tipo parcial, figura 2.2, por tener más similitudes con los problemas que se presentan en la industria pues del total de las máquinas que puedan existir en un taller, no siempre todas pueden procesar todos los trabajos como sucede en un ambiente de flexibilidad total.

2.3.1 Tipos de problemas de calendarización flexible

En la literatura se reconocen dos tipos de problemas de calendarización, de manera general: flexibilidad-total y flexibilidad-parcial [Ho y Tay, 2004; Kacem *et al.*, 2001; Noureddine *et al.*, 2007]. En el caso de flexibilidad-total todas las máquinas están disponibles para procesar el total de las operaciones involucradas en el proceso como se puede ver en la figura 2.1.

		M1	M2	M3
Job1	O1	6	3	5
	O2	2	4	2
	O3	6	6	6
Job2	O4	4	6	5
	O5	6	5	6
	O6	4	5	3
Job3	O7	2	3	4
	O8	4	2	6
	O9	5	4	2

Figura 2. 1 Calendarización de flexibilidad-total

Se sabe que revisten de mayor complejidad los problemas de tipo parcial [Kacem *et al.*, 2001; Ho y Tay, 2004], razón por la cual se llevó a cabo esta investigación, ya que muchas empresas tienen máquinas que no realizan todas las operaciones y deshacerse de ellas implica una pérdida económica, la representación de una instancia del tipo parcial, se muestra en la figura 2.2.

		M1	M2	M3
Job1	O1	6	*	5
	O2	2	4	*
	O3	6	*	*
Job2	O4	4	6	5
	O5	6	*	6
	O6	*	5	3
Job3	O7	2	3	*
	O8	4	2	6
	O9	5	4	*

Figura 2. 2 Calendarización de flexibilidad-parcial

2.3.2 Representación de un FJSSP

El presente trabajo propone una representación gráfica de manera general para instancias del FJSSP. La figura 2.2, es una instancia de tres máquinas con tres trabajos [Martínez-Rangel, 2008], basándose en el grafo disyuntivo clásico del JSSP se agregan arcos disyuntivos en las operaciones que se ejecutan dentro de una misma máquina, tomando en cuenta la diferencia de que en el FJSSP existe un conjunto de máquinas candidatas para ejecutar una operación, se les asignan colores a los arcos disyuntivos que pertenecen a la misma máquina, para aquellas operaciones que tienen más de una máquina candidata para su ejecución, se asigna un color diferente para el conjunto de máquinas que realizan dicha operación. En la figura 2.3 se muestran los datos de una instancia de tres trabajos y tres máquinas [Martínez-Rangel, 2008] donde, el trabajo 1 consta de la secuencia de operaciones O1, O2 y O3 (orden de precedencia). Las operaciones O4, O5 y O6, pertenecen al trabajo 2 y las operaciones O7, O8 y O9 al trabajo 3.

		M1	M2	M3
Job1	O1	6	*	5
	O2	2	4	*
	O3	6	*	*
Job2	O4	4	6	5
	O5	6	*	6
	O6	*	5	3
Job3	O7	2	3	*
	O8	4	2	6
	O9	5	4	*

Figura 2. 3 Instancia de un FJSSP de flexibilidad parcial 3 x 3

En la figura 2.4 se muestra la representación mediante un arco disyuntivo que representa la instancia 3 x 3 de un FJSSP (Figura 2.3) donde, las operaciones que se realizan en las mismas máquinas se representan mediante un arco de un color específico. Por ejemplo, las operaciones que se realizan solamente en la máquina 1 (Maq1), se ponen mediante un arco de color rojo. Las operaciones que se realizan en una máquina en común con otras operaciones, como lo son, la operación O1 y O4 en la máquina 3 (Maq3) se representan mediante un arco de color morado. Los colores de los arcos van a depender del número de combinaciones que se tenga con cada una de las operaciones en máquinas comunes, para este ejemplo, el color azul cielo, representa a las operaciones comunes en la máquinas 1 y 2. El color morado a las máquinas 1 y 3. El color verde máquinas 1, 2 y 3.

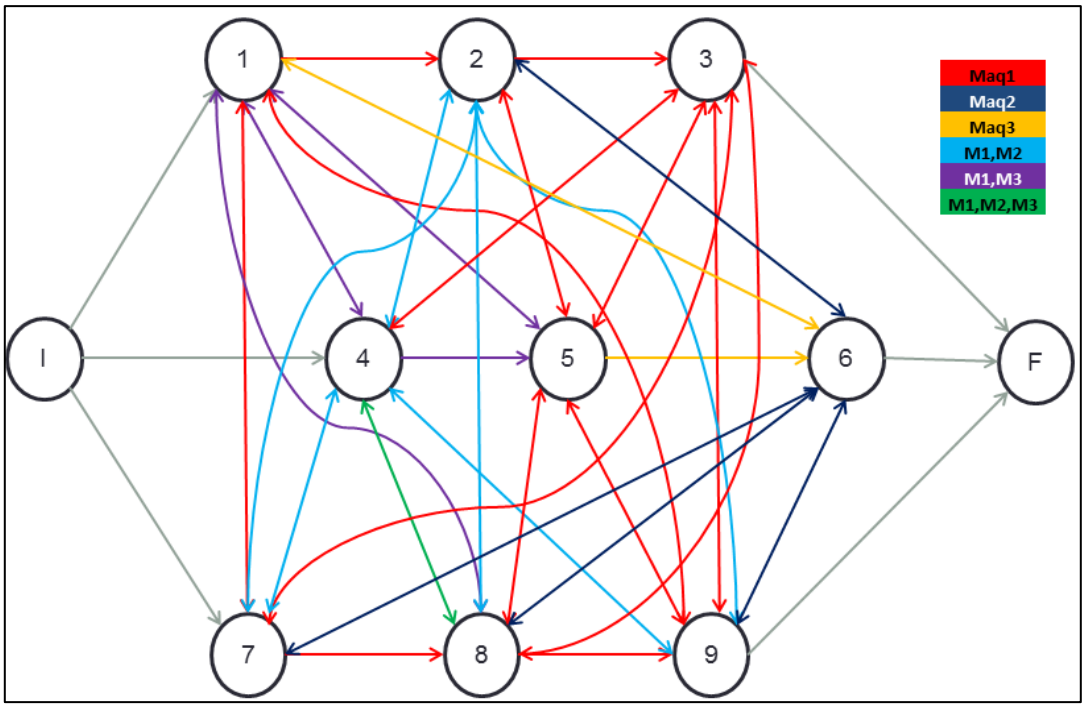

Figura 2. 4 Representación propuesta de un FJSSP de 3 trabajos x 3 máquinas

2.4 Modelo matemático

De manera formal el FJSSP consiste en organizar la ejecución de *n* trabajos sobre *m* máquinas. En este problema, hay un conjunto de máquinas M = { M_k } *1 ≤ k ≤ m*, y un conjunto de trabajos J = { J_i } *1 ≤ i ≤ n.* Cada trabajo consiste en una secuencia de operaciones $O_{ij,}\ j = 1, \dots, n_i$, donde O_{ij} y n_i indica la j-ésima operación del trabajo i y el número de operaciones requeridas para el trabajo i, respectivamente. Cada operación O_{ij} debe ser procesada sobre una máquina llamada M_k de un conjunto de máquinas candidatas llamado $M_{ij} \subseteq M$ con un tiempo de procesamiento predefinido fijado $P_{i,h,k}$. La flexibilidad de los problemas puede ser clasificado en flexibilidad total y flexibilidad parcial. Es total cuando cada operación puede ser procesada sobre cualquiera maquina perteneciente al conjunto M. El problema entonces, se le llama total FJSSP (T-FJSSP). Por otro lado, el problema es parcial FJSSP (P-FJSSP) tiene algunos supuestos de la siguiente manera: todos los trabajos y todas las máquinas están disponibles en el tiempo cero. Cada máquina puede procesar solamente una operación en el tiempo cero. La interrupción de operaciones no está permitida, es decir, no puede ser detenido hasta su finalización una vez que inicie.

Para el modelo matemático, se utilizó la siguiente notación para la formulación del FJSSP.

Índices y conjuntos

i →Índice de Trabajos (i=1,…..,n).

j →Índice de operaciones.

k →Índice de máquina (k=1,…….,m).

r →Índice de prioridad de procesamiento.

Oi,j →La j-ésima operación del trabajo i.

J →El conjunto de trabajos

M →El conjunto de máquinas

$\boldsymbol{M_{ij}}$ →El conjunto de máquinas candidatas en los que la operación $\boldsymbol{O_{ij}}$ puede ser procesada. $\boldsymbol{M_{ij}} \subseteq$ **M**

Parámetros

n →Número de trabajos

m → Número de máquinas

$\boldsymbol{P_{i,j,k}}$→ Tiempo de procesamiento $\boldsymbol{O_{ij}}$ sobre la máquina k ($\boldsymbol{P_{i,j,k}} \geq 0$).

L→ Un número muy grande.

Variables de decisión

$$\boldsymbol{X_{i,j,k}} = \begin{cases} \mathbf{1}, si\ la\ máquina\ k \text{ es seleccionada para la operación } \boldsymbol{O_{ij}} \\ \mathbf{0}, \text{ si no es seleecionada} \end{cases}$$

$$\boldsymbol{Z_{i,j,k,r}} = \begin{cases} \mathbf{1}, si\ \boldsymbol{O_{ij}} \text{ es ejecutada en la máquina } k \text{ con prioridad } r \\ \mathbf{0}, \text{ si no es seleecionada} \end{cases}$$

$\boldsymbol{S_{i,j}}$ →El tiempo de inicio de la operación $\boldsymbol{O_{ij}}$**.**

$\boldsymbol{C_{i,j}}$ →El tiempo de terminación de la operación $\boldsymbol{O_{ij}}$.

$\boldsymbol{C_i}$ →El tiempo de terminación del trabajo *i*.

$\boldsymbol{C_{max}}$ →El tiempo máximo de ejecución de todos los trabajos.

$\boldsymbol{S^m_{k,r}}$ →El tiempo de inicio de la operación ejecutada en la máquina *k* con prioridad *r.*

$\boldsymbol{q_k}$→Número de operaciones asignadas a la máquina *k.*

Bajo estos supuestos y notaciones, el modelo matemático propuesto para el problema FJSSP, empezando por la función objetivo, se define como sigue:

$$\text{Min } C_{max} = \frac{max}{\forall O_{ij}} \{S_{ij} + p_{ij}\} \quad (1)$$

Sujeto a:

$$S_{i,j} + \sum_{k \in M_{ij}} (p_{i,j,k} . x_{i,j,k}) \leq S_{i,j+1} \; \forall j = 1, \dots, n_i - 1; i = 1, \dots, n; \quad (2)$$

$$\sum_{k \in M_{ij}} (x_{i,j,k}) = 1 \; \forall j = 1, \dots, n_i; i = 1, \dots, n; k = 1, \dots, m; \quad (3)$$

$$\sum_r z_{i,j,k,r} = x_{i,j,k} \; \forall j = 1, \dots, n_i; i = 1, \dots, n; k = 1, \dots, m; r = 1, \dots, q_k; \quad (4)$$

$$S^m_{k,r} + p_{i,j,k} . z_{i,j,k,r} \leq S^m_{k,r+1} \; \forall j = 1, \dots, n_i; i = 1, \dots, n; k = 1, \dots, m; r = 1, \dots, q_k - 1; \quad (5)$$

$$S^m_{k,r} + (1 - z_{i,j,k,r}) . L \geq s_{i,j} \; \forall j = 1, \dots, n_i; i = 1, \dots, n; k = 1, \dots, m; r = 1, \dots, q_k; \quad (6)$$

$$S^m_{k,r} \leq S_{i,j} + (1 - z_{i,j,k,r}) . L \; \forall j = 1, \dots, n_i; i = 1, \dots, n; k = 1, \dots, m; r = 1, \dots, q_k; \quad (7)$$

$$S_{i,j} \geq 0 \; \forall j = 1, \dots, n_i; i = 1, \dots, n; \quad (8)$$

$$p_{i,j,k} \geq 0 \; \forall j = 1, \dots, n_i; i = 1, \dots, n; \quad (9)$$

$$S^m_{k,r} \geq 0 \; \forall k = 1, \dots, m; r = 1, \dots, q_k; \quad (10)$$

$$x_{i,j,k} \in \{0,1\} \; \forall j = 1, \dots, n_i; i = 1, \dots, n; k = 1, \dots, m; \quad (11)$$

$$z_{i,j,k,r} \in \{0,1\} \; \forall j = 1, \dots, n_i; i = 1, \dots, n; k = 1, \dots, m; r = 1, \dots, q_k; \quad (12)$$

La restricción (2) asegura que las secuencias de precedencia entre las operaciones de un trabajo no sean violadas, es decir, la operación O_{ij} no puede iniciar hasta que su predecesor O_{ij-1} se haya completado. La

restricción (3) se asegura de que cada operación sea asignada a una sola máquina del conjunto de máquinas candidatas. La restricción (4) obliga a cada operación ser procesada en un cierto orden de prioridad dentro de una máquina. La restricción (5) se asegura de que cada máquina procese una sola operación en un mismo tiempo a la vez. Las restricciones (6) y (7) se hacen cargo de cuidar el requerimiento de que cada operación puede ser procesada después de que su máquina asignada este libre y su operación previa haya sido procesada. La restricción (8) corresponde al tiempo de inicio de procesamiento para O_{ij}. La restricción (9) determina el tiempo de procesamiento de la operación O_{ij} después de seleccionar una máquina. La restricción (10) corresponde al tiempo de inicio de la operación en un turno para la máquina *k* con prioridad *r*. La restricción (11) toma valores de 1; si la operación O_{ij} es ejecutada sobre la máquina *k* y toma el valor de cero para el caso contrario. La restricción (12) toma valores de 1 si la máquina *k* es seleccionada para ejecutar la operación O_{ij} y toma el valor de cero para el caso contrario.

CAPÍTULO III. ALGORITMO BÚSQUEDA LOCAL ITERADA

En el presente capítulo se presenta un algoritmo de búsqueda local iterada secuencial, así como el desarrollo e implementación de una estructura de vecindad híbrida aplicada en búsqueda local para resolver el problema de calendarización de tareas en talleres de manufactura flexible. Se incluye la función temporal del algoritmo y su complejidad.

3.1 Introducción

En este capítulo se propone un algoritmo de Búsqueda Local Iterada secuencial, así como el desarrollo e implementación de una estructura de vecindad híbrida aplicada en búsqueda local para resolver el FJSSP. Se incluye una amplia descripción del procedimiento utilizado para el desarrollo de la estructura de vecindad híbrida mencionada con anterioridad y finalmente se anexa la función temporal del algoritmo y su complejidad.

Para llevar a cabo la selección de las estructuras de vecindad que conforman la estructura híbrida, se hizo una búsqueda exhaustiva de diversas publicaciones que abordan el problema tratado (Capitulo 2), para, de esta forma identificar aquellas estructuras que cuentan con menor complejidad computacional y que han obtenido buenos resultados en su aplicación.

En primer lugar se desarrolló el algoritmo de Búsqueda Local Iterada, para mejorar la solución obtenida, se llevó a cabo la implementación de una estructura de vecindad híbrida para la búsqueda local. En las secciones siguientes se explica ampliamente el procedimiento realizado para el desarrollo de dicha estructura.

3.2 Búsqueda local iterada

El procedimiento implementado se basa en una búsqueda local iterada, la cual es una heurística que aplica iterativamente un procedimiento de Búsqueda Local que genera una sucesión de soluciones que se aproxima a un resultado mucho mejor de los que se esperaría aplicando repetidamente el mismo método de búsqueda local [Micheli M., 2009]. De acuerdo a los estudios de Hoos y Stützle, [2005] donde afirman que ILS es una de las metodologías más sencilla y eficaz para evitar quedar atrapados en óptimos locales.

El procedimiento de búsqueda local iterada requiere de una estructura de vecindad y de conocer una función objetivo que se requiera maximizar o

minimizar, empieza con una solución cualquiera factible *s* y el conjunto de soluciones en *N(s),* del cual se elige una solución factible ***s'*** a través de un movimiento *σ,* el tipo de movimiento a realizar para seleccionar un vecino define la estructura de la vecindad que mejore la función objetivo (minimizar el tiempo de ejecución de la última operación) por medio de un proceso estocástico es decir ***f(s') < f(s),*** si esto se cumple se reemplaza la solución *s* por la solución ***s'*** que la mejore, esto se repite hasta alcanzar el criterio de paro de la búsqueda local (CP_BL) y sigue iterando la búsqueda local hasta que la solución no siga mejorando, se evalúa por medio de un procedimiento (f(s') < f(MS_BLI)), si esto se cumple se reemplaza la solución *s* por la solución MS_BLI, esto se repite hasta alcanzar el criterio de paro CP_BLI. Los valores de CP_BL y CP_BLI, se obtienen mediante la sintonización del algoritmo que se muestra en la sección 3.5. En la figura 3.1, se muestra el algoritmo general de búsqueda local iterada.

```
MS_BLI← M; // donde M tiene un valor muy grande
Hacer // búsqueda local iterada
        Genera solución inicial s
        Hacer // búsqueda local
                s' ← solución movimiento δ
                sí (f(s') < f(s)) entonces
                        s' ← mejor solución encontrada
                        s ← s'
                fin-sí
        Mientras CP_BL
        sí (f(s) < f(MS_BLI)) entonces
                MS_BLI ← s
        fin-sí
Mientras CP_BLI
```

Figura 3. 1 Algoritmo general de Búsqueda Local Iterada

Para mejorar la calidad de las soluciones encontradas, diversos autores han recurrido al uso de estructuras de vecindad aplicadas a búsqueda local [Guo *et al.,* 2007; Gómez *et al.,* 2007; Zhou *et al.,* 2007;]; Pei y Shih, 2007; Chun, 2008; Mateo, 2009].

