UNIVERSIDAD AUTÓNOMA DEL
ESTADO DE MORELOS

FACULTAD DE CIENCIAS QUÍMICAS E INGENIERÍA
CENTRO DE INVESTIGACIÓN EN INGENIERÍA
Y CIENCIAS APLICADAS

ALGORITMO DE ACEPTACIÓN POR UMBRAL CON APLICACIÓN DE BÚSQUEDA LOCAL HÍBRIDA PARA EL EMPAREJAMIENTO DE PESO MÁXIMO EN UN GRAFO

TESIS PROFESIONAL

PARA OBTENER EL GRADO DE:
MAESTRÍA EN INGENIERÍA Y CIENCIAS APLICADAS
OPCIÓN TERMINAL EN TECNOLOGÍA ELÉCTRICA

P R E S E N T A:

I.I YESSICA YAZMIN CALDERON SEGURA

ASESOR: DR. MARCO ANTONIO CRUZ CHÁVEZ
CO- ASESOR:DR.MARTIN HERIBERTO CRUA ROSALES

CUERNAVACA, MOR. JULIO 2011

Resumen

En este trabajo de investigación se desarrollaron cinco estructuras de vecindad para el problema de emparejamiento de peso máximo, donde cada una fue aplicada a una búsqueda local y a una búsqueda local iterada con la finalidad de mejorar el desempeño del problema tratado. Una vez que se determinó la mejor estructura de vecindad, se implementó el algoritmo de aceptación por umbral, en donde se llevó a cabo un análisis de sensibilidad para obtener la sintonización de todos los parámetros de control del algoritmo. Una vez sintonizado el algoritmo, se llevó a cabo el análisis de eficiencia y eficacia del algoritmo de aceptación por umbral con búsqueda local hibrida para el problema de emparejamiento de peso máximo.

El problema de emparejamiento de peso máximo es considerado como un problema de tipo NP, por lo que se tomó un modelo de programación lineal entera binaria ya existente en la literatura. En este trabajo se realizaron dos versiones del algoritmo de aceptación por umbral, la primera versión con una estructura de vecindad hibrida aleatoria y la segunda con una estructura de vecindad con un par aleatorio.
Las pruebas experimentales fueron ejecutadas en el clúster ciicap, el clúster nopal y el clúster upemor con problemas de prueba generados de forma aleatoria, los cuales fueron utilizados en ambas versiones del algoritmo.

El análisis experimental realizado demostró que la estructura de vecindad con un par aleatorio aplicada al algoritmo de aceptación por umbral es de mayor calidad que la estructura de vecindad hibrida tanto en eficiencia como en eficacia.

Abstract

In this research five neighborhood structures were developed for the problem of maximum weight matching.where each structure was applied to a local search and iterated local search,in order to improve the performance of the tackled problem. once the best neighborhood structure was identified the acceptance threshold algorithm was implemented,then a sensitivity analysis was applied to obtain the appropriate values of all the control parameters of the algorithm was tuned,there was conducted the efficiency and effectiveness of the threshold acceptance algorithm with local search hybridized to the problem of maximum weight matching.

Since the problem of maximum weight matching is considered as an NP problem,thus there was taked an existing integer linear programming model . In this research there were developed two versions of the threshold acceptance algorithm, the first version with a hybrid random neighborhood structure and the second one involved a random pair neighborhood structure.

The experimental tests were executed in ciicap cluster, the cluster and the cluster upemor cactus using testing problems randomly generated ,which were used in both versions of the algorithm.

Agradecimientos

A CONACYT por brindarme el apoyo económico sin el cuál no hubiera sido posible la realización de mis estudios de maestría.

A Doctor Marco por proponerme estudiar esta maestría y cambiar mi vida.

A los integrantes de mi comité tutoral y revisores de tesis: Dr. Marco Antonio Cruz Chávez, Dra. Margarita Tecpoyotl Torres,M.C. Jesús del Carmen Peralta Abarca,Dra. Ocotlan Diaz Parra, Dr. Martín Heriberto Cruz Rosales , por sus acertados consejos y comentarios para la realización de este trabajo de investigación.Ustedes son la base de crecimiento personal e intelectual y nunca los olvidare .

Dedicatoria

Dedico esta tesis a mis Padres

A Norma Nelli Segura Beltrán por ser la mejor madre del mundo por su apoyo incodicional, paciencia y amor. Una vez me dijiste si caes una vez levante, no llores yo estaré siempre para ti una no es ninguna dos forman una inténtalo de nuevo. Es verdad siempre has cumplido tu palabra. Gracia mami.

A Jose Calderón Vargas por ser el mejor padre por apoyarme con la pequeña Michelle su comprensión y amor.

A mi Esposo Edmundo Fuentes Zuniga por su apoyo, comprensión por su amor que nos une.

A mi pequeña Michelle Itzayana Fuentes Calderón que es la fuente de motivación para continuar, porque en mi vientre requerías de descanso y aun así continúe, por que recordar , tu risas tus enfermedades ,tus llantos y saber que me necesitas me motivas y aunque me duele dejarte sé que falta un camino más grande donde solo recordaras lo mejor que yo puedo brindarte. Mi vida tiene sentido gracias a ti, te amo Michelle.

A mi hermanito Erik Jose Calderon segura Solo puedo decir que pase lo que pase siempre te voy a querer.

Índice de figuras

Figura 1-1 Se muestra una clasificación de los principales métodos de optimización. 15
Figura 1-2Tiempo necesario para ejecutar un algoritmo, si cada paso se realiza en un 20
Figura 2-1Emparejamiento perfecto 28
Figura 2-2Emparejamiento no perfecto 29
Figura 2-3 Matriz Adyacente de Vértices 31
Figura 2-4 Matriz Adyacente de Costo 31
Figura 2-5 Matriz Adyacente de Pesos 32
Figura 2-6 Solución de la matriz costos 32
Figura 2-7Restricciones del problema de Emparejamiento 33
Figura 2-8Modelo matemático para el emparejamiento de peso máximo 34
Figura 2-9Función Objetivo 35
Figura 2-10Función Objetivo desarrollada 35
Figura 3-1Pseudocódigo del algoritmo de aceptación 38
Figura 3-2Grafica del parámetro de control Inicial. 39
Figura 3-3 Diagrama de flujo del problema de emparejamiento de peso máximo 42
Figura 3-4Grafica del parámetro de control final 43
Figura 3-5Algoritmo general de búsqueda local iterada 48
Figura 3-6 Representa el primer movimiento de con un par aleatorio. 49
Figura 3-7*Estructura de vecindad con un par aleatorio* 50
Figura 3-8Pseudocódigo de una estructura de datos con un par aleatorio. 50
Figura 3-9 Solución inicial para dos pares aleatorios 51
Figura 3-10Estructura de vecindad con dos pares aleatorios 51
Figura 3-11Pseudocódigo de la estructura de vecindad con dos pares aleatorios 52
Figura 3-12Solución inicial para tres pares aleatorios 53
Figura 3-13*Estructura de vecindad con tres pares aleatorios* 54
Figura 3-14 Pseudocódigo de la estructura de vecindad con tres pares aleatorios 54
Figura 3-15Solución inicial para tres pares aleatorios 55
Figura 3-16Estructura de vecindad con tres pares aleatorios 55
Figura 3-17Pseudocódigo de la estructura de vecindad con cuatro tres pares aleatorio 56
Figura 3-18Solución inicial para un par aleatorio 57
Figura 3-19Estructura de vecindad con un par aleatorio 57
Figura 3-20Estructura de vecindad con dos pares aleatorios 57
Figura 3-21Estructura de vecindad con tres par aleatorio 57
Figura 3-22Estructura de vecindad Hibrida 58

Figura 3-23Diagrama de flujo de la estructura de vecindad Hibrida 58
Figura 3-24Emparejamiento {1,3},{4,5} 60
Figura 3-25Emparejamiento {3,4},{2,5} con un 60
Figura 4-1Graficas de Pruebas de la Estructura de Vecindad con un Par Aleatorio 67
Figura 4-2Grafica de Pruebas de la Estructura de Vecindad con Dos Pares Aleatorios 68
Figura 4-3Graficas de Pruebas de la Estructura de Vecindad con Tres Pares Aleatorios 69
Figura 4-4Grafica de Pruebas de la Estructura de Vecindad con Cuatro Pares Aleatorios 70
Figura 4-5Pruebas de la Estructura de Vecindad Hibrida Aleatoria 71
Figura 4-61Variables del Algoritmo de Aceptación por Umbral 73

Índice de tablas

Tabla 1Tiempo necesario para ejecutar un algoritmo, si cada paso se realiza en un microsegundo. 18
Tabla 2Crecimiento de t(n) con 60 n^2 [Johnsonbaugh,1999]. 20
Figura 4-61Variables del Algoritmo de Aceptación por Umbral 73
Tabla 4-2Rangos del parámetro de control inicial utilizados para realizar el análisis de sensibilidad para el algoritmo de aceptación por umbral 74
Tabla 4-3Rangos del parámetro de control final utilizados para realizar el análisis de sensibilidad para el algoritmo de aceptación por umbral 75
Tabla 4-4Rangos de Delta utilizados para realizar el análisis de sensibilidad para el algoritmo de aceptación por umbral 76
Tabla 4-5Rangos de D utilizados para realizar el análisis de sensibilidad para el algoritmo de aceptación por umbral 76
Tabla 4-6Resultados para 100 Vértices. 30 ejecuciones con la búsqueda local iterada para cada estructura. 78
Tabla 9Funciones polinomiales 85

Contenido

Resumen *2*

Abstract *3*

Agradecimientos *4*

Dedicatoria *5*

Índice de figuras *6*

Índice de tablas *8*

Contenido *9*

Capítulo 1 Introducción *10*

1.1 Metodología de optimización **13**

1.2 Complejidad del Problema **16**

1.3 Objetivo de la Investigación **22**

1.4 Alcance de la investigación **23**

1.5 Contribución de la Tesis **24**

1.6 Organización de la tesis **25**

Capítulo 2 Emparejamiento de peso máximo *26*

2.1 Problema del Emparejamiento de peso máximo **27**

2.2 Descripción del Problema del Emparejamiento de Peso Máximo **29**

Capítulo 3 ..... *36*

Referencias ..... *¡Error! Marcador no definido.*

Capítulo 2 Apéndices ..... *¡Error! Marcador no definido.*

Capítulo 1 Introducción

Podemos encontrar gran cantidad de problemas tanto en la industria como en la ciencia .Desde los clásicos problemas de diseños de redes de telecomunicación u organización de la producción hasta los más actuales de ingeniería y re-ingeniería de software ,existe una infinidad de problemas teóricos y prácticos para aplicar al problema de emparejamiento de peso máximo.

Si consideramos a la industria está contiene una gran variedad de problemas que requieren solución, ya que estos llevan a las empresas al gasto excesivo y uso poco eficiente de los recursos. Considerando que la industria siempre requiere del aumento de producción, ventas y obtener la mayor maximización para la utilización de los recursos [trabajador-tela], [trabajador-material], [individuo-tarea]. Este tipo de problemas con los que tenemos contacto en la vida diaria, pueden ser representados por medio de modelos matemáticos que involucren una función objetivo para ser tratados por la Optimización Combinatoria.

Esta es una rama muy importante de las ciencias computacionales, dedicada a la investigación de operaciones así como al estudio y tratamiento de problemas considerados difíciles de resolver [Papadimitriou y Steiglitz, 1998];El objetivo es abordar el problema con el menor esfuerzo computacional posible y buscar la mejor solución mediante la métodos que lo permitan. Para representar un problema de optimización es necesario utilizar modelos matemáticos .Cada modelo cuenta con ciertas particularidades como es el caso de la función objetivo y las restricciones propias del problema. La función objetivo es el problema tratado, donde el modelo aplicado en este caso es maximizar el emparejamiento.

Todos los problemas pueden ser clasificados de acuerdo al grado de dificultad para resolverlos, de aquí surge la Teoría de la Complejidad, de la cual se encarga de dar una clasificación a estos problemas, dividiéndolos en P, NP y NP-Duros.

Para los problemas de carácter combinatorio (donde el dominio de soluciones es finito y se elige la mejor solución posible de éste), existen distintas formas de resolverlos, una de ellas es la búsqueda exhaustiva del conjunto de soluciones y así poder encontrar la óptima, es decir generar todas las soluciones factibles (que cumplan con todas las restricciones), calcular su costo respectivo asociado y de éstas elegir la mejor. Pero el tiempo de cálculo crece de manera exponencial de acuerdo al número de items del problema.

Podemos encontrar problemas en que se produce una explosión combinatoria (donde el tiempo de ejecución es no polinomial), de acuerdo al tamaño del problema, de los que solo se conocen algoritmos que encuentran una solución exacta en tiempos excesivamente largos.Cuando nos enfrentamos a un problema concreto, habrá una serie de algoritmos aplicables. Se suele decir que el orden de complejidad de un problema es el del mejor algoritmo que se conozca para resolverlo. Así se clasifican los problemas, y los estudios sobre algoritmos que se aplican a la realidad.

Estos estudios han llevado a la constatación de que existen problemas muy difíciles, problemas que desafían la utilización de los ordenadores para resolverlos.Los problemas computacionales los podemos dividir en dos conjuntos problemas tratables para los cuales existe un algoritmo de complejidad polinomial y los problemas intratables para los que no se conoce ningún algoritmo de complejidad polinomial(Apéndice A)

A los problemas tratables se les conoce también como problemas P (de orden polinomial). Asimismo a los problemas no tratables se les llama también NP (de orden no determinantico polinomial) y se dice que son

tratables en el sentido que suelen ser abordables en la práctica. Entre estos últimos podemos encontrar los siguientes ejemplos:
Agente Viajero.Caminos Hamiltonianos. Encontrar un camino en un grafo que pasa una sola vez por cada nodo. O(n!).Caminos Eulerianos. Encontrar un camino en un grafo que pasa una sola vez por cada arco.

Para mencionar algunos de los problemas de optimización, a continuación se presenta una clasificación, así como una breve explicación de cada uno de ellos.

Clase P

Los problemas P, son aquellos que pueden ser resueltos por una máquina de Turing determinista en tiempo polinomial, es decir, que la relación entre el tamaño del problema y su tiempo de ejecución es polinómica (Pinedo, 2008).

Clase NP

Los problemas NP,se consideran problemas intratables pueden caracterizarse por el curioso hecho de que puede aplicarse un algoritmo polinómico para comprobar si una posible solución es válida o no. Esta característica lleva a un método de resolución no determinista consistente en aplicar heurísticos (máquina de Turing) para obtener soluciones hipotéticas que se van desestimando (o aceptando) a ritmo polinómico. Los problemas de esta clase se denominan NP (la N de no-deterministicos y la P de polinómicos). Es evidente que $P \subseteq NP$.
Una de las características de estos problemas, es que el tamaño de su espacio de soluciones es de comportamiento exponencial conforme se incrementa el tamaño de la instancia(Pinedo, 2008). Algunos de los problemas NP más conocidos son: el problema del Agente Viajero, Máquinas en Paralelo no Relacionadas[Garey et al., 1976], Caminos Hamiltonianos, etc.

Clase NP-completos.

En 1971 Cook demostró que hay problemas en NP que son especialmente difíciles , son los denominados NP-completos. Se conoce una amplia variedad de problemas de tipo NP, de los cuales destacan algunos de ellos de extrema complejidad. Gráficamente podemos decir que algunos problemas se hallan en la "frontera externa" de la clase NP. Son problemas NP, y son los peores problemas posibles de clase NP. Estos problemas se caracterizan por ser todos "iguales" en el sentido de que si se descubriera una solución P para alguno de ellos, esta solución sería fácilmente aplicable a todos ellos.

Clase NP-duros.

Los problemas NP-Duros presentan las mismas características que los NP, pero se diferencian en que estos son aún más difíciles de tratar y sus instancias resueltas hoy en día son más pequeñas en comparación con las resueltas para problemas NP.

Cualquier problema de decisión, pertenezca o no a los problemas NP, el cual pueda ser transformado a un problema NPC (NP-completo) tendrá la propiedad que no podrá ser resuelto en tiempo polinomial a menos que P=NP. Podríamos entonces decir que dicho problema es al menos tan difícil como uno NP Completo.

1.1 Metodología de optimización

El método de optimización combinatorio aplicado al problema de emparejamiento de peso máximo consiste encontrar el mejor valor de la función objetivo ,de modo que se satisfagan las restricciones propias del problema tratado para una instancia dada. La literatura indica que para

obtener una buena solución a un problema de optimización clasificado como NP, es necesaria la implementación de un algoritmo no determinantico de tipo heurístico [Papadimitriou y Steiglitz, 1998], debido a que este tipo de algoritmos permiten obtener soluciones cercanas al óptimo para instancias medianas y grandes en un tiempo computacional razonable. Una heurística es un método bien estructurado, desarrollado en base a la experiencia que permite obtener soluciones aproximadas de un problema sin garantizar el optimo (Wetzel, 1983).

Este tipo de métodos se aplican cuando no es posible encontrar una solución óptima por un método exacto, debido a la naturaleza y complejidad del problema.

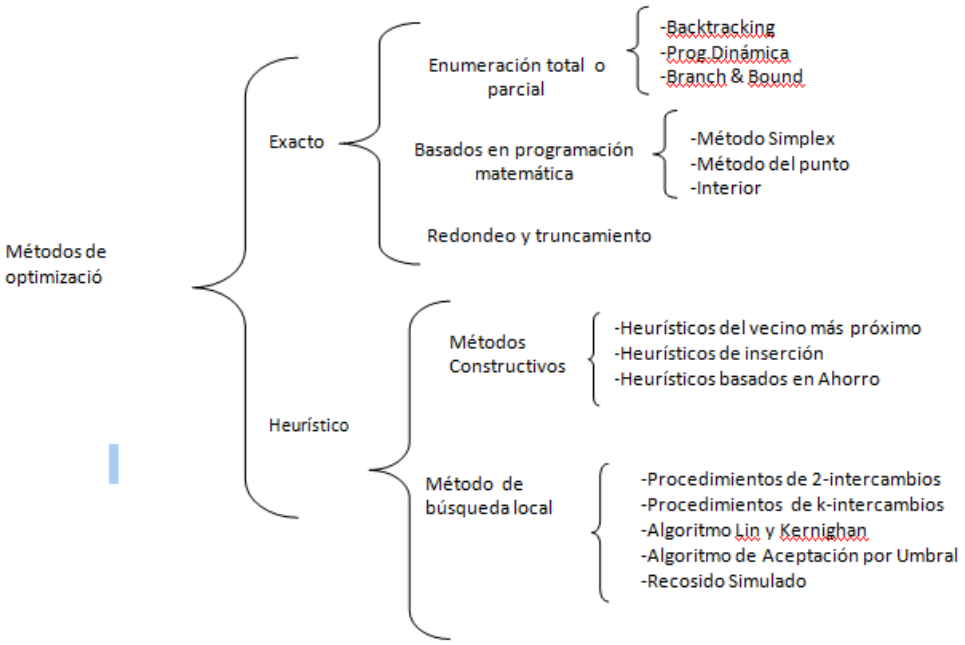


Figura 1-1 Se muestra una clasificación de los principales métodos de optimización.

Métodos de optimización

Una heurística es un método bien estructurado, desarrollado en base a la experiencia que permite obtener soluciones aproximadas de un problema sin garantizar el optimo (Wetzel, 1983). Este tipo de métodos se aplican cuando no es posible encontrar una solución óptima por un método exacto, debido a la naturaleza y complejidad del problema.

Métodos Constructivos

Métodos que consisten en ir construyendo paso a paso una solución a un problema dado. Normalmente son métodos deterministas y suelen estar basados en la elección de la mejor solución en cada iteración. Estos métodos han sido muy utilizados en problemas de optimización clásicos,

como es el caso del problema del Agente Viajero. Un ejemplo de este tipo de métodos son los algoritmos voraces (Michalewicz y Fogel, 2002).

Métodos de Búsqueda Local

A diferencia de los métodos mencionados anteriormente, los procedimientos de búsqueda local comienzan con una solución inicial y la van mejorando progresivamente. El procedimiento se lleva a cabo realizando en cada iteración, un movimiento de una solución a otra que mejore la solución anterior. Este movimiento puede ser una permutación, inserción o eliminación. El método finaliza cuando, para una solución, no existe ninguna solución accesible que la mejore. La diferencia con los métodos analíticos es que estos no necesariamente encontraran una solución óptima. Ejemplos de éste tipo de métodos son Búsqueda Tabú, Recocido Simulado, etc [Michalewicz y Fogel, 2002].

1.2 Complejidad del Problema

La complejidad del problema de emparejamiento de peso máximo se encuentra clasificada como un problema Np-Completo ya que este problema encuentra una nueva solución en cada iteración.

Para conocer la complejidad del problema de emparejamiento de peso máximo es necesario tomar en cuenta dos aspectos principales: el tiempo y los recursos necesarios para resolver un problema computacional (espacio). Estos parámetros son de vital importancia para definir si un algoritmo es eficiente o no; ya que si este requiere mucho tiempo para resolver un problema, no será de utilidad, lo mismo sucede si requiere gran cantidad de memoria.

