FACULTAD DE CIENCIAS QUÍMICAS E INGENIERÍA

CENTRO DE INVESTIGACIÓN EN INGENIERÍA
Y CIENCIAS APLICADAS

SOLUCIÓN AL PROBLEMA DEL ÁRBOL DE EXPANSIÓN MÍNIMA APLICANDO RECOCIDO SIMULADO CON BÚSQUEDA TABÚ

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. BEATRIZ MARTÍNEZ BAHENA

ASESOR: DR. MARCO ANTONIO CRUZ CHÁVEZ

COASESOR: DRA. OCOTLÁN DÍAZ PARRA

CUERNAVACA, MOR. JULIO 2011

Resumen

En este trabajo de investigación se desarrolló un algoritmo de Recocido Simulado con Lista Tabú para tratar instancias del problema del Árbol de Expansión Mínima. Se generó una estructura de vecindad híbrida, con la finalidad de mejorar el desempeño del algoritmo. Se utilizó una metodología de sintonización, la cual permitió realizar el análisis de sensibilidad de los parámetros de control del algoritmo Recocido Simulado, lo que permite trabajar al algoritmo con mejor desempeño tanto en eficiencia como en eficacia. El análisis de resultados se llevó a cabo en base a pruebas de eficiencia y eficacia de las estructuras de vecindad propuestas, ejecutándose éstas en un algoritmo de búsqueda local iterada. Los resultados del análisis mostraron que la estructura híbrida es la que trabaja con mejor eficacia y en eficiencia fue competitiva, en base a estos resultados se implementó la estructura híbrida en el algoritmo de Recocido Simulado. También se aplicó a esta metaheurística una lista tabú, lo que permitió mejorar aún más la eficacia del mismo al incrementar el tamaño de la lista tabú y de igual manera se generó un aumento en la eficiencia. Las pruebas experimentales se realizaron tomando cuatro instancias del problema del árbol de expansión mínima generadas de forma aleatoria. Las pruebas experimentales del algoritmo fueron realizadas en equipo del laboratorio de Optimización del CIICAp.

Abstract

This work developed a Simulated Annealing algorithm with Tabu list for use in instances of the Minimum Spanning Tree. It generated a hybrid neighborhood structure, with the goal of improving the algorithm's performance. A tuning method was employed, which allowed a sensibility analysis of the control parameters of the Simulated Annealing algorithm, thereby improving both the efficiency and efficacy of the algorithm. The results analysis was based on tests of the efficiency and efficacy of the proposed neighborhood's structures, by executing them in an iterated local search algorithm. The results of the analysis showed that the hybrid structure was the one that performed with greatest efficiency and with competitive efficacy. Based on these results, the hybrid structure was implemented in the Simulated Annealing algorithm. In addition, a tabu list was applied to this metaheuristic, which enabled further improvement of the efficacy upon increasing the size of the tabu list. The experimental tests were performed using four randomly generated instances of the minimum spinning tree problem. The tests were carried out using equipment of the optimization laboratory of CIICAp.

Agradecimientos

A Dios porque me ha permitido despertar cada mañana, porque es la persona más importante la que decide si te vas o continúas en esta vida. Gracias a Dios por darme la oportunidad de cumplir con mis objetivos y metas planteadas en mi vida.

A CONACYT por brindarme el apoyo económico para ser posible la realización de mis estudios de maestría.

A mi asesor Dr. Marco A. Cruz Chávez por el tiempo dedicado, por su paciencia y apoyo incondicional para dirigir este trabajo de investigación. Gracias a sus conocimientos y sabios comentarios dados con la finalidad de obtener el mejor resultado.

A los integrantes de mi comité tutoral y revisores de tesis: Dr. Marco Antonio Cruz Chávez, Dra. Margarita Tecpoyotl Torres, Dr. Martín Heriberto Cruz Rosales, Dra. Ocotlán Díaz Parra, M.C. Carmen Peralta, por sus observaciones y comentarios para la realización de este trabajo de investigación.

Dedicatorias

A mis padres Víctor y Francisca porque ellos fueron la parte esencial. Gracias a su gran esfuerzo, apoyo y sacrificio para lograr que se cumplieran mis metas y ser lo que soy ahora.

A mis sobrinos Yatziry Lizzet, Leslie Evelyn y Alex Jhovany porque gracias a sus lindas sonrisas, dulces besos y pequeños abrazos, me impulsaron a seguir adelante para que algún día pueda ser un gran apoyo en su vida.

A mi familia porque gracias a ellos pude lograr mi formación profesional y personal, con su sacrificio, apoyo y dedicación pude concluir una etapa más en mi vida y me ayudaron a seguir adelante.

A mis compañeros y amigos Jazmín, Yessica, Maricruz, Jorge, Luis, Abraham, Eduardo, Christian Eduardo y Christian con los que he compartido grandes momentos, por su gran apoyo incondicional.

Nomenclatura

NOMENCLATURA GENERAL

MST Siglas en inglés para el árbol de expansión mínima (Mínimum Spanning Tree).

SA Siglas en inglés para el algoritmo de Recocido Simulado (Simulated Annealing).

TS Siglas en inglés para Búsqueda Tabú (Tabu Search).

ILS Siglas en inglés para el algoritmo de Búsqueda Local Iterada (Iterated Local Search).

Parámetros de Control de Recocido Simulado

T_0 Valor inicial del parámetro de control

α Coeficiente del parámetro de control

T_f Valor final del parámetro de control

L_k Longitud de la cadena de Markov

L_T Tamaño de la lista tabú

Tabla de Contenido

Lista de Figuras

Lista de Tablas

Capítulo 1

Introducción

El problema del árbol de expansión mínima MST (por sus siglas en inglés, Minimum Spanning Tree) es un problema de optimización combinatoria y es uno de los problemas más importantes en el área de computación distribuida y redes de comunicación [Maleq, 2007]. Fue formulado por Otakar Borukva en 1926, quien lo planteó para resolver el problema de hallar la forma más económica de distribuir energía eléctrica en el sur de Moravia. La formulación de este problema ha sido útil para realizar muchas investigaciones en diversos campos como sistemas eléctricos e hidráulicos, transporte, diseño de redes de telecomunicaciones, sistemas computacionales, sistemas telefónicos y en otros problemas de investigación de operaciones [Hillier y Lieberman, 2010].

Para resolver el problema MST se han utilizado varios algoritmos exactos y de aproximación. El problema del árbol de expansión mínima dentro de la teoría de la complejidad se clasifica como un problema P, este tipo de problema es un conjunto de problemas de decisión que puede ser resulto por un algoritmo determinístico en tiempo polinomial [Papadimitriou y Steiglitz, 1998] [Talbi, 2009].Para resolver el problema MST existen algoritmos exactos de tiempo polinomial, como los desarrollados por [Kruskal, 1956]; [Prim, 1957]; [Dijkstra, 1959] y [Sollin, 1965]. Sin embargo existen diferentes extensiones del problema del MST tal como dc-MST por [Narula y

Ho, 1980], p-MST por [Bertsimas, 1990], s-MST por Ishii y sus colaboradores [Ishii et al., 1981], q-MST por [Assad y Xu, 1991], g-MST por Myung y otros [Myung et al., 1995] y MStT por [Gilbert y Pollak, 1968] y más. Estos problemas son clasificados como problemas NP-duros en los cuales no existen soluciones en tiempo polinomial. Por su complejidad se hace uso de metaheurísticas. El método típico llamado Greedy es de los más utilizados para tratar variantes del problema MST [Papadimitriou y Steiglitz, 1998].

En este trabajo de investigación el problema a tratar es el MST de tipo P. Este problema se representa por medio de un grafo no dirigido, para obtener una solución particular por medio del desarrollo de un algoritmo de Recocido Simulado para el cuál se generó una estructura de vecindad híbrida.

1.1 Estado del Arte

Las metaheurísticas utilizadas para el problema del árbol de expansión mínima y sus extensiones se caracterizan por búsquedas mediante vecindades, una vecindad es un conjunto de soluciones que pueden ser alcanzadas desde una solución dada por medio de un movimiento. Las *metaheurísticas* 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 metaheurísticas proporcionan un marco general para crear nuevos algoritmos híbridos combinando diferentes conceptos derivados de la inteligencia artificial, la

evolución biológica y los mecanismos estadísticos [Osman y Nelly, 1996]. Las técnicas metaheurísticas son capaces de proporcionar muy buenas soluciones (no necesariamente la óptima pero si aproximada) en tiempo y en recursos razonables.

En la literatura existen varios métodos que permiten obtener buenas soluciones para el problema del árbol de expansión mínima y sus extensiones anteriormente mencionadas.

Pop y sus colaboradores [Pop *et al*., 2007] desarrollaron un método combinando, un algoritmo de Recocido Simulado con un algoritmo local voraz (greedy) para resolver el problema del árbol de expansión mínima generalizado (GMST, por sus siglas en inglés Generated Minimum Spanning Tree). La heurística que propone soluciones encontradas que fueron óptimas para grafos con hasta 280 nodos y para instancias más grandes proporciona soluciones sub-optimas de mejora en un tiempo razonable en ejecución. Para este mismo problema [Hu *et al*., 2005] proponen una variable de búsqueda por vecindad (VNS, por sus siglas en inglés Variable Neighborhood Search), combinando dos tipos de estructura de vecindad, estructura de vecindad de intercambio de un nodo (NEN, por sus siglas en inglés Node Exchange Neighborhood) y una estructura de vecindad de intercambio de un vértice global (GEEN, por sus siglas en inglés Global Edge Exchange Neighborhood). Los resultados obtenidos con este método superan a otras

metaheurísticas, en particular para instancias con gran número de nodos por clúster.

Yoon-Teck y sus colaboradores [Yoon-Teck *et al*., 2008] diseñaron dos métodos de Optimización de Colonia de Hormigas (ACO, por sus siglas en inglés Ant Colony Optimization) y sus variantes mejoradas en el problema del árbol de expansión mínima grado-restricciones (d-MST, por sus siglas en inglés Degree-Constrained Minimum Spanning Tree), el primer método es llamado p-ACO (p se refiere al algoritmo Prim), este método utiliza los vértices para la construcción de un grafo como componente de solución, y está determinado por el algoritmo de Prim para la construcción del MST. El segundo método, conocido como k-ACO (k se refiere al algoritmo Kruskal) utiliza las aristas de un grafo como componentes de la solución, y está determinada por el algoritmo de Kruskal para el problema MST. Los resultados muestran que k-ACO se comporta mejor que p-ACO y es competitivo a comparación con otras heurísticas. Otro método de optimización utilizado para resolver el CMST (por sus siglas en inglés Capacitated Minimum Spanning Tree) es un algoritmo ACO híbrido, el cuál combina la construcción de una solución desarrollada por el algoritmo de Ahorro para el VRP (por sus siglas en inglés Vehicle Routing Problem) capacitado, con una búsqueda local que incorpora un algoritmo exacto (algoritmo Prim) para encontrar Árboles de Expansión Mínima. Obteniendo buenos resultados en cuanto a eficacia y eficiencia [Reimann y Laumanns, 2009].

1.1.1 Métodos de Optimización Combinatoria

Los métodos de optimización combinatoria consisten en la solución algorítmica de problemas donde se busca maximizar o minimizar el valor de la función objetivo. La optimización combinatoria es una rama de la optimización en matemáticas aplicadas y en ciencias computacionales, relacionada a la investigación de operaciones, teoría de algoritmos y teoría de la complejidad computacional [Cook et al., 1997]. Los algoritmos de optimización combinatoria resuelven instancias de problemas que son difíciles en general, explorando el espacio de soluciones. Varios problemas de Optimización Combinatoria pueden ser resueltos en tiempo polinomial utilizando los métodos exactos y teoría de la programación lineal.

Los métodos de optimización tratan de encontrar la mejor solución evaluando la función objetivo del problema para encontrar su valor máximo o mínimo. Generalmente, estos métodos utilizan métodos de búsqueda local en su procedimiento. Los métodos basados en Recocido Simulado [Cruz et al., 2006], Algoritmos Meméticos [Cruz et al., 2008], Búsqueda Tabú [Glover, 1989]; [Glover, 1990] y Colonia de Hormigas [Martínez, 2010] son métodos de optimización que involucran búsquedas locales, los cuales tratan de determinar la mejor solución de acuerdo con los criterios definidos de la función objetivo. Estos métodos se aplican cuando no es posible encontrar una solución óptima por un método exacto debido a la complejidad del problema. La optimización combinatoria se clasifica en métodos exactos y métodos aproximados (o heurísticos). Los métodos exactos son aquellos que

proporcionan una solución óptima de cualquier problema, estos métodos no son muy utilizados para encontrar soluciones a problemas NP-duros, debido a que el tiempo crece exponencialmente con el tamaño del problema. Los métodos heurísticos encuentran resultados óptimos en tiempo razonables. Una *heurística* es un método bien estructurado, desarrollado en base a la experiencia que permite obtener soluciones aproximadas de un problema sin garantizar la optimalidad [Wetzel, 1983]. A continuación en la figura 1-1 se muestra la clasificación de los métodos de optimización.

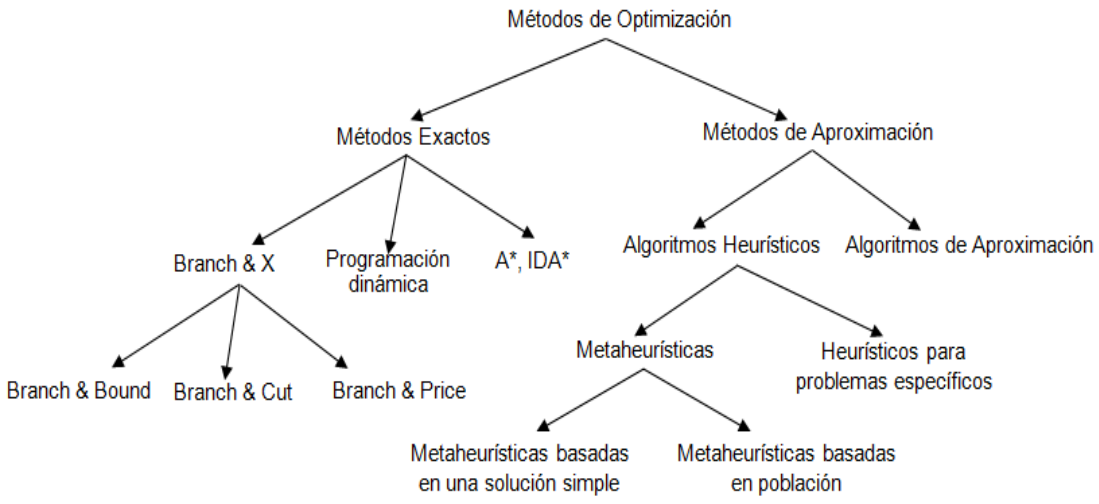


Figura 1-1 Clasificación de Métodos de Optimización Combinatoria traducido de [Talbi, 2009]

En la figura 1-1se presenta la clasificación de los métodos de optimización combinatoria se utiliza para saber porque métodos se puede resolver un problema ya sea por un método exacto o por un método de aproximación para resolver el problema del árbol de expansión mínima existen métodos exactos en tiempo polinomial, pero en este trabajo de investigación se propuso utilizar un método de aproximación. Este método se clasifica en algoritmos heurísticos y de aproximación, los algoritmos heurísticos se clasifican metaheurísticas y heurísticos para problemas específicos, las metaheurísticas se dividen en dos clases Metaheurísticas basadas en una solución simple como Recocido Simulado, Búsqueda Tabú y Colonia de Hormigas y en metaheurísticas basadas en población como algoritmos Genéticos. En esta tesis se propuso una Metaheurística basada en una solución simple (Recocido Simulado) para resolver el problema del árbol de expansión mínima.

1.2 Objetivo de la Investigación

Objetivo General

1.- Aplicar el algoritmo de Recocido Simulado al Problema de Árbol de Expansión Mínima.

Objetivo Particular

1.- Mejorar la eficiencia y eficacia de Recocido Simulado mediante la implementación de una estructura de vecindad híbrida y una lista tabú.