3.3 Estructura de vecindad híbrida

Todo problema de optimización tiene un conjunto, ya sea finito o infinito de soluciones posibles, por lo que se requiere la utilización de técnicas que permiten la mejor explotación del espacio de soluciones, y por ende obtener soluciones de buena calidad. Este tipo de técnicas aplicadas a búsqueda local son mejor conocidas como estructuras de vecindad.

El funcionamiento de las estructuras de vecindad es iterativo, debido a que permiten la mejor explotación del espacio de soluciones, de modo que define así como la forma de acceder a soluciones vecinas de una solución *s* mediante una operación denominada *movimiento,* donde el tipo de movimiento aplicado define la estructura de vecindad y tamaño de una vecindad [Martínez-Oropeza, 2010].

Las estructuras de vecindad son técnicas utilizadas con el propósito de mejorar una solución, para lo cual es necesario moverse paso a paso desde una solución inicial hasta una solución vecina que proporcione el valor mínimo o máximo de la función objetivo según sea el caso [Cruz-Chávez y Rivera-López, 2007]. Estas técnicas son utilizadas en problemas de optimización las cuales permiten la mejor explotación del espacio de soluciones mediante su implementación dentro del algoritmo de búsqueda local.

Una vecindad se define como un conjunto de todas aquellas soluciones que pueden ser alcanzables a partir de una solución inicial *s,* por medio de un movimiento σ que puede ser una perturbación, inserción o eliminación entre elementos que conforman la solución *s* para realizar la explotación del espacio de soluciones [Papadimitriou y Steiglitz, 1998]. El tipo de movimiento define el tipo de estructura y tamaño de la vecindad [Martínez, 2006].

De acuerdo a esto, una estructura de vecindad se define como una función *N(s)* presentada en la ecuación: $N(s) = \left\{ s' \in S : s \xrightarrow{\sigma} s' \right\}$

Una función de vecindad *N(s)* específica para cada solución *s* ∈ *S* es un conjunto *N(s)* ∈ S, el cual es llamado vecindario de *S*, esto indica que cada solución ***s'*** es un vecino de *s* si *s´*∈ *N(s). S* representa el conjunto total de soluciones posibles de una instancia del problema [Martínez *et al.*, 2011].

El punto fundamental para explotar el espacio de soluciones, consiste en empezar desde un punto factible *s,* del conjunto de soluciones en *N(s)* se elige mediante un movimiento una solución vecina ***s'*** que mejore f(s') *(mejor valor de la función objetivo encontrado hasta el momento)*, esto es *f(s') < f(s),* con esto, se hace un reemplazo de la solución *s* y se posiciona ***s'*** en este nuevo punto tal y como se puede observar en la figura 3.2**,** de modo que la estructura de vecindad es la función de vecindad o tipo de movimiento que aplicado a búsqueda local, permite mejorar la explotación del espacio de soluciones [Gu y Huang, 1994; Stattenberger *et al.*, 2007].

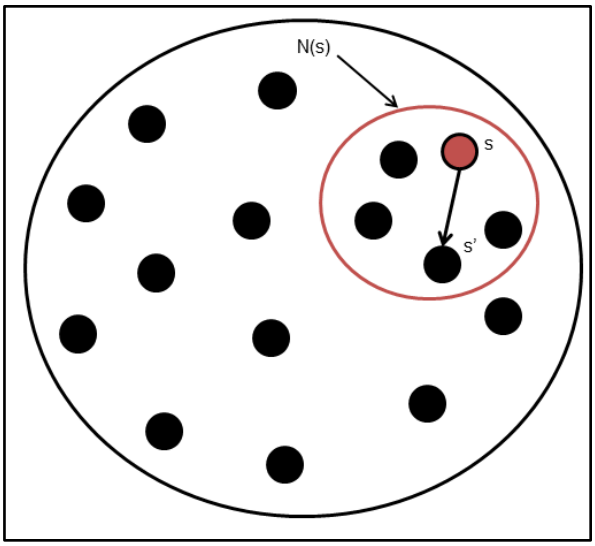

Figura 3. 2 Representación del espacio de soluciones con una vecindad N(s) = {s'} [Gu y Huang, 1994].

Un aspecto crítico del diseño de algunos algoritmos de optimización es la elección de una estructura de vecindad adecuada, es decir, elegir aquella estructura de vecindad que permita la mejor explotación del espacio de soluciones de acuerdo al algoritmo utilizado al problema tratado.

Para determinar el funcionamiento de una estructura de vecindad, se debe definir el criterio de selección de un vecino, es decir, el movimiento que se llevará a cabo para alcanzar una solución *s'* desde una solución *s*, de modo que si el criterio de selección se cumple, se lleva a cabo el movimiento, el proceso se repite hasta que la solución encontrada no pueda ser mejorada, por lo que se dice que se ha llegado a un *óptimo local.* A continuación, en la figura 3.3 se muestra el algoritmo general de una búsqueda por vecindad.

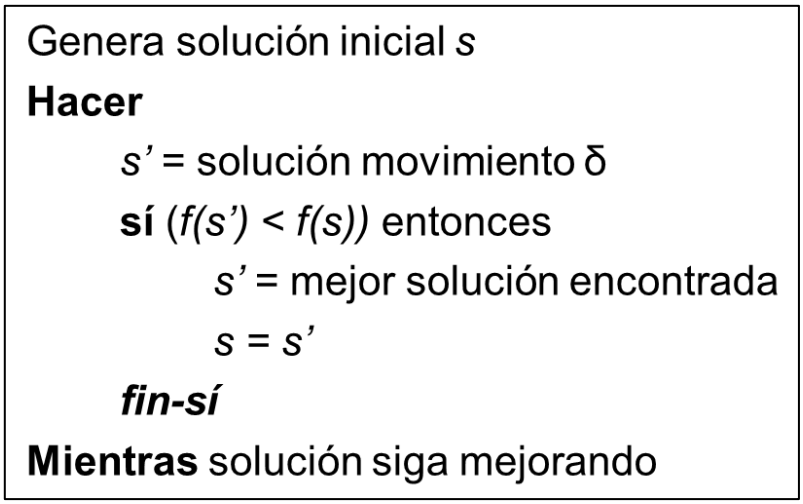

Figura 3. 3 Algoritmo general de una búsqueda por vecindad

Se dice que *s* es un óptimo local si no se encuentra una solución en *N(s)* que la mejore. La idea básica de la estructura de vecindad es que a partir de una solución inicial *s,* se explore y evalué a sus vecinos en *N(s),* para encontrar una nueva mejor solución *s'*∈ *N(s)* [Gu y Huang, 1994 y Stattenberger *et al.*, 2007].

Para desarrollar una estructura de vecindad híbrida, se realizó una revisión en la literatura, para buscar las estructuras de vecindad que contaran con baja complejidad computacional y que hayan obtenido buenos resultados en su aplicación. Las pruebas experimentales se realizaron para el FJSSP. Se tomó este problema debido a la complejidad, la similitud que tiene con los sistemas de manufactura reales y que no hay trabajos de investigación que utilicen una hibridación de estructuras de vecindad. En las siguientes

secciones (3.3.1 a la 3.3.5) se mencionan las características de cada una de las estructuras de vecindad, así como la estructura de vecindad híbrida propuesta.

3.3.1 Vecindad N1

Laarhoven et al., [1992] describieron la función de vecindad conocida en la literatura como N1. Que se compone de los vecinos que se forman al realizar una perturbación en un par de operaciones adyacentes que pertenecen a una misma máquina. Dado un diagrama de la solución D tal como se presenta en el grafo de la figura 3.4, un movimiento es generado invirtiendo un arco (v, W) en la ruta crítica, tal que la operación v en una máquina K. La revocación de (v, W) da lugar siempre a una solución (a cíclica) factible. Por otra parte, la revocación de arcos en la ruta crítica son las únicas revocaciones del arco que pueden reducir el valor del makespan. N1 tiene la característica de la conectividad. Por ejemplo, considerar la solución representada por el diagrama D en la figura 3.4 que es la ruta más larga (fuente, O_{31}, O_{11}, O_{12}, O_{21}, O_{32}, O_{33}, O_{13}, destino), que se muestra de color más oscura. La vecindad de D, N1 (D), consiste en las soluciones obtenidas invirtiendo los arcos siguientes: (O_{31}, O_{11}), (O_{12}, O_{21}), (O_{21}, O_{32}) y (O_{33}, O_{13}).

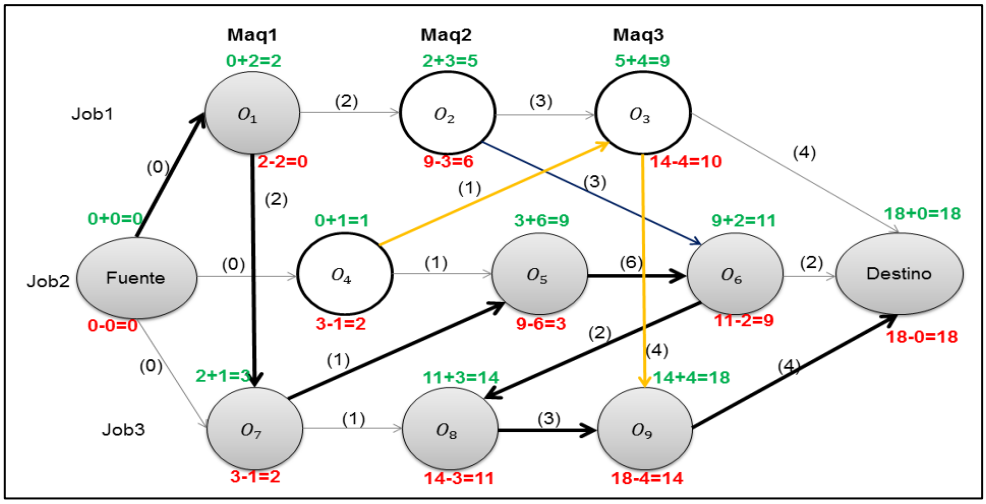

Figura 3. 4 Ruta crítica de una solución factible [Martínez-Rangel, 2008]

La figura 3.5 presenta el esquema de movimientos de la estructura de vecindad N1 para una máquina, se aplica a los pares de operaciones adyacentes obtenidos de la solución factible de la figura 3.4, en esta estructura se propone tomar aleatoriamente un par de operaciones que no tengan tiempo de ocio y pertenezcan a la misma máquina. Teniendo en cuenta las restricciones del problema, esta estructura no se puede aplicar si las dos operaciones a intercambiar pertenecen al mismo trabajo, ya que si esto ocurre, se violaría la restricción de precedencia. Los intercambios de las operaciones factibles, están marcados con flechas azules.

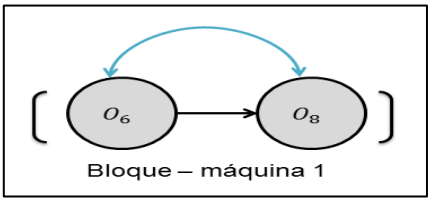

Figura 3. 5 Esquema de movimientos N1 [Laarhoven *et al.,* 1992].

3.3.2 Vecindad N4

La estructura de vecindad conocida como N4, propuesta por Dell'Amico y Trubian [1993], implica mover una operación interna (toda operación que no es la primera ni la última de un bloque) al principio o al final del mismo bloque.

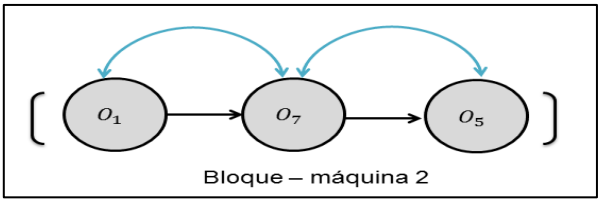

Figura 3. 6 Esquema de movimientos N4 [Dell'Amico y Trubian, 1993].

La figura 3.6 muestra los intercambios posibles de la estructura N4, está basada en el fundamento de bloques, que es, el conjunto de operaciones consecutivas que se ejecutan en la misma máquina y que pertenezcan a la ruta crítica. La estructura de vecindad N4 solo se puede aplicar si, el bloque está compuesto por más de 2 operaciones, para poder así, seleccionar una

operación como interna. De igual manera se debe tomar en cuenta la restricción de precedencia, la cual, se verifica que las operaciones que estén entre la primera o la última operación (dependiendo si el intercambio es hacia delante o hacia atrás) y la operación interna del bloque que se va a intercambiar no pertenezcan al mismo trabajo, de lo contrario no se puede realizar dicho movimiento.

3.3.3 Vecindad N5

Nowicki y Smutniki [1993] propusieron el uso de bloques como agrupamientos para operaciones que se efectúen en la misma máquina, ahí los movimientos son como sigue; en el primer bloque de operaciones, intercambia el último para de operaciones, en el segundo bloque, se intercambia el primer par de operaciones del bloque.

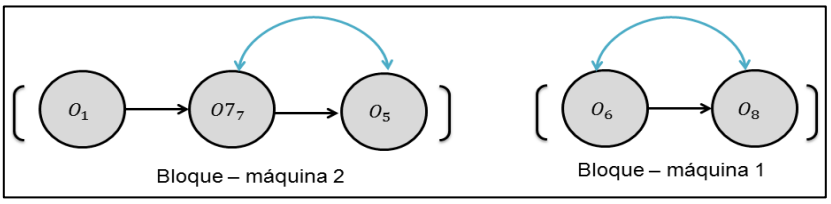

Figura 3. 7 Esquema de movimientos N5 [Nowicki y Smutniki, 1996]

En la figura 3.7 se muestra los intercambios posibles de la estructura de vecindad N5, se lanza un número aleatorio para seleccionar cada uno de los bloques en los que se aplicará la estructura de vecindad, en la figura 3.7, el primer bloque seleccionado es el de la máquina 2 y el otro bloque corresponde a la máquina 1.

3.3.4 Vecindad N6

La idea de Balas y Vazacopoulos [1998] fue proponer un método de búsqueda local guiada, la cual se basa en la inversión de más de un arco disyuntivo a la vez. Esto conduce a un vecindario considerablemente mayor que las estructuras anteriores. Por otra parte, los vecinos se definen mediante el intercambio de un conjunto de arcos de diferentes tamaños, por lo tanto, la búsqueda es de profundidad variable y apoya la diversificación de

búsqueda en el espacio de soluciones. La estructura de vecindad N6 (figura 3.8) es una extensión de todas las estructuras de vecindad mencionadas en los puntos 3.3.1 al 3.3.4 del presente capítulo.

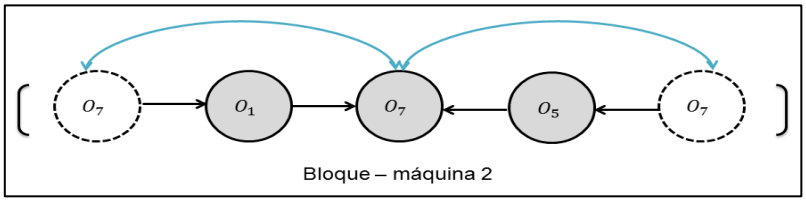

Figura 3. 8 Esquema de movimientos N6 [Balas y Vazacopoulos, 1998].

Al igual que en la estructura de vecindad N4, se debe considerar las restricciones de precedencia, ya que algunos movimientos de la estructura de vecindad N6 puede llevar a una solución infactible, por tal motivo, si el conjunto de operaciones que están entre la primera o última operación (dependiendo si el intercambio es hacia atrás o hacia delante) y la operación interna a extraer para intercambiar, el movimiento no se puede llevar a cabo, en este caso se selecciona otra operación al azar para ser intercambiada.

3.3.5 Vecindad híbrida

Uno de los aspectos fundamentales del presente trabajo es definir la estructura de vecindad que sea eficiente y eficaz. Las estructuras de vecindad pueden generar diferentes soluciones vecinas, con el fin de encontrar mejores soluciones, haciendo la explotación del espacio de soluciones eficiente. En este trabajo, se implementaron cuatro estructuras de vecindad basadas en el método de la ruta crítica. Este método, menciona que el *makespan* no puede ser reducido manteniendo la ruta crítica actual, por lo tanto, con el objetivo de obtener una mejor solución, se deben identificar las rutas críticas existentes y romper cada una de ellas. Moviendo diferentes operaciones críticas dentro de la ruta crítica se puede llegar a diferentes vecindarios [Zhang *et al.,* 2009], la descripción detallada de la estructura de vecindad híbrida se muestra en la figura 3.9.