La complejidad temporal de un algoritmo se encuentra representada por medio de una función temporal, la cuál es dependiente del tamaño de la entrada y del número de instrucciones que requieren ser evaluadas para resolver el problema.

Un claro ejemplo de lo explicado anteriormente, es que si un algoritmo presenta un tiempo de ejecución constante $T(n) = k$, se puede decir que este algoritmo es eficiente, debido a que si el tamaño de la entrada aumenta, el tiempo necesario para su ejecución permanecerá constante. En el caso de un algoritmo con complejidad logarítmica $T(n) = \log(n)$, también es considerado eficiente, debido a que, por ejemplo, si el tamaño de la entrada se hace 100 veces más grande, el tiempo requerido para su procesamiento únicamente se duplica; por el contrario, si un algoritmo presenta una complejidad de tipo exponencial.$T(n) = 2^n$, éste es considerado ineficiente (Bisbal, 2009). En la Tabla 1.3 se muestran los tipos de complejidad existentes.

Tabla 1 Tiempo necesario para ejecutar un algoritmo, si cada paso se realiza en un microsegundo.

Número de pasos hasta concluir el algoritmo para una entrada de tamaño N	Tipo de complejidad	Tiempo de ejecución de N= 3	6	9	12
k	Constante	10-6 seg	10-6 seg	10-6 seg	10-6 seg
log(n)	Logarítmica	2x10-6 seg	3x10-6 seg	3x10-6 seg	4x10-6 seg
n	Lineal	3x10-6 seg	6x10-6 seg	9x10-6 seg	10x10-5 seg
n log (n)	Cuasi-lineal	5x10-6 seg	2x10-6 seg	3x10-6 seg	10-5 seg
n2	cuadrática	9x10-6 seg	4x10-5 seg	8x10-6 seg	10x10-4 seg
nk	polinómica	3x10-5 seg	2x10-4 seg	7x10-6 seg	2x10-3 seg
2n	Exponencial	8x10-6 seg	6x10-5 seg	5x10-6 seg	4x10-3 seg

El tiempo de ejecución está definido por el número de iteraciones de un ciclo While. Con esta definición, los tiempos en el peor de los casos, en el mejor de los caso y el caso promedio para un algoritmo .El tamaño de entrada es de n son n-1 cada uno, pues el ciclo siempre se ejecuta n-1 veces.

Con frecuencia estamos menos interesados en los tiempos exactos en el peor o en el mejor de los casos para un algoritmo, que en la forma de incremento el tiempo en ambos casos, cuando el tamaño de la entrada se incrementa.

Por ejemplo, supongamos que el tiempo de un algoritmo en el peor de los casos es : $T(n)=60n^2+5n+1$

El estudio de eficiencia se lleva a cabo analizando el caso extremo de los algoritmos, es decir, el peor caso, el cual se encuentra definido como el valor en el que el algoritmo tendrá que realizar la mayor cantidad de operaciones.

En la Figura 1-3 se muestra el comportamiento de algunas de las funciones de complejidad con respecto al tiempo.

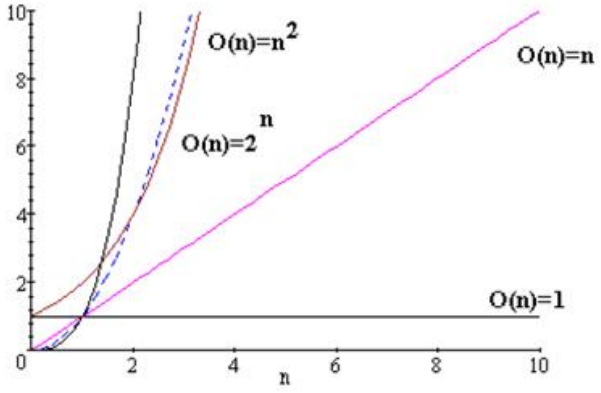


Figura 1-2Tiempo necesario para ejecutar un algoritmo, si cada paso se realiza en un microsegundo.

Para una entrada de tamaño n. Para n grande, el término $60n^2$ es aproximadamente igual a t(n).(véa la tabla 1-2) . En este sentido,t(n) crece como $60n^2$.

Tabla 2 Crecimiento de t(n) con 60 n^2 [Johnsonbaugh,1999].

N	$T(n)=60n^2+5n+1$	$60n^2$
10	6,051	6,000
100	600,501	600,000
1,000	60,005,001	60,000,000
10,000	6,000,050,001	6,000,000,000

Si (tabla 2) mide el tiempo, en segundos, en el peor de los casos para una entrada de tamaño n, entonces : $T(n)= n^2+(5n+1)/ 60$.

Mide el tiempo, en minutos en el peor de los casos para una entrada de tamaño n. Este cambio de unidades no afecta la forma en el que el tiempo en el peor de los casos crece cuando el tamaño de entrada se incrementa, sino sólo las unidades con las cuales se mide el tiempo en el

peor de los casos para una entrada de tamaño n. Así, al describir la forma en que el tiempo en el mejor de los casos o en el peor de los casos crece cuando el tamaño de la entrada crece, no sólo buscamos el término dominante [por ejemplo $60n^2$] en (figura 1-2) sino que podemos ignorar los coeficientes constantes .Bajo estas hipótesis, t(n) crece como n^2 en (figura 2) sino que podemos ignorar los coeficientes constantes constantes. Bajo estas hipótesis, t(n) crece como n^2 cuando n crece. Decimos que t(n) es de orden n^2 y escribimos.$T(n)=O(n^2)$.

Lo cual se lee " t(n) es theta de n^2".La idea básica consiste en reemplazar una expresión como t(n)= 60n2+5n+1 con una expresión más sencilla, cómo n^2,que crece como la misma razón que t(n).A continuación proporcionamos la definición formal[Johnsonbaugh,1999].

Definición figura 2

Sean f y g funciones con dominio

$$\{1,2,3\ldots.\}.$$

Escribimos

$$f(n)=O(g(n))$$

Y decimos que f(n) es de orden a lo más g(n) si existe una constante positiva C1 tal que

$$| f(n)|<= C1| g(n) |$$

Para todos los enteros positivos n,excepto para un número finito. Escribimos

$$f(n)= \Omega(g(n))$$

y decimos que f(n) es de orden

$$g(n) \text{ si } f(n)= O(g(n)) \text{ y } f(n)= \Omega(g(n)).$$

Teorema 1

Sea

$$a_k n^k + a_{k-1} n^{k-1} + \ldots\ldots a_k n + a_0$$

un polinomio en n de grado k, donde ak no es negativo.

Entonces

$$a_k n^k + a_{k-1} n^{k-1} + \ldots\ldots a_k n + a_0 = \Theta(n^k).$$

Demostración. Sea

$$C = a_{k-1} n^{k-1} + \ldots\ldots a_1 + a_0$$

Entonces

$$a_k n^k + a_{k-1} n^{k-1} + \ldots\ldots a_k n + a_0$$
$$\leq a_k n^k + a_{k-1} n^{k-1} + \ldots\ldots a_k n + a_0$$
$$= (= a_{k-1} + a_{k-1} + \ldots\ldots + a_1 + a_0) n^k = c n^k.$$

La importancia de los algoritmos en cuanto a la complejidad, se basa en diseñar algoritmos que cuenten con una complejidad logarítmica, debido a que independientemente del equipo en el que sea ejecutado, el tiempo de cómputo será razonable.

1.3 Objetivo de la Investigación

El objetivo principal de este trabajo fué la utilización de metaherísticas para resolver el problema de emparejamiento de peso máximo ,aplicando dos estructuras de vecindad a una búsqueda local con la finalidad de mejorar la eficiencia y eficacia del algoritmo. Las estructuras de vecindad parten de una solución factible que permitan mejorar considerablemente las soluciones iniciales obtenidas.

1. Desarrollar pruebas de cada estructura de vecindad propuestas, comparar la calidad de las soluciones para determinar cual de las cinco estructuras ejecutadas era conveniente aplicar al algoritmo

de aceptación por umbral. Las pruebas se llevaron a cabo aplicando 100 vértices por instancia de 30 iteraciones para poder realizar una comparación directa y confiable.

2. Desarrollo un análisis de sensibilidad para el algoritmo de aceptación por umbral. Las pruebas se llevaron a cabo aplicando 100 vértices por instancia de 30 iteraciones para poder realizar una comparación directa y confiable.

3. Se Implementaron las dos estructuras de vecindad al TA y se analizaron los resultados obtenidos de mayor calidad para determinar cual de las dos estructuras de vecindad es conveniente aplicar al algoritmo de aceptación por umbral.

1.4 Alcance de la investigación

1. El problema de emparejamiento de peso máximo propuesto cuenta con tres partes fundamentales, la primera es la implementación de dos estructuras de vecindad para observar la explotación de espacio de soluciones y la segunda parte es la aplicación del algoritmo de aceptación por umbral ,la tercer parte es establecer los parámetros de control del algoritmo de aceptación por umbral que mejoran el tiempo y la calidad del algoritmo.

2. Se desarrollo una comparación de los resultados obtenidos del algoritmo de aceptación por umbral con una estructura de vecindad con un par aleatorio y el TA con una estructura de vecindad hibrida aleatoria, tomando en cuenta que son evaluados con las mismas instancias.

3. Las instancias de pruebas son generadas de forma aleatoria y se generaron tanto para las estructuras de vecindad como para el

algoritmo de aceptación por umbral, de esta forma los algoritmos propuestos demostraron su eficiencia y eficacia.

4. Para el problema de emparejamiento de peso máximo se propuso un modelo matemático de programación lineal entera en donde no se toma en cuenta las interrupciones ni el tiempo de ejecución.

1.5 Contribución de la Tesis

1 En este trabajo se desarrolló un algoritmo de aceptación por umbral con aplicación a una búsqueda local hibrida aleatoria para el problema de emparejamiento de peso máximo.

1 Una de las contribuciones presentadas en este trabajo de investigación es el modelo de grafos no dirigidos aplicado al problema de emparejamiento de peso máximo con una matriz de [N][N] , se desarrolló el algoritmo para emparejamientos perfectos he imperfectos ,además del desarrollo e implementación de las cinco estructuras de vecindad aplicadas a la búsqueda local .

2 Una de las contribuciones de este algoritmo que es parte fundamental del algoritmo de aceptación por umbral es el análisis de sensibilidad de los parámetros de control, esta metodología para la realización del análisis de sensibilidad, lleva a cabo una sintonización de las variables de entrada del algoritmo TA. Se observa en las pruebas ejecutadas que la aplicación de una estructura de vecindad con un par aleatorio es de mayor calidad que la estructura de vecindad hibrida aleatoria propuesta.

3 Se propone como contribución hacer la comparación entre estas dos estructuras de datos pero aplicadas al algoritmo de aceptación por umbral.

1.6 Organización de la tesis

En el capítulo 1 se da una introducción del problema a tratar ,Se da una breve explicación dela metodología aplicada, la complejidad del problema, objetivos ,los alcances, las contribuciones así como la organización de la tesis. En el capítulo 2 se da una introducción al problema de emparejamiento de peso máximo ,se describe el problema y se detalla el problema tratado mediante un grafo no dirigido, la representación de este grafo en programación lineal y la formulación matemática del mismo.

En el capítulo 3 se da una explicación del algoritmo de aceptación por umbral, se explica ampliamente el pseudocódigo utilizado, se describen las variables utilizadas ,se explica ampliamente las estructuras de vecindad propuestas ,la metodología de sintonización de parámetros de control y el análisis de complejidad del algoritmo de aceptación por umbral.

En el capítulo 4 se explica el proceso realizado para la sintonización de parámetros, se explica la aplicación de la estructura de vecindad propuesta aplicada a la búsqueda local para el algoritmo de aceptación por umbral ,además de presentan los resultados en cuanto a la eficiencia y eficacia .

El capítulo 5 se dan las conclusiones finales del trabajo desarrollado, así como
trabajos futuros.

Capítulo 2
Emparejamiento de peso máximo

En este capítulo se da una explicación del problema de Emparejamiento de peso máximo, se presenta la formulación matemática por medio de un modelo de programación entera binaria, así como su representación para el problema de emparejamiento y su modelado por medio de un grafo no dirigido , también se dan a conocer los tipos de emparejamiento utilizados en este trabajo de tesis .

Definimos a continuación los principales conceptos de Teoría que utilizaremos en este trabajo. Básicamente las notaciones utilizadas son las que se encuentran en los libros de Reinaldo Guiudici E,Angeles Bris Llusch(1997), (Alfredo Caicedo Barrero et al 2010) , Gary Chartrand (1977), J.C.Fernando,V.Gregori (2002) y Claude Berge(2001).

Un grafo simple es un par G = (V, E),donde V es un conjunto finito de vértices y/o nodos y E es un conjunto de conjuntos de dos vértices definidos como aristas tambien se denomina arcos .Dos vertices v1 ,v2 € V son ayacentes o vecinos si { v1 ,v2}€ E. Si a={v,x}, decimos que la arista a incide en el vértice V. Gráficamente, los vértices se representan por puntos y las aristas por líneas que los unen.
Un vértice puede tener 0 o más aristas, pero toda arista debe unir exactamente dos vértices.
El orden de un grafo es el número de vértices que lo compone |V|.

Un grafo en programación lineal está compuesto por una estructura de datos .Cada elemento del grafo mantiene una relación con la matriz declara de N*M.

Para encontrar el emparejamiento de peso máximo se utilizaron varias estructuras adyacentes, dónde básicamente es un arreglo de NxN,donde N es la cantidad de vértices en un grafo y cada casilla de la matriz es llenada con falsos o verdaderos ,según exista la arista que conecte los dos vértices involucrados en un grafo no dirigido.

2.1 Problema del Emparejamiento de peso máximo

La literatura indica que cualquier problema de la vida real puede ser representado fácilmente utilizando una estructura de datos que represente un grafo. Las estructuras son simples y eficaces en la resolución de problemas ya que los nodos de este pueden representar muchas cosas como: servios,personas,trabajos,maquinas,moléculas,material,servicios.etc.En esta tesis se toma el emparejamiento de peso máximo de forma teórica y su representación en mediante un grafo no dirigido.

Un emparejamiento (en inglés, significa matching) en un grafo no dirigido y simple G es un conjunto de aristas, tal que dos aristas cualquiera de M no tengan un extremo común (Thanh Minh Hoang 2010). Dado un emparejamiento M en G, los vértices incidentes con alguna arista de M se denominan vértices saturados por M o M-saturados. El resto de los vértices que no cumplen dicha condición se llaman vértices no saturados o insaturados por M, aunque el nombre más común que se les asigna es el de vértice libre.

Un emparejamiento máximo en un grafo G es cualquier emparejamiento de G de orden máximo y que a satura todos sus vértices, es decir, con el mayor número de vértices posibles conectados.

Sea M un emparejamiento, se denomina arista emparejada respecto de “M” a cada una de las aristas de G que están en “M”. Si dicha arista que no estén “M” se dice que no está emparejadas.

Se denomina como pareja de un vértice (v1) al vértice (v2) del extremo opuesto en la arista perteneciente al emparejamiento M que tiene como extremo al primero de los vértices, otra forma de decirlo es que el vértice v1 está emparejado con el vértice v2.

La figura 2-1 muestra el emparejamiento máximo en un grafo no dirigido.

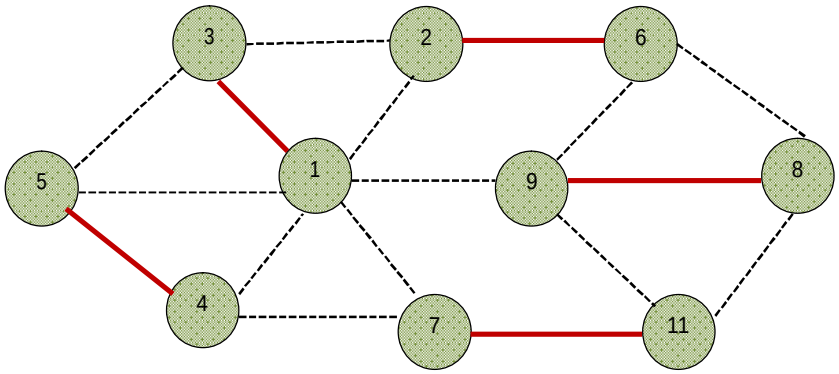


Figura 2-1Emparejamiento perfecto

Un emparejamiento máxima de peso óptimo en un grafo G es un emparejamiento de G de orden máximo donde la suma de las ponderaciones de las aristas es máxima o mínima según corresponda la función objetivo planteada, en este caso nuestra función objetivo es maximizar esta se detallara más adelante.

Se llaman vértice emparejado con respecto a “M” a cada uno de los vértices incidentes con alguna arista de “M”, en otro caso se trata de un vértice no emparejado también llamado emparejamiento imperfecto.

El algoritmo desarrollado puede dar emparejamientos imperfectos, o perfectos por que las soluciones son encontradas de forma son de forma aleatoria.La figura 2-2 Muestra el emparejamiento imperfecto.

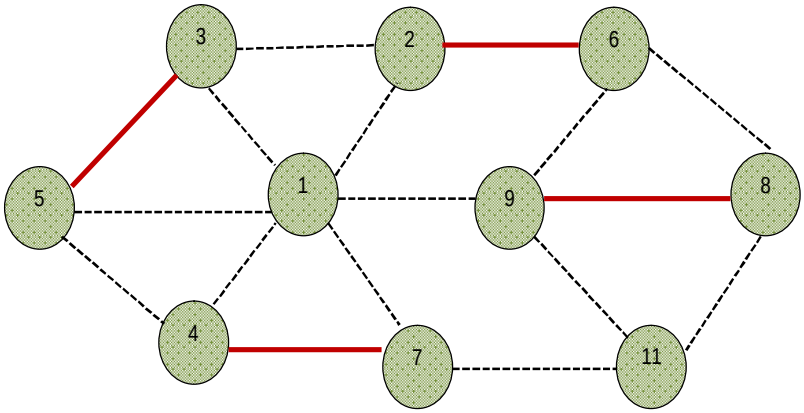


Figura 2-2Emparejamiento no perfecto

2.2 Descripción del Problema del Emparejamiento de Peso Máximo

El presente trabajo de investigación trata al problema de emparejamiento de peso máximo en un grafo no dirigido; En la actualidad

muchos problemas de la vida real han sido esquematizados a través de grafos, para poder hallarles una solución a través de la optimización, dado a sus diversas aplicaciones en la vida real, este puede ser adaptado a problemas que lo requieran, un ejemplo es aplicado en el área laboral .Que en diversos estudios sobre del comportamiento de las relaciones entre la educación superior y el empleo constituye siempre una cuestión de máximo interés en los egresados (Cristina Borra Marcos..etc,2006)

Podemos encontrar una gran cantidad de problemas de en donde podemos aplicar el emparejamiento de peso máximo, tanto en la industria como en la ciencia. Desde los clásicos problemas de diseño de redes de telecomunicación u organización de la producción hasta los más actuales en ingeniería y re-ingeniería de software, existe una infinidad de problemas teóricos y prácticos en donde el problema de emparejamiento de peso máximo puede ser aplicado de forma teórica.

El problema de emparejamiento de peso máximo, es un modelo de tipo NP-completo (L. Valiant, 1979) que puede ser formulado por medio de programación Lineal Entera Binaria, lo cual permite tratar al problema por medio de método no determinístico, mejor conocido como heurístico (R. Kulkarni…etc.,2009) . Este modelo de programación Lineal Entera Binaria utilizo una matriz incidente para almacenar las relaciones entre vértices y aristas en una matriz de *n*x*m*, en donde *n* representa el número de vértices del grafo y *m* el número de aristas (Sánchez Enriquez et al,2006) .

En el siguiente ejemplo mostramos la matriz de incidencia para el grafo en la figura 2-6.

Dado el Grafo=(V,E) es representado por la matriz adyacente 2-3 , en donde V={ 0, 1, 2, 3, 4,5,6,7,8,9,10} y E={ e0=(1,1), e0=(1,2), e0=(1,3), e0=(1,4), e0=(1,5), ,e0=(1,6), e1=(2,1), e1=(2,2), e1=(2,3), e2=(3,1),

e2=(3,2), e2=(3,3),e3(4,1), e3(4,2), e3(4,3), e4(5,1), e4(5,2), e5(6,1), e5 (6,2), e6(7,1), e6(7,2), e6(7,3), e6(7,4), e7(8,1), e7(8,2) , e7(8,3) , e8(9,1), e8(9,2), e8(9,3), e8(9,4), e9(10,1), e9(10,2), e9(10,3)};

Entonces decimos que :

e0={1,2},{1,3},{1,4},{1,7},{1,9},e1{2,1},{2,3},{2,6},e2{3,2}, e2{3,1}, {3,5},e3{4,1},e3{4,5},e3{4,7},e4{5,3},e4{5,4}, e5(6,2), e5(6,8), e6(7,1) , e6(7,4) , e6(7,10) , e6(7,9), , e7(8,9), e7(8,10), e7(8,6), e8(9,8) , e8(9,1) , e8(9,7) , e8(9,10), e9(10,9), e9(10,8), e9(10,7):

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

Figura 2-3 Matriz Adyacente de Vértices

	1	2	3	4	5	6	7	8	9	10
1	2	23	1	7	54	-1	0	0	0	0
2	11	32	56	-1	0	0	0	0	0	0
3	66	67	58	-1	0	0	0	0	0	0
4	16	87	77	-1	0	0	0	0	0	0
5	85	66	-1	0	0	0	0	0	0	0
6	79	88	-1	0	0	0	0	0	0	0
7	64	47	10	9	-1	0	0	0	0	0
8	91	10	63	-1	0	0	0	0	0	0
9	8	18	71	10	-1	0	0	0	0	0
10	9	82	78	-1	0	0	0	0	0	0

Figura 2-4 Matriz Adyacente de Costo

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

Figura 2-5 Matriz Adyacente de Pesos

La matriz adyacente de pesos mostrada en la figura 2-5 indica las conexiones realizadas para formar un emparejamiento perfecto mostrado en la figura 2-6 . El diseño del programa toma el contenido de cada fila y este es el que se conecta en cada iteración del programa el modelo matemático mostrado en la figura 2-8 permite cumplir las restricciones de la figura 2-7 .