1.3 Alcance de la Investigación

1. Implementación de un algoritmo de Recocido Simulado con Búsqueda Tabú para obtener la solución al problema del árbol de expansión mínima (MST).

2. Generación e Implementación de una estructura de vecindad híbrida para mejorar el funcionamiento del algoritmo.

3. Sintonización de los parámetros de control del algoritmo de Recocido Simulado, para mejorar el tiempo de convergencia y la eficacia de las soluciones de dicho algoritmo.

4. Las instancias de pruebas utilizadas para el algoritmo de Recocido Simulado generadas de forma aleatoria, sirven para realizar una comparación en cuanto a eficiencia y eficacia del algoritmo. Para generar dichas pruebas, se desarrolló un programa en visual c++.

1.4 Contribución de la Tesis

En esta tesis se desarrolló un algoritmo de Recocido Simulado con Búsqueda Tabú y la generación e implementación de una estructura de vecindad híbrida que prueba ser eficiente y eficaz. Este algoritmo de Recocido Simulado se aplica al problema del Árbol de Expansión Mínima.

Este trabajo se llevó acabo en base a una investigación sobre estructuras de vecindad híbridas, en donde se encontró que las estructuras híbridas pueden tener un mejor desempeño que las estructuras simples. Pero esto depende del problema a tratar debido a que en algunos problemas no resulta ser cierta esta afirmación por ejemplo para el problema de Máquinas en Paralelo no Relacionadas la estructura híbrida no funciono. Por lo que en esta tesis se evalúa el desempeño de la estructura de vecindad híbrida para el problema del Árbol de Expansión Mínima y se concluye que para este problema el desempeño de la estructura híbrida es mejor que las estructuras simples con las que se realizó la comparación la cual se presenta en el capítulo 4 sección 4.2.

1.5 Organización de la Tesis

En el capítulo 1 se da una introducción del problema a tratar; se da una explicación del estado del arte y se presentan los objetivos, alcances y la

contribución de esta investigación y por último la organización general de la tesis, en el capítulo 2 se expone el problema del árbol de expansión mínima, una descripción conceptual y representación del problema, por último la formulación matemática del mismo.

En el capítulo 3 se da a conocer el algoritmo de Recocido Simulado, se presenta la descripción del algoritmo para encontrar la solución inicial, se explica la estructura de vecindad híbrida propuesta, la implementación de la lista tabú, se presenta un esquema generalizado del algoritmo de Recocido Simulado para mejorar las soluciones obtenidas para el problema del Árbol de Expansion Minima, la metodología de sintonización para la realización del análisis de sensibilidad, el análisis de la complejidad del algoritmo, es decir la función temporal.Por último se presentan las instancias de prueba generadas de forma aleatoria.

En el capítulo 4 se muestran de los resultados experimentales del algoritmo de Recocido Simulado, se presenta la descripción del equipo utilizado para realizar las pruebas. Se presentan las pruebas de las estructuras de vecindad propuesta para el algoritmo de Recocido Simulado, el análisis de sensibilidad, las pruebas de Recocido Simulado y por último se muestra el análisis de eficacia y eficiencia del algoritmo. En el capítulo 5 se dan las conclusiones finales del trabajo de investigación, así como los trabajos futuros.

Capítulo 2

Árbol de Expansión Mínima

En este capítulo se da una explicación del problema del árbol de expansión mínima, se muestra la representación del problema del árbol de expansión mínima MST mediante un grafo así como la formulación matemática de dicho problema.

El problema del árbol de expansión mínima es uno de los problemas más utilizados y conocidos en optimización combinatoria. Dentro de la teoría de la complejidad el problema se clasifica como P [Papadimitriou y Steiglitz, 1998]; [Talbi, 2009], para el cual existen métodos exactos que lo resuelve en tiempo polinomial.

El objetivo de este problema es diseñar una red de trabajo entre los vértices dados, los cuales tengan posibles conexiones mediante aristas entre ellos donde cada una de estas aristas cuenta con un costo, el cual puede

representar tiempo, dinero, distancia u otra medida. Otro punto importante es que todos los vértices deben estar conectados, sin que existan ciclos, es decir, que no existan caminos que inicien y terminen en el mismo vértice, y que la suma total de costo de todas las aristas sea mínima.

2.1 Descripción Conceptual y representación del Problema del Árbol de Expansión Mínima

El problema del Árbol de Expansión Mínima puede ser representado de forma general mediante un grafo *G* como un conjunto de vértices *V* que necesitan ser conectado con un conjunto de aristas *E*, donde cada arista tiene un costo correspondiente W_e.

Características propias del árbol a generar en el grafo *G*:

1. Todos los vértices deben estar conectados.
2. No deben existir ciclos, es decir, el árbol debe contener *n-1* aristas.
3. La suma total de costos de todas las aristas debe ser mínima. Esta es una de varias funciones objetivo del problema.

El problema del árbol de expansión mínima en la literatura es representado mediante un grafo, el cuál es definido como un grafo no dirigido, conexo y ponderado G= (V, E), donde V= $\{v_1, v_2 \ldots . v_n\}$ es un conjunto finito de vértices y E = $\{e_{ij} \mid e_{ij} = (v_i, v_j), v_i, v_j \in V\}$ es un conjunto finito de aristas. Se dice que grafo es ponderado si en cada arista tiene asociado un número real positivo

denotado con $W = \{w_{ij} \mid w_{ij} = w(v_i, v_j), w_{ij} > 0, v_i, v_j \in V\}$ representando distancia, costo u otra medida por lo cual es un grafo ponderado. El grafo es no dirigido debido a que las aristas no tienen una dirección, un grafo es conexo si todos los vértices están conectados. El grado de un vértice es un número de aristas conectadas a este. Para el problema tratado en esta investigación, el grado de cada vértice en el grafo utilizado para encontrar su árbol de expansión mínima varía en un rango de *1* a *n-1*. En la figura 2-1 se muestra un ejemplo de un grafo no dirigido, conexo y ponderado.

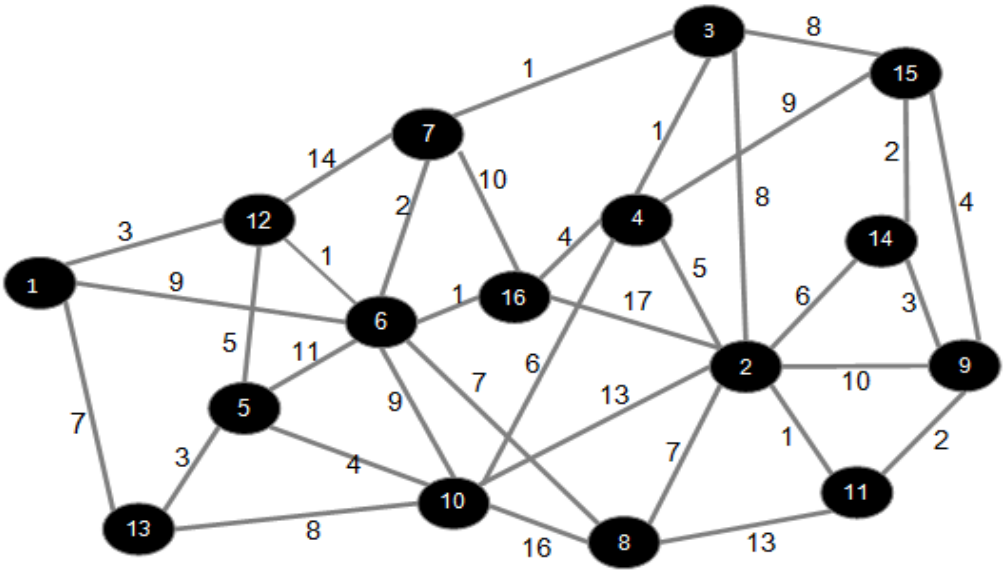


Figura 2-1 Grafo no dirigido, conexo con 35 vértices, 16 aristas y con costo aleatorio en cada arista.

Se puede decir en términos de grafos, que para que un árbol de expansión sea mínimo debe de cumplir ciertas condiciones:

1. Ser un subgrafo de *G* conectado con *n*-1 aristas, donde *n* es el número total de vértices.
2. Ser un subgrafo de *G* sin ciclos con *n-1* aristas.
3. Ser un subgrafo de *G* donde todos los vértices estén conectados.
4. La suma total de los costos de todas las aristas asociadas al subgrafo sea la mínima.

En la figura 2-1 se presenta un grafo no dirigido donde su árbol de expansión mínima con las características anteriormente mencionadas se presenta en la figura 2-2.

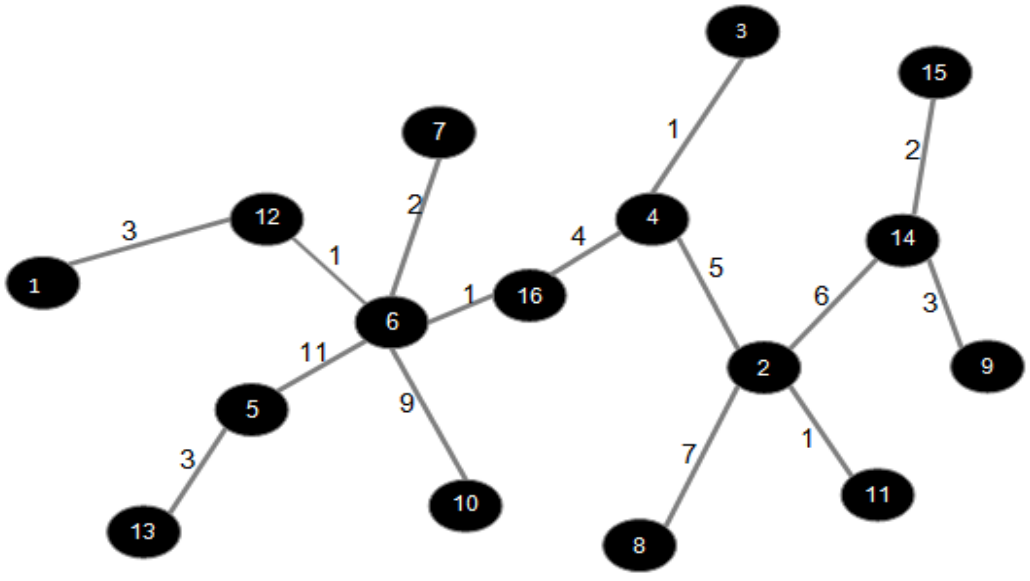


Figura 2-2 Ejemplo Árbol de Expansión Mínima

Cabe mencionar que un vértice hoja tiene solo una arista conectada. Así que el grado del vértice hoja en un árbol es uno y de los otros vértices son más de uno como se muestra en la figura 2-2.

2.2 Modelo Matemático del Problema del Árbol de Expansión Mínima

Dado un grafo conectado, no dirigido *G = (V , E),* con un costo W_e para todas los aristas en *E*, encontrar un árbol de expansión $G_T = (V_T, E_T)$ de un costo total mínimo.

Las variables de decisión x_e para la formulación de programación entera del MST son:

$$x_e = \begin{cases} 1, & Si \quad la \quad arista \quad e \in E_T \\ 0, & En \quad caso \quad contario \end{cases}$$

La formulación matemática que define al problema del MST y que se resuelve en este trabajo de investigación, se presenta a continuación.

$$\min c = \sum_{e \in E} w_e x_e \qquad (1)$$

$$sujeto\ \ a:$$

$$\sum_{e \in E} x_e = n - 1 \qquad (2)$$

$$\sum_{e \in (S,S)} x_e \leq |S| - 1 \qquad \forall S \subseteq V \qquad (3)$$

$$x_e \in \{0,1\} \qquad \forall e \in E \qquad (4)$$

La ecuación (1) es la función objetivo del problema para minimizar la suma total del costo de todas las aristas que conforman al árbol de expansión mínima. El conjunto de restricciones en (2), denota que todos los vértices deben estar conectados, por lo cual la suma total de aristas sea igual a *n-1*, donde *n* representa el número total de vértices, por lo cual también se cumple la restricción de que no existan ciclos [Hochbaum y Moreno-Centeno, 2008].

El conjunto de restricciones (3) se cumple la restricción de que las aristas de E_T no puedan formar ciclos, donde *(S, S)* denota todas las aristas que van desde un vértice en el conjunto *S* a otro vértice en el conjunto *S*.

El conjunto de restricciones en (4), ésta indica si una arista conecta a un par de vértices, donde x_e de forma que si x_e=1, la arista conecta un vértice *i* con un vértice *j*, de lo contrario x_e=0.

Capítulo 3

Algoritmo de Recocido Simulado

3.1 Introducción

Este capítulo describe el algoritmo de Recocido Simulado o SA (por sus siglas en inglés, simulated annealing) propuesto en este trabajo de investigación, y se explica a detalle el desarrollo e implementación de la estructura de vecindad híbrida y una lista tabú aplicada a SA para resolver el problema del árbol de expansión mínima. También se presenta la metodología de sintonización de los parámetros de entrada al algoritmo. Por último se presenta la función temporal y la complejidad del algoritmo.

3.2 Algoritmo de Recocido Simulado

El algoritmo de Recocido Simulado (SA) es una metaheurística de búsqueda aleatoria utilizada en la solución de problemas de optimización combinatoria, la cual fue propuesta por [Kirkpatrick et al., 1983] para el diseño de circuitos VLSI, y Cerny (1985) para el problema del agente viajero TSP, ellos mostraron que este proceso podría tener una asociación de los conceptos del proceso original de simulación, con elementos de problemas de optimización combinatoria, y por lo mismo podría ser aplicado a estos problemas. Este algoritmo se basa en la analogía entre el proceso de recocido de sólidos y los problemas de optimización combinatoria. La analogía del proceso de recocido, es cuando un material se somete a un calentamiento a temperatura muy alta llegando al punto de fusión y después es enfriado gradualmente, sus moléculas se acomodan de tal forma que la energía potencial de la configuración de las moléculas es mínima.

Métodos de búsqueda local como Recocido Simulado, son procedimientos de búsqueda local iterada que parten con una solución inicial del problema y la mejoran progresivamente. Muchos investigadores han aplicado este método para problemas de optimización combinatoria y han obtenido buenos resultados [Cruz, 2005]; [Cruz el at, 2006]; [Torres y Velez, 2007]; [Martínez, 2008]; [Cruz y Juárez, 2010].

El procedimiento del algoritmo de Recocido Simulado es partir de una solución inicial dada, de una función objetivo que depende del problema y de un parámetro de control conocido como temperatura, la cual es una función de decremento del número de iteraciones. En cada iteración se genera una nueva solución del vecindario denominado *N(x).* La nueva solución es aceptada como la mejor solución hasta el momento si $f(x') \leq f(x)$ y continua la búsqueda, de lo contrario es rechazada o aceptada en base a la probabilidad de aceptación de boltzmann, la cual involucra el parámetro de control *T* y la diferencia de los valores de la función de costos $(f(x') - f(x))$. Si la temperatura tiene valores muy altos hay más probabilidad de aceptar soluciones malas como buenas, esto para evitar óptimos locales, pero en cuanto *T* decrementa solo se aceptan soluciones buenas [Martínez, 2008].

El algoritmo de Recocido Simulado para su procedimiento requiere de una solución inicial dada. A continuación se explica a detalle el algoritmo propuesto para obtener una solución al problema del árbol de expansión mínima.

3.3 Algoritmo para encontrar una solución inicial

Como el algoritmo de Recocido Simulado requiere de una solución inicial dada. A continuación se describen los pasos a seguir para obtener dicha solución

1. Generar una representación simbólica que representa al problema, en la figura 3-1 se presenta un grafo el cual fue generado de forma aleatoria con un conjunto de aristas y un conjunto vértices y un costo en cada arista, a partir de ese grafo encontrar un árbol de expansión mínima. En la tabla 3-1 se muestra la matriz de incidencia que representa el grafo.