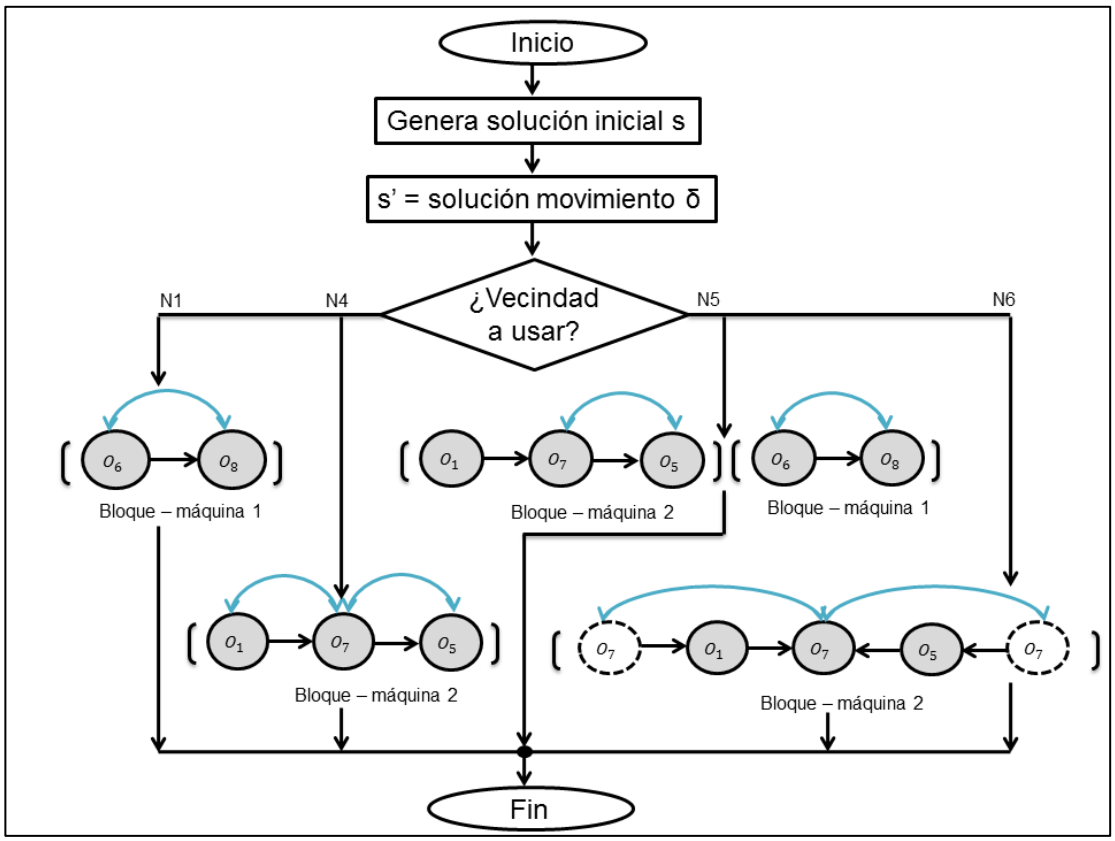

Figura 3. 9 Estructura Híbrida de vecindad

3.4 Algoritmo secuencial para FJSSP

Para proponer una solución al FJSSP, se desarrolló un algoritmo de Búsqueda Local Iterada secuencial, en el cual fue aplicada una estructura de vecindad híbrida, misma que se explica en la sección 3.3.5.

Tomando en cuenta el diagrama mostrado en la figura 3.9, se tiene el tipo de estructura de vecindad a aplicar en cada iteración durante la ejecución de la búsqueda local que se selecciona de manera aleatoria, es decir, se genera un número aleatorio entre 1 y 4, y dependiendo al número se selecciona la estructura de vecindad. A continuación se explican los módulos involucrados en el algoritmo propuesto para resolver el FJSSP, donde los primeros módulos corresponden a la inicialización de la Búsqueda Local Iterada (figura 3.10).

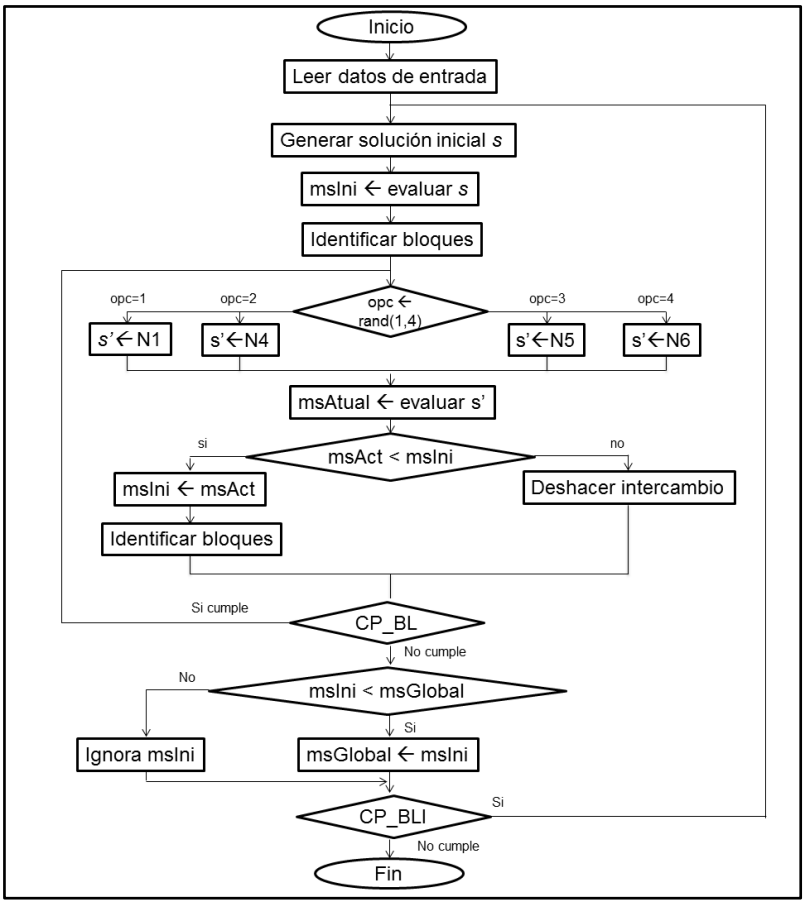

Figura 3. 10 Algoritmo BLI propuesto para el FJSSP

Módulos del algoritmo propuesto implementando una estructura de vecindad híbrida aplicada a búsqueda local:

Módulo Genera solución inicial s

Este módulo está conformado por dos principales funciones:

- Asigna Máquina: Se encarga de mantener un equilibrio de la carga de operaciones en las máquinas, asignando a cada operación una máquina de un conjunto de máquinas candidatas (Ruteo).
- Scheduling: Planifica la secuencia de la asignación de las operaciones en todas las máquinas, para obtener una planificación factible o schedule (Planificación).

Módulo Evaluar s

La función *evaluar s*, corresponde al método seleccionado para conocer el valor del makespan, dado un schedule. El método de la ruta crítica, conocido como CPM (por sus siglas en inglés) [Hillier y Lieberman, 2005], se implementó de manera modular con el objetivo de poder aplicar una parte del método para evaluar una nueva solución generada por la aplicación de una perturbación, estos es, mediante la aplicación de una estructura de vecindad.

El método de la ruta crítica se utiliza en el campo de la investigación de operaciones para la planeación y el control de proyectos. En este caso se usará para identificar la ruta crítica y las operaciones que la conforman, de tal manera que, se pueda identificar los bloques críticos de operaciones en los cuales se aplicarán las estructuras de vecindad, cabe mencionar que un bloque crítico es un conjunto de operaciones que se ejecutan en la misma máquina.

El algoritmo de *CPM*, inicialmente obtiene datos de un dígrafo, cada arco del dígrafo representa la actividad llevada a cabo por la operación que la precede. Cada una de las actividades se ejecutan en un tiempo *t*, que es igual a la duración del proceso de la operación que le precede, [Cruz *et al*., 2004].

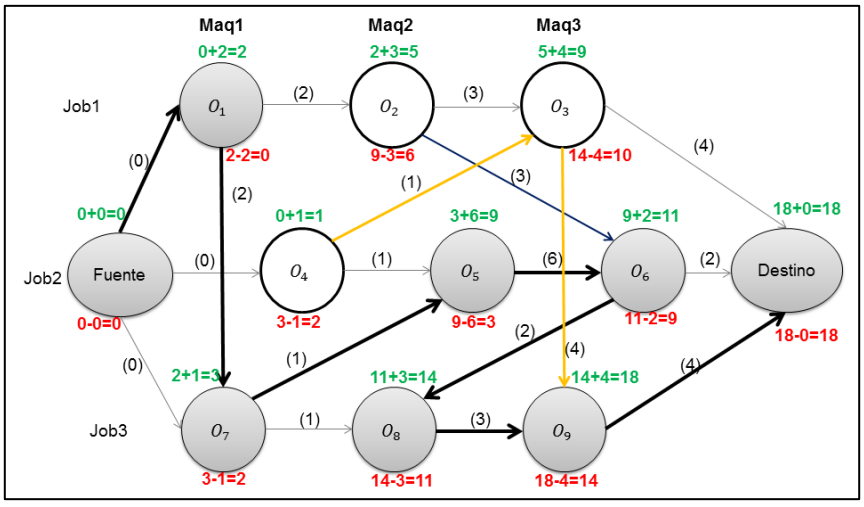

Figura 3. 11 Ejemplo del cálculo de la ruta crítica.

La Figura 3.11 muestra un ejemplo del Cálculo de la ruta crítica, el algoritmo *CPM* pasa a través de cada nodo del grafo dos veces, una hacia adelante, de la operación *I* hacia la operación *F*, y una hacia atrás, de la operación *F* hacia la operación *I*. En la pasada hacia adelante a través del grafo, a cada operación se le asigna un Máximo Tiempo más próximo (*Maximum Earliest Time*), ($MET = ET + t_{act}$), donde el tiempo más próximo *ET* (*Earliest Time*) de la operación es el tiempo en el cual la operación empezará, si las actividades precedentes son llevadas a cabo lo más rápido posible.

En la pasada hacia atrás a través del grafo, a cada operación se le asigna un Mínimo Tiempo más lejano (*Minimum Latest Time*), ($MLT = LT - t_{act}$), donde el tiempo más lejano *LT* (*Latest Time*) de la operación es igual al último momento en el cual la operación podría empezar sin demora en la finalización de cualquier otra operación. Después de calcular el Máximo Tiempo más próximo y el Mínimo Tiempo más lejano, se calcula la diferencia y al resultado se le llama Holgura (*Slack*). La ruta crítica está conformada por todas aquellas operaciones las cuales en su holgura tienen un valor de cero [Cruz *et al*, 2004].

Este método consiste en recorrer dos veces el schedule (ida y vuelta). La principal razón de implementar este método, es que al aplicar ida solamente, se conoce la calidad de la solución, es decir, el makespan. De esta forma, la solución generada es comparada con la solución anterior, sí no mejora, esta nueva solución se ignora y se aplica otra perturbación. En el caso que la solución mejoró, esta reemplaza a la solución anterior y se aplican la parte correspondiente al módulo vuelta, que es el que recorre el schedule del nodo final al inicio, para obtener los valores mínimos más prontos. Con esta propuesta, se espera reducir los tiempos de ejecución al no perder tiempo aplicando CPM en soluciones que no mejoran el schedule actual.

Módulo Identificar bloques

Como se puede observar en el módulo “Evaluar s”, consiste en recorrer dos veces el schedule solución (ida y vuelta). La principal razón de implementar este método, es que al aplicar ida, se puede conocer la calidad de la solución, es decir, el makespan. De esta forma, la solución generada mediante una perturbación es comparada con la solución original, en el caso que no mejore, esta solución generada se ignora y el módulo “Identificar bloques” no se aplica, y se ejecuta otra perturbación. Pero en el caso que la solución encontrada si mejoré la calidad de la solución, se lleva acabo los siguientes pasos dentro de este módulo:

- El primer paso a seguir es obtener todas las operaciones que pertenecen a la ruta crítica, es decir, las operaciones que no pueden ser retardadas y no tienen tiempo de ocio. Tomando en cuenta el ejemplo de la figura 3.11, las operaciones que no tienen tiempo de ocio se muestran en la tabla 3.1.

Tabla 3. 1 Operaciones sin tiempo de ocio.

operación	tiempo inicio
1	0
5	3
6	9
7	2
8	11
9	14

- Segundo paso, de la lista de operaciones que se obtuvo en el primer paso, se ordenan de menor a mayor en función del tiempo de inicio de cada operación. Dando preferencia a las operaciones que se ejecutan dentro de la misma máquina.

Tabla 3. 2 Operaciones ordenadas por tiempo de inicio.

operación	tiempo inicio	máquina
1	0	1
7	2	1
5	3	1
6	9	2
8	11	2
9	14	3

- El tercer paso es identificar de la lista ordenada (segundo paso) las operaciones que se ejecutan dentro de la misma máquina. Tomando el ejemplo de la tabla 3.2, los bloques identificados se muestran en la figura 3.12.

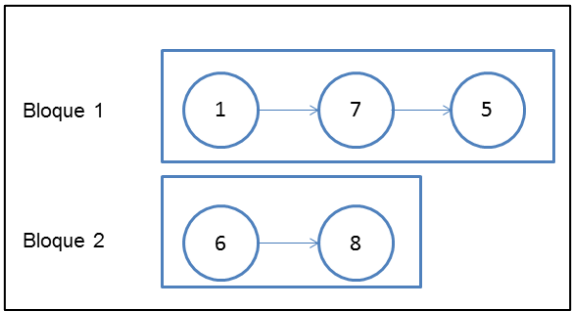

Figura 3. 12 Bloques de la ruta crítica

Módulo Deshacer cambio

Después de identificar los bloques que hay en una ruta crítica (modulo identificar bloques), se selecciona de manera aleatoria un bloque y una estructura de vecindad para realizar una perturbación. Una vez elegido el bloque (para este ejemplo, bloque 1 de la figura 3.12) y la estructura a aplicar (N1 intercambio de un par aleatorio), se genera un nuevo número aleatorio para seleccionar la operación a intercambiar (operación 7). Hecho esto, se evalúa la calidad de la solución generada con la perturbación, si resulta obtener una mayor calidad (minimizando o maximizando según sea el caso), la nueva solución reemplaza a la solución que sirvió como base para la perturbación. En caso contrario, donde la solución perturbada no mejore la calidad de la solución base, se procede a deshacer el cambio. El proceso de deshacer cambio implica copiar la configuración de una máquina específicamente. La máquina seleccionada es aquella en la que se ejecutan las operaciones del bloque elegido para la perturbación (figura 3.13).

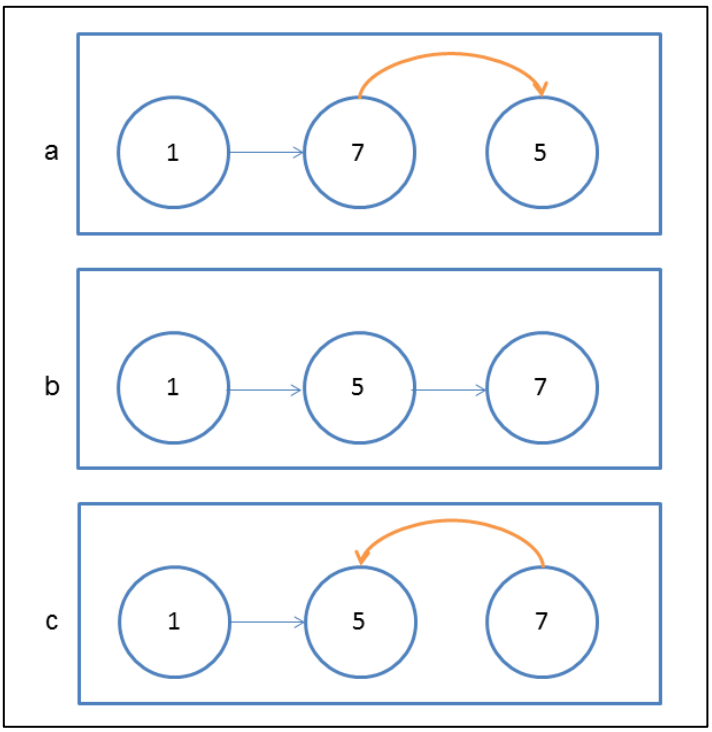

Figura 3. 13 Deshacer perturbación

Para el caso de la figura 3.13 (a) representa la calendarización original de la máquina 1 sobre la cual se aplica la estructura N1, intercambiando la operación 7 por la 5, quedando como el inciso (b). Después de evaluar la nueva calendarización se tiene que no mejoro la calidad de la solución, es decir no disminuyo el makespan y se procede a deshacer el cambio como se ve en (c) de la figura 3.13.

3.5 Estrategia de sintonización

Para llevar a cabo una adecuada sintonización de los parámetros de control de nuestro algoritmo, es necesario llevar a cabo un análisis de sensibilidad.

El análisis de sensibilidad es una evaluación del comportamiento de las variables críticas de un problema (parámetros de control), con la finalidad de establecer un rango numérico, dentro del cual la solución obtenida por el algoritmo sigue siendo buena, además que permite conocer que tan sensible

es el algoritmo a cambios en los valores de ciertas variables propias del problema.

El análisis de sensibilidad tiene su origen en la programación lineal (PL) debido a que facilita la toma de decisiones. Los cambios llevados a cabo durante dicha evaluación pueden ser en el entorno general del problema, en la empresa o bien en los datos característicos del problema.

El objetivo primordial del análisis de sensibilidad, es encontrar la adecuada sintonización de los parámetros de control del algoritmo, de modo que este tenga una mejora en cuanto la eficiencia y eficacia. Los valores más estudiados en este análisis de forma general, son los coeficientes de la función objetivo y los términos independientes de las restricciones.

Por lo tanto, un objetivo fundamental en el análisis de sensibilidad es identificar los parámetros sensibles (es decir, los parámetros cuyos valores no pueden cambiar sin que cambie la solución óptima) [Hillier y Lieberman, 2010].

A continuación se explica una metodología de sintonización propuesta por Martínez-Oropeza [2010] para encontrar la adecuada proporción en los valores correspondientes a los parámetros de control, tomando en cuenta tanto el problema como el método de solución implementado.

1. Selección de los parámetros de control

Para determinar los parámetros de control del algoritmo, es necesario llevar a cabo una revisión de publicaciones que se relacionen con el algoritmo implementado, de esta manera es posible analizar los parámetros tomados en cuenta en investigaciones anteriores.