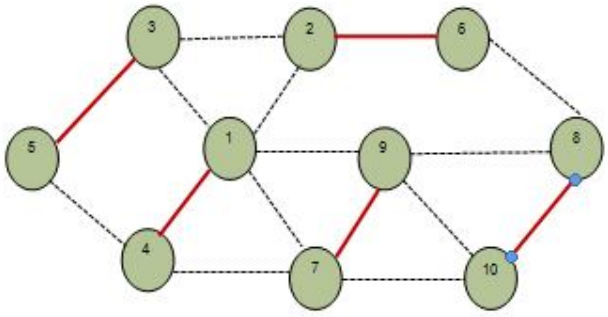


Figura 2-6 Solución de la matriz costos

Las restricciones básicas del problema se listan a continuación:

- Sólo podemos unir dos vértices entre sí.
- Se toma el emparejamiento con mayor ponderación o peso.
- Si existe emparejamiento se representa con 1 en caso contrario 0

Figura 2-7Restricciones del problema de Emparejamiento

Hallar un emparejamiento de peso máximo en un grafo puede resolverse en tiempo polinomial en la cantidad de nodos del grafo. Pero, por otro lado si aplicamos el problema de emparejamiento de peso máximo en el área laboral, Sea un grafo G = (V, E); donde V define los vértices o trabajos a desarrollar, V = {1, …, n} y E representa las aristas o eficiencia de cada trabajo desarrollado el vértice i a uno j, Se supone que cada individuo puede realizar algunas de las tareas. Se pretende asignar a cada individuo una única tarea de modo que se realicen el mayor número posible de estas. Si se supone que cada individuo sólo va a realizar una tarea, las demandas son todas 1 y las disponibilidades 1. Los costos de cada celda se definirán con 1 ó 0. Un 1 significa que el individuo puede hacer esa tarea y un 0 que no puede hacerla[15]. por lo que E = {(i, j) : i, j € V}, y sea c_{ij} el costo asociado al vértice (i, j) sea x_{ij} el emparejamiento que solo contiene una sola arista, de acuerdo a esto se muestra la formulación matemática para el emparejamiento máximo de un grafo que se presenta en la figura 2-8 .

La figura 2-8 muestra el modelo matemático utilizado para la solución inicial del problema de emparejamiento de peso máximo.

$$Max\ \mathrm{f} = \sum_{\mathrm{i=1}}^{\mathrm{2n\text{-}1}} \sum_{\mathrm{j=i+1}}^{\mathrm{2n}} \mathrm{c_{ij}} x_{ij}$$

$$s.a.$$

$$(1) \quad \sum_{\mathrm{k=1}}^{\mathrm{i\text{-}1}} x_{ki} + \sum_{j=i+1}^{2n} x_{ij} = 1, i = 1..2n$$

$$(2) \quad i < j$$

$$(3) \quad x_{ij} \in \{0,1\}$$

Figura 2-8Modelo matemático para el emparejamiento de peso máximo

De acuerdo al modelo matemático presentado en la figura 2-8 , tenemos que las restricciones representadas por (1), (2) y (3) que si se representara un problema de la vida real, se especificaría que cada trabajador debe desarrollar un trabajo sólo una vez de todos los demás trabajos asignados , con la finalidad de cumplir la función objetivo que se presenta en la ecuación (3) y maximizar el costo total de la eficiencia de cada trabajo.

La función objetivo es el emparejamiento de peso máximo de cada trabajador asi que la función objetivo indica la sumatoria del costo del emparejamiento de peso máximo donde N debe ser un número par para que la formula de emparejamiento de peso máximo sea correcta.

El programa desarrollado respeta las tres restricciones Indicadas en la figura 2-9 y cumple con el objetivo principal de esta tesis que es encontrar el máximo emparejamiento en un grafo no dirigido.

$$Max\ \mathrm{f} = \sum_{\mathrm{i=1}}^{\mathrm{2n\text{-}1}} \sum_{\mathrm{j=i+1}}^{\mathrm{2n}} \mathrm{c_{ij}} x_{ij}$$

Figura 2-9Función Objetivo

La primera restricción de la figura 2-9 indica la sumatoria independiente entre las conexiones y el emparejamiento de peso máximo.

$$Max\ \mathrm{f} \sum_{\mathrm{i=1}}^{\mathrm{2n\text{-}1}} \sum_{\mathrm{j=i+1}}^{\mathrm{2n}} \mathrm{c}_{\mathrm{ij}} x_{ij} = \sum_{\mathrm{i=1}}^{3} \sum_{\mathrm{j=1+1}}^{4} \mathrm{c}_{12} x_{12} + \sum_{\mathrm{i=1}}^{3} \sum_{\mathrm{j=2+1}}^{4} \mathrm{c}_{13} x_{13} + \sum_{\mathrm{i=1}}^{3} \sum_{\mathrm{j=3+1}}^{4} \mathrm{c}_{14} x_{14}$$
$$\sum_{\mathrm{i=2}}^{3} \sum_{\mathrm{j=2+1}}^{4} \mathrm{c}_{23} x_{23} + \sum_{\mathrm{i=2}}^{3} \sum_{\mathrm{j=3+1}}^{4} \mathrm{c}_{24} x_{24} + \sum_{\mathrm{i=3}}^{3} \sum_{\mathrm{j=3+1}}^{4} \mathrm{c}_{34} x_{34}$$

Figura 2-10Función Objetivo desarrollada

Si se desarrolla esta función objetivo mostrada en la figura 2-10 en un programa lineal i y j son la variables utilizadas para obtener el costo y los emparejamiento en cada iteración.

La segunda restricción de la figura 2-8 indica el emparejamiento entre i <j .

La tercera restricción de la figura 2 -8 indica que cuando exista un emparejamiento de peso máximo se marca con un 1 la matriz adyacente utilizada ;Esta es similar a la mostrada en la figura 2-5 y en caso contrario el emparejamiento no existe y se marca como falsa representándolo con 0.

Capítulo 3 Algoritmo de Aceptación por Umbral

3.1 Introducción

En este capítulo se presenta el algoritmo de aceptación por umbral aplicado a la búsqueda local hibrida , además el algoritmo de aceptación por umbral, así como el desarrollo y la implementación de dos estructuras de vecindad utilizadas. Se incluye una amplia descripción del procedimiento utilizado para el desarrollo de estas estructuras de vecindad ,se presenta una explicación de la primera solución para la manipulación del algoritmo de aceptación por umbral ,se muestran las variables utilizadas, las instancias de prueba manejadas, la sintonización de la metodología utilizada, y finalmente se anexa la función temporal del algoritmo y el análisis de complejidad.

Para llevar a cabo la selección de la estructura de vecindad se desarrollo la sintonización que conforma la estructura de vecindad hibrida, se llevó a cabo una búsqueda en diversas publicaciones que abordan el problema tratado en (Capítulo 2), para, de esta forma, identificar aquellas estructuras que cuentan con menor complejidad computacional y que han obtenido buenos resultados en su aplicación.

Como primer paso se desarrolló el problema de emparejamiento de peso máximo para mejorar la solución obtenida por dicho algoritmo, se llevó a cabo la implementación de una estructura de vecindad híbrida a búsqueda local. En esta sección se explica ampliamente el procedimiento realizado para el desarrollo de dicha estructura.

3.2 Algoritmo de Aceptación por Umbral

El algoritmo de aceptación por umbral (TA por sus siglas en inglés threshold accepting) está considerado como una técnica heurística de propósito general que permite solucionar problemas de optimización combinatoria el cual fue propuesto independientemente por (Dueck y Scheuer 1990) y por (Moscato y Fontanari 1992). Este es una variante de recocido simulado (SA por sus siglas en inglés) y la principal diferencia entre estos métodos consiste en el criterio de aceptación de soluciones peores en el valor de la energía. El algoritmo de aceptación por umbral propone aceptar un movimiento siempre y cuando rebase un cierto Umbral determinado con respecto a cada iteración.

La implementación del algoritmo de aceptación por umbral requiere de algunas variables similares a las que el recocido simulado , se debe resaltar que las variables utilizadas para el equilibrio térmico es el factor de disminución de la temperatura y las condiciones de paro del algoritmo.

En la figura 3-0 se muestra el pseudocódigo del algoritmo de aceptación por umbral. Más adelante se detallan las variables y se describe el algoritmo utilizado.

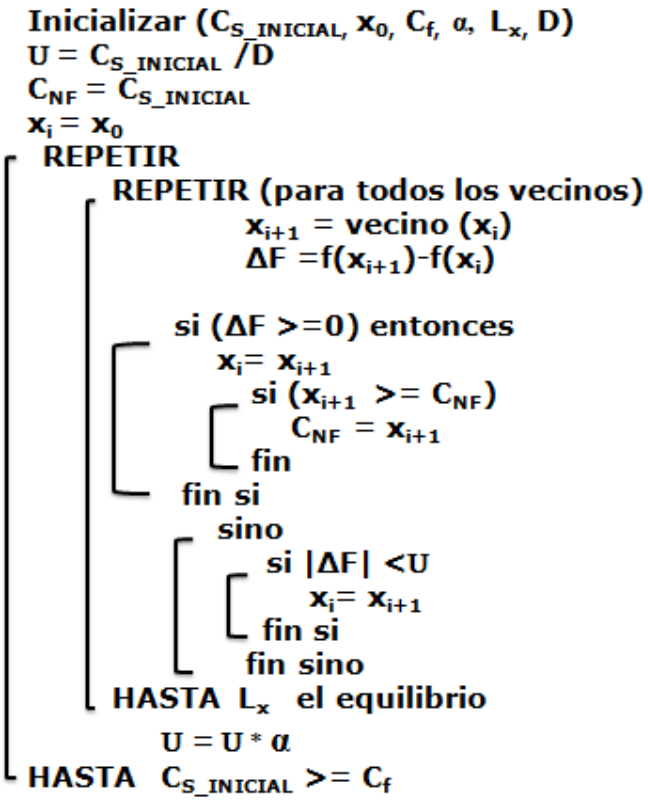


Figura 3-1Pseudocódigo del algoritmo de aceptación

Parámetro de Control Inicial $C_{s\_INICIAL}$

Tanto en Recocido Simulado como en Aceptación por umbral, requieren de una variable que indique la temperatura esta se inicializa alta para que el algoritmo gaste mucho tiempo y permita cualquier solución entrante .Si se inicializa con una temperatura baja, el sistema no alcanzará a explorar de manera suficiente el espacio de soluciones y posiblemente se estancará en un mínimo local.

Para este algoritmo la temperatura se tomó el promedio, debido a que los resultados graficados muestran los óptimos más cercanos y contienen una degradación estratégica de la solución en el decremento.

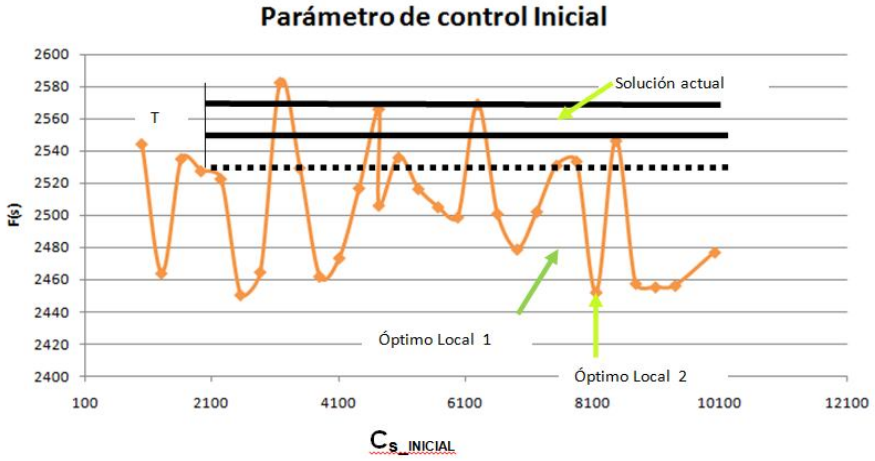


Figura 3-2Grafica del parámetro de control Inicial.

Solución inicial X_o

Para comenzar con la ejecución del algoritmo de aceptación por umbral se debe contar con una solución factible inicial del problema a resolver, que en este caso es el problema del emparejamiento de peso máximo.

Una configuración inicial es un conjunto de variables enteras del problema a tratar, en este caso es el problema de emparejamiento de peso máximo donde se hace uso de tres matrices adyacentes y del contenido de estas matrices ,Es tomado de forma aleatoria para la solución de problemas de emparejamiento perfecto o imperfectos .

Las tres matrices adyacentes están configuradas para contener los vértices, el peso y las conexiones realizadas de forma aleatoria para el emparejamiento de peso máximo sus dimensiones son de [N][N].
Estas están descritas en el capítulo 2.

A continuación se muestra el pseudocódigo que se utilizo para encontrar los emparejamientos perfectos he imperfectos de nuestra solución inicial.

Desarrollo de algoritmo de la solución inicial en pseudocódigo

Para desarrollar este algoritmo se analizó diversos artículos sobre el problema tratado, en donde de forma aleatoria se hace el emparejamiento de los dos vértices con su respectivo peso y/o ponderación para formar el emparejamiento máximo en un grafo.

Desarrollo de algoritmo de la solución inicial en pseudocódigo

Para desarrollar este algoritmo se analizó diversos artículos sobre el problema tratado, en donde de forma aleatoria se hace el emparejamiento de los dos vértices con su respectivo peso y/o ponderación para formar el emparejamiento máximo en un grafo.

A continuación presentamos el algoritmo que resuelve el problema del emparejamiento máximo:

Paso 1: inicializar el grafo no dirigido con los vértices candidatos con todas las aristas y pesos correspondientes.

Paso 2: sea v1 el vértice generado aleatoriamente que exista en el grafo no dirigido.

Paso 3: Se verifican los grados del vértice v1 .

Paso 4: Se toma en cuenta el grado del vértice V1 para generar de forma aleatoria la conexión que tendrá el vértice v2 con el vértice v1.

Paso 5: Agregar a la solución. La arista(v1 , v2)

Paso 6: si los vértice son factibles ir al paso 5, si no rechazar dicho vértice y retornar al paso 2.

Paso 7: sacar todos del vértice v2 candidatos propuesto con todas las aristas incidentes con dichos vértices (v1, v2).

Paso 8: retornar al paso 2, si aun hay aristas para valuar en los candidatos; si no lo hay, ir al paso 9.

Paso 9: retornar el conjunto solución, y fin del algoritmo.

En este algoritmo, se hace mención de dos conjuntos cuando para obtener la solución: el conjunto de vértices que contendrán el emparejamiento de peso máximo siempre y cuando un par de vértices sean factibles se añade la solución y el conjunto de candidatos rechazados para hacer el emparejamiento son eliminados con-1, y al final retorna la solución inicial que contiene el emparejamiento de peso máximo del grafo no dirigido.

La figura 3-2 muestra el diagrama de flujo utilizado para el desarrollo del emparejamiento para la primera solución del problema tratado.

Diagrama de flujo para el problema del emparejamiento peso máximo

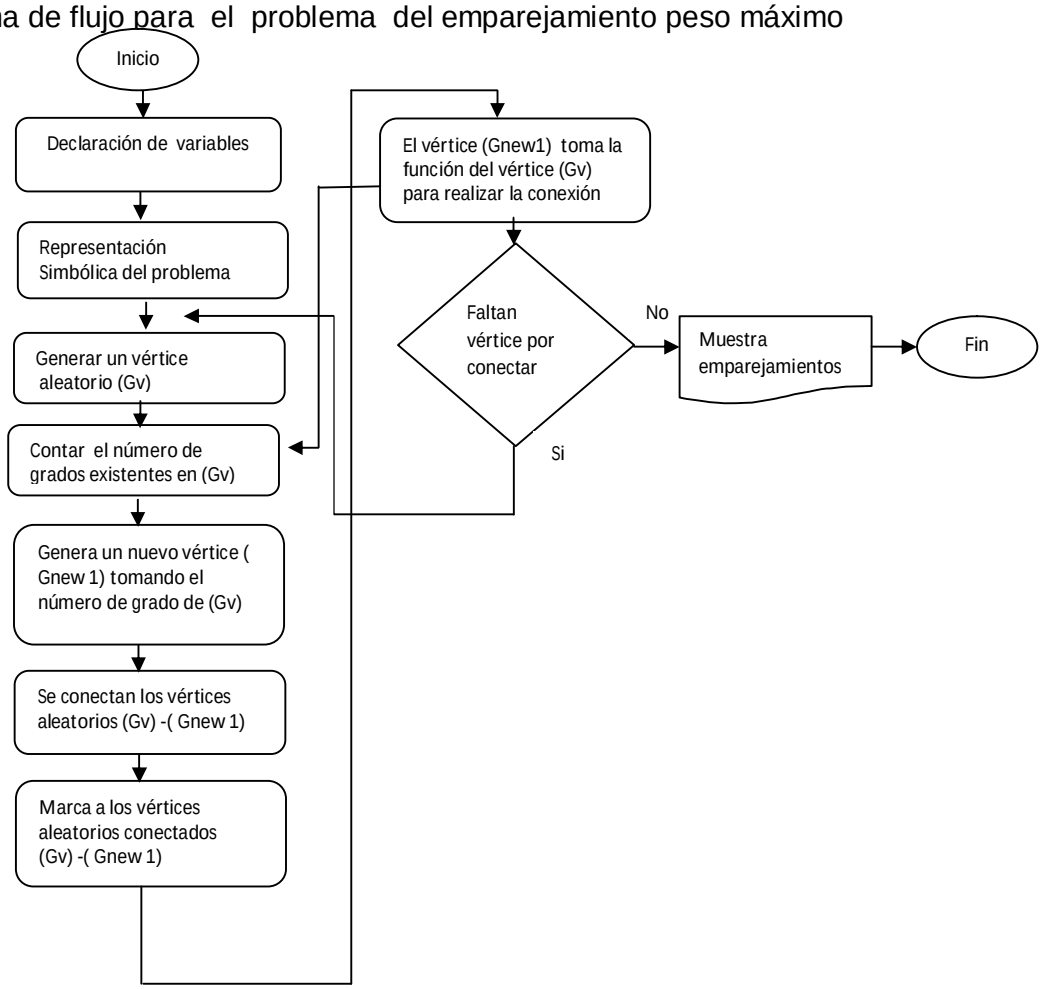


Figura 3-3 Diagrama de flujo del problema de emparejamiento de peso máximo

Parámetro de control final C_f

El parámetro de control final se está considerando como el parámetro que determina el equilibrio del algoritmo donde se estableció el número máximo de iteraciones para el parámetro de control inicial.

La figura 3-3 muestra los valores sintonizados del algoritmo de aceptación por umbral que se utilizaron para determinar el parámetro de control final.

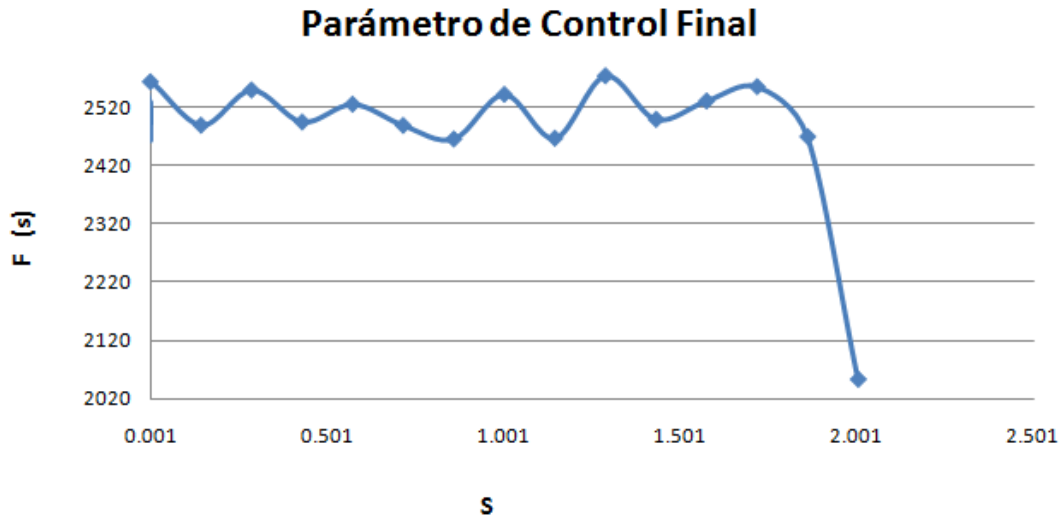


Figura 3-4Grafica del parámetro de control final

Variables esenciales para determinar el Umbral son "α ,D "

La aceptación por umbral se implementa simpre y cuando no se cumpla la condición de que el valor absoluto de la diferencia en costo del vecino con la solución actual sea mayor o igual a cero $\Delta F=f(xi+1) - f(xi) >= 0$.
El umbral comienza a aceptar soluciones malas para poder escapar del óptimo local.
El umbral se encuentra definido por una variable U que es dividida por D y en cada iteración desarrolla el siguiente proceso $U=U *\alpha$

Disminución de la temperatura.