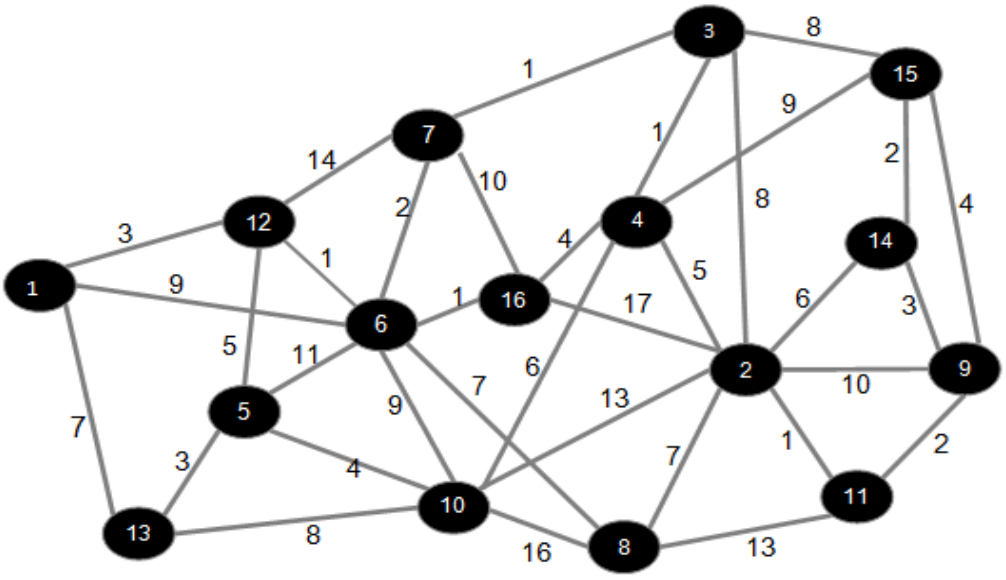


Figura 3-1 Representación de un grafo no dirigido, conexo y ponderado para el MST

Tabla 3-1 Representación simbólica para el problema MST

Vértices	V	P	V	P	V	P	V	P	V	P	V	P	V	P	V	P
1	6	9	12	3	13	7										
2	3	8	4	5	8	7	9	10	10	13	11	1	14	6	16	17
3	2	8	4	1	7	1	15	8								
4	2	5	3	1	10	6	15	9	16	4						
5	6	11	10	4	12	5	13	3								
6	1	9	5	11	7	2	8	7	10	9	12	1	16	1		
7	3	1	6	2	12	14	16	10								
8	2	7	6	7	10	16	11	13								
9	2	10	11	2	14	3	15	4								
10	2	13	4	6	5	4	6	9	8	16	13	8				
11	2	1	8	13	9	2										
12	1	3	5	5	6	1	7	14								
13	1	7	5	3	10	8										
14	2	6	9	3	15	2										
15	3	8	4	9	9	4	14	2								
16	2	17	4	4	6	1	7	10								

En la tabla 3-1 se muestra un ejemplo de la instancia de prueba generada de forma aleatoria, esta instancia contiene una columna V misma que

corresponde al vértice y una columna P representa el costo en un intervalo de 1 a 100 requerido para conectar el vértice *i* con el vértice *j*. Los renglones representan el número total de vértices en el grafo.

1. Generar solución factible de forma aleatoria

Para generar una solución aleatoria se generó un programa en visual C++. A continuación se presenta el diagrama de flujo del procedimiento del algoritmo para generar la solución aleatoria.

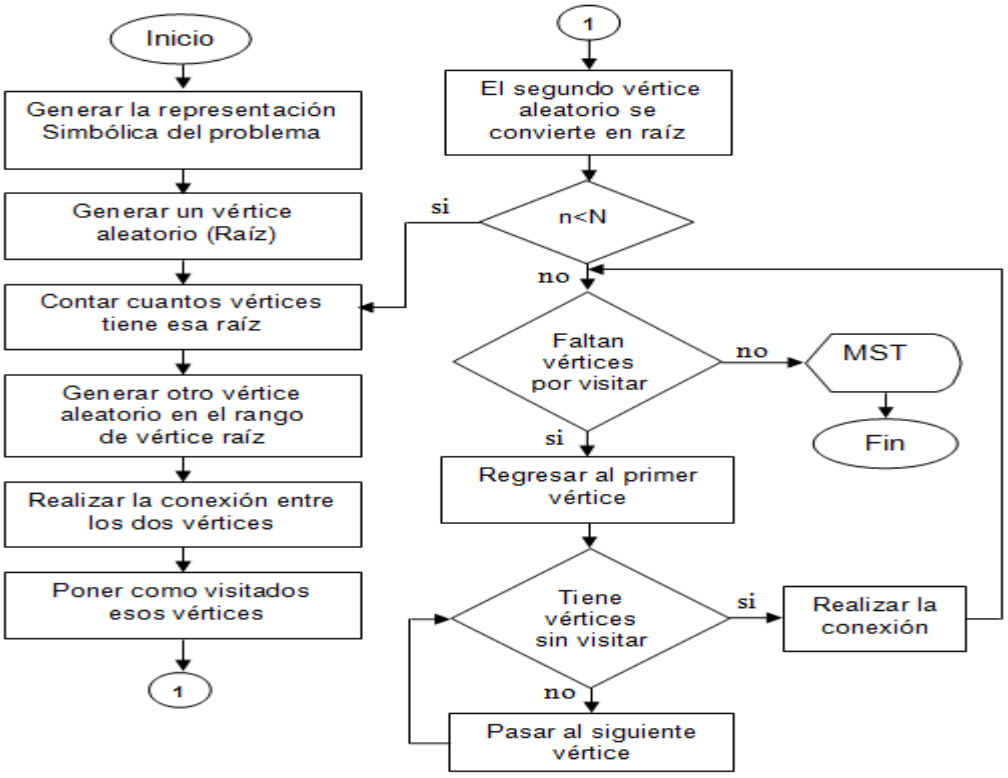


Figura 3-2 Diagrama de flujo para encontrar una solución inicial

Una vez teniendo la solución inicial factible para aplicarla al algoritmo de Recocido Simulado también se requiere de una estructura de vecindad para su funcionamiento, en este trabajo de investigación se propone una estructura de vecindad híbrida con el objetivo de mejorar la calidad de las soluciones encontradas. A continuación se explica a detalle la estructura de vecindad utilizada en el algoritmo de Recocido Simulado.

3.4 Estructura de Vecindad Híbrida

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

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

acuerdo a esto, una estructura de vecindad se define como una función *N(s)* presentada en la ecuación (5).

$$N(s) = \left\{ s' \in S : s \xrightarrow{\sigma} s' \right\} \qquad (5)$$

Una función de vecindad *N(s)* específica para cada solución $s \in S$ es un conjunto $N(s) \in S$, el cual es llamado vecindario de *s*, esto indica que cada solución *s'* es un vecino de *s* si $s' \in N(s)$. S representa el conjunto total de soluciones posibles de una instancia del problema.

Para mejorar una solución, es necesario partir de una solución factible *s* hacia una solución *s'* que proporcione el valor mínimo de la función objetivo, es decir la función de costo *f* es una asignación $f: S \rightarrow R$ que asigna un valor real a cada solución en *S*, llamado el *costo* de la solución [Michiels et al., 1998]; [Cruz y Rivera, 2007].

El procedimiento de búsqueda local requiere de una estructura de vecindad y de conocer una función objetivo que se requiera maximizar o minimizar, empieza con una solución factible *s* y el conjunto de soluciones en *N(s),* del cual se elige una solución *s'* a través de un movimiento σ, el tipo de movimiento a realizar para seleccionar un vecino define la estructura de la vecindad que mejore la función objetivo por medio de un procedimiento estocástico es decir $f(s') \leq f(s)$, si esto se cumple se reemplaza la solución *s*

por la solución *s'* que la mejore, esto se repite hasta alcanzar el criterio de paro de la búsqueda local. Las estructuras de vecindad son un aspecto importante, las cuales permiten tener una mejor explotación del espacio de soluciones.

En la figura 3-3 se muestra el algoritmo general de búsqueda local.

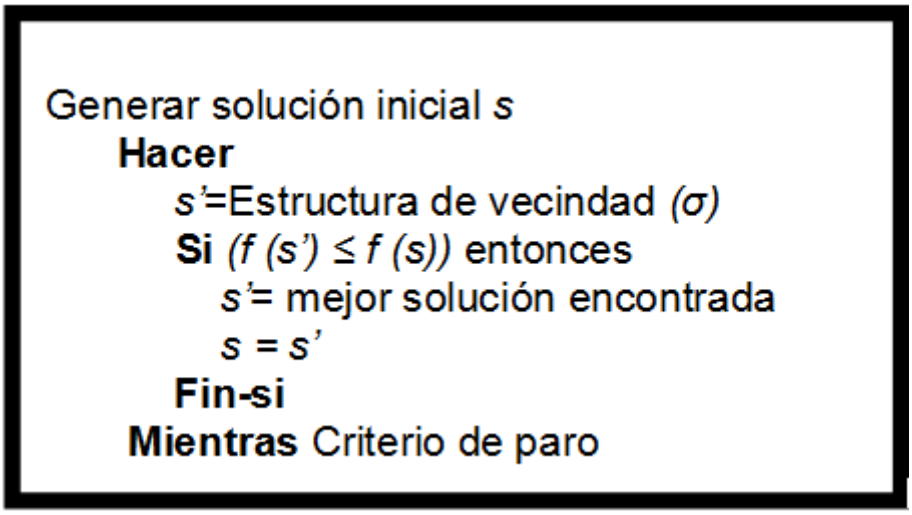


Figura 3-3 Algoritmo general de búsqueda local.

De acuerdo a trabajos presentes en la literatura [Hansen y Mladenovic, 2001]; [Arajy y Abdullah, 2010] para la Reducción de los Atributos en la Teoría de Conjuntos en Bruto se demuestra que la implementación de una estructura de vecindad híbrida da buenos resultados en su aplicación, también en el trabajo de investigación de [Cruz et al., 2010] utilizaron una

estructura híbrida para el problema del agente viajero obteniendo muy buenos resultados tanto en eficiencia como en eficacia, pero algo importante es que se comprobó en el trabajo de investigación de [Martínez, 2010], que para el problema de Máquinas en Paralelo no Relacionadas no se obtuvieron buenos resultados con respecto a la estructura de vecindad híbrida. A continuación se mencionan los movimientos realizados de cada una de las estructuras simples de vecindad, aplicados estos al problema MST. Estas estructuras conforman también a la estructura híbrida de vecindad. Cabe mencionar que cada estructura simple de vecindad realiza diferente tipo de movimiento, el cual se utiliza para ir mejorando una solución inicial hasta alcanzar el óptimo local.

Un Par Aleatorio. Este procedimiento inicia con una solución factible *s* a partir de la cuál se elige un número aleatorio *num1* considerado raíz, se elige otro número aleatorio *num2* considerado vértice vecino, entre los cuáles se realiza una perturbación, dicho movimiento genera un ciclo simple y se elimina una arista que pertenezca al ciclo generado. Si al eliminar esa arista quedan vértices sin conectar, se vuelve a conecta la arista eliminada y se elimina otra arista y así sucesivamente hasta que no queden vértices sin conectar. En la figura 3-4 se muestra el movimiento realizado por la estructura de un par aleatorio y en la figura 3-5 el algoritmo de dicha estructura.

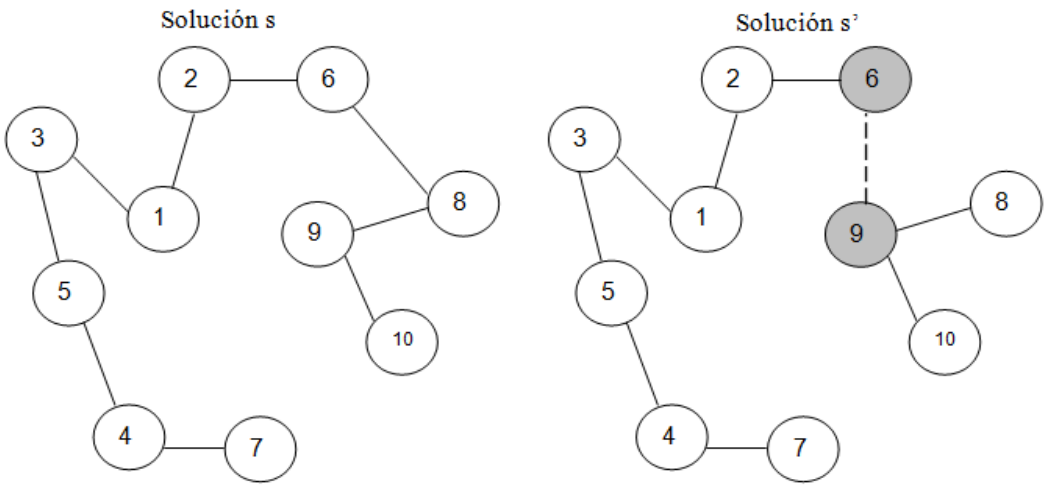


Figura 3-4 Estructura de vecindad un par aleatorio.

```
Solución inicial
Hacer
        Generar número aleatorio
         num1=rand () %N+1 // Donde  N = # de vértices
         Nvértices =contarvértices (num1)
       Si (Nvértices!= 0) entonces
          Generar otro número aleatorio
          num2= rand () % Nvértices +1 // Donde Nvértices = #
                                          de vértices de num1
          Conectar vértices num1 y num2
          Hacer
             Eliminar un arista pertenezca num1 o
             num2
             Desconectados=verificarVertices ();
             Si (Desconectados<N) entonces
                Conectar arista desconectada
             Fin-si
          Mientras (Desconectados!=N)
       Fin-si
Mientras (Nvértices ==0)
```

Figura 3-5 Algoritmo de la estructura de un par aleatorio

Dos Pares Aleatorios. Para esta estructura de vecindad, se lleva a cabo el mismo procedimiento explicado para un par aleatorio, la diferencia está en que requiere generar dos números aleatorios considerados como raíz, los cuáles deben ser distintos y elegir otros dos números aleatorios considerados

como vértices vecinos, estos números pueden ser iguales, por lo tanto se eliminan dos aristas que pertenezca a los números generados. La figura 3-6 muestra el movimiento realizado por la estructura utilizada y en la figura 3-7 se presenta el algoritmo de la estructura.

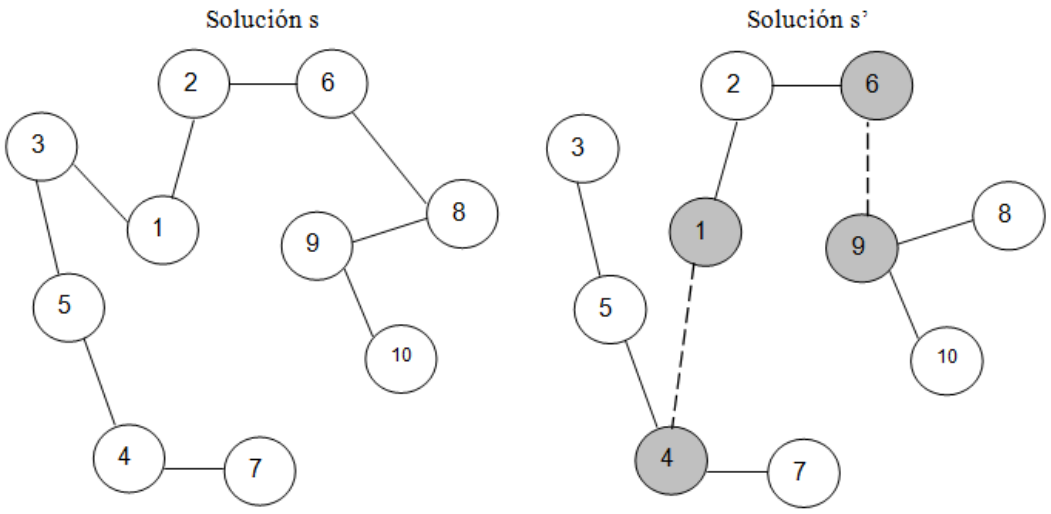


Figura 3-6 Estructura de vecindad dos pares aleatorios.

```
Solución inicial
Hacer
    Generar dos números aleatorios
    num1=rand () %N+1
    Hacer
        num2=rand () %N+1
    Mientras (num2==num1)
    Nvértices =contarvértices (num1)
    Si (Nvértices!= 0) entonces
        Generar otros dos números aleatorios
        Num3= rand () % Nvértices +1
        Num4= rand () % Nvértices +1
        Conectar num1,num2,num3,num4
        Hacer
            Eliminar dos aristas pertenezcan a num1,
            num2, num3 o num4
            Desconectados=verificarVertices ()
            Si (Desconectados<N) entonces
                Conectar arista desconectada
            Fin-si
        Mientras (Desconectados!=N)
     Fin-si
Mientras (Nvértices ==0)
```

Figura 3-7 Algoritmo de la estructura de dos pares aleatorios

Tres Pares Aleatorios. Una técnica de búsqueda por vecindad que requiera tres pares aleatorios, se lleva a cabo el mismo procedimiento explicado para un par aleatorio, la diferencia está en que requiere generar tres números aleatorios, considerados raíz, los cuáles deben ser distintos, se eligen otros tres números aleatorios considerados como vértices vecinos, estos números pueden ser iguales, por lo tanto se eliminan tres aristas que pertenezcan a los números generados. En la figura 3-8 se muestra el movimiento realizado por la estructura de vecindad de tres pares aleatorios y en la figura 3-9 el algoritmo.