Otra forma de identificar los parámetros de control, es realizar un análisis tanto del problema como del algoritmo, a manera de identificar los

parámetros críticos que influyen de cierta manera en la calidad de las soluciones.

2. Establecer rangos de evaluación

Una vez que los parámetros de control se han identificado, es necesario establecer los rangos que van a ser utilizados para realizar el análisis de sensibilidad a cada uno de los parámetros, para lo que es necesario llevar a cabo una revisión en la literatura, a manera de identificar y analizar los valores utilizados por diferentes autores para el algoritmo propuesto, de ésta manera es más sencillo especificar un rango de acción para cada uno de los parámetros.

Cabe mencionar que es de vital importancia llevar a cabo una adecuada sintonización debido a que éste permite identificar la proporción adecuada entre los parámetros de control, dentro de la cual, el algoritmo obtiene buenas soluciones.

3. Pruebas a rangos de evaluación

Para llevar a cabo el análisis de sensibilidad a los parámetros de control mencionados anteriormente, se toman en cuenta los rangos establecidos, calculando series de muestras (la cantidad de muestras varía de acuerdo al tamaño del rango) que permitan evaluar el comportamiento del algoritmo cuando los parámetros de control toman valores distintos.

Una vez que se tiene el conjunto de muestras, se procede a realizar las pruebas experimentales, para lo cual se recomienda realizar conjuntos mínimos de 30 pruebas para cada una de las muestras. Para llevar a cabo una adecuada sintonización de los parámetros de control, es necesario realizar un barrido de los valores correspondientes a una de las variables, manteniendo fijos los demás, hasta identificar el valor que mejore la calidad de las soluciones. Una vez obtenido el mejor valor para esta variable, se fija

dicho valor y se comienza con la variación de otro parámetro, llevando a cabo el mismo proceso hasta obtener el conjunto de valores que permitan al algoritmo mejorar la eficiencia y eficacia.

Una vez obtenido los mejores valores para cada parámetro, realizamos muestras para un rango más pequeño, tomando como punto medio, el valor fijado, de modo que se vuelva a realizar el proceso de sintonización hasta fijar nuevamente los valores.

4. Sintonización de parámetros

Los valores obtenidos para cada uno de los parámetros de control al término de la evaluación de todas y cada una de las muestras, son considerados como los valores de sintonización.

La sintonización de parámetros se lleva a cabo con la finalidad de identificar aquellos valores correspondientes a las variables de control, que influyen de manera positiva en la calidad de la solución, lo que permite obtener una mejora en el desempeño del algoritmo.

3.6 Complejidad del algoritmo búsqueda local iterada

Una vez definidos los parámetros de control que hacen que el algoritmo funcione correctamente, es necesario realizar un estudio que permita conocer su comportamiento y así poder medir su rendimiento, centrándose principalmente en su simplicidad y el uso eficiente de los recursos.

La complejidad computacional de un algoritmo se puede clasificar de acuerdo a la dificultad de resolverlo en función de los siguientes parámetros:

*Espacio: c*antidad de memoria requerida para el almacenamiento de los datos durante la ejecución del algoritmo.

Tiempo: duración de la ejecución del algoritmo.

Ambos parámetros representan el costo requerido por el algoritmo según el tipo de problema que se está tratando, para encontrar una solución.

El tiempo de ejecución de un algoritmo o complejidad temporal T(n), donde n es el tamaño de la entrada, está en función de los parámetros: Datos de entrada, velocidad del procesador y complejidad del algoritmo. La complejidad temporal representa el número de instrucciones simples (asignaciones, comparaciones, operaciones aritméticas, etcétera) que serán ejecutadas por el algoritmo. Por lo regular se considera la complejidad del algoritmo en el peor de los casos, aunque también es importante conocer la complejidad en el mejor y el caso promedio.

Para clasificar si un algoritmo es considerado como bueno o malo nos basamos en las siguientes convenciones:

1. Todos los algoritmos, desde constantes hasta polinomiales, son polinomiales y son clasificados cómo buenos algoritmos.
2. Todos los algoritmos exponenciales y factoriales, son exponenciales y son clasificados cómo malos algoritmos.

La complejidad del algoritmo Búsqueda Local Iterada se calcula en una primera etapa, un análisis de las instrucciones requeridas por el algoritmo para encontrar una solución al problema tratado, de esta forma obtenemos la función de complejidad temporal del algoritmo.

$$T(n) = mn^3 + 4mn^2 + 7n + 39$$

Para el caso de la Búsqueda Local Iterada, *n* representa el número de trabajos a calendarizar y *m* la cantidad de máquinas donde se ejecutarán los trabajos. Por lo tanto la complejidad para el algoritmo de Búsqueda Local Iterada con una estructura sencilla para el peor de los casos tiene una complejidad de: $O(mn^3)$.

CAPÍTULO IV. RESULTADOS EXPERIMENTALES

En este capítulo se presentan los resultados obtenidos en las pruebas experimentales realizadas al algoritmo de búsqueda local iterada, donde se implementa la estructura de vecindad híbrida a búsqueda local. Estos resultados son comparados con los obtenidos por tres diferentes autores que abordan el mismo problema y con las mismas instancias de pruebas.

4.1 Descripción del equipo utilizado

Para llevar a cabo las pruebas experimentales correspondientes al algoritmo de Búsqueda Local Iterada, se utilizó un equipo tipo laptop con las siguientes características (Tabla 4.1).

Tabla 4. 1 Equipo de cómputo utilizado

Descripción del equipo utilizado		
Procesador	**Memoria**	**S.O.**
Pentium a 2.0 GHz	2 GB	Windows 7

Para probar el funcionamiento del algoritmo, se usó un conjunto de benchmarks para FJSSP de tamaño pequeño, mediano y grande propuestas por Brandimarte [1993], las características de las instancias de prueba, se muestran en la tabla 4.2 ordenadas por la cantidad de operaciones.

Tabla 4. 2 Características de los benchmarks utilizados

Instancia	Trab/Maq/Op
MK01	**10/6/55**
MK02	**10/6/58**
MK07	**20/5/100**
MK03	**15/8/150**

4.2 Análisis de factibilidad de estructuras N4 y N6

Las estructuras de vecindad son técnicas utilizadas con el propósito de mejorar una solución, para la cual es necesario moverse paso a paso desde una solución inicial hacia una solución vecina que proporcione el valor mínimo de la función objetivo, según sea el caso. El tipo de movimiento define el tipo de estructura y tamaño de la vecindad [Martínez *et al.*, 2011]. Es sabido que si un par de operaciones que pertenecen a la ruta crítica de una solución son intercambiadas, el resultado es una solución factible [Cruz-Chávez *et al.,* 2009].

La rápida convergencia de la búsqueda local es relativamente lenta por que el tamaño de la vecindad es grande cuando se resuelven problemas grandes de FJSSP. Un enfoque prometedor para mejorar la velocidad de

convergencia es usar el concepto de la ruta crítica en la fase de planificación de operaciones. La ruta crítica es el camino más largo. Puede haber varias rutas críticas en un grafo disyuntivo y el tamaño para cada ruta crítica tiene exactamente el mismo makespan. Una operación de la ruta crítica es llamada operación crítica. Una operación crítica no puede ser retrasada sin incrementar el makespan de la planificación. Es posible descomponer la ruta crítica en un número de bloques. Un bloque es una secuencia máxima de operaciones adyacentes críticas procesadas en la misma máquina. La primera y la última operación de este bloque son llamadas cabeza y cola de bloque, respectivamente. Las otras operaciones son llamadas operaciones internas [Mastrolli, 1998].

En esta sección se presentan dos estructuras de vecindad (N4 y N6) que son aplicadas en métodos de búsqueda local para el FJSP. Moviendo una operación se produce un vecino de la solución actual S. Con el objetivo de minimizar el makespan, el conjunto de operaciones candidatas a ser movidas deben pertenecer a la ruta crítica y estas operaciones no deben ser la cabeza ni la cola del bloque.

En primer lugar, se crea una solución factible como se puede ver en la figura 4.1 y se calcula la ruta crítica, que es la ruta más larga (fuente, O_1, O_7, O_5, O_6, O_8, O_9 destino), se muestra de color más oscura. Los bloques de operaciones críticas que se forman a partir de la solución presentada son los siguientes: **Bloque 1→**(O_1, O_7, O_5) **Bloque 2→** (O_6, O_8). Los intercambios posibles para las estructuras N4 y N6 se describen en las secciones 3.3.2 y 3.3.4 respectivamente. Para usar estas estructuras de vecindad se deben considerar las siguientes propiedades:

- El bloque debe contener al menos tres operaciones. Por ejemplo: bloque 2.
- Verificar que no se viole la restricción de precedencia.

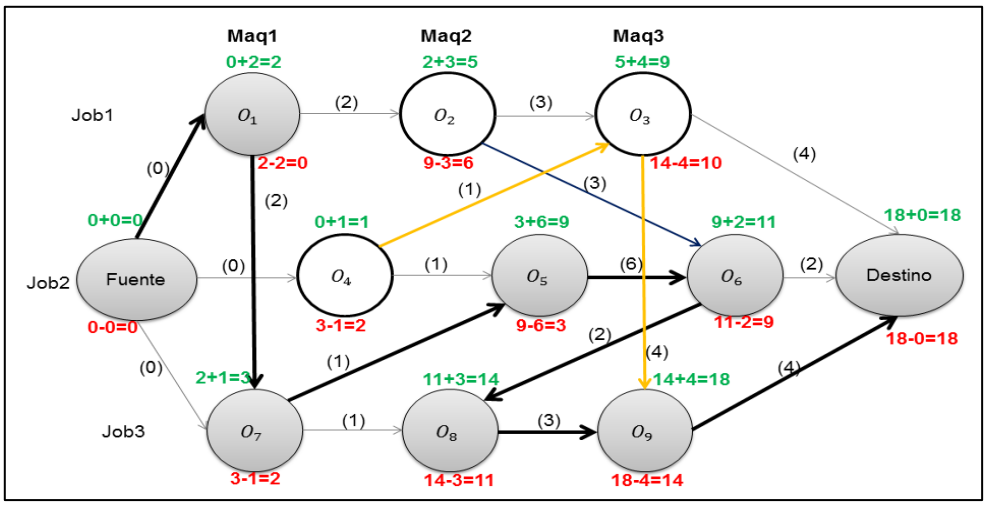

Figura 4. 1 Solución factible de 3 trabajos y 3 máquinas

Para conocer el número de movimientos factibles dada una solución, se contabiliza los movimientos que no violen las restricciones (capacidad de recursos y precedencia), los movimientos no permitidos y el total de movimientos, como se muestra en la figura 4.2 y 4.3.

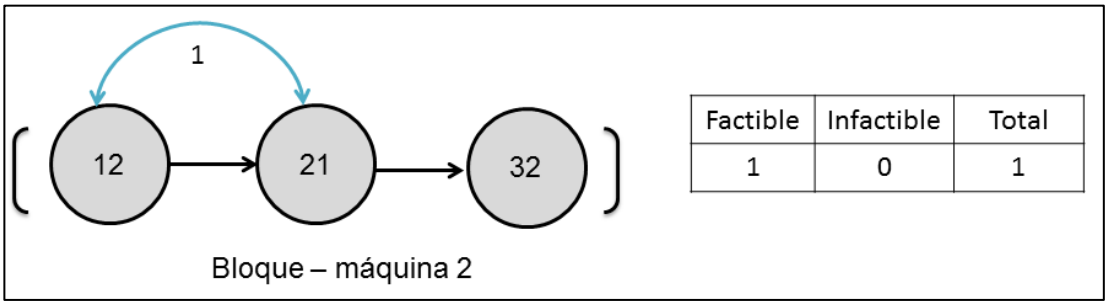

Figura 4. 2 Movimiento N4 factible hacia atrás

Para contabilizar un movimiento como permitido, se evalúa que no se rompa la restricción de precedencia (hacia atrás) como se muestra en la figura 4.2, si no se rompe, se incrementa el contador de movimientos factibles (factible) y el contador de total de movimientos (total). Después, se valora con la misma operación pero al lado contrario (hacia delante) como se puede ver en la figura 4.3.

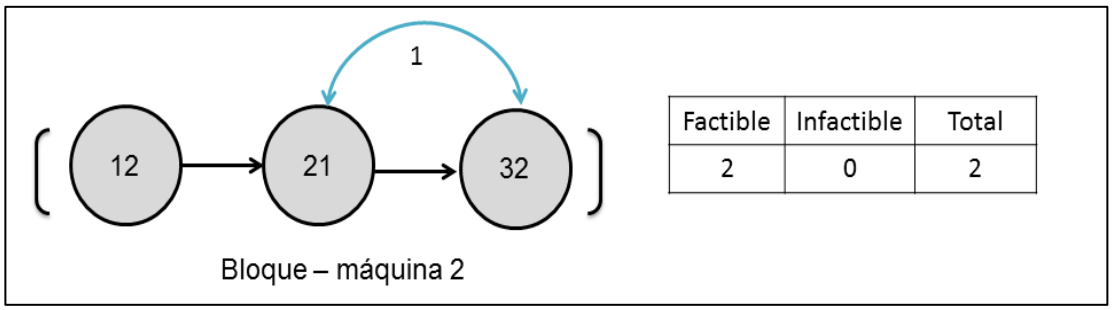

Figura 4. 3 Movimiento N4 factible hacia adelante

Se cuentan cada uno de los movimientos permitidos de cada operación dentro de un mismo bloque, al terminar el bloque, se evalúa la primera operación interna del bloque siguiente y de igual manera hasta la última operación interna. Una vez terminados de evaluar los bloques que se encuentren de una solución, se genera una nueva solución, se calcula la ruta crítica e identifican los bloques para empezar a contabilizar los movimientos permitidos de las operaciones internas.

Para el procedimiento anterior, se realizaron un conjunto de 30 pruebas. Las cuales consistieron en generar 100 soluciones aleatorias y contar los movimientos factibles de cada una de las estructuras de vecindad aplicadas a cada una de las instancias de prueba (MK01, MK02, MK03 y MK07). La tabla 4.3 muestra el total de los movimientos factibles e infactibles, de cada instancia de prueba aplicando la estructura de vecindad N4.

Tabla 4. 3 Movimientos factibles e infactibles de la estructura de vecindad N4

Estructura de vecindad N4				
Instancia	**Factibles**	**%**	**Infactibles**	**%**
MK01	20891	**75**	6913	**25**
MK02	19358	**86**	3082	**14**
MK03	40014	**80**	9984	**20**
MK07	47559	**79**	12622	**21**

La tabla 4.4 presenta los movimientos permitidos y no permitidos, para las mismas instancias de pruebas pero aplicando la estructura de vecindad N6.

Tabla 4. 4 Movimientos factibles e infactibles de la estructura de vecindad N6

Estructura de vecindad N6				
Instancia	**Factibles**	**%**	**Infactibles**	**%**
MK01	21024	**75**	7066	**25**
MK02	19834	**89**	2436	**11**
MK03	41562	**83**	8622	**17**
MK07	46733	**78**	13436	**22**

En las tablas 4.3 y 4.4 se puede observar que el porcentaje de movimientos permitidos para las estructuras de vecindad N4 y N6, aplicadas a las instancias de prueba, es mayor que el porcentaje de movimientos no permitidos. Teniendo en cuenta esta información, las posibilidades de obtener un movimiento permitido por los movimientos generados al utilizar estas estructuras de vecindad son: tres de cada cuatro intentos en promedio.

4.3 Análisis de sensibilidad

Para llevar a cabo el análisis de sensibilidad de los parámetros de control del algoritmo de Búsqueda Local Iterada aplicado al problema de Calendarizar Trabajos en Talleres de Manufactura Flexible, se siguió la metodología propuesta por Martínez-Oropeza [2010], la cual se explica en el capítulo 3. A continuación se aplica la metodología al FJSSP.

1. *Selección de parámetros de control*

De acuerdo al análisis realizado tanto al método aplicado como al problema tratado, las variables utilizadas para llevar a cabo el análisis de sensibilidad del algoritmo Búsqueda Local Iterada aplicado al FJSSP, son los las siguientes:

- Tamaño de la vecindad (CP_BL), donde *m* es la cantidad de máquinas disponibles y *n* es el número de trabajos a realizar.
    - N1 → m (n-1)

- N5 → m (n – 1)
- N4 y N6 → 2m (n - 2)

- Cantidad de repeticiones de la Búsqueda Local (CP_BLI)

2. Establecer rangos de evaluación

Una vez que los parámetros de control se han identificado, es necesario establecer los rangos que van a ser utilizados para realizar el análisis de sensibilidad a cada una de las variables identificadas en el paso uno de la metodología. En el criterio de paro de la Búsqueda Local Iterada (CP_BLI), se corrió el algoritmo durante aproximadamente una hora, con el objetivo de conocer el número de iteraciones donde la solución dejaba de mejorar, obteniendo el valor promedio de 30 pruebas, es decir la convergencia del algoritmo. Para el caso del parámetro CP_BL, en la Tabla 4.5 se muestra el rango de evaluación (Límite inferior = 1, Límite superior = 5), que se refiere al número de veces que se repite el tamaño de la vecindad en la búsqueda local.

Tabla 4. 5 Rangos utilizados en los parámetros de control para el análisis de sensibilidad.