El cálculo de la temperatura final se definió mediante una contante .Se desarrollaron 810000 pruebas para definir el la variable de temperatura final. Este parámetro definirá la disminuye la temperatura cada vez que se logre el equilibrio. La literatura propone algunos esquemas estos son:

- Cuando se alcanza un número determinado de iteraciones (L. Morales, R.et al,1992),(H. Telley. et al,1992),(Bruce S.et al.1992).
- Cuando la temperatura actual toma un valor constante especificado (J. Pérez.et al.2000),(Jose R.Medina et a ,2006).
- Cuando el porcentaje de aceptados rebasa un límite . (Jacques A. Ferland,1996),(S. Kirkpatrick et al ,1983),(N. Bouhmala, P. A. et al1996), (Cruz-Chávez M. A, Rivera-López R.,2007).

El procedimiento del TA es el siguiente:

El algoritmo comienza con la mejor solución factible $x_0 \in N$ y $N \subset X$ (donde X es el conjunto de todas las soluciones factibles) y un parámetro de control $C_{s\_INICIAL}$ (llamado temperatura debido a la analogía con la mecánica estadística) inicializado con un valor adecuadamente alto. Se genera una solución vecina $y \in S(x)$ donde $S(x) \subset X$ definida por el programador. (en esta tesis se trabajo con una estructura de vecindad con un par aleatorio y la estructura de vecindad hibrida aleatoria) ;Si se cumple la condición donde el valor absoluto es igual a la diferencia en

costo del vecino con la solución actual y esta es mayor o igual a cero **∆F=f(xi+1) - f(xi) >= 0** se dice que la solución encontrada es factible entonces es aceptada como la nueva solución actual en caso contario la solución actual permanece sin cambio. El valor de la temperatura $C_{s\_INICIAL}$ se decrementa cada vez que se alcanza el equilibrio térmico. De otra manera, la probabilidad de que la nueva configuración sea aceptada es ∆F<=UMBRAL la cual es comparada con un número intervalo y si la condición se cumple es aceptada como nueva solución factible .Este proceso se repite hasta que se alcanza una condición de equilibrio. para que al final sólo se acepten las configuraciones que mejoren la solución. Todas las soluciones o valores cambian cada vez que se forme un conjunto de S soluciones factibles ,a diferencia de (Romero,1990) donde se usa un conjunto de soluciones aceptadas. El valor de $C_{s\_INICIAL}$ se decrementa multiplicándolo por un factor de enfriamiento en este caso llamado Delta hasta que el sistema alcance el congelamiento.

3.3 Estructura de vecindad Hibrida

En esta sección, se presenta la idea básica de la búsqueda por vecindad, la aplicación de la búsqueda local, la búsqueda local iterada y el desarrollo de las estructuras de vecindad.

Se utiliza la búsqueda por vecindad porque es una técnica utiliza en los problemas de optimización para mejorar una solución inicial en donde el tamaño del espacio de soluciones permite obtener el óptimo local (Jianhong-Yang et al.2009) La parte fundamental de una vecindad radica en el tamaño y la estructura (Cruz et al., 2010b). Mientras mayor sea el tamaño de la vecindad, mejor será la calidad de las soluciones localmente óptimas, así como la precisión de la solución final encontrada.El tamaño de vecindad es el número promedio de

movimientos, intercambios, inserciones o eliminaciones entre los elementos de una solución s, un movimiento es un parámetro importante para determinar el tiempo computacional. Este movimiento sobre una solución actual, puede conducir a una solución factible o que no mejore la solución actual.

Para aplicar la búsqueda local a un problema particular, sólo se necesita para especificar la estructura de vecindad y una solución factible inicial (R. Jensen and Q. Shen , 2003).La búsqueda por vecindad trata de encontrar la mejor solución evaluando la función objetivo del problema de emparejamiento para encontrar su valor máximo. Para determinar la estructura de una vecindad, se debe definir un vecindario y el criterio de selección de un vecino. Es decir, si se cumple el criterio de selección, 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 hemos llegado a un óptimo local. El tipo de movimiento a realizar para seleccionar un vecino se define en la estructura de la vecindad propuesta .De lo contrario, la solución inicial se mantiene como el óptimo local con respecto a la zona de la estructura . De modo que la estructura de vecindad es la función de vecindad o tipo de movimiento que aplicado a la búsqueda local, permite mejorar la explotación del espacio de soluciones (Stattenberger et al…1)y[Gu,Huang,1994].Para mejorar una solución, es necesario realizar una búsqueda local, además de moverse dentro de una vecindad desde una solución inicial factible hacia una solución vecina, la cuál debe ser evaluada de acuerdo al criterio establecido por la función objetivo, misma que generalmente involucra costos (a maximizar o minimizar), con la finalidad de encontrar una solución que mejore el valor de la solución anterior.

La literatura indica que una heurística con una vecindad más grande será más eficaz, por otro lado el tiempo requerido para llevar a cabo una iteración dentro de la vecindad se incrementará con respecto a la

tamaño de la vecindad. De acuerdo a esto, se debe buscar un tamaño adecuado para la vecindad permita generar el mejor desempeño posible de la función de vecindad.

Para probar las estructuras de vecindad propuestas se hiso uso de la búsqueda local iterada esta es también una heurística que propone una estrategia exploratoria de mejora de las soluciones obtenidas por una determinada heurística mediante la iteración. Un esquema en el se incluye una heurística base que mejora los resultados mediante la repetición de dicha heurística. Esta idea ha sido propuesta en la literatura con distintas denominaciones, como descenso iterado, grandes pasos con cadenas de Markov, Lin-Kerningan iterado, búsqueda perturbada o ruidosa o la búsqueda de entorno variable con agitación donde la solución aportada por una heurística de búsqueda por entornos es agitada para producir una solución de partida para la heurística de búsqueda.

La búsqueda local iterada se empieza con una solución inicial s , y a partir de esta, se genera un proceso de búsqueda local dentro de un subespacio definido por los óptimos relativos(Lourenco et al.,2001).Un algoritmo de búsqueda transforma una solución cualquiera s en otra s' que es el óptimo local. Una vez que el proceso de solución encuentra un óptimo local en la que se estanca este óptimo se perturba hacia una solución diferente que no incluya las soluciones de la última búsqueda local este procedimiento se desarrollo mediante las estructuras de vecindad. La nueva solución, después de haber sido perturbada o hecho un movimiento, sirve de punto de partida para generar un segundo grupo de soluciones con un nuevo óptimo local s' ,a la cual se vuelve a aplicar el algoritmo de búsqueda local iterada para alcanzar el nuevo optimo local s* esto se repite hasta alcanzar el criterio de paro de la búsqueda local iterada.

En la figura 3-5 Se muestra el algoritmo general de búsqueda local iterada.

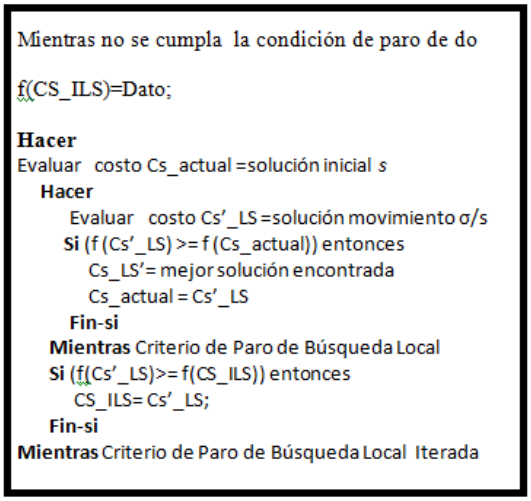


Figura 3-5Algoritmo general de búsqueda local iterada.

Las pruebas experimentales se realizaron para observar el comportamiento del el problema de emparejamiento máximo en cada estructura de vecindad. A continuación se mencionan los movimientos realizados para cada estructura de vecindad.

Estructura de Vecindad Par Aleatorio

En esta sección, se presenta la estructura de vecindad con un par aleatorio y se da una breve explicación del procedimiento a seguir.Para la estructura de búsqueda por vecindad con un par aleatorio se genera una solución s

Inicial factible a partir de la cual se elige un número aleatorio Gv (Generar vértice aleatorio), mismo que corresponde a una posición de una matriz en donde se encuentra almacenada la solución inicial.

Una vez que se tiene el vértice Gv, se verifica que este corresponda a la solución inicial (figura 3-5) se toma en cuenta el número de arcos del vértice elegido y se elige uno nuevo vértice V1 de los posibles vértices a conectar, se examina si este es factible si este es factible se realiza el movimiento y se obtiene la nueva solución vecina s´ en caso contrario se mantiene la solución si modificaciones .

Solución *S*

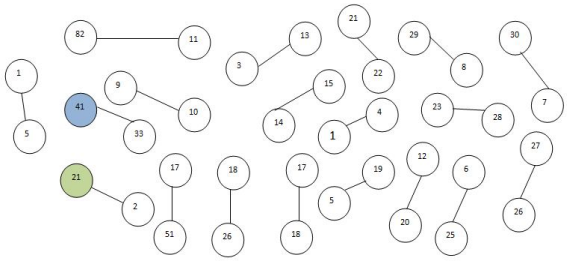


Figura 3-6 Representa el primer movimiento de con un par aleatorio.

Solución *S'*

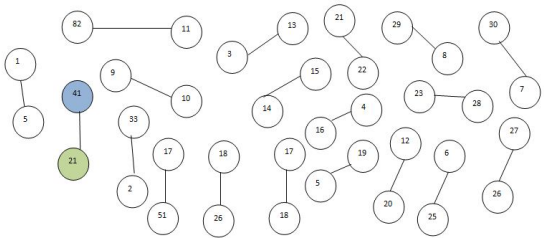
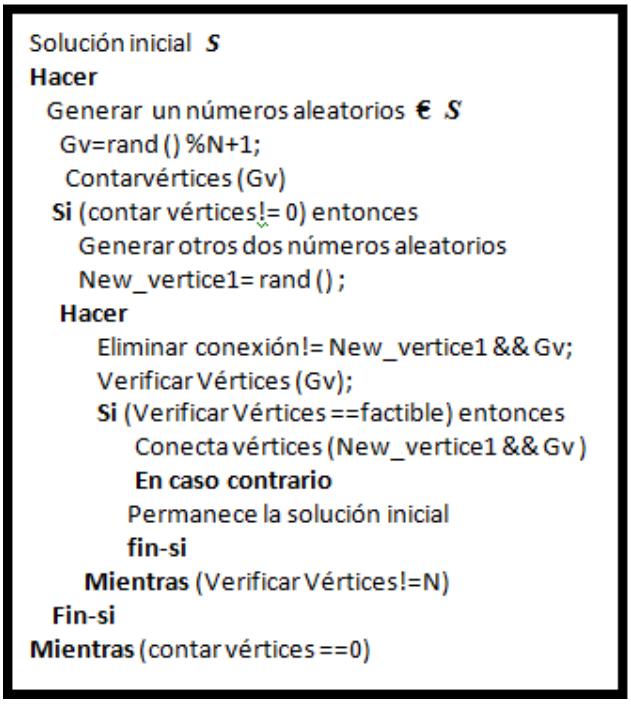


Figura 3-7*Estructura de vecindad con un par aleatorio*



```
Solución inicial S
Hacer
  Generar un números aleatorios € S
   Gv=rand () %N+1;
    Contarvértices (Gv)
  Si (contar vértices!= 0) entonces
     Generar otros dos números aleatorios
     New_vertice1= rand () ;
   Hacer
       Eliminar conexión!= New_vertice1 && Gv;
       Verificar Vértices (Gv);
       Si (Verificar Vértices ==factible) entonces
            Conecta vértices (New_vertice1 && Gv )
            En caso contrario
           Permanece la solución inicial
           fin-si
      Mientras (Verificar Vértices!=N)
  Fin-si
Mientras (contar vértices ==0)
```

Figura 3-8Pseudocódigo de una estructura de datos con un par aleatorio.

Estructura de vecindad con dos pares aleatorios

La estructura de vecindad con dos pares aleatorios, se desarrolla mediante dos vértices que sean generados de forma aleatoria, que cumplan con la condición de generar emparejamiento perfectos o imperfectos dependiendo de la ejecución del programa .El primer vértice generado de forma aleatoria no debe ser igual al segundo vértice aleatorio .En la figura 3-8 se muestra un ejemplo de la posible solución

inicial para la estructura de vecindad con dos pares aleatorios y el movimiento desarrollado en la estructura de vecindad que son los vértices marcados de color verde y azul . Debido a que los vértices elegidos cumplen con la condición se hace el movimiento mostrado en la figura 3-9 Y la figura 3-10 muestra el pseudocódigo utilizado para implementar esta estructura de vecindad con dos pares aleatorios.

Solución s

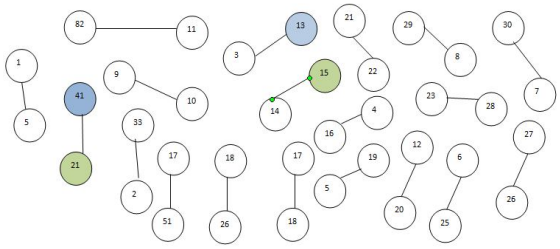


Figura 3-9 Solución inicial para dos pares aleatorios

Solución s'

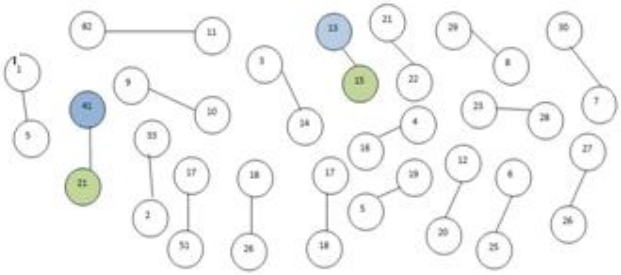
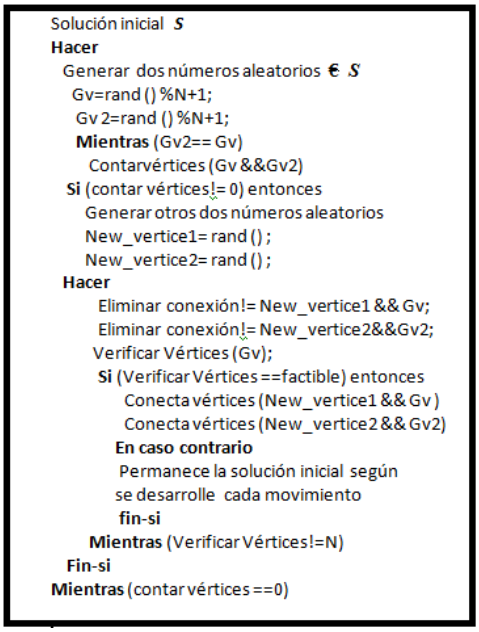


Figura 3-10Estructura de vecindad con dos pares aleatorios



```
Solución inicial  S
Hacer
  Generar  dos números aleatorios € S
    Gv=rand () %N+1;
    Gv 2=rand () %N+1;
    Mientras (Gv2== Gv)
      Contarvértices (Gv &&Gv2)
  Si (contar vértices!= 0) entonces
     Generar otros dos números aleatorios
     New_vertice1= rand () ;
     New_vertice2= rand () ;
  Hacer
        Eliminar conexión!= New_vertice1 && Gv;
        Eliminar conexión!= New_vertice2&&Gv2;
       Verificar Vértices (Gv);
        Si (Verificar Vértices ==factible) entonces
            Conecta vértices (New_vertice1 && Gv )
            Conecta vértices (New_vertice2 && Gv2)
          En caso contrario
           Permanece la solución inicial según
          se desarrolle  cada movimiento
           fin-si
      Mientras (Verificar Vértices!=N)
  Fin-si
Mientras (contar vértices ==0)
```

Figura 3-11Pseudocódigo de la estructura de vecindad con dos pares aleatorios

Estructura de vecindad con tres pares aleatorios

La estructura de vecindad con tres pares aleatorios , es necesario el generar tres números aleatorios que cumplan con la condición de generar emparejamiento perfectos o imperfectos dependiendo de la ejecución del programa la figura 3.12 representa la primera solución para llevar a cabo el movimiento con tres pares aleatorios ,y en el código de este programa se agrega la siguiente condición : el tercer vértice generado no debe ser igual al segundo vértice y el primer vértice debe ser distinto al segundo y tercero y se cumple la condición se desarrolla el movimiento en caso contrario no se desarrolla ningún movimiento.

En la figura 3-13 se muestra el movimiento desarrollado en la estructura de vecindad con tres pares aleatorios.

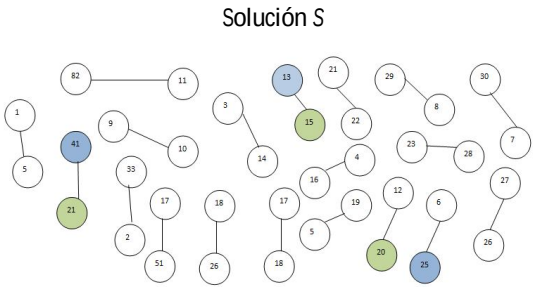


Figura 3-12Solución inicial para tres pares aleatorios

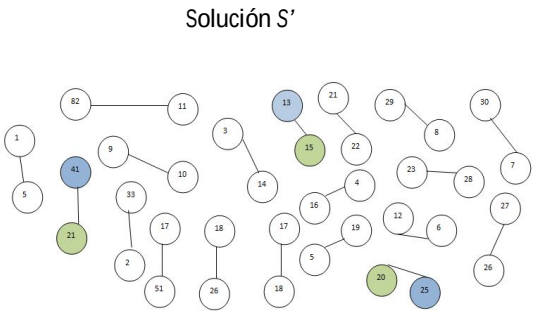


Figura 3-13*Estructura de vecindad con tres pares aleatorios*

```
Solución inicial S
Hacer
  Generar tres números aleatorios ϵ S
    Gv=rand () %N+1;
     Gv 2=rand () %N+1;
     Gv 3=rand () %N+1 ;
  Mientras (Gv3== Gv| | Gv3== Gv2)
      Contarvértices (Gv &&Gv 1&& Gv3)
  Si (contar vértices!= 0) entonces
      Generar otros dos números aleatorios
      New_vertice1= rand () ;
      New_vertice2= rand () ;
      New_vertice3= rand () ;
    Hacer
        Eliminar conexión!= New_vertice1 && Gv;
        Eliminar conexión!= New_vertice2&&Gv2;
        Eliminar conexión!= New_vertice3&&Gv3;
       Verificar Vértices (Gv);
        Si (Verificar Vértices ==factible) entonces
            Conecta vértices (New_vertice1 && Gv )
            Conecta vértices (New_vertice2 && Gv2)
            Conecta vértices (New_vertice3&& Gv3)
          En caso contrario
           Permanece la solución inicial según
           se desarrolle  cada movimiento
           fin-si
      Mientras (Verificar Vértices!=N)
  Fin-si
Mientras (contar vértices ==0)
```

Figura 3-14 Pseudocódigo de la estructura de vecindad con tres pares aleatorios

Estructura de vecindad con cuatro pares aleatorios

La estructura de vecindad con cuatro pares aleatorios , se generan cuatro vértices aleatorios que cumplan el mismo procedimiento mostrados anteriormente ,y se coloca la siguiente condición al código :el cuarto vértice aleatorio debe ser distinto al tercer vértice generado y no debe ser igual al segundo vértice ni al primer vértice. Los cuatro deben ser distinto y si se cumple la condición se desarrolla el movimiento de la nueva estructura de vecindad en caso contrario no se desarrolla ningún movimiento. La figura 3-15 muestra la solución inicial que indica los movimientos de los nodos.

En la figura 3-16 se muestra el movimiento desarrollado en la estructura de vecindad con cuatro pares aleatorios.