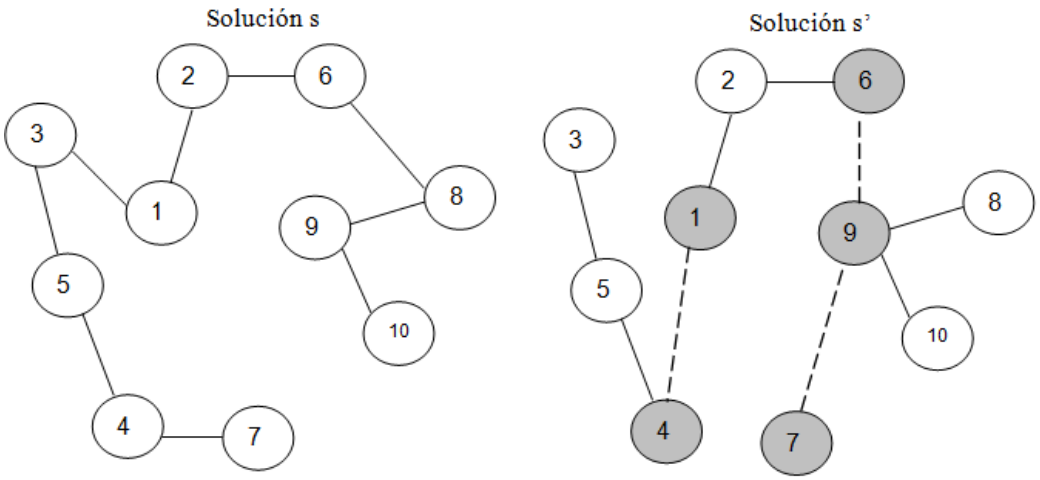


Figura 3-8 Estructura de vecindad tres pares aleatorios

```
Solución inicial
Hacer
      Generar tres números aleatorios
      num1=rand () %N+1
   Hacer
       num2=rand () %N+1
   Mientras (num2 == num1)
   Hacer
       num3=rand () %N+1
   Mientras (num3 == num1 || num3 == num2)
       Nvértices =contarvértices (num1)
   Si (Nvértices!= 0) entonces
          Generar otros tres números aleatorios
          num4= rand () % Nvértices +1
          num5= rand () % Nvértices +1
          num6= rand () % Nvértices +1
          Conectar num1,num2,num3,num4,num5,num6
      Hacer
             Eliminar tres aristas pertenezca num1,
             num2, num3, num4, num5, num6
             Desconectados=verificarVertices ();
             Si (Desconectados<N) entonces
                 Conectar arista desconecta
             fin-si
      Mientras (Desconectados!=N)
   Fin-si
Mientras (Nvértices ==0)
```

Figura 3-9 Algoritmo de la estructura de tres pares aleatorios

Cuatro Pares aleatorios. Una técnica de búsqueda por vecindad que requiera cuatro pares aleatorios, se lleva a cabo el mismo procedimiento explicado para las anteriores estructuras, la diferencia está en que se requiere generar cuatro números aleatorios, considerados raíz, los cuáles

deben ser distintos y elegir otros cuatro números aleatorios considerados como vértices vecinos, estos números pueden ser iguales, por lo tanto se eliminan cuatro aristas que pertenezca a los números generados. En figura 3-10 se observa el movimiento realizado por la estructura y en la figura 3-11 se presenta el algoritmo.

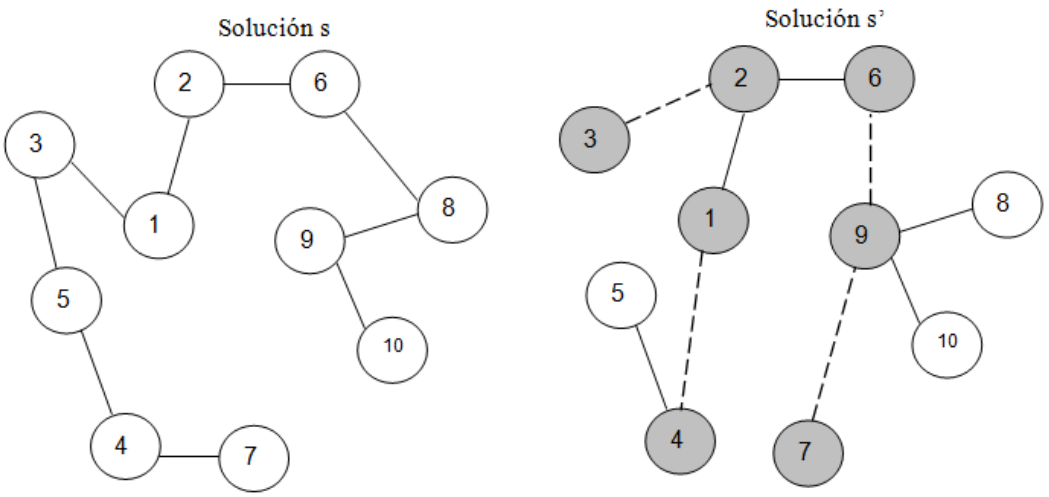


Figura 3-10 Estructura de vecindad de cuatro pares aleatorios.

```
Solución inicial
Hacer
      Generar tres números aleatorios
      num1=rand () %N+1
   Hacer
       num2=rand () %N+1
   Mientras (num2 == num1)
   Hacer
      num3=rand () %N+1
   Mientras (num3 == num1 || num3 == num2)
   Hacer
      num4=rand () %N+1
   Mientras (num4 == num1 || num4 == num2 || num4 ==
                num3)
     Nvértices =contarvértices (num1)
     Si (Nvértices!= 0) entonces
          Generar otros cuatro números aleatorios
           num5= rand () % Nvértices +1
           num6= rand () % Nvértices +1
           num7= rand () % Nvértices +1
           num8= rand () % Nvértices +1
          Conectar num1,num2,num3,num4,num5,num6,
          num7,num8
       Hacer
             Eliminar cuatro aristas pertenezcan num1, num2,
             num3, num4, num5, num6, num7, num8
             Desconectados=verificarVertices ();
         Si (Desconectados<N) entonces
             Conectar arista desconecta
         Fin-si
       Mientras (Desconectados!=N)
   Fin-si
Mientras (Nvértices ==0)
```

Figura 3-11 Algoritmo de la estructura de cuatro pares aleatorios

Considerando las funciones de las estructuras de vecindad presentadas anteriormente y su desempeño reportado en la literatura [Cruz et al., 2010], se propone el desarrollo de una ***estructura de vecindad híbrida*** la cual es una combinación de las estructuras explicadas anteriormente, estructura de un par aleatorio, dos pares aleatorios, tres pares aleatorios y cuatro pares aleatorios donde el tipo de movimiento a aplicar se determina aleatoriamente durante la ejecución del algoritmo.

A continuación se muestra el funcionamiento de forma general de la estructura de vecindad híbrida.

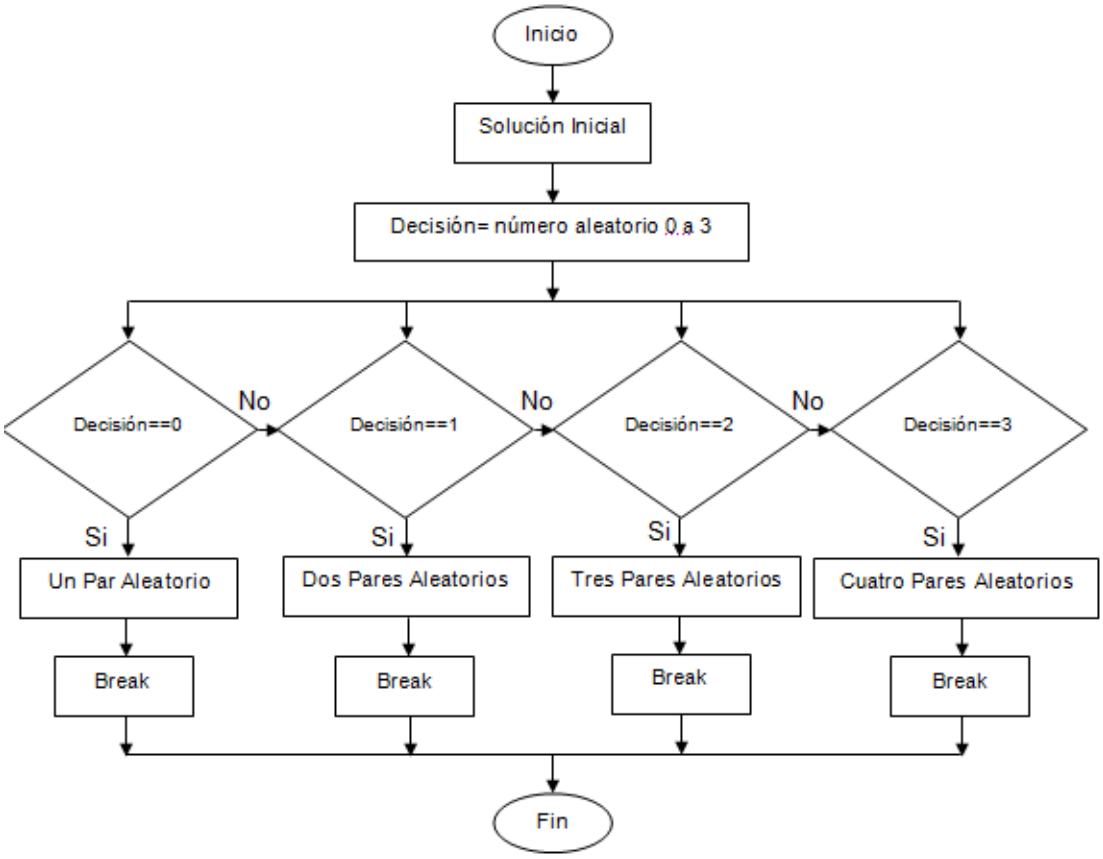


Figura 3-12 Diagrama de flujo estructura de vecindad híbrida

En la figura 3-12 se muestra la estructura de vecindad híbrida se observa que esta estructura se compone de las otras cuatro estructuras, el procedimiento de este algoritmo es partir de una solución inicial factible, y se elige de forma aleatoria que tipo de movimiento se aplica, si la decisión es 0 se aplica la estructura de un par aleatorio explicada en las figuras 3-4 y 3-5, si la decisión

es 1 entonces se aplica la estructura dos pares aleatorios explicada en la figura 3-6 y 3-7, si la decisión es 3 aplica la estructura tres pares aleatorios explicada en la figura 3-8 y 3-9, si la decisión es 4 se aplica el tipo de estructura de cuatro pares aleatorios explicada en la figura 3-10 y 3-11. Para mejorar aún más la eficiencia y eficacia de Recocido Simulado, aparte de la implementación de una estructura de vecindad híbrida también se aplicó una lista tabú que a continuación se explica a detalle.

3.5 Implementación de Lista Tabú

La búsqueda tabú TS (por sus siglas en inglés, *Tabu Search*) es una de las técnica más utilizadas en algoritmos no determinísticos, proviene de la inteligencia artificial y está basada en una memoria adaptativa que le permite explorar un espacio de soluciones de manera eficiente a través de la estructura de vecindad [Glover, 1989]. TS considera dos tipos de memoria a largo y corto plazo.

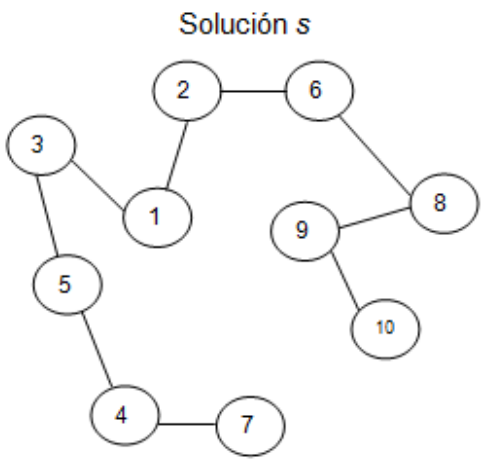
En este trabajo de investigación se emplea una lista tabú en Recocido Simulado para mejor la búsqueda local. La lista Tabú debe ser dinámica después de un cierto número de iteraciones la búsqueda está en una región distinta y las soluciones antiguas pueden liberarse del status tabú [Glover y Laguna, 1997].

Esta implementación ayuda a que las soluciones no queden atrapadas en óptimos locales, lo que permite mejorar aún más la eficacia y eficiencia del algoritmo. Esta lista tabú utilizada contiene los movimientos que se realizan entre vértices para encontrar una solución vecina, esta lista tiene la función de evitar que una solución sea visitada nuevamente en base a una repetición del movimiento.

A continuación se presenta el procedimiento del uso de la lista tabú para el problema del árbol de expansión mínima.

Paso 1:

Se parte de una solución inicial y una lista tabú vacía, cabe mencionar que los tamaños de la lista tabú son de 5 a 9 tomados de [Pinedo, 2008] en la figura 3-13 se presenta un ejemplo de una lista con un tamaño de 5 x 8, donde 8 representa el número máximo de vértices generados para los movimientos.



Lista Tabú

	1	2	3	4	5	6	7	8
1	0	0	0	0	0	0	0	0
2	0	0	0	0	0	0	0	0
3	0	0	0	0	0	0	0	0
4	0	0	0	0	0	0	0	0
5	0	0	0	0	0	0	0	0

Figura 3-13 Solución inicial y Lista Tabú vacía

Paso 2

Se elige de forma aleatoria un tipo de estructura de vecindad (un par aleatorio, dos pares aleatorios, tres pares aleatorios y cuatro pares aleatorios), por ejemplo se elige el movimiento de un par aleatorio, se generan dos vértices de forma aleatoria y se busca en la lista tabú si dicho movimiento no existe de ser así se agrega a la lista tabú como se muestra en la figura 3-14.