Parámetro de control	Límite inferior	Límite superior
CP_BLI	Convergencia del algoritmo	1 Hora
CP_BL	1	5

3. Pruebas a rangos de evaluación

Para realizar el análisis de sensibilidad a los parámetros de control mencionados anteriormente, se realizó un conjunto de 30 repeticiones, utilizando la instancia de prueba más grande con 15 trabajos y 8 Máquinas (ver tabla 4.2) del conjunto de instancias de pruebas Brandimarte [1993] donde:

- Se ejecutó el algoritmo durante una hora y se analizaron los datos pertinentes para conocer el valor donde el algoritmo convergió y se

fijó, la prueba más representativa para cada función de vecindad se muestra en la Figura 4.4.

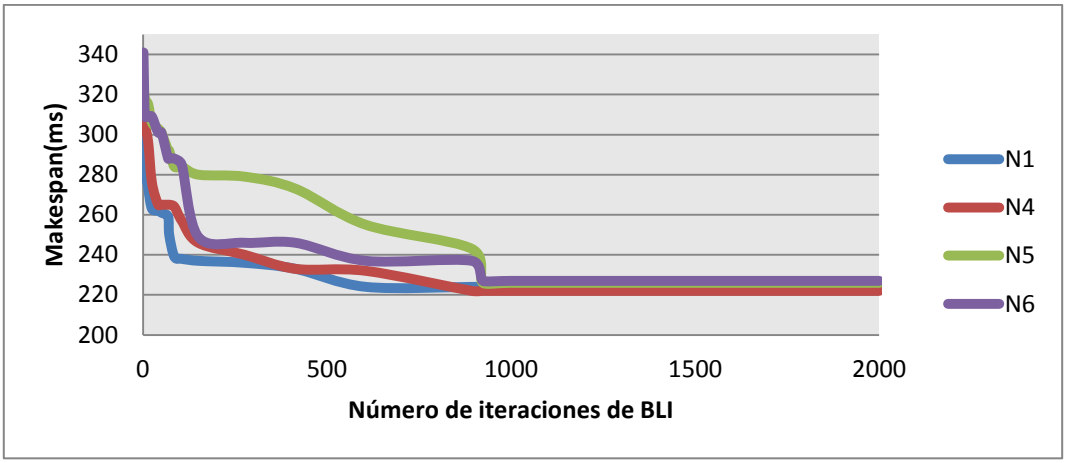

Figura 4. 4 Valores de CP_BLI donde el algoritmo converge para las estructuras de vecindad

- Una vez fijado el valor de la convergencia del algoritmo, se probaron las repeticiones del tamaño de vecindad (CP_BL), de acuerdo a los rangos establecidos (de uno a cinco veces el tamaño de la vecindad).

Tabla 4. 6 Valores de CP_BL

Estructura de vecindad	Rep. Tan. Vecindad
N1	2
N4	2
N5	2
N6	2

En el análisis realizado a cada una de las variables, se pudo observar que, mientras el tamaño de la vecindad sea más grande, la calidad de las soluciones óptimas locales van a ser mejores, así como la precisión de la solución final encontrada, esto es, ya que aumentando el tamaño de vecindad existen un mayor número de combinaciones (soluciones) a las cuales se puede acceder desde una solución inicial. Por lo tanto, una heurística con vecindario grande va a ser más efectivo, de otra manera, el tiempo requerido para realizar una iteración dentro de un vecindario va a ir

incrementando en función al tamaño de la vecindad. Conforme a esto, se requiere encontrar un tamaño de vecindad adecuado, que permita generar el mejor rendimiento posible del algoritmo (Tabla 4.6).

4. Sintonización de parámetros

Los valores obtenidos para cada uno de los parámetros de control evaluados, al término de las pruebas experimentales con las variables mostradas en la Tabla 4.5, dan como resultado la sintonización de los parámetros de control.

La sintonización de los parámetros de control se lleva a cabo con la finalidad de identificar aquellos valores correspondientes a las variables de control, que influyen de manera positiva en la calidad de la solución, lo que permite tener una mejora en el desempeño del algoritmo.

Con base en el análisis de sensibilidad realizado, se obtuvieron los siguientes valores de sintonización Tabla 4.7.

Tabla 4. 7 Valores sintonizados correspondientes a los parámetros de control evaluados durante el análisis de sensibilidad

Estructura de vecindad	Rep. Tan. Vecindad	CP_BLI
N1	2	498
N4	2	550
N5	2	502
N6	2	580

4.4 Resultados experimentales

Para conocer el comportamiento de un algoritmo, es necesario recurrir al cálculo de parámetros estadísticos como son la media, desviación estándar y error relativo. La media permite obtener el promedio de las soluciones encontradas para cada instancia. La desviación estándar permite conocer que tan dispersas están las soluciones con respecto a la media aritmética. El error relativo permite conocer el porcentaje de error de la mejor solución

encontrada con respecto a una cota establecida, que para este caso es la solución óptima correspondiente a cada instancia. De acuerdo a los resultados obtenidos por dichos parámetros, se puede determinar si un algoritmo es bueno o no para cierto problema. Para el cálculo del error relativo se utilizó la ecuación:

$$ER = \frac{valorSolucion - óptimo}{óptimo} * 100 \; \forall \; óptimo > 0;$$

Para realizar las pruebas experimentales, se ejecutaron 30 veces cada uno de los problemas de prueba antes mencionados, tomando como criterio de paro el número de iteraciones y el tamaño de la vecindad, de acuerdo al análisis de sintonización hecho (ver tabla 4.7). Primero, se ejecutaron cada una de las estructuras de vecindad implementas, incluyendo la estructura de vecindad híbrida, con el objetivo de conocer el rendimiento de cada una de ellas por separado. Los resultados se presentan en la sección 4.5.

4.5 Análisis de eficacia y eficiencia del algoritmo

El análisis de eficacia y eficiencia se realiza para evaluar tanto la calidad de las soluciones obtenidas por un algoritmo, como los recursos y tiempo necesarios para su ejecución. En cuanto a la eficacia, se realizaron dos comparaciones del algoritmo propuesto:

- Evaluando las estructuras de vecindad simples y la estructura de vecindad híbrida (sección 4.5.1).
- Con algoritmos de la literatura como Recocido simulado [Martínez-Rangel, 2008], Algoritmo Genético [Zhang *et al.,* 2008] y Búsqueda Tabú [Jun-Bing *et al.,* 2010] (sección 4.6).

Por otro lado, para conocer la eficiencia del algoritmo se compararon los resultados obtenidos por cada una de las estructuras de vecindad bajo las mismas condiciones en la misma máquina, esto porque no se cuenta con los ejecutables de los algoritmos reportados en la literatura, lo cual hace imposible hacer una comparación con los mismos (sección 4.5.2).

4.5.1 Análisis de eficacia

Se llevó a cabo el análisis de eficacia de las estructuras de vecindad, comparando los resultados obtenidos por cada una de ellas contra la estructura de vecindad híbrida, con el fin de conocer si la propuesta de hibridación obtiene resultados eficaces. En primer instancia se compararon la calidad de las soluciones, así como, el error relativo y la desviación estándar para conocer el error de la mejor solución encontrada con respecto a una cota establecida y que tan dispersas están las soluciones del óptimo respectivamente.

Tabla 4. 8 Resultados de las estructuras de vecindad N1 y N4.

		N1					N4				
Benchmarking	**optó**	**mejor**	**media**	**peor**	**ER**	**DS**	**mejor**	**media**	**peor**	**ER**	**DS**
MK01	**40**	46	50.8	56	**15.0**	5.00	46	55.6	63	**15.0**	8.52
MK02	**26**	29	44.4	304	**11.5**	35.67	29	37.7	43	**11.5**	7.07
MK03	**204**	226	245.8	264	**10.8**	19.01	249	293.3	322	**22.1**	36.78
MK07	**140**	150	167.8	181	**7.1**	15.56	157	181.1	196	**12.1**	19.68

Tabla 4. 9 Resultados de las estructuras de vecindad N5 y N6.

		N5					N6				
Benchmarking	**optó**	**mejor**	**media**	**peor**	**ER**	**DS**	**mejor**	**media**	**peor**	**ER**	**DS**
MK01	**40**	47	53.1	58	**17.5**	5.51	50	56.7	63	**25.0**	6.50
MK02	**26**	30	35.0	41	**15.4**	5.51	34	46.0	51	**30.8**	8.74
MK03	**204**	268	329.1	381	**31.4**	56.56	275	312.7	338	**34.8**	31.70
MK07	**140**	186	213.2	231	**32.9**	22.66	183	199.8	214	**30.7**	15.52

De acuerdo a esto, se tiene que ***mejor*** representa la mejor solución encontrada de acuerdo al total de ejecuciones realizadas para cada instancia, ***media*** es el promedio de todas las operaciones, ***peor*** es la solución más lejana al óptimo encontrado, **o*ptó*** representa el mejor resultado reportado en la literatura para cada una de las instancias, ***ER*** representa el error relativo de la mejor solución encontrada con respecto al óptimo y ***DS***

muestra que tan dispersas están las soluciones respecto al óptimo (desviación estándar σ).

Como se puede observar en los resultados obtenidos en las Tablas 4.8 y 4.9, al calcular el error relativo del mejor resultado obtenido para cada una de las instancias, por el algoritmo de Búsqueda Local Iterada con cada una de las estructuras de vecindad, se obtiene que la estructura de vecindad que menor porcentaje de error tiene, es la función de vecindad N1 en todas las instancias de pruebas tratadas. En base a este resultado, se decidió comparar los resultados obtenidos por la estructura N1 con la implementación de la estructura de vecindad híbrida aplicada a Búsqueda Local, tomando como parámetro de control el mayor número de repeticiones y del tamaño de vecindad más grande. Los cuales son: 580 y (m * (n - 1) * 2) respectivamente. Los resultados se presentan a continuación en la Tabla 4.10.

Tabla 4. 10 Resultados de la mejor estructura contra la estructura híbrida de vecindad

		Hibrida					N1				
Benchmarking	**optó**	**mejor**	**media**	**peor**	**ER**	**DS**	**mejor**	**media**	**peor**	**ER**	**DS**
MK01	**40**	**41**	**46.6**	**51**	**2.5**	5.01	**46**	**50.8**	**56**	**15.0**	5.00
MK02	**26**	**28**	**32.3**	**42**	**7.7**	7.17	**29**	**44.4**	**304**	**11.5**	35.67
MK03	**204**	**212**	**236.4**	**249**	**3.9**	18.81	**226**	**245.8**	**264**	**10.8**	19.01
MK07	**140**	**147**	**253.2**	**172**	**5.0**	51.95	**150**	**167.8**	**181**	**7.1**	15.56

De acuerdo a los resultados mostrados en la tabla 4.10, se puede observar que, la estructura de vecindad híbrida propuesta en este trabajo de investigación, la cual es una combinación de las cuatro estructuras de vecindad mostradas en las secciones anteriores, aplicadas a búsqueda local tiene un menor porcentaje de error relativo en todas las instancias de pruebas. Se puede observar claramente los resultados de la tabla 4.10 en la figura 4.4, que muestra la calidad de las soluciones que cada una de las estructuras obtiene al ser aplicadas a una instancia de prueba.

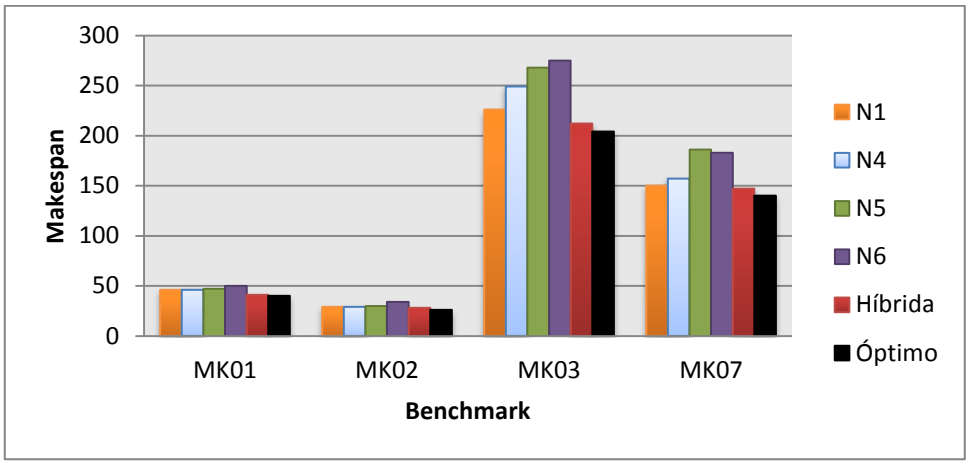

Figura 4. 5 Resultados obtenidos de las estructuras de vecindad

En la figura 4.5 se observa que la estructura simple que mejores resultados tiene es la N1, siendo superada en casi todos los casos por la estructura híbrida que combina las cuatros estructuras simples estudiadas. Ya que comparada con la columna en color negro que corresponde al óptimo reportado en la literatura, la estructura de vecindad híbrida obtiene resultados cercanos a dicha columna.

A continuación (tabla 4.11), se muestran los resultados de los errores relativos de las estructuras de vecindad, esto con el objetivo de conocer el porcentaje de error de la mejor solución encontrada con respecto a una cota establecida.

Tabla 4. 11 Error relativo obtenido de la ejecución del algoritmo Búsqueda Local Iterada

Benchmarking	N1	N4	N5	N6	Híbrida
MK01	15.0	15.0	17.5	25.0	2.5
MK02	11.5	11.5	15.4	30.8	7.7
MK03	10.8	22.1	31.4	34.8	3.9
MK07	7.1	12.1	32.9	30.7	5.0

Como se puede observar, los resultados obtenidos por cada una de las estructuras de vecindad tienen un error relativo alto con respecto a los obtenidos con la implementación de una estructura de vecindad híbrida, es decir, la estructura de vecindad híbrida encuentra soluciones más cercanas a

los óptimos reportados por otros autores. Para observar esta diferencia de forma más clara, se muestra la figura 4.6 comparando las estructuras de vecindad contra la estructura híbrida propuesta, los datos graficados corresponden a los errores relativos de cada uno de estas estructuras de vecindad.

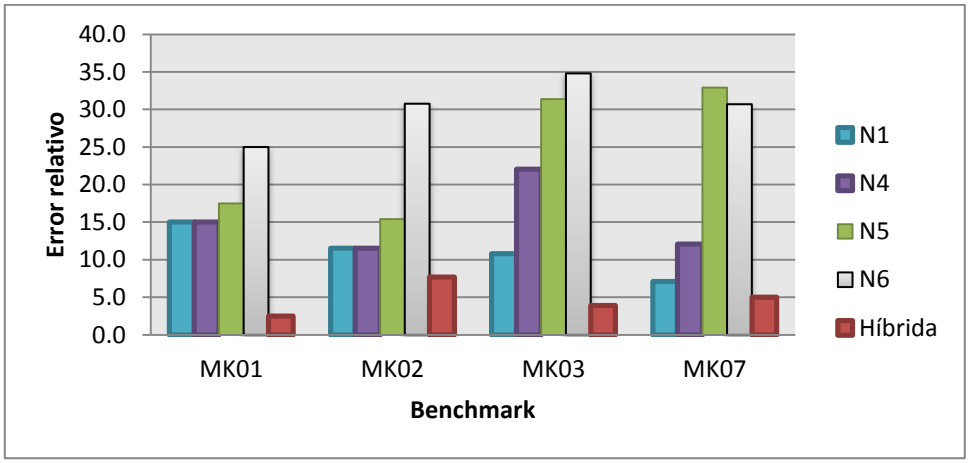

Figura 4. 6 Gráfica comparativa del erro relativo de las estructuras de vecindad e híbrida.

De acuerdo a la figura 4.6, es posible observar claramente en base al error relativo, que las mejores soluciones al FJSSP utilizando Búsqueda Local Iterada, se obtuvieron al ejecutar la estructura híbrida de vecindad, ya que tiene un menor error relativo, quedando en segundo lugar la estructura de vecindad N1. Por lo que se comprueba que la aplicación de la estructura de vecindad híbrida hace eficaz al algoritmo de BLI.

4.5.2 Análisis de eficiencia

Además de la calidad de las soluciones, otro factor a evaluar es el tiempo requerido para un algoritmo obtenga una solución al problema tratado.

El primer estudio del comportamiento del BLI se realizó sobre el benchmarking mk01 ampliamente utilizado por todo tipo de algoritmos de calendarización y relacionado con muchas publicaciones científicas, se trata de una instancia de diez trabajos y seis máquinas.

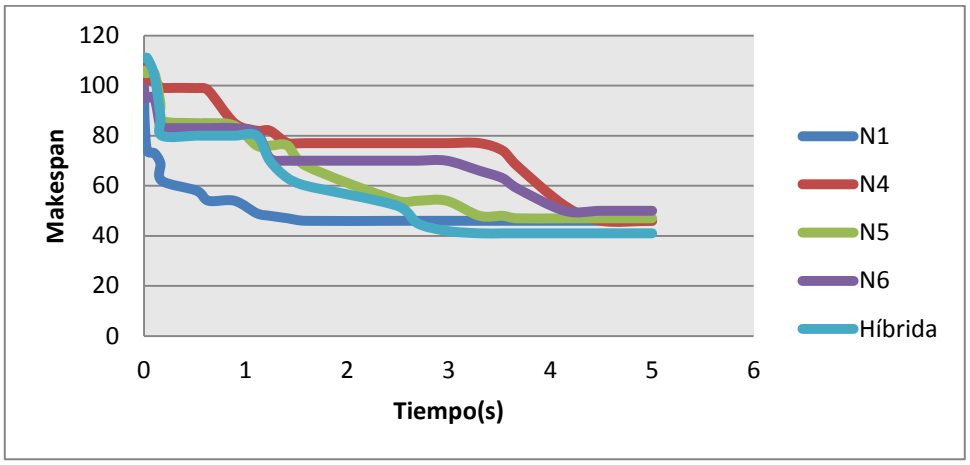

Figura 4. 7 Análisis de eficiencia benchmarking mk01

En la figura 4.7 se puede observar que la estructura de vecindad N1obtiene mejores soluciones más rápido, es decir, más eficiente, aunque al final de la ejecución de las estructuras de vecindad, la estructura híbrida obtiene la mejor calidad de las soluciones.