Solución s

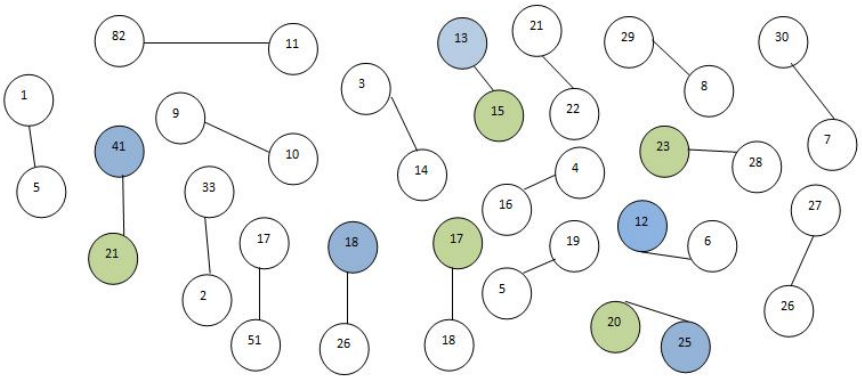


Figura 3-15Solución inicial para tres pares aleatorios

Solución s'

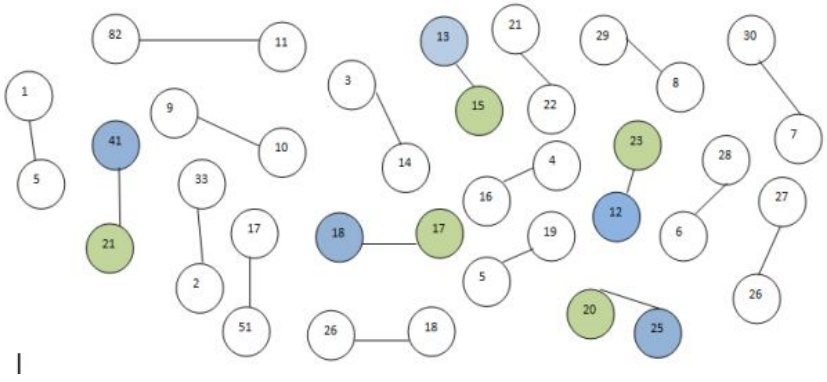


Figura 3-16Estructura de vecindad con tres pares aleatorios

En la figura 3-17 muestra el pseudocódigo utilizada para desarrollar la estructura de vecindad con tres pares aleatorios.

```
Solución inicial S
Hacer
  Generar cuatro  números aleatorios € S
    Gv=rand () %N+1;
     Gv 2=rand () %N+1;
     Gv 3=rand () %N+1;
     Gv 4=rand () %N+1;

  Mientras (Gv4== Gv|| Gv4== Gv2 || Gv4== Gv3)
      Contarvértices (Gv &&Gv 1&& Gv3&& Gv4)
  Si (contar vértices!= 0) entonces
     Generar otros dos números aleatorios
     New_vertice1= rand () ;
     New_vertice2= rand () ;
     New_vertice3= rand () ;
     New_vertice4= rand () ;
   Hacer
        Eliminar  conexión!= New_vertice1 && Gv;
        Eliminar  conexión!= New_vertice2&&Gv2;
        Eliminar  conexión!= New_vertice3&&Gv3;
        Eliminar  conexión!= New_vertice4&&Gv4;
       Verificar Vértices (Gv);
       Si (Verificar Vértices==factible) entonces
            Conecta vértices (New_vertice1 && Gv )
            Conecta vértices (New_vertice2 && Gv2)
            Conecta vértices (New_vertice3&& Gv3)
           Conecta vértices (New_vertice 4&& Gv4)
         En caso contrario
           Permanece la solución inicial según
          se desarrolle  cada movimiento
           fin-si
     Mientras (Verificar Vértices!=N)
   Fin-si
Mientras (contar vértices ==0)
```

Figura 3-17Pseudocódigo de la estructura de vecindad con cuatro tres pares aleatorio

Estructura de vecindad hibrida aleatoria

Esta estructura parte de una solución inicial factible que de forma aleatoria elige que tipo de estructura de vecindad que va a ejecutar de los procedimientos mostrados anteriormente .Esta estructura de vecindad hibrida cuenta con un menú de 4 opciones aleatorias, cada opción contiene una estructura lista para ejecutarse .Dependiendo del número aleatorio generado será la estructura que lleve a cabo el movimiento para obtener la nueva solución vecina. La figura 3-22 muestra el funcionamiento de forma general de la estructura hibrida.

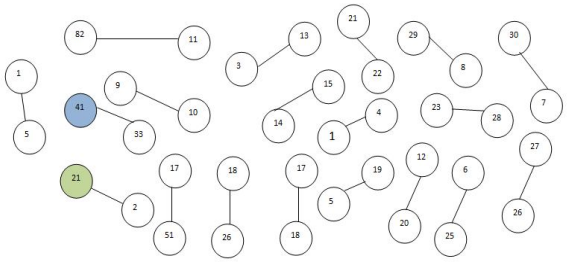


Figura 3-18Solución inicial para un par aleatorio

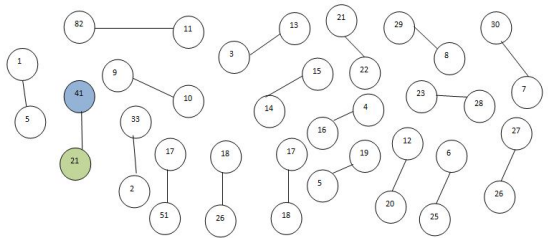


Figura 3-19Estructura de vecindad con un par aleatorio

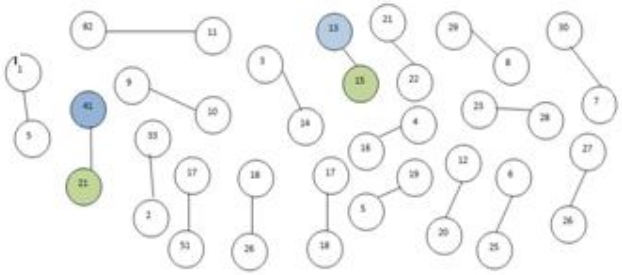


Figura 3-20Estructura de vecindad con dos pares aleatorios

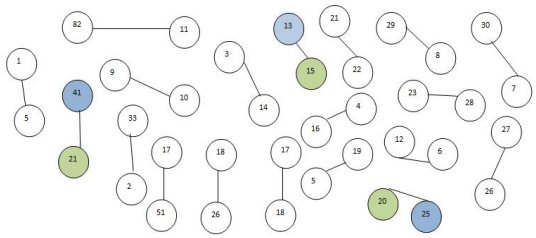


Figura 3-21Estructura de vecindad con tres par aleatorio

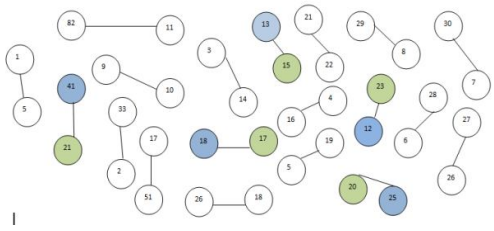


Figura 3-22Estructura de vecindad Hibrida

La figura 3-23 Diagrama de flujo de la estructura de vecindad hibrida

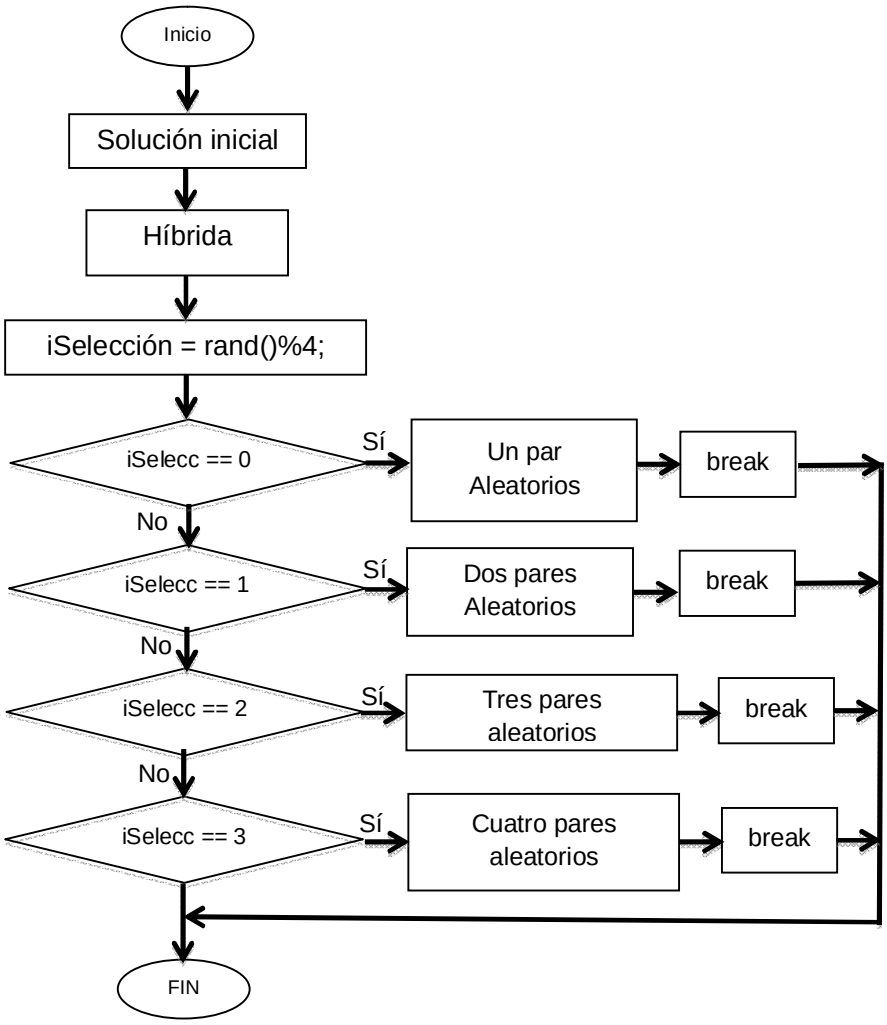


Figura 3-23Diagrama de flujo de la estructura de vecindad Hibrida

La literatura a demostrado que la estructura híbrida aplicada a búsqueda local obtiene resultados de mejor calidad que una estructura sencilla (Cruz et al., 2010b) y (Alina Martinez-Oropeza,2010)

(Yahya Z.Arajy ,Salwani Adbullah,2010),por esta razón se decidió aplicar la estructura híbrida al problema de emparejamiento de peso máximo.

Cabe mencionar que las estructuras de vecindad explicadas anteriormente fueron desarrolladas en Visual C++ 2008, sobre Windows 7 Premium.

3.4 Generación Aleatoria de Instancias de Prueba

Para generar las instancias, se desarrolló un programa en Visual C++ 2008,
el cuál permite generar las instancias de manera aleatoria para los vertices y para los pesos de los vértices .Este programa se desarrollo tomando en cuenta la función objetivo del problema de emparejamiento de peso máximo.se debe resaltar que el costo de cada emparejamiento es sumamente importante en este problema. A continuación se muestra un ejemplo de una instancia pequeña para observar el comportamiento del emparejamiento de peso máximo.

La figura 3-24 representa un ejemplo del emparejamiento de peso máximo en un grafo no dirigido con una posible solución parcial de 2 emparejamientos esta unión es entre los siguientes vértices: {{1,3}},{{4,5}} donde cada arista tiene un costo asociado. Él costo en este caso es de 17. Otra posible solución sería {{3,4}}{{2,5}}, obteniendo un ahorro o peso de 15 representado en la figura 3-25. El algoritmo de aceptación por umbral con aplicación con búsqueda local

hibrida determina el mejor costo para el problema de emparejamiento de peso máximo.

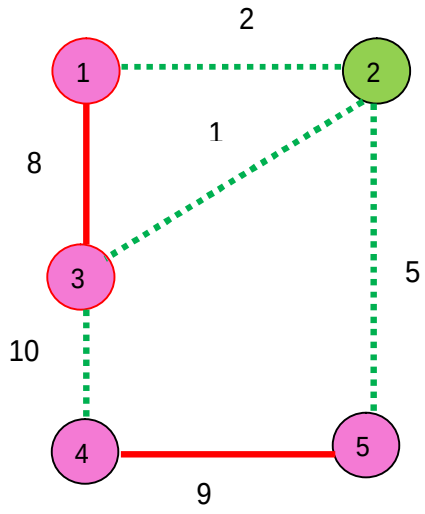


Figura 3-24Emparejamiento {1,3},{4,5}

con un costo de emparejamiento $17

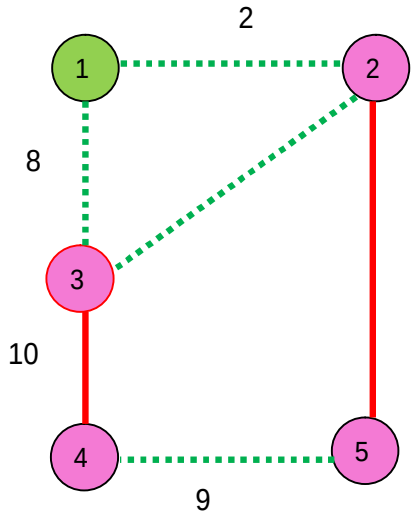


Figura 3-25Emparejamiento {3,4},{2,5} con un

Costo de emparejamiento $15

Como este trabajo se requiere del emparejamiento de peso máximo.De la dos soluciones mostradas se toma la mostrada en la figura 3-24 con un costo de 17.

3.5 Metodología de Sintonización

Se debe obtener una adecuada sintonización y análisis de sensibilidad de los parámetros utilizados para el algoritmo de aceptación por umbral ya que se debe buscar un tamaño adecuado para la vecindad que permita generar el mejor desempeño posible de la función de vecindad. Así como también determinar el comportamiento del algoritmo de aceptación por umbral.

Dentro del análisis de sensibilidad se evalúa el comportamiento de las variables, con la finalidad de establecer un rango numérico, dentro del cual el algoritmo de aceptación por umbral obtenga una mejora en cuanto a la eficiencia y eficacia. Los valores sintonizados para este problema se lleva a cabo con la finalidad de identificar aquellos valores que influyen de forma positiva en la calidad de la solución lo que nos permite obtener el mejor desempeño del algoritmo. Las variables sintonizadas son: parámetro de control inicial, tamaño de la vecindad, los parámetros del umbral y el parámetro de control final.

A continuación se explica una metodología de sintonización propuesta para encontrar la adecuada proporción en los valores correspondientes a los parámetros de control.

Selección de parámetro de control inicial:

Para establecer el parámetro de control inicial se realizó un análisis del problema tratado y del algoritmo para identificar los parámetros críticos que influyen de cierta manera en la calidad de las soluciones. Como es necesario establecer los rangos de cada parámetro que vas ser utilizado y poder hacer el análisis de sensibilidad se llevó a cabo una revisión en la literatura , pero no se encontró ningún valor establecido para cada uno de los parámetros de control.

El parámetro de control requiere de un análisis de sensibilidad ,para esto requiere tomar en cuenta los rangos establecidos de las muestras candidatas con un rango de 30 pruebas por parámetro, Con este rango se puede evaluar el comportamiento del algoritmo cuando los parámetros de control toman valores distintos.

Una vez que se ha obtenido el mejor valor para estas variables, se fija dicho valor y se comienza con la variación de otro parámetro, llevando a cabo el mismo proceso, hasta que se obtengan el conjunto de valores que permitan que el algoritmo mejore en eficacia y eficiencia.

Estos valores obtenidos se encuentran cumplido con la sintonización de los parámetros de control de entrada, esta sintonización se lleva a cabo con la finalidad de mejorar el desempeño del algoritmo de aceptación por umbral.

3.6 Análisis de Complejidad del Algoritmo de Aceptación por Umbral

Debido al análisis del análisis detallado de la función principal se ha determinado que el algoritmo funciona correctamente y es necesario realizar un estudio que permita conocer el comportamiento y medir su rendimiento ,centrándose principalmente en su simplicidad y el uso eficiente de los recursos.

Para conocer la complejidad del algoritmo de aceptación por umbral es necesario tomar en cuenta dos aspectos principales: el tiempo y los recursos necesarios para resolver un problema computacional (espacio). Estos parámetros son de vital importancia para definir si un algoritmo es eficiente o no; ya que si este requiere mucho tiempo para resolver un problema, no será de utilidad, lo mismo sucede si requiere gran cantidad de memoria.

El tiempo de ejecución de un algoritmo o complejidad temporal (T(n)), se encuentra en función de los siguientes parámetros: Datos de entrada, calidad del código objeto compilado, naturaleza y rapidez del procesador así como la complejidad del algoritmo.

La expresión (T(n)), n) representa el número de instrucciones simples (asignaciones, comparaciones, operaciones aritméticas, etc.) que son ejecutadas en el algoritmo.

Para calcular la complejidad del algoritmo de aceptación por umbra les neseasrio realizar un analis del número de instrucciones requeridas por el algoritmo

De acuerdo a esto, se muestra la ecuación correspondiente a la función temporal del algoritmo de aceptación por umbral.

Capítulo 4
Resultados Experimentales del Problema de Emparejamiento de Peso Máximo

En este capítulo se presentan los resultados obtenidos en las pruebas experimentales realizadas al Algoritmo de aceptación por umbral, donde se integra la estructura de vecindad hibrida a la búsqueda local .Estos resultados son comparados con los obtenidos por el mismo algoritmo cambiando la estructura a un más sencilla que es la estructura de vecindad con un par aleatorio.

Se desarrollo un análisis de la eficiencia y la eficacia del algoritmo con la estructura de vecindad con un par aleatorio y la estructura de vecindad hibrida aleatoria.

4.1 Descripción del Equipo Utilizado

Las pruebas experimentales realizadas para las estructuras de vecindad para el emparejamiento de peso máximo fueron realizadas en una computadora portátil con las siguientes características.

*** Procesador :** Intel Core i7-740QM quad processor (1.73GHz) whith Turbo Boost up to 2.93 Ghz
***Memoria:** 6 GBytes
***Sistema operativo:** Genuine Windows 7 Home Premium 64-bit .

El compilador utilizado fué Visual C 2008 se ejecutó la estructura de vecindad con un par aleatorio, la estructura de vecindad con dos pares aleatorios, la estructura de vecindad con tres pares aleatorios ,la estructura de vecindad con cuatro pares aleatorios y la estructura hibrida aleatoria .

En el caso de las pruebas para sintonizar las variables del algoritmo de aceptación por umbral se utilizó el Cluester Tarantula , ClusterNopal-IT Veracruz y Cluester UPEMOR con las siguientes características:

Caracteristicas Cluster Tarantula:

Proc Max 3.20GHz

RAM 7GB

HD 1.12TB

: : FRONTEND

Proc Intel(R) Pentium(R) D CPU 2.80GHz

RAM 1GB

HD 160GB

: : 6 NODOS

Procs Intel(R) Celeron(R) CPU E1400 @ 2.00GHz

RAM 1GB

HD 160GB

Compilador: Microsoft Visual C++ versión 6.0

Caracteristicas Cluster Nopal

Proc Max 3.20GHz

RAM 57GB

HD 1.2TB

: : FRONTEND

Proc Dual Intel(R) Pentium(R) 4 CPU 3.20GHz

RAM 1GB

HD 80GB

: : 5 NODOS

Procs Quad Intel(R) Core(TM)2 Quad CPU Q8200 @ 2.33GHz

RAM 4GB

HD 80GB

: : 9 NODOS

Procs Dual Intel(R) Core(TM)2 Duo CPU E8200 @ 2.66GHz

RAM 4GB

HD 80GB

Caracteristicas Cluster UPEMOR.

Proc Max 2.0GHz
RAM 20GB
HD 2.5TB

: : FRONTEND

Proc Quad Core Intel Xeón a 2.0GHz 64bits
RAM 4GB
HD 500GB

: : 4 NODOS

Procs Quad Core Intel Xeón a 2.0GHz 64bits
RAM 4GB
HD 500GB

Compilador: Microsoft Visual C++ versión 6.0

4.2 Pruebas de Estructuras de Vecindad

Las pruebas experimentales realizadas para las estructuras de vecindad para el emparejamiento de peso máximo fueron realizadas en una computadora portátil con procesador Intel Core i7-740QM quad processor (1.73GHz) whith Turbo Boost up to 2.93 Ghz con sistema operativo Genuine Windows 7 Home Premium 64-bit . El compilador utilizado fué Visual C 2008 se ejecuto la estructura de vecindad con un

par aleatorio ,la estructura de vecindad con dos pares aleatorios, la estructura de vecindad con tres pares aleatorios ,la estructura de vecindad con cuatro pares aleatorios y la estructura hibrida aleatoria .

Los tamaños de instancia probadas fueron de 100 vértices fueron generados de forma aleatoria. Las estructuras de vecindad fueron ejecutadas 30 veces para cada tamaño de instancia, teniendo un número total de 100 iteraciones realizadas durante la ejecución y registrando los resultados obtenidos de la mejor solución, en cuanto su función de costo.

Se realizó el cálculo del tiempo para encontrar 100 soluciones para cada instancia 100 nodos .