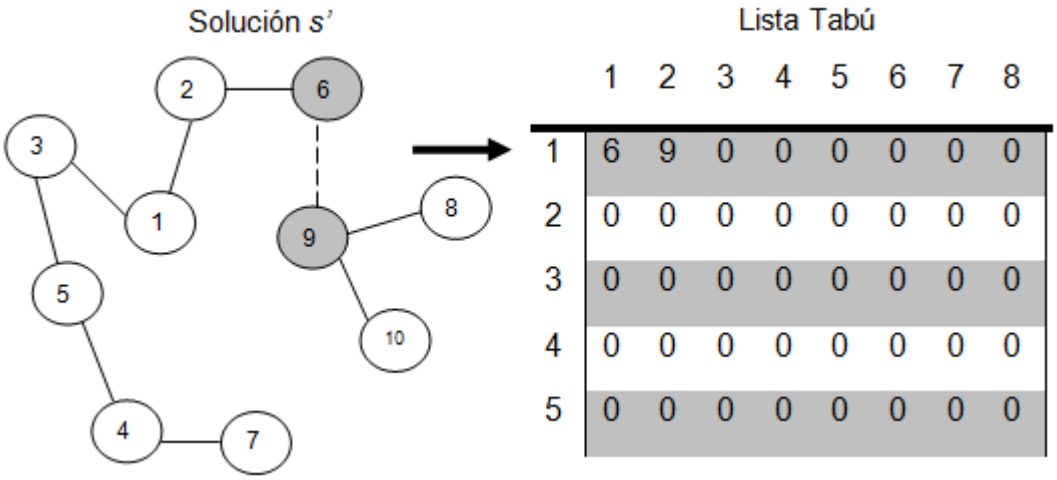
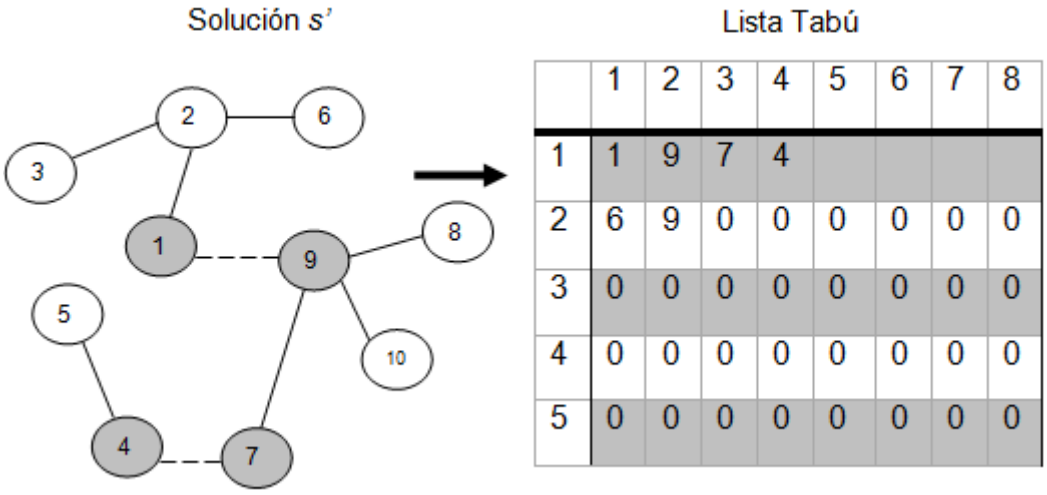


Figura 3-14 Agregar movimiento par aleatorio a lista tabú

Paso 3:

Se vuelve a elegir un movimiento de forma aleatoria, en este caso fue de dos pares aleatorios, se verifica que no exista en la lista tabú y de no existir se agrega a la lista tabú, cabe mencionar que se los movimientos se van recorriendo con forme se agrega el nuevo movimiento como se muestra en la figura 3-15.

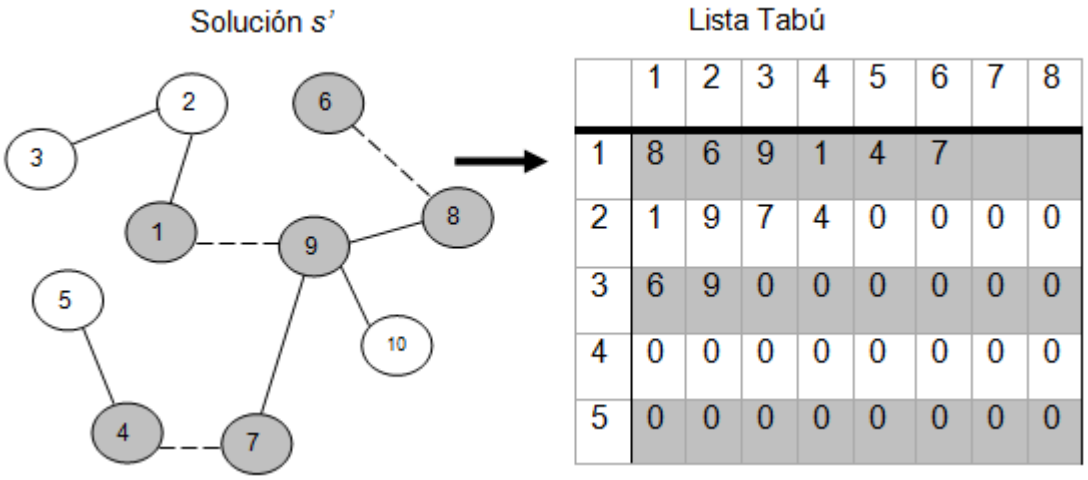


	1	2	3	4	5	6	7	8
1	1	9	7	4				
2	6	9	0	0	0	0	0	0
3	0	0	0	0	0	0	0	0
4	0	0	0	0	0	0	0	0
5	0	0	0	0	0	0	0	0

Figura 3-15 Agregar movimiento dos pares aleatorios a la lista tabú

Paso 4:

Se vuelve a elegir un movimiento de forma aleatoria, en este caso fue de tres pares aleatorios, se verifica que no exista en la lista tabú y de no existir se agrega a la lista tabú, cabe mencionar que se los movimientos se van recorriendo con forme se van agregan nuevos movimientos como se muestra en la figura 3-16.

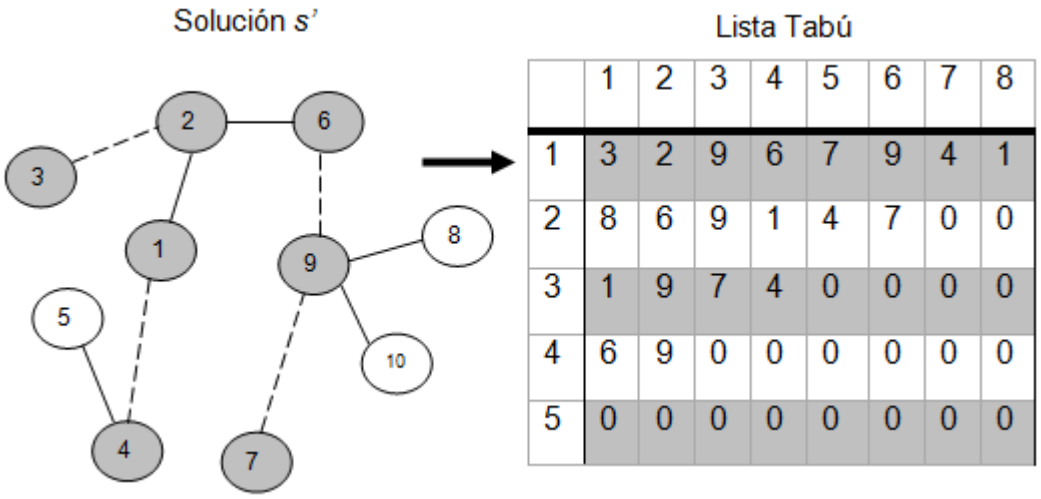


	1	2	3	4	5	6	7	8
1	8	6	9	1	4	7		
2	1	9	7	4	0	0	0	0
3	6	9	0	0	0	0	0	0
4	0	0	0	0	0	0	0	0
5	0	0	0	0	0	0	0	0

Figura 3-16 Agregar movimiento tres pares aleatorios a la lista tabú

Paso 5:

Se vuelve a elegir un movimiento de forma aleatoria, en este caso fue de cuatro pares aleatorios, se verifica que no exista en la lista tabú y de no existir se agrega a la lista tabú, los movimientos se van recorriendo con forme se agrega el nuevo movimiento como se muestra en la figura 3-17.

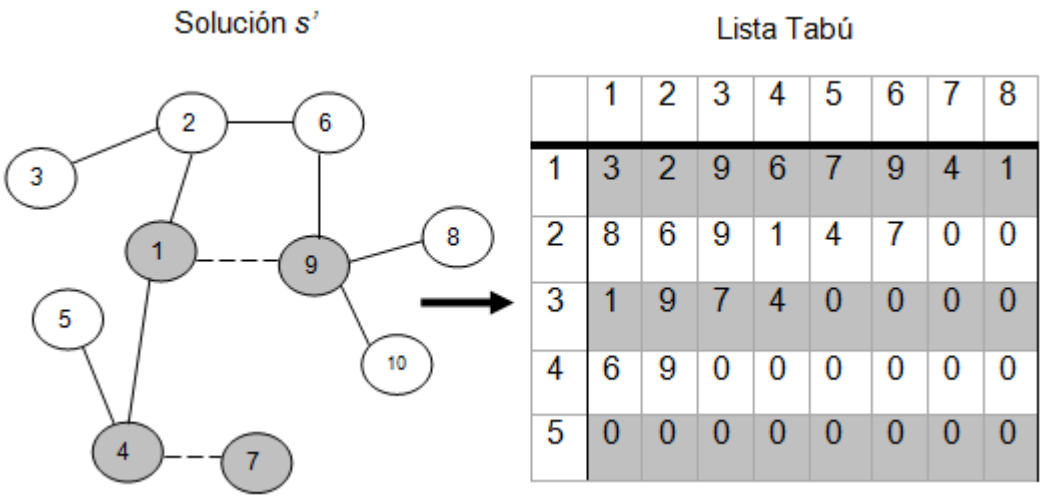


	1	2	3	4	5	6	7	8
1	3	2	9	6	7	9	4	1
2	8	6	9	1	4	7	0	0
3	1	9	7	4	0	0	0	0
4	6	9	0	0	0	0	0	0
5	0	0	0	0	0	0	0	0

Figura 3-17 Movimiento cuatro pares aleatorios y agregar a la lista tabú

Paso 6:

En la figura 3-18 se vuelve a elegir otro movimiento de forma aleatoria, en este caso se eligió dos pares aleatorios se verifica en toda la lista tabú que no exista dicho movimiento, como el movimiento ya existe se desecha y se continúa eligiendo otro movimiento.

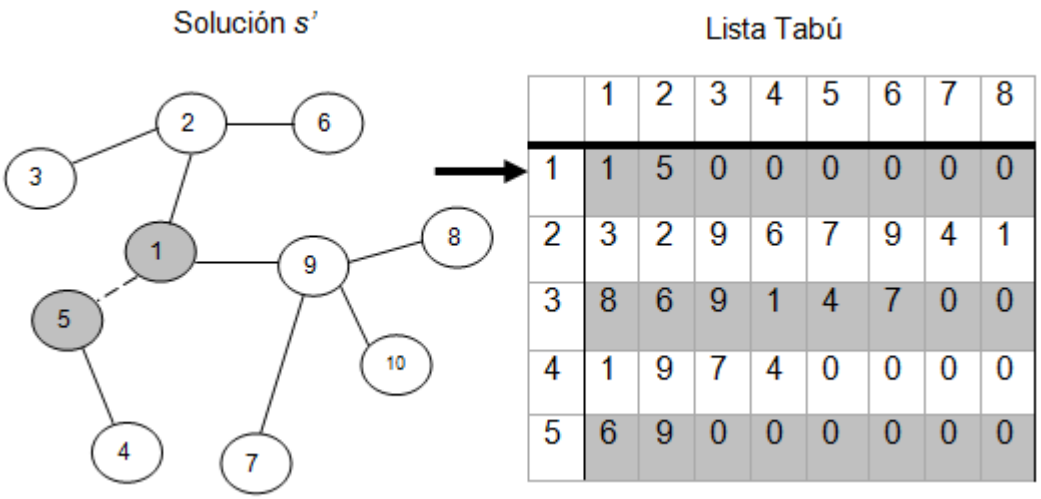


	1	2	3	4	5	6	7	8
1	3	2	9	6	7	9	4	1
2	8	6	9	1	4	7	0	0
3	1	9	7	4	0	0	0	0
4	6	9	0	0	0	0	0	0
5	0	0	0	0	0	0	0	0

Figura 3-18 Agregar Movimiento un par aleatoria a la lista tabú

Paso 7:

Se continúa eligiendo movimientos de forma aleatoria, se verifican en toda la lista tabú que no existan dichos movimientos, se agregan y si la lista ya está llena el último movimiento se libera de la lista y así sucesivamente como se ve en la figura 3-19.



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

Figura 3-19 Agregar Movimiento un par aleatoria a la lista tabú

```
Inicializar Variables
Repetir
        Si ListaTabu ==0 entonces
          Repetir
                ListaTabu=movimiento
          Hasta i<N
        Fin-si
        Sino
          Bandera=BuscarListaTabu(movimiento)
          Si Bandera ==0 entonces
              Repetir
                  Repetir
                      Recorrer Movimiento
                  Hasta i<N
              Hasta j<N
          Fin-si
        Fin-sino
Hasta k<N
```

Figura 3-20 Algoritmo de la lista tabú

3.6 Esquema Generalizado del Algoritmo de Recocido Simulado

En esta sección se presenta un esquema generalizado del algoritmo de Recocido Simulado propuesto en este trabajo de investigación. También se muestra el diagrama de flujo correspondiente al funcionamiento de dicho algoritmo.

El procedimiento esquematizado de SA es que visita el espacio de soluciones del problema de optimización reduciendo lentamente el valor de la función objetivo. A partir de una solución actual se visita una solución vecina y se acepta en el caso de ser mejor que la anterior y en caso contrario se acepta o se rechaza respetando una función de probabilidades definida como la función de *probabilidad de Boltzmann,* esto para evitar quedar atrapado en óptimos locales. En la figura 3-21 se presenta el algoritmo de Recocido Simulado.

Inicializar (T_0, s_0, T_f, α, L_k)
$T_k = T_0$
$s = s_0$
Repetir
  Repetir
    s' = Estructura de vecindad híbrida (s)
    Lista tabu(s')
    si $f(s') \leq f(s)$ entonces
      $s = s'$
    fin-si
    sino
      $Paceptar = \exp\left(-\frac{f(s') - f(s)}{T_k}\right)$
      $\rho = número\ aleatorio\ [0,1)$
      si $\rho < Paceptar$ entonces
        $s = s'$
      fin-si
    fin-sino
  Hasta alcanzar el equilibrio (L_k)
  $T_{k+1} = \alpha\ T_k$
Hasta $T_{k+1} \leq T_f$

Figura 3-21 Algoritmo de Recocido Simulado

A continuación se describen los pasos del esquema generalizado del Algoritmo.

1.- Inicializar un valor de inicio a T_0, un valor final T_f el cuál permitirá decrementar T_0 y una configuración inicial s_0 .

$T_k = T_0$

$s = s_0$

2.- Para cada iteración $k, k = 1....k_f$ hacer lo siguiente:

a. Repetir hasta alcanzar el equilibrio (L_k);

- Calcular el valor de la solución actual *s* mediante la función de costo *f(s).*
- Generar una nueva solución *s'* utilizando una estructura de vecindad híbrida (movimiento), *s' = N (s).*
- Verificar en la lista tabú si no existe dicho movimiento, si existe se queda con el mismo valor de la solución anterior.
- Calcular el valor de la nueva solución *s'* mediante la función de costo *f(s').*
- Evaluar la función objetivo donde la función de costo de la solución vecina *f(s')* sea menor que la función de costo de la solución actual *f(s)* de cumplirse se reemplaza *s = s'* de lo contrario, se acepta o rechaza de acuerdo al criterio de aceptación de la función de probabilidad de Boltzmann *P(s).*

b. Decrementar T_0 para *k+1* utilizando el coeficiente del parámetro de control α, $T_{k+1} = \alpha T_k$, donde α es una constante tal que $0 < \alpha < 1$.

3.- Cuando $T_{k+1} \leq T_f$, terminar.

A continuación de definen puntos importantes en el algoritmo de Recocido Simulado.

F(s): Función de costo de la solución actual a ser minimizada, en este trabajo es equivalente al peso del árbol de expansión mínima.

F(s'): Función de costo de la solución vecina.

N(s): Es la estructura de vecindad (genera nuevos vecinos).

P(s): Es la función que define el criterio de aceptación e indica si el nuevo estado se acepta o se rechaza, esta función da la distribución de probabilidad de boltzmann.

Para el problema del árbol de expansión mínima, un estado *s* es definido por una solución *S* (MST) del problema. La función de costo *f(s)* es definido como la función de costo que es minimizar la suma total de las aristas del MST. La vecindad *N(s)* de *S* se define como el conjunto de soluciones factibles que pueden ser generadas a partir de *S* mediante un movimiento, este puede ser, una perturbación, inserción o eliminación.