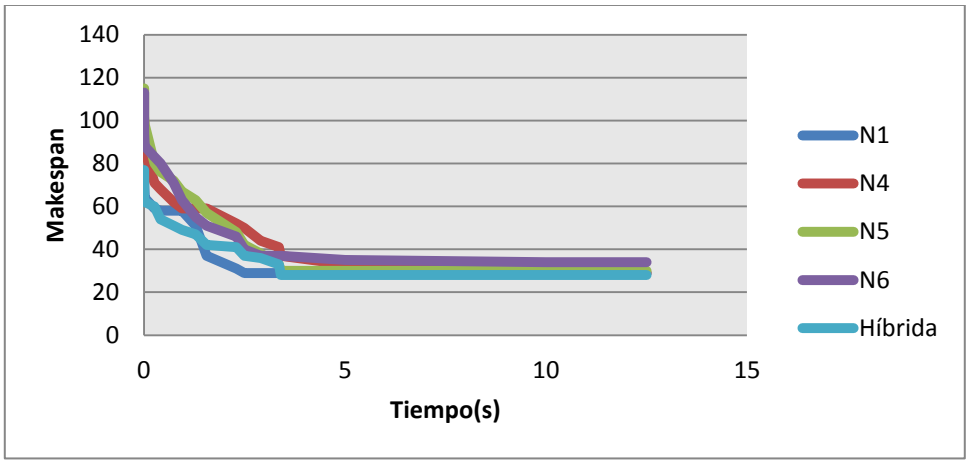

Figura 4. 8 Análisis de eficiencia benchmarking mk02

De acuerdo a la figura 4.8, es posible observar que la estructura más eficaz para los benchmarking mk02 es la estructura híbrida propuesta en este trabajo de investigación, en cuanto a la eficiencia la estructura de vecindad N1 es la mejor de todas las estructuras estudiadas.

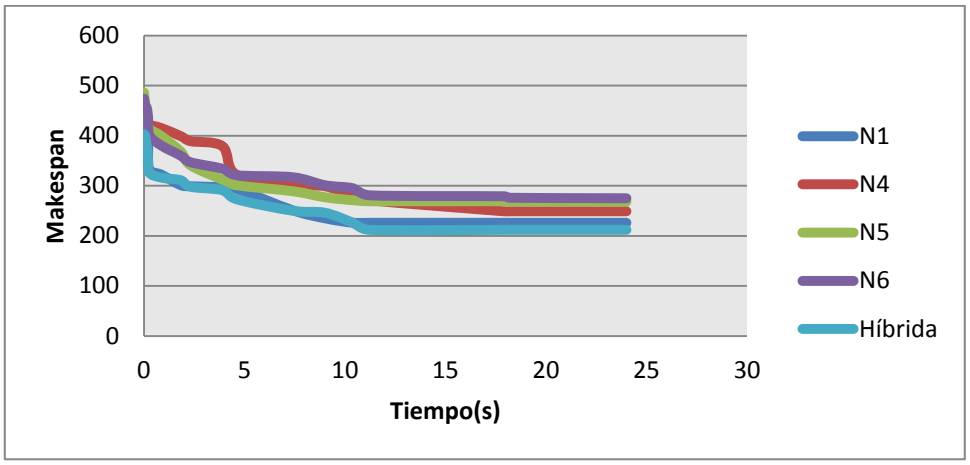

Figura 4. 9 Análisis de eficiencia benchmarking mk03

En la figura 4.9 se puede observar que la estructura de vecindad N1y la estructura híbrida obtienen soluciones eficientemente, al término de la ejecución, la estructura híbrida obtiene la mejor calidad de las soluciones.

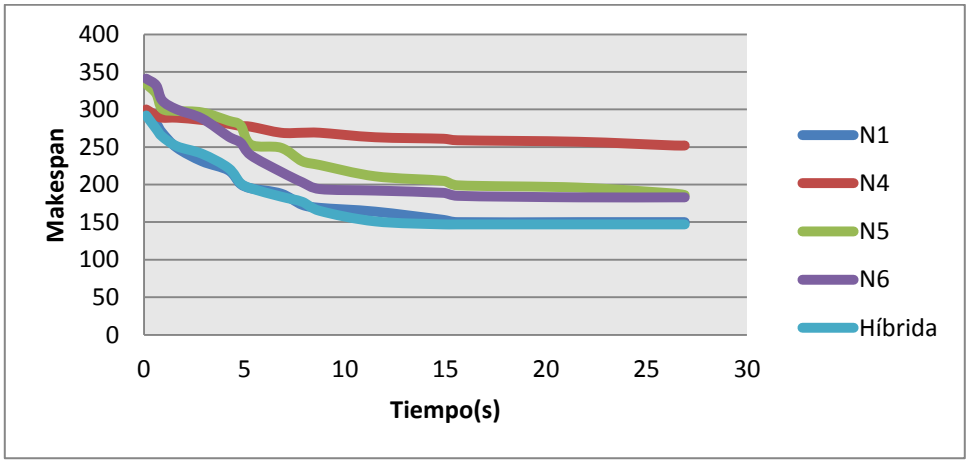

Figura 4. 10 Análisis de eficiencia benchmarking mk07

Por lo que se puede observar en la figura 4.10, la estructura de vecindad N1 como en todos los benchmarks, es la más eficiente, quedando en segundo lugar la estructura híbrida propuesta, siendo así competitiva en eficiencia.

Por otro lado, al final de las ejecuciones del algoritmo la estructura que mejor calidad de las soluciones, es decir, la más eficaz, es la estructura de vecindad híbrida, quedando en segundo lugar la estructura de vecindad N1.

Con la finalidad de dar una explicación al comportamiento de la estructura híbrida aplicada al algoritmo de búsqueda local iterada, la cual obtuvo bueno resultados comparada con la estructura N1 que en muchos estudio se menciona que es de las estructuras más utilizadas, ya que es sencilla de aplicar y obtiene muy buenos resultados en un tiempo considerablemente bajo.

De acuerdo a esto, se puede decir que una estructura de vecindad híbrida funciona adecuadamente al combinar diversas características de las estructuras, ya que implementa propiedades consideradas como sencillas en el casi de la N1 y N5, así como propiedades con una complejidad mayor. Estas características le ayudan al algoritmo de búsqueda local iterada, a tener una mayor exploración del espacio de soluciones y poder acceder soluciones que con estructuras simples sería inaccesible.

Los resultados del tiempo de ejecución consumido en promedio por cada una de las estructuras de vecindad y de la estructura híbrida propuesta se muestran en la Tabla 4.12.

Tabla 4. 12 Tiempos promedio de ejecución en segundos requeridos por cada estructura de vecindad

Benchmark	N1	N4	N5	N6	Híbrida
MK01	3.6	7.8	4.14	9.9	3.53
MK02	3.3	12.5	3.8	11.6	3.4
MK03	10.4	17.9	18.2	24	11.2
MK07	15.6	26.4	26.9	21.6	14.9

En la tabla 4.12 se muestran el tiempo promedio que tarda el algoritmo con cada estructura de vecindad en encontrar la mejor solución para una ejecución, es decir, si se lanza una ejecución con una instancia específica, la mayoría de las veces tarda este tiempo en encontrar una solución expresado en segundos. Se puede observar que la estructura de vecindad híbrida es la competitiva con respecto a la más eficiente para los cuatro tipos de instancia (estructura de vecindad: N1). Para ver mejor el desempeño en cuanto a

eficiencia de las estructuras de vecindad y de la estructura de vecindad híbrida, a continuación se muestra la figura 4.11.

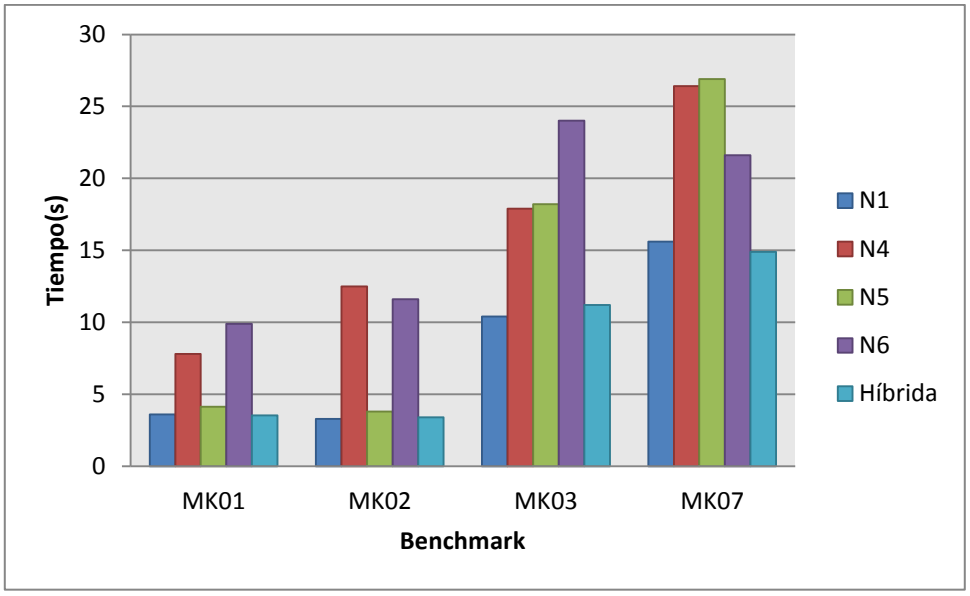

Figura 4. 11 Eficiencia de las estructuras de vecindad vs estructura híbrida

En la figura 4.11 podemos ver claramente que la estructura de vecindad N1 es la más eficiente, ya que la calidad de la solución mejora con mayor rapidez, esto es, el makespan disminuye en gran medida con que pasa el tiempo, esto se debe a que los movimientos simples son más rápidos de realizar además de tener la certeza de que no habrá ningún movimiento que genere soluciones infactibles.

4.6 Análisis de resultados

Para comparar la mejora del algoritmo de búsqueda local iterada utilizando una estructura de vecindad híbrida, como es la aportación de la tesis, se realizaron pruebas experimentales al algoritmo con cada una de las estructuras de vecindad implementadas, así como también, la estructura de vecindad híbrida propuesta tal y como se muestra en la literatura, es decir, para cada una de las instancias de prueba se ejecutó 30 veces el algoritmo bajo los parámetros de sintonización obtenidos. Cabe mencionar que estas

pruebas fueron realizadas bajos las mismas condiciones manejadas para cada estructura.

Una vez obtenidos los resultados de las estructuras de vecindad implementadas y de la estructura de vecindad híbrida, se llevó a cabo una recopilación de los resultados obtenidos por otros autores reportados en la literatura. Con la finalidad de realizar una comparación de los resultados obtenidos por la estructura de vecindad híbrida, se ejecutaron cuatro instancias de prueba (ver tabla 4.2) en un algoritmo de Recocido Simulado secuencial, mismo que fue desarrollado dentro del grupo de trabajo [Martínez-Rangel, 2008]. También, se tomó como punto de comparación, otros algoritmos aplicados al mismo tipo de problema, un Algoritmo Genético secuencial propuesto por [Zhang *et al.,* 2008] y un algoritmo de Búsqueda Tabú secuencial [Jun-Qing *et al.,* 2010].

En la Figura 4.12**,** se muestra claramente la calidad de las soluciones encontradas por cada uno de los algoritmos evaluados (Tabla 4.11) con respecto a la solución óptima de cada instancia.

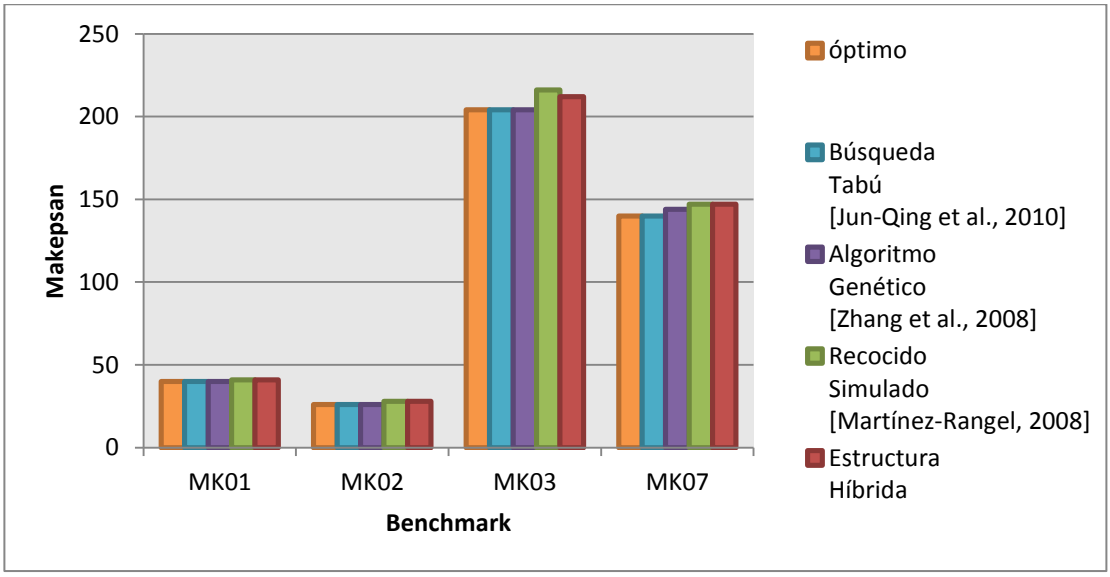

Figura 4. 12 Gráfica comparativa del funcionamiento de los algoritmos evaluados con respecto a la solución óptima.

Para tener un panorama más claro en cuanto la calidad de las soluciones, se muestra en la figura 4.13 una gráfica comparativa de los errores relativos obtenidos por casa uno de los cuatro algoritmos evaluados para cada tamaño de instancia. Cabe mencionar que los algoritmos Búsqueda Tabú y Genético, tienen un error relativo de cero. En el caso específico del Benchmark Mk07 para Búsqueda Tabú el autor reporta una disminución del makespan.

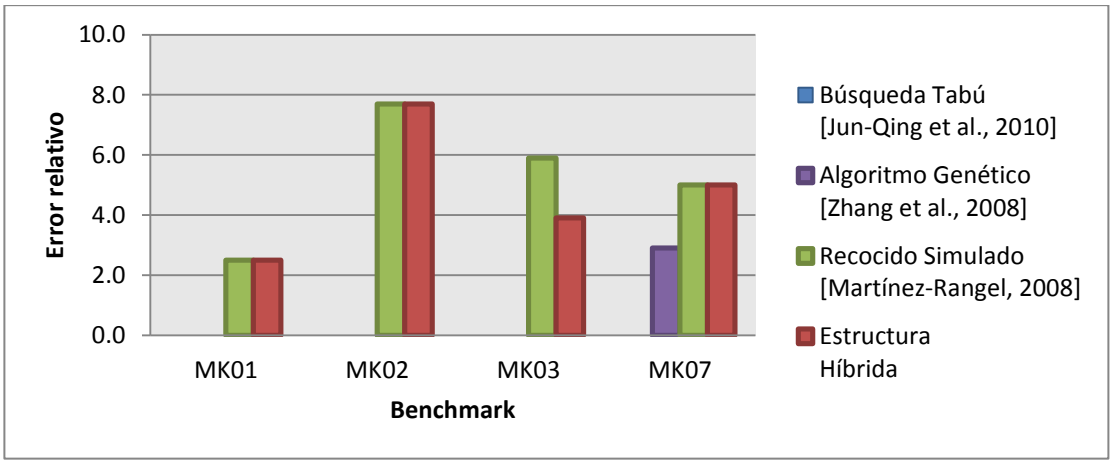

Figura 4. 13 Gráfica comparativa del error relativo obtenido por cada uno de los algoritmos evaluados.

En la figura 4.13 se observa claramente que el algoritmo de recocido simulado y la estructura de vecindad híbrida tienen un comportamiento muy parecido, mientras que los algoritmos genético búsqueda tabú no aparecen, ya que su error relativo en todos los casos es de cero, es decir, estos algoritmos tienen resultados iguales en las ejecuciones de los mismos. Cabe señalar que la propuesta de este trabajo de investigación logró mejorar en al menos una instancia de prueba los resultados del algoritmo recocido simulado, encontrando una mejor calidad de la solución y así mismo disminuir el error relativo de la misma instancia.

CAPÍTULO V. CONCLUSIONES Y TRABAJOS FUTUROS

En este capítulo, con base en los resultados obtenidos de las pruebas experimentales realizadas con el algoritmo de búsqueda local iterada, utilizando la estructura híbrida propuesta, se presentan las conclusiones y los trabajos derivados de la investigación.

5.1 Conclusiones

De acuerdo a las pruebas experimentales realizadas al algoritmo búsqueda local iterada, utilizando la estructura híbrida de vecindad propuesta, se concluye que la estructura de vecindad híbrida propuesta es más eficaz y competitiva en eficiencia comparada con cada una de las estructuras simples aplicadas al algoritmo, ya que al analizar el error relativo de las cuatro estructuras que la conforman, la estructura híbrida tiene un menor porcentaje de error relativo y en cuanto al tiempo alcanza soluciones buenas en un tiempo similares a la más eficiente que es la estructura de vecindad N1.