Grafica de la estructura de vecindad con un par aleatorio

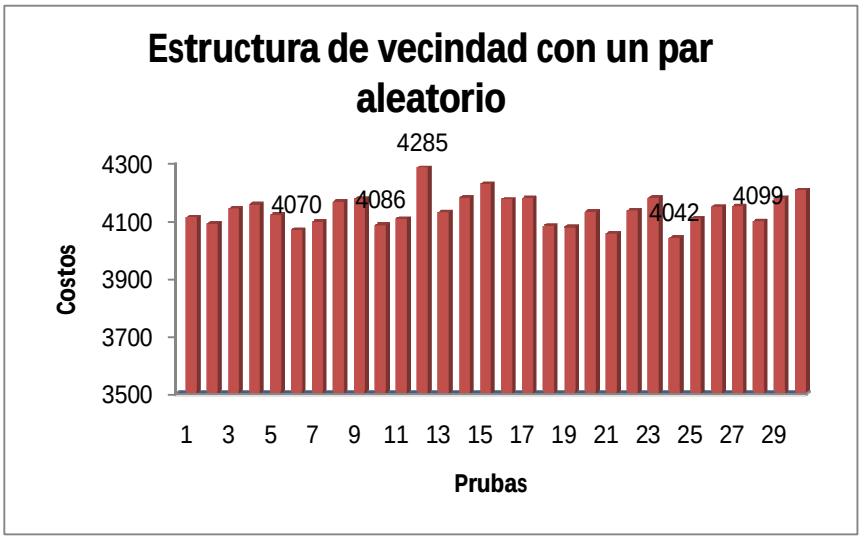


Figura 4-1Graficas de Pruebas de la Estructura de Vecindad con un Par Aleatorio

La figura 4-1 muestra los resultados obtenidos de 30 ejecuciones realizadas para la estructura de vecindad con un par aleatorio la función de costo muestra que para esta estructura la mejor solución de las 30 ejecuciones es la de 4285 y la peor es con un costo de 4042.

Grafica de la estructura de vecindad con dos pares aleatorio

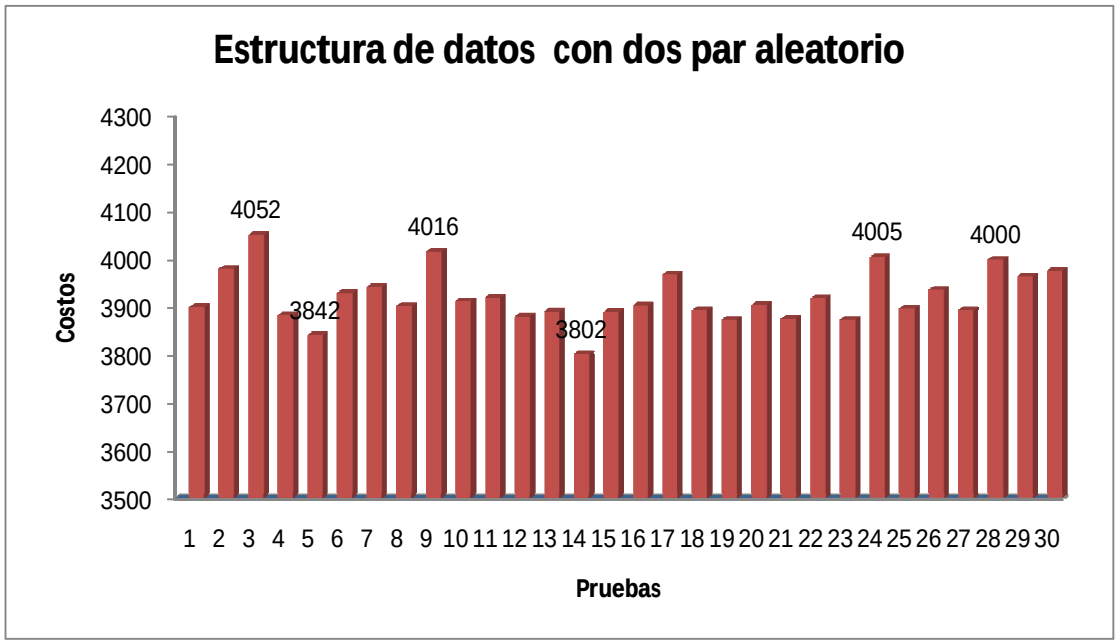


Figura 4-2Grafica de Pruebas de la Estructura de Vecindad con Dos Pares Aleatorios

La figura 5-1 muestra los resultados obtenidos de las mejores soluciones de la función de costo para la estructura de vecindad con dos pares aleatorios. Se puede observar que para esta estructura la mejor solución encontrada de las 30 ejecuciones es de 4052.El peor costo de esta estructura de vecindad con dos pares aleatorios es de 3802.

Grafica de la estructura de vecindad con tres pares aleatorio

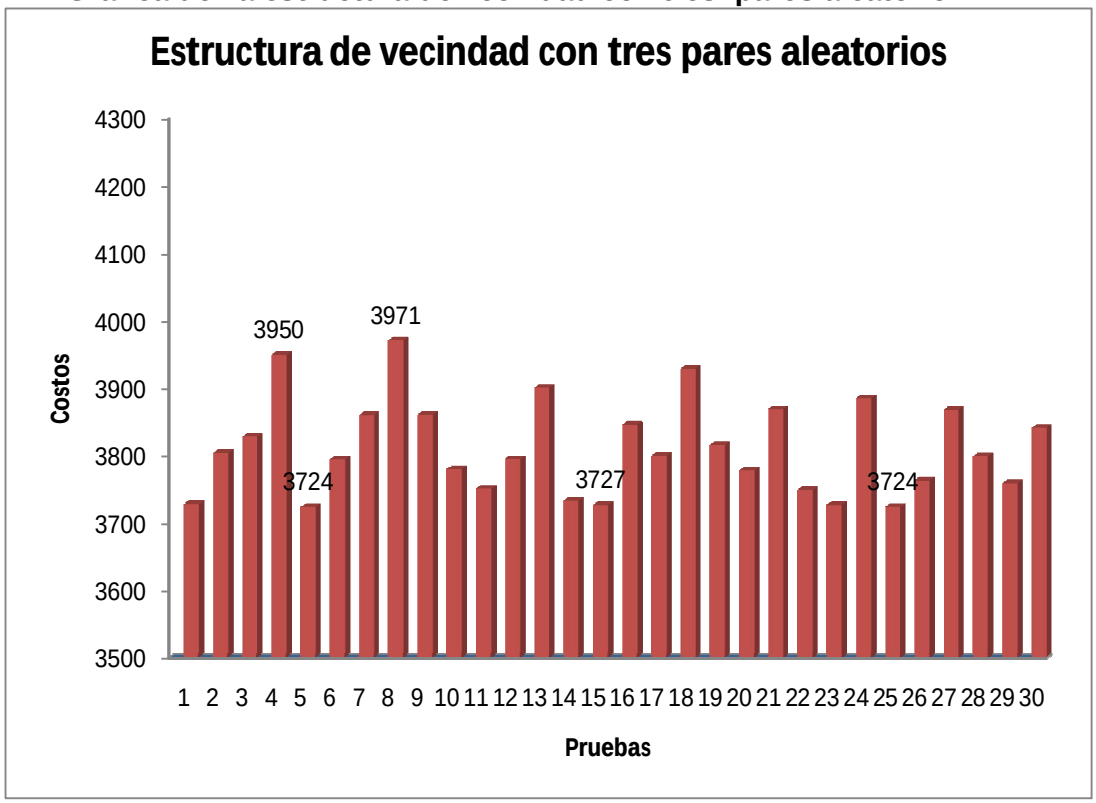


Figura 4-3Graficas de Pruebas de la Estructura de Vecindad con Tres Pares Aleatorios

La figura 5-2 muestra los resultados obtenidos de las mejores soluciones de la función de costo para la estructura de vecindad con tres pares aleatorios. Se puede observar que para esta estructura la mejor solución encontrada de las 30 ejecuciones es de 3971.El peor costo de esta estructura de vecindad con tres pares aleatorios es de 3724.

Grafica de la estructura de vecindad con cuatro pares aleatorio

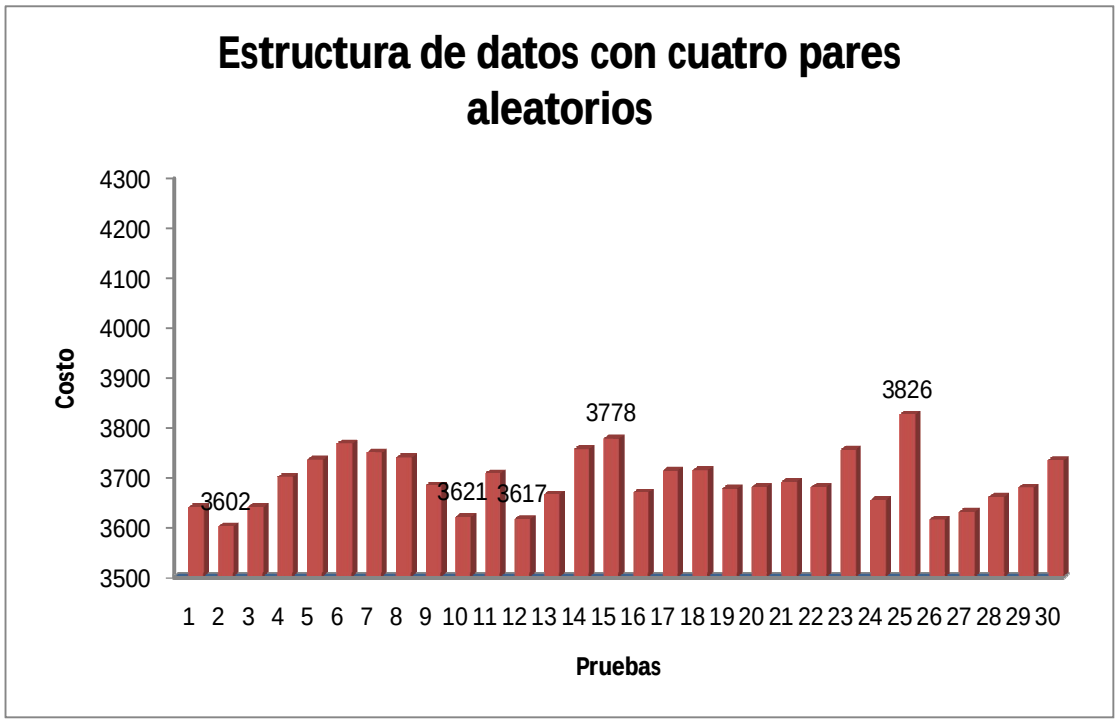


Figura 4-4Grafica de Pruebas de la Estructura de Vecindad con Cuatro Pares Aleatorios

La figura 5-3 muestra los resultados obtenidos de las mejores soluciones de la función de costo para la estructura de vecindad con cuatro pares aleatorios. Se puede observar que para esta estructura la mejor solución encontrada de las 30 ejecuciones es de 3826.El peor costo de esta estructura de vecindad con tres pares aleatorios es de 3602.

Grafica de la estructura de vecindad hibrida aleatoria

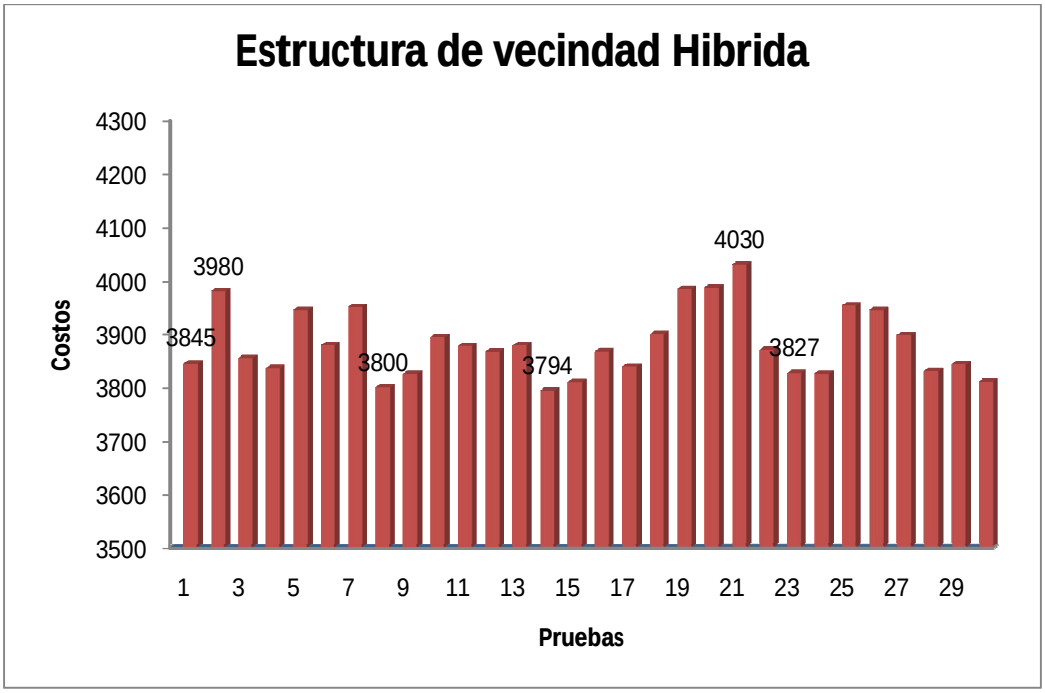


Figura 4-5Pruebas de la Estructura de Vecindad Hibrida Aleatoria

La figura 5.4 muestra los resultados obtenidos de las mejores soluciones de la función de costo para la estructura de vecindad hibrida aleatoria. Se puede observar que para esta estructura la mejor solución encontrada de las 30 ejecuciones es de 4030.El peor costo de esta estructura de vecindad hibrida aleatoria es de 3794.

Como se menciona en un principio el problema de emparejamiento de peso máximo se ha clasificado dentro de la teoría de complejidad como

NP-completo, debido a que es un problema difícil de resolver, por ello se hace uso de las técnicas de búsqueda por vecindad y encontrar el óptimo local.

Este trabajo de investigación demuestra que la estructura con un par aleatorio en cuanto a eficacia es la mejor a comparación con las otras cuatro estructuras. En eficiencia la estructura híbrida esta en un rango intermedio a comparación con las otras estructuras, es decir no es la de mejor tiempo de ejecución, pero tampoco es la de peor rendimiento. Podemos decir que este tipo de estructura puede ser aplicada a una meta heurística.

Se concluye que la estructura hibrida propuesta no es la más eficiente para el problema de emparejamiento máximo.

4.3 Análisis de Sensibilidad

Para hacer el análisis de sensibilidad se identifican los parámetros sintonizados del algoritmo de aceptación por umbral mostrados en tabla 3 .

Figura 4-61 Variables del Algoritmo de Aceptación por Umbral

Algoritmo de Aceptación por Umbral	
Variable	**Descripción**
$C_{s\_INICIAL}$	Es el parámetro de control
X_o	Solución inicial o configuración inicial del problema de emparejamiento
C_f	Parámetro de control final.
α	Es el coeficiente del parámetro de control.
L_x	La longitud de la cadena de Marlov .

Una vez que sean identificados los parámetros de control se establece el rango para realizar el análisis de sensibilidad a cada una de las variables mostradas en la tabla anterior.

Se Hace un análisis de los valores de cada una de las variables de control, se tiene que sí C_{s_INICIA} , que es el parámetro de control inicial se da al inicio como la temperatura del Algoritmo de Aceptación por Umbral este debe tener un valor adecuadamente alto en comparación con el parámetro de control final.

Para encontrar el valor de la temperatura se desarrolló una sintonización de temperatura de un valor de 100019 con un decremento de 311 por prueba, de esta forma observar el rendimiento del algoritmo. Se analizaron 180000 archivos para encontrar la temperatura inicial .
La temperatura se tomo en función de la estructura de vecindad con un par aleatorio y la estructura de vecindad hibrida. En donde los tamaños de instancia probadas fueron de 100 vértices que fueron generados de forma aleatoria.
El algoritmo de aceptación por umbral para este parámetro fue ejecutado 30 veces para cada tamaño de instancia, teniendo un número total de 100 iteraciones realizadas durante la ejecución y registrando los resultados obtenidos de las mejores soluciones hasta encontrar el criterio de paro.

La tabla 3-1 muestra los rangos utilizados para encontrar la proporción adecuada para el parámetro de control inicial.Para realizar el análisis de sensibilidad para el TA.

Tabla 4-2Rangos del parámetro de control inicial utilizados para realizar el análisis de sensibilidad para el algoritmo de aceptación por umbral

Parámetro de control	Límite Inferior	Límite superior
$C_{s\_INICIAL}$	10019	1000

La variable de **Xo** utilizada en este algoritmo de aceptación por umbral por que es la variable que contiene la primera solución del problema de emparejamiento de peso máximo al momento de entrar al algoritmo de aceptación por umbral esta es asignada a una variable llamada costo_new_funcion= $C_{s\_INICIAL}$ .Cuando el algoritmo encuentre una buena solución acepta esa solución y continúe con el decremento de la temperatura del algoritmo de aceptación por umbral en caso contrario se aplica el umbral que se describirá mas adelante.

La variable de **Xo** es ejecutada con 100, 200,300,400,500 vértices para resolver emparejamientos de peso máximo las pruebas son mostradas en el capitulo 4.5,4.6.

Parámetro de control final se sintonizó como dato de entrada al primer ciclo del algoritmo de aceptación por umbral y este determina el criterio de paro cuando se alcanza el valor de la temperatura inicial .Para identificar el adecuado parámetro de control final $\mathbf{C_f}$ se ejecutaron 900 pruebas iniciando 15 incrementos de 0.143 hasta llegar a un límite superior de 2.003 y con este límite superior se hicieron 15 decrementos de 0.00006 hasta llegar a un límite inferior de 0.00016. El tamaños de instancia probadas fueron de 100 vértices que fueron generados de forma aleatoria.
El algoritmo de aceptación por umbral para este parámetro fue ejecutado 30 veces para cada tamaño de instancia, teniendo un número total de 100 iteraciones realizadas durante la ejecución y registrando los resultados obtenidos de las mejores soluciones hasta encontrar el criterio de paro.

La tabla 3-2 Muestra los rangos utilizados para encontrar la proporción adecuada para el parámetro de control final.

Tabla 4-3Rangos del parámetro de control final utilizados para realizar el análisis de sensibilidad para el algoritmo de aceptación por umbral

Parámetro de control	Límite Inferior	Límite superior
$\mathbf{C_f}$	0.001 0.0016	2.003 0.00094

$\mathbf{L_x}$ la longitud de la cadena de Marlov que traducido a un problema de optimización combinatoria se ha demostrado que en este tipo de algoritmos la L_x equivale al tamaño de la vecindad de nuestro problema de una solución X_0 que va tener una cantidad de vecinos finita dependiendo del problema a resolver. Se establecion con un valor de 2*(N-1);

Ya que se encuentra los parámetros control antes mencionados se continua a sintonizar el parámetro de control llamado Delta iniciando la sintonización con decrementos de 0.001 los limites utilizados para la sintonización de delta se muestran en la tabla 3-3. Y se continúa con la sintonización de la variable llamada D con un rango mostrado en la tabla 3-4.

Estos dos parámetros son utilizados para el cálculo de Umbral.

Si en la ejecución del algoritmo no se obtienen resultados positivos se pasa a la ejecución del umbral logrando de esta forma no caer en el mínimo local.Este no solo toma la mejor solución sino también del deterio de la función objetivo siempre y cuando no supere el umbral establecido y el criterio de paro se cumpla.

Tabla 4-4Rangos de Delta utilizados para realizar el análisis de sensibilidad para el algoritmo de aceptación por umbral

Parámetro de control	Límite Inferior	Límite superior
Delta	0.996	0.965

Tabla 4-5Rangos de D utilizados para realizar el análisis de sensibilidad para el algoritmo de aceptación por umbral

Parámetro de control	Límite Inferior	Límite superior
D	1	10

4.4 Aplicación de la Estructura de Vecindad al Algoritmo de Aceptación por Umbral

Para mejorar la calidad de las soluciones obtenidas por el algoritmo de aceptación por umbral,se decidió implementar dos estructuras de vecindad [Cruz et al. 2010b],para observar como explotan los espacios de soluciones.

Para aplicar la estructura de vecindad con un par aleatorio y la estructura de hibrida aleatorio fue necesario evaluar todas la estructuras y evaluar el comportamiento de estas dos estructuras con el algoritmo de aceptación por umbral con 100 vértices de esta forma en las pruebas desarrolladas se identifica la mejor solución hasta el momento , Una vez que se tiene el mejor valor de la función objetivo hasta el momento, se le aplica la búsqueda local con la estructura de vecindad híbrida ,de modo que e n cada iteración se vaya mejorando el valor de la función objetivo hasta el momento.

Se resalta que este procedimiento permite mejorar la solución ya mejorada que lleva mayor explotación en el espacio de soluciones obteniendo de esta forma los resultados de mayor calidad para el problema de emparejamiento de peso máximo .

4.5 Resultados Experimentales

La tabla 3-5 presenta los resultados obtenidos para cada estructura de vecindad. Desarrolladas en este trabajo de tesis para encontrar el emparejamiento de peso máximo de las cuáles se hizo una evaluación de la mejor solución, peor solución, la función de costo promedio, así como su desviación estándar σ de las 30 pruebas realizadas con 100 iteraciones para cada estructura de vecindad. Se puede observar que la estructura de vecindad de par aleatorio en este trabajo es la más eficiente y eficaz.