Es importante mencionar que el algoritmo de Recocido Simulado utiliza el algoritmo de Metrópolis para optimización combinatoria. Donde se desea minimizar una función de costo *f* haciendo las siguientes substituciones. Se toma el estado *s* como una solución posible del problema de optimización; *f(s)* como el costo del estado *s*, se genera un nuevo estado ***s'*** y *f(s)* como el costo de estado ***s'*** aplicando una pequeña perturbación a ***s'*** (estructura de vecindad) y se define un parámetro de control $c = T_0$ cuyo valor controla en su mayor parte el criterio de aceptación de malas soluciones [Metrópolis et al., 1953], esto es, si T_0 es muy grande la probabilidad de aceptación será

mayor, y si T_0 es muy pequeño, la probabilidad de aceptación será prácticamente cercana a cero. El criterio de aceptación es:

$$P\{Aceptar\ nueva\ \ solución\ s'\} = \begin{cases} 1, & si\ f(s') \le f(s) \\ \exp\left(-\dfrac{f(s') - f(s)}{c}\right), & si\ f(s') > f(s) \end{cases}$$

3.7 Metodología de Sintonización

Para la sintonización de los parámetros de control de un algoritmo, es necesario llevar a cabo un análisis de sensibilidad. El análisis de sensibilidad es un componente importante en la construcción de modelos matemáticos, computacionales y de simulación [Pannell, 2009]. El análisis de sensibilidad puede ser utilizado con simulación numérica para estudiar el comportamiento de un algoritmo. La finalidad del análisis de sensibilidad, es encontrar la mejor sintonización de los parámetros de control del algoritmo, de manera que este tenga el mejor funcionamiento posible en cuanto a la eficiencia y eficacia.

A continuación se explica la metodología de sintonización tomada del trabajo de investigación de [Cruz, 2005] y [Martínez, 2010], quienes proponen esta metodología para encontrar la proporción adecuada en los valores

correspondientes a los parámetros de control, tomando en cuenta el problema y el método implementado para obtener la solución.

1. Selección de los Parámetros de Control.

El primer paso es seleccionar los parámetros de control del algoritmo, es necesario llevar a cabo una revisión en la literatura en donde se haya implementado dicho algoritmo, de esta forma es posible hacer un análisis de los parámetros de control que utiliza el algoritmo tomado en investigaciones anteriores.

2. Establecer Rangos de Evaluación

Para establecer los rangos que van a ser utilizados para realizar el análisis de sensibilidad, es necesario ya haber identificado los parámetros de control, para establecer los rangos a cada uno de los parámetros, se hizo una revisión en la literatura para identificar y analizar los valores utilizados por varios autores en el algoritmo propuesto, de esta forma es más fácil definir un rango para cada uno de los parámetros de entrada al algoritmo. Es muy importante mencionar que si se lleva a cabo una adecuada sintonización de los parámetros de control, esto influye dentro del algoritmo para obtener buenas soluciones.

3. Pruebas para Rangos de Evaluación

Una vez establecido los rangos para cada parámetro de control mencionados anteriormente, se calcula una cantidad de muestra que permitan evaluar el comportamiento del algoritmo. Se realizan las pruebas experimentales, ejecutando 30 pruebas por cada una de las muestras calculadas, para una adecuada sintonización de los parámetros de control, es necesario realizar las pruebas de los valores correspondientes a una de las variables, pero manteniendo fijos los demás, hasta encontrar el valor que mejore la calidad de las soluciones. En cuanto se obtenga el mejor valor de dicha variable, se fija el valor y se comienza con la variación de otra variable, llevando a cabo el mismo proceso, hasta obtener el conjunto de valores que permita al algoritmo mejorar en eficacia y eficiencia.

4. Sintonización de Parámetros

Una vez a ver terminado las pruebas experimentales, para obtener los valores para cada uno de los parámetros de control definidos para todas y cada una de las muestras, estos valores son definidos como los *valores de sintonización.*

La sintonización de parámetros se lleva a cabo con el objetivo de identificar los valores correspondientes a los parámetros de control, los cuales permitan obtener una mejora en el desempeño del algoritmo tanto en eficacia como en eficiencia.

3.8 Análisis de la Complejidad del Algoritmo de Recocido Simulado

Para obtener la complejidad del algoritmo de Recocido Simulado, es necesario realizar un estudio para conocer su comportamiento y medir su rendimiento, y el uso eficiente de los recursos.

Un algoritmo necesita dos importantes recursos para resolver un problema: tiempo y espacio. El tiempo de complejidad de un algoritmo es el número de pasos necesarios para resolver un problema de tamaño n. El espacio de complejidad de un algoritmo es la cantidad de memoria utilizada durante la ejecución del algoritmo.

La complejidad es generalmente definida en términos del análisis del peor de los casos aunque también puede analizarse mejor caso y el caso promedio.

En la ecuación (6) se presenta la complejidad Asintotica de la estructura de vecindad.

Peor Caso

$$O(n^5) \qquad (6)$$

Para calcular la complejidad del algoritmo de Recocido Simulado es necesario realizar un análisis del número de instrucciones requeridas por el algoritmo para encontrar una solución al problema tratado. En la ecuación (7) se muestra la complejidad asintótica en el peor de los casos, en Apéndice A se presenta el desarrollo de dicha complejidad.

$$O\left((n^7 + n^6)\ln n\right) \qquad (7)$$

Donde *n* representa el número de vértices de la instancia.

3.9 Generación Aleatoria de Instancias de Prueba

Las instancias fueron generadas de forma aleatoria, debido a que en la literatura no se encontraron benchmarks para MST. Para generar las instancias de prueba, se desarrolló un programa en Visual C++ 2008, el cuál permite generar las instancias de forma aleatoria. A continuación en la tabla 3-2 se muestra un ejemplo de una instancia pequeña de 10 vértices.

Tabla 3-2 Matriz de vértices y costos para MST correspondiente a una instancia de 10 vértices generados de forma aleatoria.

		V	P	V	P	V	P	V	P	V	P	V	P	V	P	V	P	V	P
Vértices	**1**	6	86	10	38	5	74	7	34	8	51								
	2	4	31	9	68	8	40	7	42	6	73	3	32	10	2				
	3	2	32	5	11	10	90	6	47	4	11	9	44	7	40				
	4	2	31	3	11	7	3	5	31	6	87	9	50						
	5	1	74	3	11	4	31	10	73	7	94	6	54	8	29	9	79		
	6	1	86	2	73	3	47	9	17	7	83	4	87	5	54	8	49	10	38
	7	1	34	2	42	4	3	5	94	6	83	3	40	9	89	10	73	8	34
	8	1	51	2	40	6	49	7	34	10	45	5	29	9	38				
	9	2	68	3	44	6	17	7	89	10	77	5	79	8	38	4	40		
	10	1	38	3	90	5	73	6	38	7	73	8	45	9	77	2	2		

En la tabla 3-2 se muestra un ejemplo de la instancia de prueba generada de forma aleatoria, esta instancia contiene una columna V misma que corresponde al vértice y una columna P representa el costo en un intervalo de 1 a 100 requerido para conectar el vértice *i* con el vértice *j*. Los renglones representan el número total vértice en el grafo.

Capítulo 4

Resultados Experimentales del Algoritmo de Recocido Simulado

En este capítulo se presentan los resultados obtenidos en las pruebas experimentales realizadas al algoritmo de Recocido Simulado, donde se aplica una estructura de vecindad híbrida, una lista tabú y se aplica un análisis de sensibilidad, todo esto con la finalidad de mejorar el desempeño del algoritmo. Estos resultados son comparados con los obtenidos por el algoritmo de Búsqueda Local Iterada, donde se lleva a cabo un análisis para mostrar la eficiencia y eficacia del algoritmo de Recocido Simulado.

4.1 Descripción del Equipo Utilizado

Para realizar las pruebas experimentales correspondientes al algoritmo de Recocido Simulado, se utilizó el equipo del laboratorio de Optimización ubicado en el CIICAp. A continuación se enlista las características del equipo utilizado:

Cluster CIICAp:

- ***Procesador:*** Max 3.20GHz
- ***Memoria distribuida:*** 7GB
- ***Nodos :*** 7
- ***Sistema Operativo:*** Linux
- ***Compilador:*** gcc

Cluster VERACRUZ:

- ***Procesador:*** Max 3.20GHz
- ***Memoria distribuida:*** 57 GB
- ***Nodos:*** *14*
- ***Sistema Operativo:*** *Linux*
- ***Compilador:*** *gcc*

Cluster UPEMOR:

- ***Procesador:*** Max 2.0GHz
- ***Memoria distribuida:*** 20 GB
- ***Nodos:*** *5*
- ***Sistema Operativo:*** *Linux*
- ***Compilador:*** *gcc*

4.2 Pruebas de estructuras de vecindad

Las pruebas experimentales para cada una de las estructuras de vecindad fueron realizadas con el algoritmo de Búsqueda Local Iterada y de acuerdo a los resultados obtenidos se determina cual estructura se aplica al algoritmo de Recocido Simulado con la finalidad de mejorar la eficiencia y eficacia del mismo en cuanto a la explotación del espacio de soluciones. Las instancias de prueba utilizadas para el problema del árbol de expansión mínima fueron dos, de 100 y 200 vértices, estas instancias fueron generadas de forma aleatoria. Cada estructura de vecindad fue ejecutada 30 veces, el número de pruebas se tomó en base al análisis estadístico donde se dice que una muestra debe tener mínimo 30 pruebas para cada tamaño de instancia, teniendo un número total de 100 iteraciones realizadas durante la ejecución de la búsqueda local.

La tabla 4-1 y 4-2 presentan los resultados obtenidos para las instancias de prueba para el problema del MST, con las cinco estructuras de vecindad presentadas en el capítulo 3, de las cuales se hizo una evaluación de la mejor solución, peor solución, la función de costo promedio, así como su desviación estándar σ para las 30 pruebas realizadas de cada instancia.

En la tabla 4-1 para el problema MST de 100 vértices con ILS se observa que la estructura con peor eficacia es la de un par aleatorio, y en cuanto a la mejor solución encontrada de las 30 pruebas es la estructura híbrida. La

estructura de cuatro pares aleatorios es la mejor en cuanto a la peor solución de mejor calidad, al promedio y a la desviación estándar. Y la estructura híbrida en cuanto a promedio queda en tercer lugar de las cinco y en segundo lugar con respecto a la desviación estándar.

Tabla 4-1 Resultados para 100 Vértices. 30 ejecuciones de ILS para cada estructura.

Tipo de Estructura	Mejor Solución	Peor Solución	Promedio	Desv.Est.
Par Aleatorio	3937	5008	4155.2	132.06
Dos Pares Aleatorios	3776	5157	4094.1	134.68
Tres Pares Aleatorios	3806	5144	4037.9	126.48
Cuatro Pares Aleatorios	3773	***4900***	***4013.7***	***109.16***
Híbrida	***3769***	4941	4056.9	122.66

En la tabla 4-2 para el problema MST de 200 vértices con ILS se observa que la estructura con peor eficacia es la de un par aleatorio, y en cuanto a la mejor solución encontrada de las 30 pruebas es la estructura híbrida también en promedio. La estructura de cuatro pares aleatorios es la mejor en cuanto a la peor solución de mejor calidad. La estructura de tres pares aleatorios es la

que tiene la mejor desviación estándar y la estructura híbrida queda en segundo lugar de las cinco.

Tabla 4-2 Resultados para 200 Vértices. 30 ejecuciones de ILS para cada estructura.

Tipo de Estructura	Mejor Solución	Peor Solución	Promedio	Desv. Est.
Par Aleatorio	8178	10221	8630.7	187.47
Dos Pares Aleatorios	7995	9969	8493.4	200.68
Tres Pares Aleatorios	8162	9912	8515.4	***158.97***
Cuatro Pares Aleatorios	8088	***9776***	8490.8	183.05
Híbrida	***7987***	10095	***8468.5***	174.20

En la figura *4-1 a la 4-5* se presentan los resultados de las 30 ejecuciones obtenidas para la instancia del problema de 100 vértices del algoritmo de ILS para cada estructura de vecindad.

De las figuras 4-1 a la 4-5 Tomando una cota superior de 3900 en la función objetivo. La figura 4-1 muestra para el par aleatorio que las 30 soluciones están por encima de la cota superior. La figura 4-2 muestra para dos pares

aleatorios que las ejecuciones 16 y 21 están por debajo de la cota superior por lo que tiene mejor eficacia que la de par aleatorio. La figura 4-3 muestra para tres pares aleatorios que las ejecuciones 11, 16 y 22 están por debajo de la cota superior por lo que tiene mejor eficacia que la de par y dos pares aleatorios. La figura 4-4 muestra para cuatro pares aleatorios que las ejecuciones 4, 13, 21, 25 y 27 están por debajo de la cota superior por lo que tiene mejor eficacia que la de par, dos pares y tres pares aleatorios. La figura 4-5 muestra para el híbrido aleatorio que las ejecuciones 11 y 17 se encuentran por debajo de la cota superior por lo que la eficacia para esa cota no es buena. Esto indica que el comportamiento de las estructuras en cuanto a eficacia varía si lo que se requiere es lograr llegar a una cota como valor de la función objetivo. Por ejemplo para la cota de 3900 la mejor eficacia es para cuatro pares aleatorios, el par aleatorio quedaría al final y el híbrido quedaría en tercer lugar.