Al comparar los resultados obtenidos por el algoritmo de recocido simulado secuencial [Martínez-Rangel, 2008], se puede decir que el algoritmo propuesto mediante la implementación de una estructura de vecindad híbrida es competitivo y en el caso del benchmark mk07 tiene una mejor eficacia. Esto es debido a que su rango de error relativo se encuentra alrededor del 4% aplicando la estructura de vecindad híbrida siendo menor al reportado en la literatura. Además, al cotejar los resultados con otros algoritmos propuestos en la literatura (genético [Zhang et al., 2008] y tabú [Jun-Qing et al., 2010]), se puede observar que la implementación de la estructura de vecindad híbrida en búsqueda local iterada hace al algoritmo eficaz, ya que ésta, aumenta la eficacia en la búsqueda por vecindades, al realizar la explotación del vecindario. De manera que, de acuerdo a los resultados obtenidos, un espacio de soluciones más complejo se ve beneficiado con la aplicación de una estructura de vecindad híbrida, debido que al incorporar características de diversas estructuras, los saltos que realiza dentro del espacio de soluciones varían en magnitud, debido a que la selección de la estructura de vecindad a aplicar se lleva de manera aleatoria, haciendo posible el acceso a otras soluciones del espacio de soluciones del FJSSP, lo que una estructura de vecindad simple no puede.

5.2 Trabajos futuros

La propuesta de solución desarrollada durante este trabajo de investigación, servirá como base para realizar a futuro lo siguiente:

- Desarrollar el análisis que permita encontrar métodos eficaces, a fin de evaluar las soluciones obtenidas por la estructura de vecindad híbrida.
- Optimizar el algoritmo de balanceo de carga para mejorar la distribución en instancias donde la cantidad de máquinas sea considerablemente menor que el número de trabajos.
- Mejorar el algoritmo propuesto, implementando una lista tabú con el objetivo de agilizar la explotación del vecindario, evitando caer en movimientos repetidos.
- Probar la estructura de vecindad híbrida en otras metaheurísticas, como por ejemplo, recocido simulado y meméticos, evaluando la eficiencia y eficacia, con las mismas instancias de pruebas.
- Realizar un algoritmo con paso de mensajes (MPI) y paralelo en el que se implemente la estructura de vecindad híbrida, para mejorar su eficiencia.

Referencias

[Adams *et al.* 1988]. Adams J., Balas E., Zawack D. The 'shifting bottleneck procedures for job shop scheduling. Management Science, vol. 34, pp. 391-401, 1988.

[Amico y Turbian, 1993]. Amico, M., Turbian, M.. Applying tabu search to the job shop scheduling problem.. *Annual Operations Research,* Volumen 40, pp. 231-252, 1993.

[Applegate y Cook, 1991]. Applegate, D. y Cook, W. A computational study of the job shop scheduling problem. *ORSA Journal on Computing,* Volumen 3, pp. 149-156, 1991.

[Bagheri *et al.,* 2010] Bagheri A., Zandieh M., Mahdavi I., Yazdani M. An artificial immune algorithm for the flexible job-shop scheduling problem, Future Generation Computer Systems 26 (4) (2010) 533–541. 2010.

[Balas y Vazacopoulos, 1998]. Balas E, Vazacopoulos A. “Guided local search with shifting bottleneck for job shop scheduling”. Manage Sci 42:262-275. 1998.

[Brandimarte *et al.*, 1993]. Brandimarte P. Routing and scheduling in a flexible job shop by tabu search, Annals of Operations Research 41 (3) (1993) 157–183. 1993.

[Burke y Smith, 1997]. Burke, E. K., Smith, A. J. A Memetic Algorithm for the Maitenance Scheduling Problem. *Proceedings of the ICONIP´97 Conference, Dunedin, New Zealand 24-28 November 1997 (published by Springer),* pp. 469-474, 1997.

[Chen *et al*. 1999]. Chen H., Ihlow J., Lehmann C. A genetic algorithm for flexible job-shop scheduling, in: IEEE International Conference on Robotics and Automation, Vol. 2, IEEE, 1999, pp. 1120–1125. 1999.

[Chun, 2008]. Chun-Lung Chen, An Iterated Local Search for Unrelated Parallel Machines Problem with Unequal Ready Times, Proceedings of the IEEE International Conference on Automation and Logistics, pp. 2044-2047, Qingdao, China September, ISBN: 978-1-4244-2502-0. 2008.

[Cruz-Chávez *et al.,* 2009]. Cruz-Chávez Marco A., Frausto-Solis Juan y Cora-Mora. Experimental Analysis of a Neighborhood Generation Mechanism Applied to Scheduling Problems. SPRINGER, 2009.

[Cruz-Chávez, 2005]. Cruz-Chávez, M.. *Cooperación de procesos para el problema de Job Shop Scheduling Problem aplicando Recocido Simulado.* s.l.:Tesis Doctoral. Instituo Tecnológico y de Estudios Superiores de Monterrey, 2005.

[Cruz-Chávez y Rivera-López, 2007]. Cruz-Chávez, M. A., Rivera-López, R. A Local Search Algorithm for a SAT Representation of Scheduling Problems, Lecture Notes in Computer Science, Springer-Verlag Pub., Berlin Heidelberg, ISSN: 0302-9743, Vol.4707, No. 3, pp. 697-709, 2007.

[Dauzère-Pérès y Paulli, 1997]. Dauzère-Pérès S. y Paulli J., An integrated approach for modeling and solving the general multiprocessor job-shop scheduling problem using tabu search, Annals of Operations Research 70 (1997) 281–306. 1997.

[Dell'Amico y Trubian, 1993]. Dell'Amico M., Trubian M., Applying taboo search to the job-shop scheduling problem, Annals of Operations Research 41(1993)231-252, Politecnico di Milano, 1-20133 Milano, Italy. 1993.

[Fattahi *et al.*, 2007]. Fattahi P., Mehrabad M. S., Jolai F. *Mathematical modeling and heuristic approaches to flexible job shop scheduling problems.* s.l.:J Intell Manuf Vol. 18:pp.331-342, 2007.

[Gao *et al.*, 2008]. Gao J., Sun L., Gen M. A hybrid genetic and variable neighborhood descent algorithm for flexible job shop scheduling problems, Computers & Operations Research 35 (9) (2008) 2892–2907. 2008.

[Garey *et al.*, 1976]. Garey M., Jhonson D., Sethi R. The complexity of Flow Shop and Job Shop Scheduling, in Mathematics Of Operation research. Volumen 2, pp. 117-129, 1976.

[Girish y Jawahar, 2009]. Girish B. y Jawahar N., A particle swarm optimization algorithm for flexible job shop scheduling problem, in: IEEE International Conference on Automation Science and Engineering, 2009. CASE 2009, IEEE, 2009, pp. 298–303. 2009.

[Gonçalves *et al.*, 2005]. Gonçalves J., de Maghl aes Mendes J., Resende M. *A hybrid genetic algorithm for the job shop scheduling problem.* s.l.:European Journal of Operational Research, 2005.

[González *et al.*, 2005]. González M. A., Sierra M., Vela, C. R., Varela, R. Genetic Algorithms Hybridized with Greedy Algorithms and Local Search over the spaces of active and semi-active Schedules. *CAEPIA 20051,* pp. 231-240, 2005.

[Gómez et al., 2007]. Gómez Ravetti, Geraldo R. Mateus, Pedro L. Rocha. A Scheduling Problem with Unrelated Parallel Machines and Sequence Dependent Setups. 380 Int. J. Operational Research, Vol. 2, No. 4. Copyright © 2007 Inderscience Enterprises Ltd. 2007.

[Gu y Huang, 1994]. Gu Jun, Huang Xiaofei. Efficient Local Search with Search Space Smoothing: A Case Study of the Traveling Salesman Problem (TSP). IEEE transactions on Systems, Man, and Cybernetics. Vol. 24, No.5. 1994.

[Guo *et al.*, 2007]. Guo Y., Lim. A., Rodrigues B., Yang L. Minimizing the Makespan for Unrelated Parallel Machines. International Journal on Artificial Intelligence Tools (IJAIT). Vol. 16. Issue 3. pp. 399 – 415 Junio, 2007.

[Hillier y Lieberman, 2005] Hiller, F. S. and Lieberman, G. J., Introduction to Operations Research, ISBN: 0-07-113989-3, International Editions, 1995.

[Hillier y Lieberman, 2010]. Hillier F. S., Lieberman G. J. Introducción a la Investigación de Operaciones, Novena Edición, ISBN: 978-607-15-0308-4, 2010.

[Ho y Tay, 2004]. Ho N. y Tay J. An efficient cultural algorithm for solving the flexible job-shop problem. *Evolutionary Computation, CEC2004 on Volumen 2, Issue,* 19-23 June, Volumen 2, pp. 1759-1766, 2004.

[Hoos y Stützle, 2005]. HOOS, H.H.; STÜTZLE, T.: Stochastic local search: foundations and applica-tions, Morgan Kaufmann Publishers, San Francisco, CA. 2005.

[Hurink *et al.*, 1994]. Hurink J., Jurisch B., Thole M. Tabu search for the job-shop scheduling problem with multi-purpose machines, OR Spectrum 15 (4) (1994) 205–215. 1994.

[Jia *et al.*, 2003]. Jia H., Nee A., Fuh J., Zhang Y. A modified genetic algorithm for distributed scheduling problems, Journal of Intelligent Manufacturing 14 (3) (2003) 351–362. 2003.

[Jun-Qing *et al.,* 2010]. Jun-Qing Li, Quan-Ke Pan, P. N. Suganthan y T. J. Chua. A hybrid tabú search algorithm with an efficient neighborhood structure for the flexible job shop scheduling problem. Springer-Verlag London Limited 2010

[Kacem *et al.*, 2001]. Kacem I., Hammadi S., Borne P. Approach by localization and genetic manipulation algorithm for flexible job-shop scheduling problem. *IEEE International Conference on Volumen 4, Issue,* pp. 2599-2604, 2001.

[Kacem *et al.*, 2002]. Kacem I., Hammadi S., Borne P. Pareto-Optimality approach based on uniform design and fuzzy evolutionary algorithms for flexible job-shop scheduling problems (FJSPs). *Systems, Man and Cybernetics, 2002 IEEE International Conference on 6-9 Oct. 2002. Volume: 7, pp. 7. ISBN:0-7803-7437-1*, 2002.

[**Klahorts, 1981**]. Klahorst T. H., Flexible Manufacturing Systems: Combining elements to lower costs, add flexibility, industrial engineering, Vol. 32, No. 11, 1981.

[Laarhoven *et al.*, 1992]. Laarhoven V., Aarts E., Lenstra, J. Job Shop Scheduling by simulated annealing. *Operations Research,* pp. 113-125, 1992.

[Li *et al.*, 2011]. Li J., Pan Q., Suganthan P., Chua T. A hybrid tabu search algorithm with an efficient neighborhood structure for the flexible job shop scheduling problem, The International Journal of Advanced Manufacturing Technology 52 (5) (2011) 683–697. 2011.

[Martínez, 2006]. Martínez, M. A. A. Algoritmo Basado en Búsqueda Tabú para el Cálculo del Índice de Transmisión de un Grafo, Vol. 1, No. 1, pp. 31-39, 2006.

[Martínez-Rangel, 2008]. Martínez, R. M. G. *Algoritmo de recocido simulado paralelizado, aplicado al problema de asignación de recursos en un taller de manufactura flexible sujeto a disposicones de tiempo.* Cuernavaca (Morelos): s.n., 2008.

[Martínez-Oropeza, 2010]. Martínez, O. A., “Solución al Problema de Máquinas en Paralelo no Relacionadas mediante un Algoritmo de Colonia de Hormigas”, Tesis de Maestría, Agosto 2010.

[Martínez *et al.*, 2011]. Martínez Bahena Beatriz, Cruz Chávez Marco A., Diaz-Parra Ocotlán. Estructura Híbrida de Vecindad para el problema del Árbol de Expansión Mínima (MST). Cicos-Springer, 2011.

[Mastrolilli y Gambardella, 1998]. Mastrolilli M. y Gambardella L. M. *Effective Neighborhood Functions for the Flexible Job Shop Shop Problem.* s.l.:Technical Report. Istituto Dalle Molle Di Studi Sull Intelligenza Artifiale, 1998.

[Mastrolilli y Gambardella, 2000]. Mastrolilli M. y Gambardella L. Effective neighborhood functions for the flexible job shop problem, Journal of Scheduling 3 (1) (2000) 3–20. 2000.

[Mateo Manuel, 2009]. Mateo Manuel, Ribas Imma, Ramón Companys. A GRASP Procedure for Scheduling Orders on Parallel Machines with Setup Times. 3rd. International Conference on Industrial Engineering and Industrial Management. XIII Congreso de Ingeniería de Organización. Barcelona-Terrassa, 2009.

[Micheli M., 2009]. Micheli, M.: Algoritmo de búsqueda local iterativa para la programación de piezas en un sistema flow shop híbrido, Universidad Politécnica de Catalunya, Departamento de Organización de Empresas 2009.

[Najid *et al.*, 2002]. Najid N., Dauzere-Peres S., Zaidat A. A modified simulated annealing method for flexible job shop scheduling problem, in: IEEE International Conference on Systems, Man and Cybernetics, 2002, Vol. 5, IEEE, 2002, p. 6. 2002.

[Nakano y Yamada, 1991]. Nakano, R. y Yamada, T. Conventional Genetic Algorithm for Job-Shop Problems, Proceedings of the fourth International Conference on Genetic Algorithm, Morgan Kaufmann; San Mateo, CA, USA, 1991, pp.477-479. 1991.

[Noureddine *et al.*, 2007]. Noureddine L., Ishen S., Slim H., Pierre B. Ant system & Local Search Optimization for flexible Job Shop Scheduling Production. *International Journal of Computers, Comunications & Control,* Volumen 2, pp. 174-184, 2007.

[Nowicki y Smutnicki, 1993]. Nowicki E. y Smutnicki C. A fast taboo search algorithm: for the job shop problem. *Managment Science ,* Volumen 42, pp. 797-813, 1996.

[Papadimitriou y Steiglitz, 1998]. Papadimitriou C. H., Steiglitz K.: Combinatorial Optmization. Algorithms and Complexity. ISBN: 0-486-40258-4. USA. 1998.

[Pezzella *et al.*, 2008]. Pezzella F., Morganti G., Ciaschetti G. A genetic algorithm for the flexible job shop scheduling problem, Computers & Operations Research 35 (10) (2008) 3202–3212. 2008.

[Pei y Shih, 2007]. Pei-Chann Chang, Shih-Hsin Chen. Integrating Dominance Properties with Genetic Algorithms for Parallel Machine Scheduling Problems with Setup Times. Applied Soft Computing. ScienceDirect, 2007.

[Pinedo M., 2001]. Pinedo, M.. *Scheduling Theory, Algorithms, and Systems.* Segunda ed. U.S.A.: Prentice Hall, 2001.

[Pinedo M., 2010]. Pinedo, M. *Scheduling: theory, algorithms and systems.* 4th ed. s.l.:Springer, 2010.

[Raeesi y Kobti, 2012]. M.N. Raeesi, Z. Kobti, A memetic algorithm for job shop scheduling using a critical-path-based local search heuristic, Memetic Computing 4 (3) (2012) 231–245. 2012.

[Rahmati y Zandieh, 2012]. Rahmati S. y Zandieh M. A new biogeography based optimization (BBO) algorithm for the flexible job shop scheduling problem, The International Journal of Advanced Manufacturing Technology 58 (9) (2012) 1115–1129. 2012.

[Rankyn P., 1983]. Rankyn P. The design and operation of FMS. Flexible manufacturing systems, IFS Publications Ltd., Bedford, 1983.

[Roy y Sussmann, 1964]. Roy, B. y Sussmann, B. *Les problemes d'ordonnancement avec contraintes disjonctives.* Paris: SEMA, 1964.

[Stattenberger *et al.*, 2007]. Stattenberger Günther, Dankesreiter Markus, Baumgartner Florian, Scheider Johannes J. On the Neighborhood Structure of the Traveling Salesman Problem Generated by Local Search Moves. J Stat Phys129:623-648. DOI 10.1007/s10955-007-9382-1. Springer Science + Business Media, LLC 2007.

[Sule D., 1997]. Sule, D. R. *Industrial Scheduling.* USA: PWS Publishing Company, 1997.

[Tung *et al.*, 1999]. Tung L., Lin L., Nagi R. Multiple-objective scheduling for the hierarchical control 1069 of flexible manufacturing systems, International Journal of Flexible Manufacturing Systems 11 (4) (1999) 379–409. 1999.

[Wang *et al.*, 2011]. Wang L., Zhou G., Xu Y., Wang S., Liu M. An effective artificial bee colony algorithm for the flexible job-shop scheduling problem, The International Journal of Advanced Manufacturing Technology (2011) 1–13.

[Wang *et al.*, 2012]. Wang L., Wang S., Xu Y., Zhou G., Liu M. A bi-population based estimation of distribution algorithm for the flexible job-shop scheduling problem, Computers & Industrial Engineering 62 (4) (2012) 917–926.

[Xia y Wu, 2005]. Xia W. y Wu Z. An effective hybrid optimization approach for multi-objetive flexible job-shop scheduling problems. *Computers and Industrial Engineering,* Volumen 48, pp. 409-425, 2005.

[Xing *et al.*, 2010]. Xing L., Chen Y., Wang P., Zhao Q., Xiong J., A knowledge-based ant colony opti-mization for flexible job shop scheduling problems, Applied Soft Computing 10 (3) (2010) 888–896, 2010.