La segunda mejor estructura en eficacia es la que realiza dos movimientos aleatorios para la instancia de 100 vértices.

La estructura promedio en eficiencia es la estructura hibrida aleatoria con un tiempo de 8.93 minutos aleatorios para la instancia de 100 vértices.

El comportamiento de la desviación estándar para cada estructura muestra que la dispersión de las soluciones en las 30 pruebas, es diferente dependiendo del permutación que se aplique, mientras mayor sea la desviación estándar, mayor será la variabilidad, es decir mayor explotación del espacio de soluciones.

Tabla 4-6 Resultados para 100 Vértices. 30 ejecuciones con la búsqueda local iterada para cada estructura.

100 VÉRTICES					
TIPO DE ESTRUCTURA	**TIEMPO**	**MEJOR**	**PEOR**	**PROMEDIO**	**σ**
Par Aleatorio	**2 min**	**4285**	**3502**	**4137.23**	**19.369**
Dos Pares Aleatorios	6.1053 min	4052	3239	3921.3	69.89
Tres Pares Aleatorios	40.38 min	3971	3168	3812	80.33
Cuatro Pares Aleatorios	558.516 min	3826	2901	72.62	3693.4
Híbrida	8.93 min	4030	3271	3881.7	69.59

La tabla 4-6 muestra el análisis completo de cada una de las 30 ejecuciones de cada estructura para 100 vértices y se resalta la estructura de vecindad con un par aleatorio que demostró ser la más eficiente con un tiempo de 2 minutos por ejecución y la más eficaz con un costo de 4285 que cumple con la función objetivo del emparejamiento de peso máximo.

Referencias

Alfredo Caicedo Barrero ,Graciela Wagner de García, Rosa María Méndez Parra (2010), ”Introducción a la teoría de grafos “,diciembre del 2010,[10-octubre-2010], Edicciones Elizcom ,Primera ediccion ,ISBN:978-958-99325-7-5

Alina Martinez-Oropeza(2010),”Solucion al Problema de maquinas en Paralelo no Relacionadas mediante un algoritmo de colonia de hormigas” ”,(Maestría en tecnología electrica).Universidad Autonoma del Estado de Morelos,CIICAp,Avenida Universidad 1001.Col. Chamilpa.C.P.62210,Morelos.,pp 1-157

(Cruz et al., 2010b). Cruz-Chávez Marco Antonio, Martínez-Oropeza Alina, Serna-Barquera Sergio A. Neighborhood Hybrid Structure for Discrete Optimization Problems.
Accepted as paper in Cerma 2010. To publish in IEEE. 2010.

(Cruz y Juárez, 2010). Cruz-Chávez Marco Antonio, Juárez Pérez Fredy. Algoritmo de Recocido Simulado con Paso de Mensajes en Ambiente Grid para el Problema de Máquinas en Paralelo no Relacionadas. A enviar a Journal of Grid Computing. ISSN (e):
1570-7873. Springer. 2010.

[Chun y Chuen, 2006]. Chun-Lung Chen, Chuen-Lung Chen. A Heuristic Method for a Flexible Flow Line with Unrelated Parallel Machines Problem. IEEE Conference on Robotics, Automation and Mechatronics. Digital Object Identifier: 10.1109/RAMECH.2006.252673. pp. 1 – 4. 2006.

[Cruz et al., 2009b]. Cruz-Chávez Marco Antonio, Martínez-Oropeza Alina, Zavala-Díaz José Crispín, Martínez-Rangel Martín G. Relajación del Problema de Calendarización de Trabajos en un Taller de Manufactura utilizando un Grafo Bipartita. 7mo. Congreso Internacional de Cómputo en Optimización y Software AGECOMP-CICos 2009. ISBN: 978-970.9750-26.3. 2009

[Cruz et al., 2010a]. Cruz-Chávez Marco Antonio, Martínez Oropeza Alina, Rivera López Rafael. Relaxation of Job Shop Scheduling Problem using a Bipartite Graph. Accepted as
paper in Cerma 2010. To publish in IEEE. 2010.

Cruz-Chavez,Alina Martinez-Oropeza(2010) “Estructura Hibrida de Vecindad para el Problema de Optimización Distribuida

(Bisbal Riera Jesús 2009)Bisbal Riera Jesús 2009. “Manual de Algorítmica“.Recursividad, Complejidad y Diseño de Algoritmos. Ed. UOC.Primera Edición en Castellano. ISBN: 978-84-9788-027-5. pp.54-59. Barcelona. 2009.

Cruz-Chávez M. A, Rivera-López R.,(2007) .”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.

Cristina Borra Marcos ,Francisco Gómez García (2006). “Los Factores Determinantes del Emparejamiento Educación-Empleo:Evidencia Partir de la Nueva Muestra de Economistas1”.Universidad de Sivilla. XVI Jornadas de la Asociación de Econimia de la Educación.ECOD2.05/038.pp1-11 ,2006

Daisuke Takafuji,Satoshi Taoka and Toshimasa Watanabe(2002).”Efficent Aproximation for Algorithms for the Maximum Weigh Matching Problem”, ISBN:0-7803-7448-7 .pp.457-460.

Dueck,G;Scheuer T.(1990).”Threshold accepting:A general purpose optimization algorithm appearing superior to simulated annealing “.Journal of computation Physics 90,pp.161-175.

Gary Chartrand (1977),“Introductory Graph Teory”,Manufactured in the United States of America Dover Plucations ,Inc, 31 Enero ,1977[Octubre-2010],First published ,ISBN 0-486-24775-9

J.C.Fernando,V.Gregori (2002),”Matemáticas Discretas”,octubre DE 2002 [Noviembre,2010]Editorial Reverté,S.A.,Segunda Ediccion. ISBN:84-291-5179-6

Jianhong-Yang , Jiangbo-Shu,Chunxiu-Xiong,Xing-Zhou(2009).”A Hybrid Tree Spatial Indexing Method Base d on the Cluester Algorithm ”,Department of Computer and Information Engineering Wuhuan Polytechnic University Wuhan,Chica.ISBN:978-1-4244-4994-1.

Yahya Z.Arajy ,Salwani Adbullah(2010),“Hybrid Variable Neighbourhood

Search Algorithm for Attribute Reduction in Rough Set Theory".Data Mining and Optimisation Research Group (DMO),Center for Artificial Intelligence Technology Universiti Kebangsaan Malaysia,43600 Bangi Selangor,Malaysia. ISBN:978-1-4244-8136

Kathryn Blackmond Laskey,Member,IEEE .(1995). "Sensitivity Analysis for Probability Assessments in Bayesian Networks", Transactions on System ,Man , and Cybernetics ISSBN. 00-18-9472,Vol 25 ,No.6,pp .901-906,1995y values in Bayesian network models.

L. Valiant(1979)."The complexity of computing the permanent" Theoretical Computer Science ",
8:189–201, 1979.

Wetzel. A.(1983)." Evaluation of the Effectiveness of Genetic Algorithms in Combinatorial Optimization. University of Pittsburgh, Pittsburgh "(unpublished). 1983.

(Romero, 1990). D. Romero, y A. Sanchez. "Methods for the one-

dimensional space allocation problem", *Computers Opns Res*. Vol. 17 No.

5. Great Britain, 1990. pp. 465-473.

Papadimitriou C. H., Steiglitz.(1998)." Combinatorial Optmization Algorithms and
Complexity". ISBN. 0-486-40258-4. USA. 1998

Pinedo,(2008). "Michael L. Scheduling Theory, Algorithms and Systems". Third Edition. New York University. ISBN: 978-0-387-78934-7. pp. 14. 2008.
Michalewicz Zbigniew, Fogel David B .(2002) ."*How to Solve It: Modern Heuristics"*. Springer-Verlag BerlinHeidelberg New York. ISBN. 3-540-66061-5. pp. 55 – 90. 2002

R. Kulkarni, M. Mahajan, and K. R,(2008). "Varadarajan Some perfect matchings and perfect halfintegral
matchings in nc. Chicago Journal of Theoretical Computer Science (CJCS)",2008.

Reinaldo Guiudici E,Angeles Bris Llusch(1997)."Introducción a la Teoría de Grafos" ,Edicciones de la Universidad Bolivar,1997[25-septiembre-2010],Primer Edicicon, ISBN:980-237-154-8

Sánchez Enriquez Heider Isaias,Rodríguez Maysundo Eduardo,Ruiz Campos Johnatan Eric(2006).”Software para Teoria de Grafos”., Universidad Nacional de Trujillo,Escuela Profesional de Informática

Claude Berge(2001),”The Theory of graphs ”.Library of Congress Cataloging-in-Publication Data Berge,Claude.,Paris .first Published in 2001,[ENERO 211].ISBN 0-486-41975-4

http://books.google.com.mx/books?id=h5BjnaoKyOwC&printsec=frontcover&hl=es&source=gbs_ge_summary_r&cad=0#v=onepage&q&f=false

L. Morales, R. Garduño, D. Romero(1992).” The Multiple-Minima Problem in Small Peptides Revisited:”The threshold Accepting Approach. In: Journal of Biomolecular Structure & Dynamics, Vol. 9, No. 5, (1992) 951-957.

H. Telley, Th. M. Liebling, A. Mocellin(1992) .”Reconstruction of polycrystalline structures: a new application of combinatorial optimization. In”. Journal computing, Vol. 38, (1987) 1-11.

Bruce S. Elenbogen, Bruce R. Maxim,(1992)” Scheduling a bridge club: a case study in discrete optimization. In” Mathematics magazine, Vol. 65, No. 1, (february, 1992) 18-26.

J. Pérez O., R. Pazos R., D. Romero y L. Cruz R.,(2000) "Vertical Fragmentation and Allocation inDistributed Databases with Site Capacity Restrictions Using the Threshold Accepting
Algorithm", *Proceedings of the Mexican International Conference on Artificial Intelligence*,
Acapulco, México, Abr. 2000, pp. 75-81

Jose R.Medina Folgado,Victor Yepes Piqueras(2006) “Optimizacion de rutas mediante la busqueda en entornos variables y aceptación por umbral estocastico”,ORSA Journal on computing 2,pp.151-166.

(Jacques A. Ferland,1996).Jacques A. Ferland, Alain Hertz, Alain Lavoie: An object oriented methodology for solving assignment type problems with neighborhood search technique. In: Operations research, Vol. 44, No. 2, (March, 1996) 347-359.

S. Kirkpatrick, C. D. Gelatt and M. P. Vecchi(1983). “Optimization by Simulated Annealing”. In: Science 220, (1983)671-680.

N. Bouhmala, P. A. Knuutti, H. H. Naegeli(1996): A distributable algorithm for optimizing mesh partitions. In: IEEE, (1996) 544-549.

R. Jensen and Q. Shen,(2003), "Finding Rough Set Reducts with Ant Colony Optimization, " *Proc. 2003 UK Workshop Computational Intelligence*, pp. 15-22, 2003.

[2] T. StÄutzle. Local search algorithms for combinatorial problems analysis, algorithms and news applications. DISKI Dissertationen zur KÄunstliken Intelligenz., 1999.

[8]. Wetzel. A. Evaluation of the Effectiveness of Genetic Algorithms in Combinatorial Optimization. University of Pittsburgh, Pittsburgh (unpublished). 1983.

[11]HOOS, H.H.; STÜTZLE, T. Stochastic local search: foundations and applications, Morgan Kaufmann Publishers, San Francisco, CA, 2005.

20. Abbas A. El Gamal, Lane A. Hemachandra, Itzha Shperling, Victor K. Wei: Using simulated annealing to design good codes. In: IEEE Trans. On information theory, Vol.it-33, No. 1, (January, 1987) 116-123.

APENDICES

Funciones polinomiales

Las funciones polinomiales tienen una gran aplicación en la elaboración de modelos que describen fenómenos reales. Algunos de ellos son: la concentración de una sustancia en un compuesto, la distancia recorrida por un móvil a velocidad constante, la compra de cierta cantidad de objetos a un precio unitario, el salario de un trabajador más su comisión, la variación de la altura de un proyectil, entre otros.

Son aquellas cuya regla de correspondencia es un polinomio. Recordando que el grado de un polinomio es el exponente mayor de la variable, podemos hablar de una función polinomial de grado n.

Tabla A-1 Representación de una función polinomial

Función Polinomial	**Llamamos a una función polinomial de grado n, si tiene la forma en donde n es un entero positivo.** $\mathbf{a_0x^n + a_1x^{n-1} + a_{n-1}x \quad + a_{n-1}x \quad + a_n, + a_0 \neq 0}$

Todas las funciones polinomiales tienen como dominio al conjunto de números reales R, pero su contradominio varía dependiendo del tipo de

función que sea. Una función polinomial puede considerarse como una suma de funciones cuyos valores son del tipo c_{xk}, donde c es un número real y k es un entero no negativo.

Ejemplos particulares de la función polinomial son, la función lineal (función polinomial de grado uno), la función cuadrática (función polinomial de segundo grado), función cúbica (función polinomial de tercer grado).Mostradas en la tabla A-2.

Tabla 7 Funciones polinomiales

Grado	Función	Definición	Características
0	Constante	f(x) = k	Asigna a cada argumento la misma imagen k. Recta horizontal. No tiene raíces.
1	Identidad	f(x) = k	Asocia a cada argumento del dominio el mismo valor en el contra dominio. Recta que pasa por el origen con un ángulo de 450. Raíz en el punto x = 0.
1	Lineal	f(x) = mk + k	Recta con inclinación aguda si $m > 0$ y obtusa si $m < 0$. Raíz en el punto $x = -k/m.$
2	Cuadrática	f(x) = ax^2k + bx + c	Parábola cuya ordenada del vértice es k. Raíces dadas por la fórmula:

			$x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$
3	Cúbica	$f(x) = ax^3 + bx^2 + cx + d$	Tiene al menos una raíz real.

Estructura de Datos Utilizados en el Algoritmo de Aceptación por Umbral

Para darle una solución factible al algoritmo de aceptación por umbral con búsqueda local hibrida para el problema de emparejamiento de peso máximo se utilizo las estructuras de datos que permiten el uso ordenado y eficaz de la información estas estructuras pueden contener uno o más elementos los cuales pueden tener distinto tipo de datos.

Las estructuras son manejadas con los mismos campos

Las estructuras manipuladas en este programa son

arista matching[N][N],permutacion[N][N],svecina[N][N],S_Actual[N][N],SOL_ILS[N][N],AUmbral[N][N];

A continuación se explica la estructura utilizada para el problema de emparejamiento de peso máximo.

```
typedef struct{
        int vértice;
        int conexión;
        int peso;
} arista;

arista matching[N][N];
```

La estructura de matching es la principal del programa que determina el emparejamiento de peso máximo en su campo vértice y peso .Estos

son cagados a la estructura de datos mediante la lectura de un archivo txt que se genero de forma aleatoria .Una vez cargados en la estructura de datos .El programa desarrolla los emparejamientos de pesos máximos por medio de su campo vértice con su respectivo peso tomado de esta estructura de datos,los vértices emparejados son marcados en su campo conexión que indica si existe emparejamiento de datos .

Cuando el programa obtiene la primera solución del emparejamiento de peso máximo se hace una copia del recorrido correspondiente a la mejor solución de la función objetivo hasta el momento en la estructura de datos llamada permutación.

Esta estructura llamada permutación solo contiene los vértices emparejados con su respectivo peso.

```
typedef struct{
        int vértice;
        int conexión;
        int peso;
}arista;

arista  permutación[N][N];
```

La estructura de datos llamada svecina[N][N],S_Actual[N][N] son utilizadas para hacer los movimientos de la estructura de vecindad con un par aleatorio y la estructura de vecindad hibrida aleatoria .La estructura llamada AUmbral[N][N] almacena los resultados del algoritmo de aceptación por umbral.

```
typedef struct{

        int vertice;
        int conexion;
        int peso;
}arista;
```

svecina[N][N],S_Actual[N][N], AUmbral[N][N]

Calculo de Complejidad Temporal por pasos

El cálculo de complejidad es importante para conocer la eficiencia del algoritmo. La lectura indica que cualquiera de las instrucciones del algoritmo puede ser contado como pasos.

BUSQUEDA_BINARIA

```
1  void Busqueda_Binaria(int *Arreglo, int buscado)
2   {
3    int inicio, fin, mitad;
4       inicio = 0;
5        fin = TAM-1;
6        mitad = (inicio+fin)/2;
7        while (inicio < fin && buscado != Arreglo[mitad]){
8             if(buscado > Arreglo[mitad])
9                  inicio = mitad+1;
10               else
11                 fin = mitad-1;
12                 mitad=(inicio+fin)/2;
13            }
14         if (buscado == Arreglo[mitad])
15         printf("\nEl elemento %d  se encuentra en la posicion  %d ",
buscado, mitad);
16         else
17            printf("\n%d NO EXISTE", buscado);
18     }
```

Cálculo de la complejidad del algoritmo de búsqueda binaria

Propuesta numero 2 del algoritmo de búsqueda binaria:

Líneas 1, 2,3 :	0 pasos
Línea 4:	C_1 1 paso
Línea 5:	C_2 1 paso
Línea 6:	C_3 1 paso
Línea 7:	C_4 n+1 veces
Línea 8	C_5 n veces
Línea 9:	C_6 n veces
Línea 10 :	0 paso
Línea 11:	C_7 n veces
Línea 12:	C_8 n veces
Línea 13:	0 pasos
Línea 14:	C_9 n paso
Línea 15:	C_{10} 1 paso
Línea 18:	C_{11} 1 paso
Línea 16,17:	0 pasos

$T(N)= C_1(1)+ C_2(1)+ C_3(1)+ C_4(n+1)+ C_5(n)+ C_6(n)+ C_7(n)+ C_8(n)+ C_9(n)\ C_{10}(1)+ C_{11}(1) =$ 6N+6pasos

EL Orden de complejidad para el mejor de los casos: O(n).

EL Orden de complejidad para el peor de los casos: O(n/2).