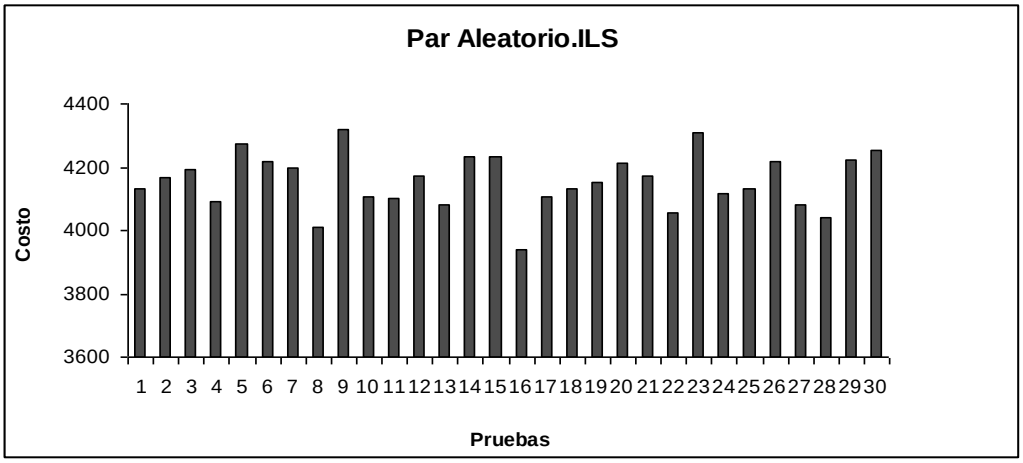


Figura 4-1 Resultados Estructura de vecindad par Aleatorio

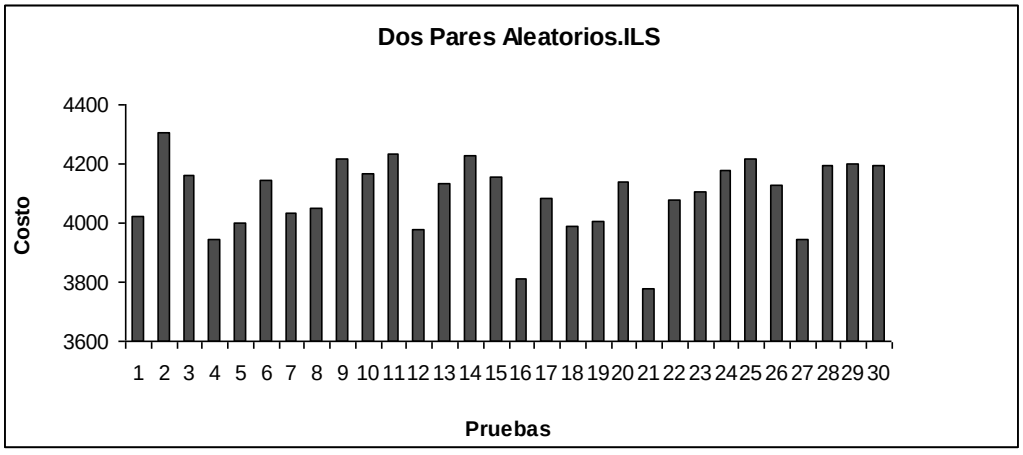


Figura 4-2 Resultados Estructura de vecindad dos pares aleatorios

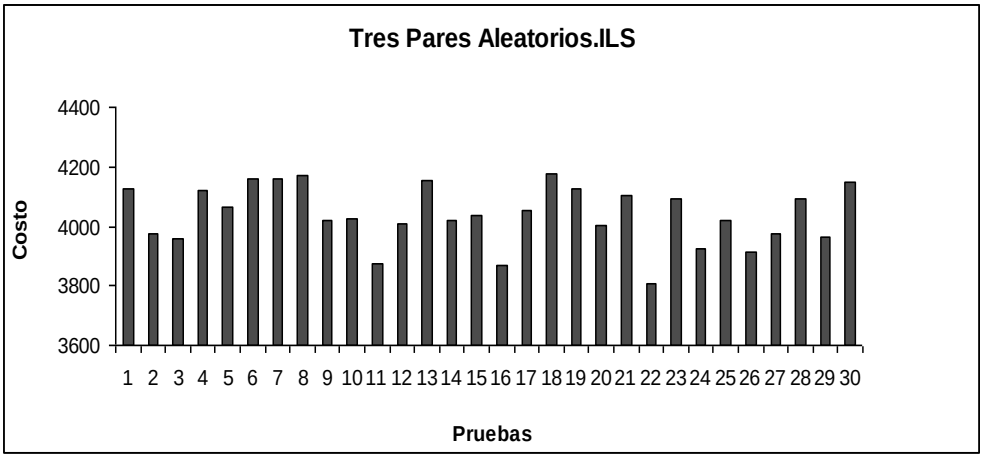


Figura 4-3 Resultados Estructura de vecindad tres pares aleatorios

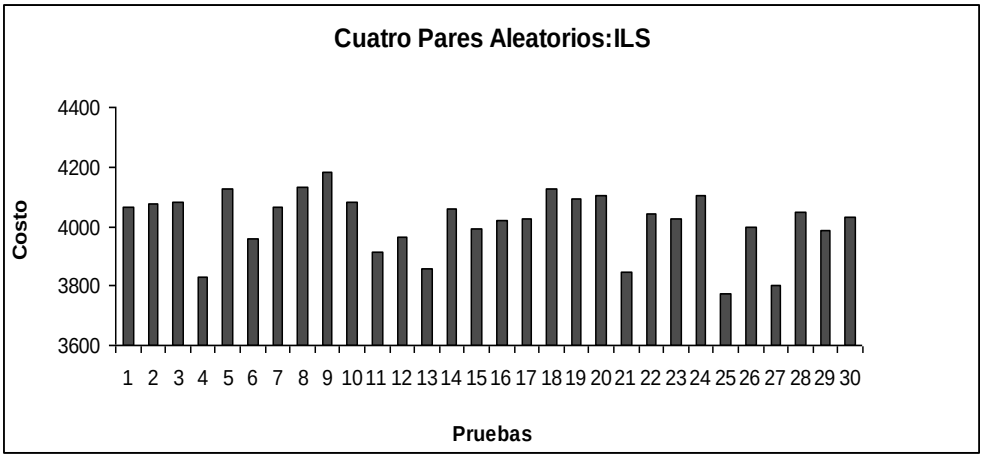


Figura 4-4 Resultados Estructura de vecindad cuatro pares aleatorios

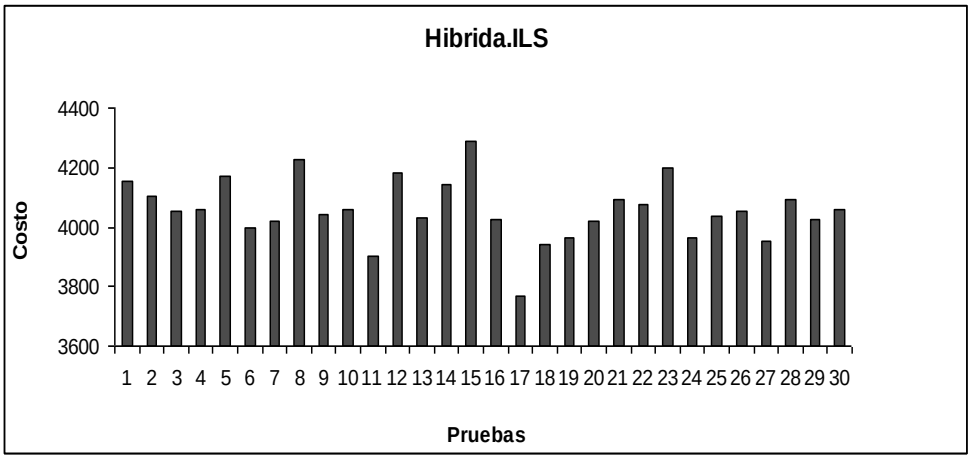


Figura 4-5 Resultados Estructura de vecindad híbrida

En la figura 4-6 a la 4-10 se presentan los resultados de las 30 ejecuciones obtenidas para la instancia del problema de 200 vértices del algoritmo de ILS para cada estructura de vecindad.

De las figuras 4-6 a la 4-10 Tomando una cota superior de 8400 en la función objetivo. La figura 4-6 muestra para el par aleatorio que las ejecuciones 13 y 17 están por debajo de la cota superior. La figura 4-7 muestra para dos pares aleatorios que las ejecuciones 3,12,13,14,15,17, 22,23 y 25 están por debajo de la cota superior por lo que tiene mejor eficacia que la de par aleatorio. La figura 4-8 muestra para tres pares aleatorios que las ejecuciones 3, 5,9,10,23,26 y 30 están por debajo de la cota superior por lo que tiene mejor eficacia que la de par aleatorio pero peor eficacia que la de dos pares aleatorios. La figura 4-9 muestra para cuatro pares aleatorios que las

ejecuciones 5, 12, 15, 18 y 22 están por debajo de la cota superior por lo que tiene mejor eficacia que la de par, pero peor eficacia que la de dos pares y tres pares aleatorios. La figura 4-10 muestra para el híbrido aleatorio que las ejecuciones 7, 8, 18, 19, 21, 22, 24, 27, 28 y 29 se encuentran por debajo de la cota superior por lo que tiene mejor eficacia que las otras cuatro estructuras, la eficacia para esa cota es buena. Esto indica que el comportamiento de las estructuras en cuanto a eficacia varía si lo que se requiere es lograr llegar a una cota como valor de la función objetivo. Por ejemplo para la cota de 8400 la mejor eficacia es para la híbrida, el par aleatorio quedaría al final y el cuatro pares aleatorios quedaría en penúltimo lugar.

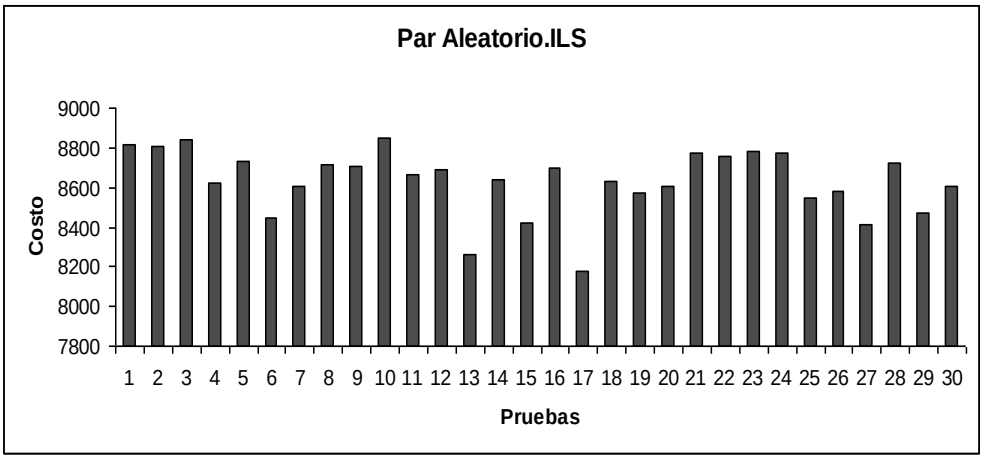


Figura 4-6 Resultados Estructura de vecindad par aleatorio

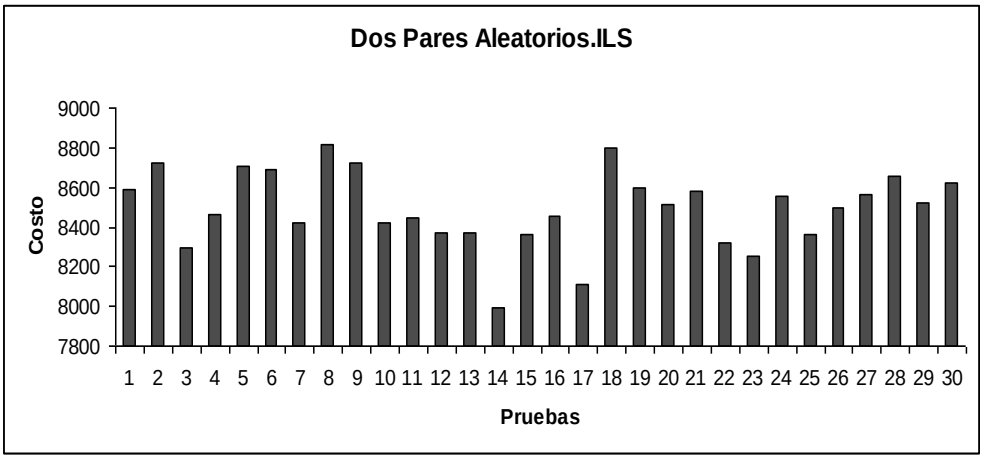


Figura 4-7 Resultados Estructura de vecindad dos pares aleatorios

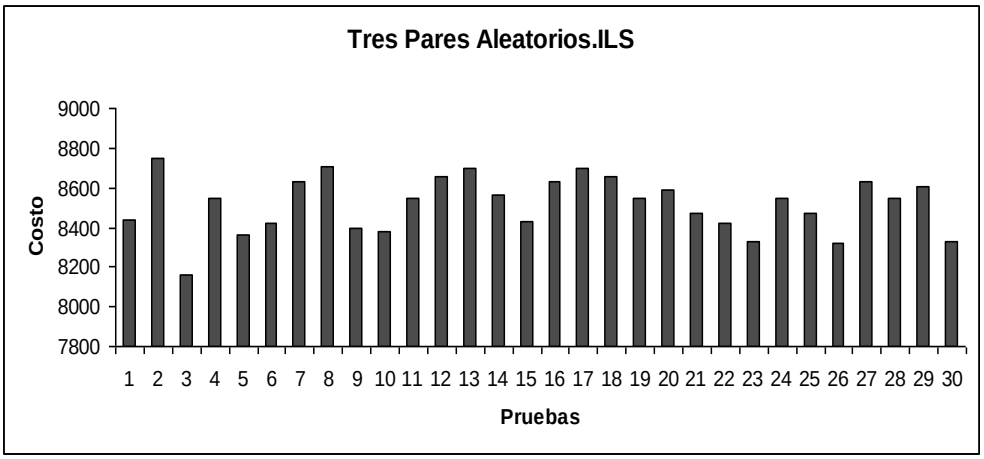


Figura 4-8 Resultados Estructura de vecindad tres pares aleatorios

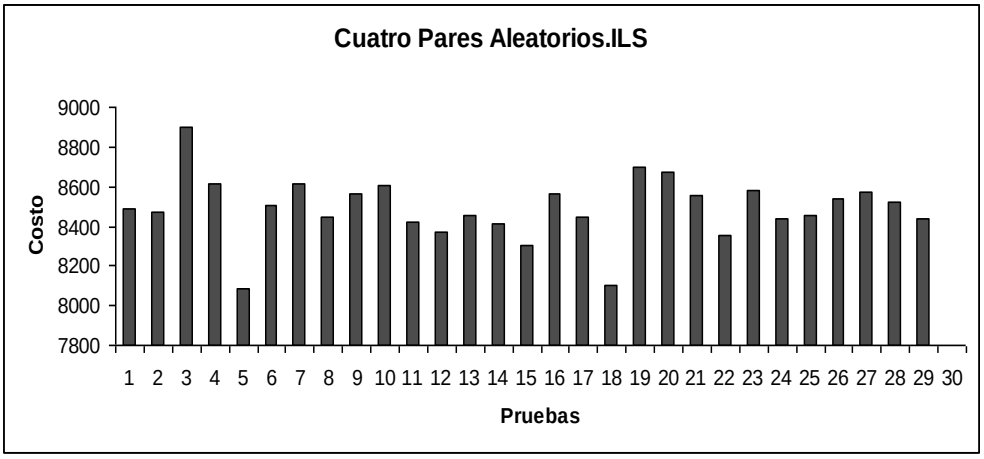


Figura 4-9 Resultados Estructura de vecindad cuatro pares aleatorios

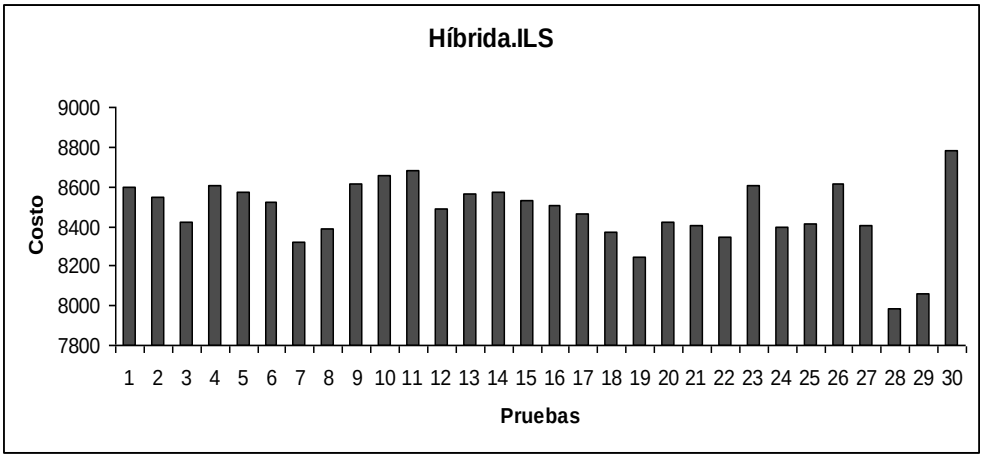


Figura 4-10 Resultados Estructura de vecindad híbrida

La figura 4-11 presenta los resultados obtenidos del tiempo de ejecución promedio del problema MST de 100 y 200 vértices con 30 pruebas para cada

instancia aplicando cada estructura de vecindad. En la figura 4-11 se puede observar que en la instancia de 200 vértices el tiempo de ejecución para cada estructura de vecindad se incrementa considerablemente, debido al aumento en el tamaño del espacio de soluciones. Entonces, podemos decir que una vecindad de mayor tamaño necesariamente requiere mayor tiempo para ser explorada, por ello el tiempo de ejecución para realizar las 30 ejecuciones con 200 vértices aumenta.