[Yazdani *et al.*, 2010]. Yazdani M., Amiri M., Zandieh M. Flexible job-shop scheduling with parallel variable neighborhood search algorithm, Expert Systems with Applications 37 (1) (2010) 678–687, 2010.

[Yuan *et al.,* 2012]. Yuan Yuan, Xu Hua, Jiadong Yang. A hybrid harmony search algorithm for the flexible job scheduling problem. 2012. Applied Soft Computing, Elsevier. 2012.

[Zhang y Gen, 2005]. H. Zhang, M. Gen, Multistage-based genetic algorithm for flexible job-shop scheduling problem, Journal of Complexity International 11 (2005) 223–232. 2005.

[Zhang *et al.,* 2008].Guohui Zhang, Liang Gao, Xinyu Li, y Peigen Li. Variable Neighborhood Genetic Algorithm for the Flexible Job Shop Scheduling Problems. Springer-Verlag Berlin Heidelberg 2008.

[Zhang *et al.,* 2009]. Zhang G., Shao X., Li P., Gao L.,. An effective hybrid particle swarm optimization algorithm for multiobjetive flexible job shop scheduling problem. *Computers & Industrial Engineering,* pp. 1309-1318. 2009.

[Zhang *et al.*, 2011]. Zhang G., Gao L., Shi Y. An effective genetic algorithm for the flexible job shop scheduling problem, Expert Systems with Applications 38 (4) (2011) 3563–3573. 2011.

[Zhou et al., 2007]. Zhou H., Li Z., Wu X. Scheduling Unrelated Parallel Machine to Minimize Total Weighted Tardiness Using Ant Colony Optimization. Proceedings of the IEEE International Conference on Automation and Logistics. pp. 132-136, August 18 - 21, Jinan, China, 2007.

APÉNDICES

A. Estructuras de datos utilizadas para la representación simbólica en el algoritmo de búsqueda local iterada.

Para guardar los datos de las operaciones que representan cada uno de los nodos en el grafo disyuntivo, además de las dos operaciones ficticias que se aumentas (inicio y fin), fue necesario la utilización de estructuras de datos que permitieran el uso ordenado y eficiente de la información, debido a que una estructura es una colección de uno o más elementos que pueden ser de diferente tipo de datos. A continuación se explican cada una de las estructuras utilizadas en el algoritmo propuesto.

```
struct trabajo{       // Maneja restricciones de precedencia
       int noMaq;     // Número de Máquina
       int tCosto;    // Tiempo de procesamiento
       int tInicio;   // Tiempo de inicio
       int noOp;      // Número de operación
       int maquinas; // No. Máquinas que pueden ejecutar Oij
       int **maqXtiempo; // Máquinas y tiempo
       int max;       // RC - hacia delante
       int min;       // RC - hacia atrás
       int flag;      // RC - bandera
       int holgura;   // Holgura
};
```

La estructura *trabajo* permite almacenar toda la información de una operación en función del trabajo al que pertenezca, en esta estructura se guarda el conjunto de máquinas candidatas que pueden procesar esta

operación, así como, los datos obtenidos en los recorridos del método de la ruta crítica.

```
struct maquina{       // Maneja restricciones de capacidad
       int noJob;     // Número de trabajo
       int tCosto;    // Tiempo de procesamiento
       int tInicio;   // Tiempo de inicio
       int noOp;      // Número de operación
       int max;       // RC - hacia delante
       int min;       // RC - hacia atrás
       int flag;      // RC - bandera
};
```

La estructura *maquina* permite almacenar toda la información de una operación en función de la máquina en la que se ejecuta, en esta estructura se guarda el trabajo al que pertenece, el tiempo de ejecución y el tiempo de inicio de esta operación. También se guarda información referente a los recorridos del método de la ruta crítica.

B. Código fuente del algoritmo búsqueda local iterada.

A continuación se presenta el código fuente correspondiente a la función que realizar el algoritmo de búsqueda local iterada, para tratar el problema de calendarización de trabajos en talleres de manufactura flexible utilizando una estructura híbrida de vecindad.

Mientras el contador de la búsqueda local (BL_COUNT) sea menor que el tamaño de vecindad, se genera un nuevo vecino aplicando una perturbación mediante una estructura de vecindad y se evalúa, almacenando el valor en msLocal, sí este valor es menor que el valor del estado anterior (msActual), el nuevo vecino se acepta como el estado actual, en caso contrario, se ignora y aplica otra perturbación generando un nuevo vecino. Esto se repite hasta alcanzar el criterio de paro de la búsqueda local iterada, que es el número de repeticiones de la búsqueda local.

```
int main(int argc, char *argv[]){
    int N, M, noOperaciones, noBlocks, bloque, bloqueDos, vecindad, opInter;
    int msActual, msLocal, msGlobal = 9999999;
    int BLI_COUNT, BL_COUNT;    // Parámetros de control
    int CP_ILS = atoi(argv[1]); // Repeticiones búsqueda local
    int CP_LS = (M * (N - 1) * 2);    // Tamaño de la vecindad
    while(BLI_COUNT <= CP_ILS){   /*----- BLI -----*/
       asignaMaquina(&N, &M);
       solucionInicial(&N, &M, &noOperaciones);
       msActual = rutaCritica(&N, &M, &noOperaciones);
       noBlocks = generaBloques(&opRc);
       BL_COUNT = 1;
       while(BL_COUNT <= CP_LS){    /* --- BL --- */
```

```
        vecindad = generaAleatorio(1,4);
        bloque = generaAleatorio(1, noBlocks);
        opInter = generaAleatorio(1, opXbloque[noBlocks] - 1);
              switch(vecindad){
              case 1: funcionN1(&M, bloque, opInter);break;
              case  2:  funcionN5(&M,  &bloque,  &bloqueDos,
noBlocks);break;
               case 3: funcionN4(&M, bloque, opInter);break;
               case 4: funcionN6(&M, bloque, opInter);break;
              }
         msLocal = vueltaDelante(&N, &M, &noOperaciones, &ok);
              if(msLocal < msActual){
                  msActual = msLocal;
                  vueltaTras(&N, &M, &noOperaciones);
                  noBlocks = generaBloques(&opRc);
              }
              else rollBack(&M, bloque, bloqueDos);
              BL_COUNT++;
       }  /* Fin BL */
       if(msActual < msGlobal)msGlobal = msActual;
       BLI_COUNT++;
    } /* Fin BLI */
    muestraResultado(M, msGlobal);
}
```

C. Análisis de la complejidad del algoritmo

En el presente apéndice se muestra como se obtuvo la complejidad de las estructuras de vecindad paso a paso, así como la complejidad del algoritmo de búsqueda local iterada propuesto en este trabajo de investigación.

Función temporal de la estructura de vecindad N1

Estructura de vecindad N1		
Instrucción	*Costo*	*Repeticiones*
`op = bloques[idB][nop];`	C1	O(1)
`pos1 = bloques[idB][1];`	C2	O(1)
`maq = bloques[idB][bloques[idB][0] + 1];`	C3	O(1)
`for(i = 1; i < opsXmac[maq]; i++)`	C4	Σ i=1, 2, …, N
`  if(opM[maq][i].noOp == pos1)`	C5	Σ i=1, 2, …, N
`  pos1 = i;`	C6	O(1)
`pos = pos1 + (nop - 1);`	C7	O(1)
`if(opM[maq][pos].noJob != opM[maq][pos + 1].noJob)`	C8	O(1)
`  ok = 1;`	C9	O(1)
`else`		
`  if(bloques[idB][0] > 2){`	C10	O(1)
`  for(i = pos + 1; i < pos1 + (bloques[idB][0] - 1); i++)`	C11	Σ i=1, 2, …, N
`  if(opM[maq][i].noJob != opM[maq][i + 1].noJob){`	C12	Σ i=1, 2, …, N
`  ok = 1`	C13	O(1)
`  pos = i; }`	C14	O(1)
`  if(ok == 0 && pos1 != pos)`	C15	O(1)
`  for(i = pos; i > pos1; i--)`	C16	Σ i=1, 2, …, N
`  if(opM[maq][i].noJob != opM[maq][i - 1].noJob) {`	C17	Σ i=1, 2, …, N
`  ok = 1;`	C18	O(1)
`  pos = i - 1; }`	C19	O(1)
`  }`		
`aux = opM[maq][pos].noOp;`	C20	O(1)
`opM[maq][pos].noOp = opM[maq][pos + 1].noOp;`	C21	O(1)
`opM[maq][pos + 1].noOp = aux;`	C22	O(1)
`aux = opM[maq][pos].noJob;`	C23	O(1)
`opM[maq][pos].noJob = opM[maq][pos + 1].noJob;`	C24	O(1)

`opM[maq][pos + 1].noJob = aux;`	C25	O(1)
`aux = opM[maq][pos].tCosto;`	C26	O(1)
`opM[maq][pos].tCosto = opM[maq][pos + 1].tCosto;`	C27	O(1)
`opM[maq][pos + 1].tCosto = aux;`	C28	O(1)

Por inducción matemática sabemos que: $\sum_{i=1}^{n} i = \frac{n(n+1)}{2} = \frac{n^2+n}{2}$

Del análisis de la tabla anterior se sustituyen los valores en la fórmula de la inducción: T(n) = n(c) + 6 ($\frac{n^2+n}{2}$)

Desarrollando la inducción matemática: ($\frac{n^2+n}{2}$ (6c)) = $\frac{6\,n^2c\,+6\,nc}{2}$

Sustituyendo en la fórmula de la complejidad temporal:

T(n) = $3n^2c + 3nc + 21\,nc$ = 3n^2c + 24nc = **c (n + 8)**

La función temporal T(n) = n + 8 que corresponde a la estructura de vecindad N1, es de tipo polinomial y tiene una cota superior en la cual la constante está determinada por c. La premisa de la función temporal para el peor de los casos es:

$$O\big(g(n)\big) = \{f(n) \exists c, n_0, \forall n \geq n_0, 0 \leq f(n) \leq cg(n)\}$$

$$f(n) = n + 8 = O(n) si\ \exists c: \quad 0 \leq n + 8 \leq cn \quad n \geq n_0 = 4$$

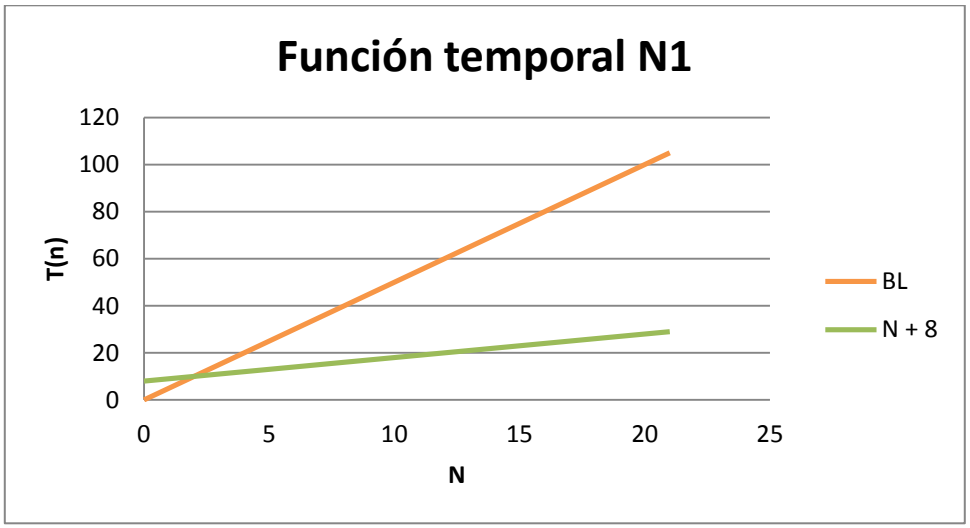

Figura C. 1 Gráfica de la función temporal de la estructura de vecindad N1

De acuerdo a la evaluación de la complejidad asintótica, para la estructura de vecindad N1, la complejidad es de *O*(*n*), con un valor para la constante *c* de 15, para un valor inicial de $n_0 = 4$.

Funciones temporales de las estructuras N4, N5 y N6.

De la misma manera en la que se calculó la función temporal de la estructura N1, se obtuvo las funciones temporales de las estructuras de vecindad N4, N5 y N6 que integran la estructura de vecindad híbrida propuesta en este trabajo de investigación. Así como para todas las funciones que implican la búsqueda local iterada, en la siguiente sección se muestra la función temporal de la estructura de vecindad híbrida aplicada a búsqueda local iterada.

Función temporal del algoritmo búsqueda local iterada.

La complejidad del algoritmo Búsqueda Local Iterada se calcula en una primera etapa, un análisis de las instrucciones requeridas por el algoritmo para encontrar una solución al problema tratado, de esta forma obtenemos la función de complejidad temporal del algoritmo.

$$T(n) = mn^3 + 4mn^2 + 7n + 39$$

Para el caso de la Búsqueda Local Iterada, *n* representa el número de trabajos a calendarizar y *m* la cantidad de máquinas donde se ejecutarán los trabajos. Por lo tanto la complejidad para el algoritmo de Búsqueda Local Iterada con una estructura sencilla para el peor de los casos tiene una complejidad de: $O(mn^3)$.

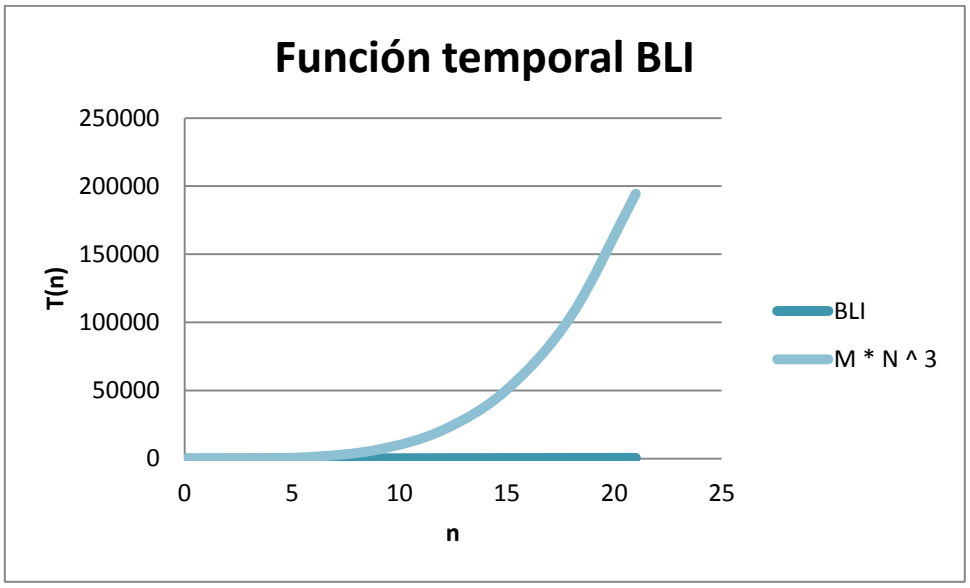

Figura C. 2 Función temporal búsqueda local iterada

La función temporal $T(n) = mn^3 + 4mn^2 + 7n + 39$ que corresponde al algoritmo de búsqueda local iterada propuesto en este trabajo de investigación, es de tipo exponencial, por lo que el tiempo requerido para obtener una solución en tiempo computacional razonable, es dependiente del número de trabajos a calendarizar en el sistema de manufactura

.

D. Glosario de términos

Algoritmo Determinístico: Es un algoritmo en el que una misma entrada obtiene siempre el mismo resultado.

Algoritmo no Determinístico: Algoritmo en el que a una misma entrada se obtienen diferentes salidas, de modo que no es posible saber de manera previa, el resultado que de la ejecución de un algoritmo de este tipo.

Análisis de Sensibilidad: Evaluación del comportamiento de las variables críticas de un problema con la finalidad de establecer un rango numérico, dentro del cual, la solución obtenida por el algoritmo sigue siendo buena, además de que permite conocer que tan sensible es el algoritmo a cambios en los valores de ciertas variables propias del problema.

Búsqueda Local: Técnica iterativa de que permite explorar el espacio de soluciones de un problema dado, a partir de una solución inicial, por medio de movimientos, de modo que vaya mejorando la solución obtenida de acuerdo a la función objetivo. Este tipo de técnicas son utilizadas para mejorar la calidad de las soluciones obtenidas por un algoritmo, así como para reducir el tiempo en que se obtienen dichas soluciones.

Búsqueda local iterada: La idea fundamental de la búsqueda local iterada, consiste en rastrear la solución a un problema combinatorio dentro del subespacio definido por los óptimos locales aplicando repetidas veces búsquedas locales.

Complejidad Espacial: Cantidad de memoria que un algoritmo consume o utiliza durante su ejecución.

Complejidad Temporal: Tiempo que necesita el algoritmo para ejecutarse.

Estructura de Vecindad: Tipo de movimiento (permutación, inserción o eliminación) utilizado para explorar el espacio de soluciones de un problema de optimización.

Heurística: Procedimiento intuitivo bien definido que proporciona buenas soluciones aproximadas a problemas difíciles de resolver sin garantizar la optimalidad, en un tiempo computacional razonable.

Instancia*:* Es un problema de prueba que puede ser definido como el tamaño de entrada de los datos del problema.

Tiempo polinomial: En las ciencias computacionales, el tiempo viene expresado generalmente en función del tamaño de la entrada. Se considera que un algoritmo puede ser resuelto en tiempo polinomial sí su función de complejidad es de orden $O(p(n))$, es decir, que puede ser representado por un polinomio.

Vecindad: Es el conjunto de soluciones que pueden ser alcanzadas desde una solución dada por medio de un movimiento (inserción, eliminación, permutación).