Complejidad Asintótica

Peor Caso:

$$O(g(n)) = \{ f(n) \mid \exists c, n_0, \forall n \geq n_0, c_1 g(n) \leq f(n) \leq c_2 g(n) \}$$

$$f(n) = 6n + 6 = O(n) si \exists c: \quad c_1 \leq n \leq c_2 n \quad n \geq n_0 = \mathbf{1750}$$

Caso promedio:

$$\Theta(g(n)) = \{ f(n) \mid \exists c_1, c_2, n_0, \forall n \geq n_0, c_1 g(n) \leq f(n) \leq c_2 g(n) \}$$

$$f(n) = \frac{n}{2} \Theta(n) si \exists c_1, c_2: \quad c_1 n \leq \frac{n}{2} \leq c_2 n \quad n \geq n_0 = \mathbf{2300}$$

C .Código Fuente de la Estructura de Vecindad con un Par Aleatorio

En esta sección se presenta el código fuente corresponde a la búsqueda local iterada y estructura de vecindad con un par aleatorio, implementada al algoritmo de aceptación por umbral aplicado al problema de emparejamiento de peso máximo.

```
/*Declacion de  librerias*/
#include "stdafx.h"
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <math.h>
#include  <time.h>
#define N 501
clock_t inicio,fin;
/*****************************************************************/
/* Declaracion de estructura de datos*/
typedef struct{
            int vertice;
            int conexion;
            int peso;
}arista;

arista matching[N][N],permutacion[N][N],svecina[N][N],S_Actual[N][N],SOL_ILS[N][N],AUmbral[N][N];
char *Archs[30]={"PAR_ALE9708txt","PAR_ALE2.txt","PAR_ALE3.txt","PAR_ALE4.txt","PAR_ALE5.txt"};
/**********************Declaracion de  matrices*********************/
int  mat_vec[N][N],verif_emp[N-1],vector_ppar[N-1],vector_structura[N-1];
int matriz_ppar[N][N];
int matriz_new[N][N];
/********Declaracion de funciones para la  estructura de vecindad hibrida**********************/
void HIBRIDO(arista(*)[N]);
void ParAleatorio(arista (*arbol)[N]);
void DosParesAleatorios(arista (*)[N]);
void TresParesAleatorios(arista (*)[N]);
void CuatroParesAleatorios(arista (*)[N]);
/************Funcion principal del programa***********/
int main()
{
   int     i=0,iteracion=0,CS_INICIAL=0,Costo_inicial=0;
   int     D=5;
   float  cs_TAC=0;
            srand(time(NULL));
```

```
        do
            {
               inicio=clock();
              Primera_solucion(matching,permutacion);
               CS_INICIAL=suma_Solucion(permutacion,matriz_new);
              inicializar_grafo(S_Actual,svecina);
               inicializar_grafo(S_Actual,AUmbral);
               cs_TAC=algoritmo_aceptacion_umbral(permutacion,S_Actual,svecina,AUmbral,CS_INICIAL,D);
         GuardarDatos(i,CS_INICIAL,cs_TAC);
    i++;
} while (i<=30);
  fin=clock();
  GuardarTiempo(i,inicio,fin);//Guardar el tiempo en el archivo
 return 0;
}


/**************DESARROLLO DEL ALGORITMO DE ACEPTACION POR  UMBRAL*********/
int algoritmo_aceptacion_umbral(arista (*permutacion) [N],arista (*S_Actual) [N],arista (*svecina) [N],arista(*
AUmbral)[N],int CS_INICIAL,int D)
{
int          Cc=0,Xi=0,j,i,Costo_vecina=0,Costo_delta=0, Delta_f=0,Costo_actual=0,Costo_New_funcion=0;
 int         LM=2*(N-1);
float        u,delta=0.977,Cf=1.288;
             u=(float)CS_INICIAL/(float)D;
             copy_structura(S_Actual,permutacion);
             Costo_New_funcion=CS_INICIAL;
             copy_structura(AUmbral,S_Actual);
        for(i=6287;i>=Cf;i=i*delta){
             for(j=1;j<=LM;j++){
                  copy_structura(svecina,S_Actual);
                  par(svecina);
                  Costo_vecina=suma_Solucion(svecina,matriz_new);
                  Costo_actual=suma_Solucion(S_Actual,matriz_new);
                   Delta_f=(Costo_vecina - Costo_actual);
                           if(Delta_f>=0)
                           {
                              copy_structura(S_Actual,svecina);
                         if(Costo_vecina>=Costo_New_funcion)
                             {
                               Costo_New_funcion=Costo_vecina;
                               copy_structura(AUmbral,svecina);
                            }
                           }else
                               {
                         if(fabs((float)Delta_f)<(float)u)
                               {
                                 copy_structura(S_Actual,svecina);
                                }
                           }
               }//fin for
               u=u*delta;
}//fin for
return Costo_New_funcion;
}//fin función


void Primer_par(arista(*permutacion)[N])
{
int    Gv=0,cont_vertices=0,E1=0,E2=0,D1=0,bandera=0,p=0,vert_encontrado=0;
       llenar_en_ceros_vector(vector_ppar,matriz_ppar);
  for(;;)
  {
    Gv=rand()%(N-1)+1;
    cont_vertices=num_vertices(permutacion,Gv);
      if(cont_vertices!=0 )
        {
```

```
            p=rand()%cont_vertices+1;
            vert_encontrado=numvertice(permutacion,Gv,p);
            E1=Eliminar_vert_ex_conectado(permutacion, Gv);
             E2=Eliminar_vert_ex_conectado(permutacion, vert_encontrado);
             ConPAleatorio(permutacion,E2, E1);
            ConPAleatorio(permutacion,vert_encontrado, Gv);
            Proceso_alamcenar(permutacion,E1,E2,Gv,vert_encontrado);
            break;
           }
    }
    Almacenar_D1_ppar(permutacion,vector_ppar);
    par_impar_vector(vector_ppar,matriz_ppar);

}
```

D. Código Fuente de la Estructura de Vecindad Hibrida Aplicada a una Búsqueda Local

```
/*Declacion de  librerias*/
#include "stdafx.h"
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <math.h>
#include  <time.h>
#define N 501
clock_t inicio,fin;
/*****************************************************************/
/* Declaracion de estructura de datos*/
typedef struct{
            int vertice;
            int conexion;
            int peso;
}arista;

arista matching[N][N],permutacion[N][N],svecina[N][N],S_Actual[N][N],SOL_ILS[N][N],AUmbral[N][N];
char *Archs[30]={"PAR_ALE9708txt","PAR_ALE2.txt","PAR_ALE3.txt","PAR_ALE4.txt","PAR_ALE5.txt"};
/**********************Declaracion de  matrices********************/
int  mat_vec[N][N],verif_emp[N-1],vector_ppar[N-1],vector_structura[N-1];
int matriz_ppar[N][N];
int matriz_new[N][N];
/********Declaracion de funciones para la  estructura de vecindad hibrida**********************/
void HIBRIDO(arista(*)[N]);
void ParAleatorio(arista (*arbol)[N]);
void DosParesAleatorios(arista (*)[N]);
void TresParesAleatorios(arista (*)[N]);
void CuatroParesAleatorios(arista (*)[N]);
/************Funcion principal del programa***********/
int main()
{
   int     i=0,iteracion=0,CS_INICIAL=0,Costo_inicial=0;
   int     D=5;
   float  cs_TAC=0;
            srand(time(NULL));
        do
            {
               inicio=clock();
```

```
            Primera_solucion(matching,permutacion);
             CS_INICIAL=suma_Solucion(permutacion,matriz_new);
            inicializar_grafo(S_Actual,svecina);
             inicializar_grafo(S_Actual,AUmbral);
             cs_TAC=algoritmo_aceptacion_umbral(permutacion,S_Actual,svecina,AUmbral,CS_INICIAL,D);
        GuardarDatos(i,CS_INICIAL,cs_TAC);
   i++;
} while (i<=30);
  fin=clock();
  GuardarTiempo(i,inicio,fin);//Guardar el tiempo en el archivo
 return 0;
}
```

```
int Umbral(int CS_INICIAL ,int delta,int D )
{
            int u;
      u=(float)CS_INICIAL/(float)D;
 return u;
}
/*************************FUNCION DE  LA ESTRUCTURA DE VECINDAD
HIBRIDA*****************************************************/
void HIBRIDO(arista(*permutacion)[N])
{
int opcion=0;
   opcion=rand()%4;

   switch(opcion){
 case 0:
             ParAleatorio(permutacion);
             break;
 case 1:
             DosParesAleatorios(permutacion);
             break;
 case 2:
             TresParesAleatorios(permutacion);
             break;
 case 3:
             CuatroParesAleatorios(permutacion);
    break;
  }
}
void ParAleatorio(arista (*permutacion)[N])
{
  int
Gv=0,Gv2=0,Gv3=0,Gv4=0,cont_vertices=0,E1=0,E2=0,D1=0,bandera=0,p=0,vert_encontrado=0,comparaD1,comp
araD2;
     llenar_en_ceros_vector(vector_ppar,matriz_ppar);
/*PRIMER  MOVIMIENTO*/
     for(;;)
       {
        Gv=rand()%(N-1)+1;
        cont_vertices=num_vertices(permutacion,Gv);
        if(cont_vertices!=0 )
                {
                 p=rand()%cont_vertices+1;
                 vert_encontrado=numvertice(permutacion,Gv,p);
                 E1=Eliminar_vert_ex_conectado(permutacion, Gv);
                 E2=Eliminar_vert_ex_conectado(permutacion, vert_encontrado);
                ConPAleatorio(permutacion,E2, E1);
```

```
                ConPAleatorio(permutacion,vert_encontrado, Gv);
                 Proceso_alamcenar(permutacion,E1,E2,Gv,vert_encontrado);
                 break;
                 }
          }
          Almacenar_D1_ppar(permutacion,vector_ppar);
         par_impar_vector(vector_ppar,matriz_ppar);
}
void DosParesAleatorios(arista (*permutacion)[N])
{
 int
Gv=0,Gv2=0,Gv3=0,Gv4=0,cont_vertices=0,E1=0,E2=0,D1=0,bandera=0,p=0,vert_encontrado=0,comparaD1,comp
araD2;
     llenar_en_ceros_vector(vector_ppar,matriz_ppar);
/*PRIMER  MOVIMIENTO*/
     for(;;)
       {
        Gv=rand()%(N-1)+1;
       cont_vertices=num_vertices(permutacion,Gv);
       if(cont_vertices!=0 )
                {
                   p=rand()%cont_vertices+1;
                 vert_encontrado=numvertice(permutacion,Gv,p);
                 E1=Eliminar_vert_ex_conectado(permutacion, Gv);
                 E2=Eliminar_vert_ex_conectado(permutacion, vert_encontrado);
                 ConPAleatorio(permutacion,E2, E1);
                 ConPAleatorio(permutacion,vert_encontrado, Gv);
                 Proceso_alamcenar(permutacion,E1,E2,Gv,vert_encontrado);
                 break;
          }
       }

             /*SEGUNDO MOVIMIENTO*/
      for(;;)
        {
                Gv2=rand()%(N-1)+1;
                while(Gv2==Gv)
                {
                Gv2=rand()%(N-1)+1;
                }
               cont_vertices=num_vertices(permutacion,Gv);
                if(cont_vertices!=0 )
                   {
                     p=rand()%cont_vertices+1;
                    vert_encontrado=numvertice(permutacion,Gv,p);
                     E1=Eliminar_vert_ex_conectado(permutacion, Gv);
                     E2=Eliminar_vert_ex_conectado(permutacion, vert_encontrado);
                    ConPAleatorio(permutacion,E2, E1);
                    ConPAleatorio(permutacion,vert_encontrado, Gv);
                     Proceso_alamcenar(permutacion,E1,E2,Gv,vert_encontrado);
                     break;
                    }
                 }
          Almacenar_D1_ppar(permutacion,vector_ppar);
          par_impar_vector(vector_ppar,matriz_ppar);
}



void TresParesAleatorios(arista (*permutacion)[N])
{
Int
Gv=0,Gv2=0,Gv3=0,Gv4=0,cont_vertices=0,E1=0,E2=0,D1=0,bandera=0,p=0,vert_encontrado=0,comparaD1,comp
araD2;
      llenar_en_ceros_vector(vector_ppar,matriz_ppar);
/*PRIMER  MOVIMIENTO*/
```

```
   for(;;)
       {
        Gv=rand()%(N-1)+1;
        cont_vertices=num_vertices(permutacion,Gv);
        if(cont_vertices!=0 )
               {
                p=rand()%cont_vertices+1;
                vert_encontrado=numvertice(permutacion,Gv,p);
                E1=Eliminar_vert_ex_conectado(permutacion, Gv);
                E2=Eliminar_vert_ex_conectado(permutacion, vert_encontrado);
                ConPAleatorio(permutacion,E2, E1);
                ConPAleatorio(permutacion,vert_encontrado, Gv);
                Proceso_alamcenar(permutacion,E1,E2,Gv,vert_encontrado);
                break;
               }
        }

/*SEGUNDO  MOVIMIENTO*/
for(;;){
         Gv3=rand()%(N-1)+1;
         while(Gv3==Gv|| Gv3==Gv2)
               {
               Gv3=rand()%(N-1)+1;
               }
               cont_vertices=num_vertices(permutacion,Gv);
               if(cont_vertices!=0 )
                  {
                   p=rand()%cont_vertices+1;
                   vert_encontrado=numvertice(permutacion,Gv,p);
                   E1=Eliminar_vert_ex_conectado(permutacion, Gv);
                   E2=Eliminar_vert_ex_conectado(permutacion, vert_encontrado);
                  ConPAleatorio(permutacion,E2, E1);
                  ConPAleatorio(permutacion,vert_encontrado, Gv);
                  Proceso_alamcenar(permutacion,E1,E2,Gv,vert_encontrado);
                 break;
                  }

/*TERCER  MOVIMIENTO*/
for(;;){
              Gv3=rand()%(N-1)+1;
               while(Gv3==Gv|| Gv3==Gv2)
               {
               Gv3=rand()%(N-1)+1;
               }
               cont_vertices=num_vertices(permutacion,Gv);
       if(cont_vertices!=0 )
                  {
                    p=rand()%cont_vertices+1;
                    vert_encontrado=numvertice(permutacion,Gv,p);
                    E1=Eliminar_vert_ex_conectado(permutacion, Gv);
                   E2=Eliminar_vert_ex_conectado(permutacion, vert_encontrado);
                   ConPAleatorio(permutacion,E2, E1);
                  ConPAleatorio(permutacion,vert_encontrado, Gv);
                  Proceso_alamcenar(permutacion,E1,E2,Gv,vert_encontrado);
                  break;
                  }
       Almacenar_D1_ppar(permutacion,vector_ppar);
        par_impar_vector(vector_ppar,matriz_ppar);
}



void CuatroParesAleatorios(arista (*permutacion)[N])
{
int
Gv=0,Gv2=0,Gv3=0,Gv4=0,cont_vertices=0,E1=0,E2=0,D1=0,bandera=0,p=0,vert_encontrado=0,comparaD1,comp
araD2;
```

```
      llenar_en_ceros_vector(vector_ppar,matriz_ppar);
/*PRIMER  MOVIMIENTO*/

for(;;)
  {
    Gv=rand()%(N-1)+1;
    cont_vertices=num_vertices(permutacion,Gv);
    if(cont_vertices!=0 )
                {
                  p=rand()%cont_vertices+1;
                 vert_encontrado=numvertice(permutacion,Gv,p);
                 E1=Eliminar_vert_ex_conectado(permutacion, Gv);
                 E2=Eliminar_vert_ex_conectado(permutacion, vert_encontrado);
                 ConPAleatorio(permutacion,E2, E1);
                 ConPAleatorio(permutacion,vert_encontrado, Gv);
                 Proceso_alamcenar(permutacion,E1,E2,Gv,vert_encontrado);
                 break;
        }
   }
/*SEGUNDO MOVIMIENTO*/
for(;;){

               Gv2=rand()%(N-1)+1;
                while(Gv2==Gv)
                {
                Gv2=rand()%(N-1)+1;
                }
               cont_vertices=num_vertices(permutacion,Gv);
               if(cont_vertices!=0 )
                   {
                     p=rand()%cont_vertices+1;
                    vert_encontrado=numvertice(permutacion,Gv,p);
                    E1=Eliminar_vert_ex_conectado(permutacion, Gv);
                     E2=Eliminar_vert_ex_conectado(permutacion, vert_encontrado);
                     ConPAleatorio(permutacion,E2, E1);
                     ConPAleatorio(permutacion,vert_encontrado, Gv);
                    Proceso_alamcenar(permutacion,E1,E2,Gv,vert_encontrado);
                     break;
                    }
    }
/*TERCER  MOVIMIENTO*/
for(;;){
               Gv3=rand()%(N-1)+1;
                while(Gv3==Gv|| Gv3==Gv2)
                {
                Gv3=rand()%(N-1)+1;
                }
                cont_vertices=num_vertices(permutacion,Gv);
        if(cont_vertices!=0 )
                   {
                     p=rand()%cont_vertices+1;
                     vert_encontrado=numvertice(permutacion,Gv,p);
                     E1=Eliminar_vert_ex_conectado(permutacion, Gv);
                    E2=Eliminar_vert_ex_conectado(permutacion, vert_encontrado);
                    ConPAleatorio(permutacion,E2, E1);
                   ConPAleatorio(permutacion,vert_encontrado, Gv);
                   Proceso_alamcenar(permutacion,E1,E2,Gv,vert_encontrado);
                   break;
                    }
}




/*Cuarto  MOVIMIENTO*/
```

```
for(;;){

                Gv4=rand()%(N-1)+1;
                while(Gv4==Gv|| Gv4==Gv2|| Gv4==Gv)
                {
                Gv4=rand()%(N-1)+1;
                }
                  //printf("\n C4Gv:%d\n",Gv);
        cont_vertices=num_vertices(permutacion,Gv);
        if(cont_vertices!=0 )
                    {
          p=rand()%cont_vertices+1;
                    vert_encontrado=numvertice(permutacion,Gv,p);
         // printf("\nC4Gv:%d\n",vert_encontrado);
                            E1=Eliminar_vert_ex_conectado(permutacion, Gv);
                            E2=Eliminar_vert_ex_conectado(permutacion, vert_encontrado);
                                    ConPAleatorio(permutacion,E2, E1);
                                    ConPAleatorio(permutacion,vert_encontrado, Gv);
                            Proceso_alamcenar(permutacion,E1,E2,Gv,vert_encontrado);
                            break;
                    }
}

Almacenar_D1_ppar(permutacion,vector_ppar);
par_impar_vector(vector_ppar,matriz_ppar);
}
```

Glosario de términos

Algoritmo Secuencial: Algoritmo programado de manera estructurada que al ser ejecutado utiliza un solo procesador.

Análisis de Sensibilidad: Evaluación del comportamiento de las variables críticas de un problema con la finalidad de establecer un rango numérico, dentro del cuál, 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.

Adyacencia: dos vértices son adyacentes si son extremos de una misma arista.

Benchmark: Palabra del inglés utilizada para nombrar a los problemas de prueba utilizados por diversos investigadores, para medir el rendimiento de un sistema o algoritmo para un problema dado. Para el caso del Problema de Máquinas en Paralelo no Relacionadas existen en la literatura un conjunto de benchmarks utilizados por diversos autores,los cuales involucran problemas de prueba de 20, 40 , 60, 80, 100 y 120 trabajos, a calendarizar en 2, 4, 6, 8, 10 y 12 máquinas respectivamente

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.

Complejidad Temporal: Es el número de pasos necesarios (tiempo) para obtener una solución a una instancia de un problema dado.

Costo: En el caso del problema de UPMP, son las unidades de tiempo requeridas para el procesamiento de uno o varios trabajos.

Camino: es un recorrido en el que no se repiten vértices ni aristas, siendo un recorrido una sucesión de vértices y aristas.

Camino alternado: dado un emparejamiento M, un camino M-alternado es un camino donde se alternan aristas de M y aristas que no están en M.

Camino de aumento: es un camino alternado que tiene como extremos vértices libres.

Estructura de datos: Una estructura es un tipo de dato compuesto que permite almacenar un conjunto de datos de diferente tipo. Los datos que contiene una estructura pueden ser de tipo simple (caracteres, números enteros o de coma flotante etc.) o a su vez de tipo compuesto (vectores, estructuras, listas, etc.).

Estructura de Vecindad: Tipo de movimiento utilizado para explorar el espacio de soluciones de un problema dado.

Emparejamiento: es un subconjunto de aristas $M \subseteq A$, tal que dos aristas de M no tengan un extremo común.

Emparejamiento perfecto: un emparejamiento es perfecto si todos los vértices de G son extremo de alguna arista de M, es decir, todos los vértices de G están saturados.

Emparejamiento maximal: es un emparejamiento que no puede ser ampliado agregando una arista.

Emparejamiento máximo: es un emparejamiento que tiene el mayor número posible de aristas.

Algoritmo: Un algoritmo se puede definir como una secuencia de instrucciones que representan un modelo de solución para determinado tipo de problemas .O bien como un conjunto de instrucciones que realizadas en orden conducen a obtener la solución de un problema.Para realizar un programa es conveniente el diseño o definición previa del algoritmo. El diseño del algoritmo requiere de creatividad y conocimientos profundos de la técnica de programación [Oliedo Regino,2004].

Grafo: es un par (V,A) donde V es un conjunto finito no vacío y desordenado (vértices) y A es un subconjunto finito de pares no ordenados (aristas). El orden de un grafo es el número de vértices que lo compone.

Grado de un vértice: número de aristas de las que el vértice es extremo.

Grafo simple: es un grafo donde A es un conjunto en el cual no debe haber aristas repetidas ni bucles.

Grafo bipartido: es un grafo simple donde se puede encontrar una partición de V tal que $V=X \cup Y$ con $X \cap Y$ = vacío tal que toda arista $uv \in A$ une vértices de distinta parte, es decir, une un vértice de la capa X con un vértice de la capa Y. Un grafo es bipartido si y solo si no contiene ciclos impares.

Grafo regular: es un grafo simple donde todos sus vértices tienen el mismo grado.

Grafo conexo: un grafo es conexo si para cada par de vértices u y v existe un camino de u a v. Si al quitar al grafo una arista se obtienen más componentes conexas que en el grafo original, la arista se designa puente.

Grafo completo: es un grafo simple en el que todo par de vértices está unido por una arista, todos los vértices están unidos entre sí.

Grafo bipartido completo: se designa por por $K_{r,s}$ donde $|X| = r$ e $|Y| = s$, y hay una arista que conecta cada vértice de X con cada vértice de Y.

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

Incidencia: un vértice y una arista son incidentes si el vértice es extremo de la arista.

Metaheurísticas: Las cuales son una clase de métodos aproximados que están diseñados para resolver problemas complejos de optimización combinatoria, en los que los heurísticos clásicos no son efectivos. Las meta heurísticos proporcionan un marco general para crear nuevos algoritmos híbridos combinados diferentes conceptos derivados de la inteligencia

artificial, la evolución biológica y los mecanismos estadísticos [Osman y Nelly,1996].

Struct: Es una palabra reservada de C que indica que los elementos que vienen agrupados a continuación entre llaves componen una estructura de datos. Se identifica el tipo de dato que se describe y del cual se podrán declarar variables. Se especifica entre corchetes para indicar su opcionalidad.

Modelo de Programación entera binaria: El formato del modelo de programación entera binaria requiere tener una función objetivo lineal, ya sea a minimizar o maximizar. La característica de este modelo, es que su conjunto de variables solo puede tomar valores binarios, es decir, 0s y 1s.

Óptimo Global: Es la mejor solución de un espacio de soluciones f(x).

Óptimo Local: Representa la mejor solución de f(x) en un entorno x .

Sintonización: Es la proporción adecuada en cuanto a los valores obtenidos mediante el análisis de sensibilidad aplicado a los parámetros de control, tomando en cuenta el problema y el método de optimización utilizado, de modo que el algoritmo muestre una mejora tanto en eficiencia como en eficacia.

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).

Subgrafo: sea G=(V,A) un grafo, un subgrafo de G es H=(V',A') tal que $V' \subseteq V$ y $A' \subseteq A$ y la asignación de extremos a las aristas en H es la misma que en G.