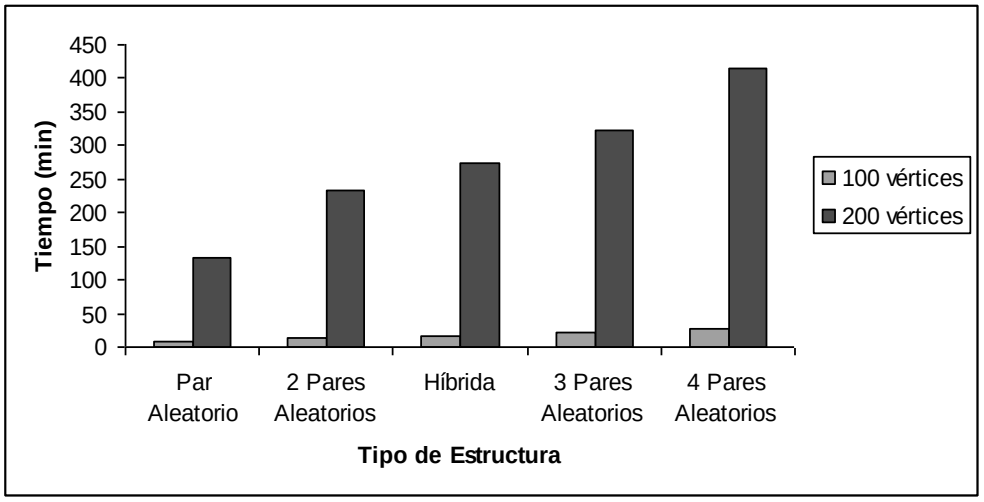


Figura 4-11 Resultados de tiempo de ejecución. 100 y 200 vértices para cada estructura

Se observa en la figura 4-11 que la estructura de par aleatorio en ambos casos es la mejor en eficiencia, ya que tiene el mejor tiempo de ejecución. La estructura híbrida propuesta en este trabajo de investigación muestra un

comportamiento competitivo en ambos casos, es decir no es la que cuenta con la mejor eficiencia, pero tampoco es la de peor eficiencia. Esta estructura se encuentra en un punto intermedio con respecto a las otras estructuras. El comportamiento de la estructura híbrida es lógico, debido a que se compone de las otras cuatro estructuras (capítulo 3) las cuales se eligen de forma aleatoria. Y la estructura con peor eficiencia es la de cuatro pares aleatorios ya que en ambos casos el tiempo de ejecución es el mayor a comparación con las otras estructuras.

De acuerdo a la evaluación de los resultados obtenidos en los dos casos en cuanto a eficacia y eficiencia, se decidió tomar la estructura híbrida ya que en ambos casos dio la mejor solución y en un caso dio el mejor promedio. La estructura de cuatro pares aleatorios en un caso es la que tiene mejor eficacia, pero es la de peor eficiencia y la estructura híbrida es competitiva es decir no es la que mejor eficiencia tiene pero tampoco la de peor eficiencia se encuentra un punto intermedio a comparación de las otras estructuras de vecindad.

4.3 Análisis de Sensibilidad

Para realizar el análisis de sensibilidad de los parámetros de control del algoritmo de Recocido Simulado aplicado al problema del Árbol de Expansión Mínima, se implementó una metodología de sintonización explicada en el

capítulo 3. A continuación se aplica dicha metodología al algoritmo de Recocido Simulado.

1. Selección de los Parámetros de Control.

Primero se selecciona las variables utilizadas para llevar a cabo el análisis de sensibilidad del dicho algoritmo aplicado al problema del árbol de expansión mínima. A continuación se enlistan los parámetros de control de entrada al algoritmo de Recocido Simulado.

- Valor inicial del parámetro de control (T_0).
- Coeficiente del parámetro de control (α).
- Valor final del parámetro de control (T_f).
- Longitud de cada cadena de Markov (L_k).
- Tamaño de la lista tabú (L_T).

2. Establecer Rangos de Evaluación

En cuanto se hayan definido los parámetros de control, es necesario determinar los rangos que serán utilizados para el análisis de sensibilidad a cada uno de los parámetros mencionados anteriormente. Se hizo una

revisión en la literatura donde se aplica el algoritmo de Recocido Simulado [Cruz, 2005]; [Martínez, 2008]; [Torres y Vélez, 2007] y de acuerdo a los valores utilizados en estos trabajos se determinó un rango para el parámetro de T_0, α y T_f, para el valor de la longitud de la cadena de Markov L_k se tomó como el tamaño de la vecindad [Van Laarhoven et al., 1992] y para el rango de la lista tabú se tomó en base a la investigación de [Pinedo, 2008].

Tabla 4-3 Rangos utilizados para realizar el análisis de sensibilidad al Recocido Simulado.

Parámetro de control	Límite inferior	Límite superior
T_0	1000	10016
α	0.965	0.994
T_f	0.0001	2
L_T	5	9
L_k	N +1	

3. Pruebas sobre Rangos de Evaluación

Para llevar a cabo el análisis de sensibilidad a los parámetros de control mencionados anteriormente, se realizan los pasos siguientes:

En principio, para cada uno de los rangos mostrados en la tabla 4-3, se calcularon 30 muestras con los incrementos mostrados en la tabla 4-4.

Tabla 4-4 Valores de incrementos utilizados para los rangos de los parámetros de control utilizados para el análisis de sensibilidad.

Parámetro de Control	Incrementos
T_0	311
α	0.001
T_f	0.143
L_T	1

El análisis de sensibilidad se llevó a cabo realizando 30 ejecuciones de T_0, de acuerdo al incremento y presentado en la Tabla 4-4, manteniendo constante el valor de los parámetros restantes con un valor que comúnmente

se utiliza en Recocido Simulado para problemas de optimización, α = 0.98, T_f = 0.001, L_k = n+1 y con L_T = 0, por cada serie de pruebas.

En cuanto se terminada de realizar la evaluación de los resultados obtenidos para cada una de las pruebas, se fija el valor de T_0 que haya obtenido los mejores resultados de acuerdo a la función objetivo del problema. En la figura 4-12 se muestran los resultados obtenidos en cuanto al promedio de 30 ejecuciones para cada incremento de T_0, donde podemos observar que el valor de 9708 es el que mejor soluciones obtuvo.

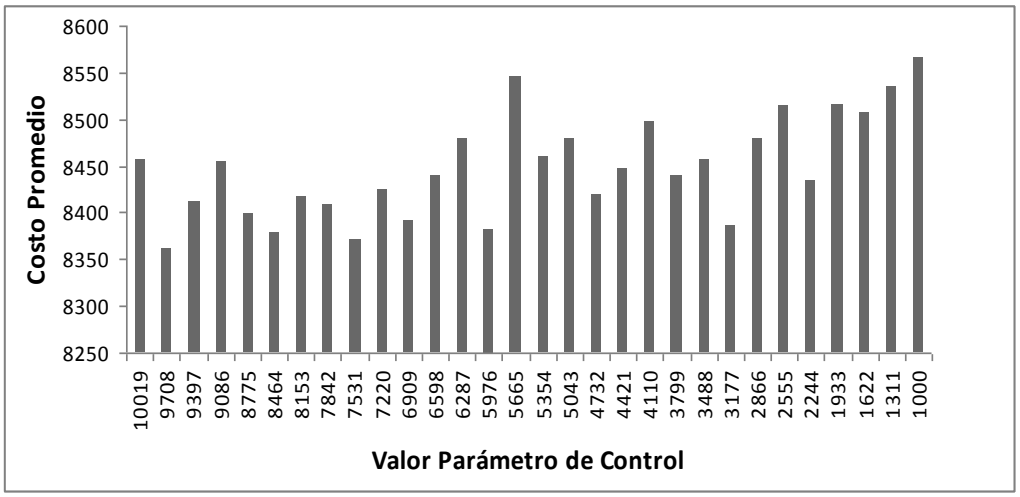


Figura 4-12 Resultados obtenidos para la sintonización del valor del parámetro de control T_0

Se continua realizando las pruebas, esta vez realizando las ejecuciones para el valor de α y manteniendo constante los valores de los parámetros T_0 =9708 ya con el valor fijado que obtuvo los mejores resultados, T_f = 0.001, L_k = n+1 y con L_T = 0 respectivamente. Al igual que en el caso de T_0, se realiza una serie de 30 ejecuciones por cada incremento evaluado, fijando el valor de α que obtenga las mejores soluciones. En la figura 4-13 se muestra los resultados obtenidos en cuanto al promedio de las 30 ejecuciones, de acuerdo a estos resultados se fijó el valor de α = 0.994 ya que ese valor es el que obtuvo mejores soluciones.

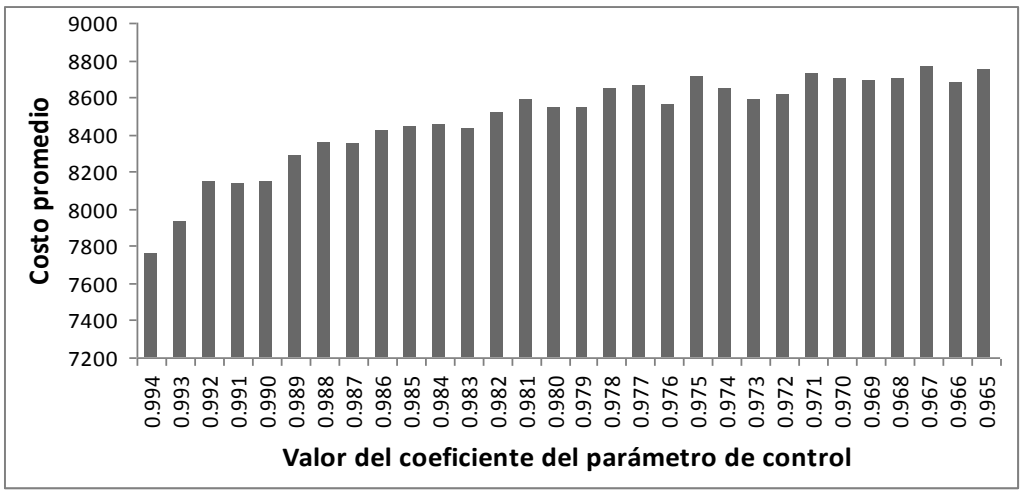


Figura 4-13 Resultados obtenidos para la sintonización del valor del coeficiente del parámetro de control α

El mismo proceso se lleva a cabo para el valor de T_f, manteniendo constante los valores de los parámetros T_0=9708, α=0.994 ya con el valor fijado de acuerdo a los resultados, L_k = n+1 y con L_T = 0. Al igual que en el caso de T_0 y α, se realiza una serie de 30 ejecuciones por cada incremento evaluado, fijando el valor de T_f que obtenga las mejores soluciones. En la figura 4-14 se muestra los resultados obtenidos en cuanto al promedio de las 30 ejecuciones, de acuerdo a estos resultados se fijó el valor de T_f = 0.00016 ya que ese valor es el que obtuvo mejores soluciones. También se puede observar que a partir del valor 0.001 al 0.00016 el comportamiento es muy similar.

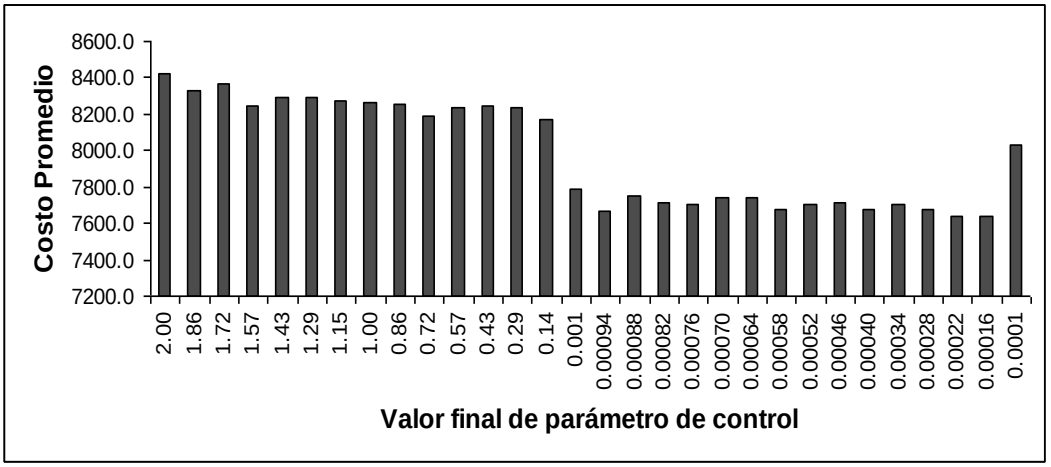


Figura 4-14 Resultados obtenidos para la sintonización del valor final del parámetro de control T_f

La sintonización del valor L_T se realizó con dos tamaños de instancia de 200 y 300 vértices con 30 ejecuciones para cada incremento manteniendo constante los valores de los parámetros T_0=9708, α=0.994 , T_f=0.00016 ya con los valores fijados de acuerdo a las pruebas realizas y L_k = n+1, fijando el valor de L_T que obtenga las mejores soluciones de acuerdo a la función objetivo del problema. En la tabla 4-5 podemos observar que los mejores resultados obtenidos para ambos casos en cuanto a la mejor solución obtenida es con un tamaño de lista tabú de 9. Y en cuanto al mejor promedio en el caso de 100 vértices también se obtiene con un tamaño de 9 y para el caso de 200 vértices se obtiene el mejor promedio con un tamaño de lista tabú de 6. De acuerdo a estos resultados se decidió utilizar el tamaño de lista tabú de 9 ya que en ambos casos obtuvo la mejor solución.

Tabla 4-5 Resultados para dos instancias de 200 y 300 vértices con 30 ejecuciones para cada tamaño de lista tabú.

	100 Vértices		**200 Vértices**	
Tamaños de Lista Tabú	**Mejor Solución**	**Promedio**	**Mejor Solución**	**Promedio**
5	6848	7560.4	12513	13098
6	7079	7543.6	12472	**12988.7**
7	7122	7633.1	12316	13064.3
8	6864	7605.2	12216	13009
9	**6667**	**7528.5**	**11845**	13042.2

4. Sintonización de Parámetros

Los valores obtenidos para cada uno de los parámetros de control evaluados, al término de la evaluación de todas y cada uno de los incrementos (Tabla 4-4), dan como resultado la *sintonización de los parámetros de control.*

La sintonización de parámetros de control se lleva a cabo con el objeto de identificar los valores correspondientes a los parámetros de control, lo cual permita obtener una mejora en el desempeño del algoritmo. De acuerdo al

análisis de sensibilidad realizado, se obtuvieron los siguientes valores sintonizados (Tabla 4-6).

Tabla 4-6 Valores sintonizados correspondientes a los parámetros de control evaluados durante el análisis de sensibilidad.

Parámetro de Control	Valor Sintonizado
T_0	9708
α	0.994
T_f	0.00016
L_T	9

4.4 Pruebas Recocido Simulado

De acuerdo a los resultados obtenidos del análisis de cada estructura de vecindad aplicada al algoritmo de búsqueda local iterada (ver sección 4.2), se demuestra que al aplicar una estructura híbrida se obtienen buenos resultados en cuanto a eficacia y en eficiencia es competitiva, es decir no es la que cuenta con la mejor eficiencia, pero tampoco es la de peor eficiencia. Por lo tanto esta estructura se encuentra en un punto intermedio con respecto a las otras estructuras. De acuerdo a esto se implementó la

estructura de vecindad hibrida al algoritmo de Recocido Simulado para mejorar la eficiencia y eficacia del algoritmo.

Para probar el funcionamiento del algoritmo, se generaron de forma aleatoria cuatro tamaños de instancias de 200, 300, 400 y 500 vértices. Cabe mencionar que los problemas de prueba fueron generados de forma aleatoria debido a que en la literatura no se encontraron Benchmarks para el problema clásico del árbol de expansión mínima que se trata en esta tesis.

Las pruebas experimentales, se realizaron ejecutando 30 veces cada uno de los problemas de prueba antes mencionados, tomando como criterio de paro el valor final del parámetro de control T_f .

Tabla 4-7 Resultados del algoritmo de Recocido Simulado aplicando una Estructura de Vecindad Híbrida.

Instancias	Mejor Solución	Peor Solución	Promedio	Desv.Est.
200	7198	10739	7816.8	204.1
300	12563	15893	13285.3	299.3
400	17740	21075	18317.5	285.1
500	22465	26290	23303.9	389.3

En la tabla 4-7, las *instancias* representa el número de vértices del grafo. La segunda columna corresponde a la mejor solución encontrada de acuerdo a las 30 ejecuciones realizas para cada instancia. La tercera columna representa la peor solución encontrada, es decir la más lejana al óptimo. La columna etiquetada con la palabra "*Promedio"* representa el promedio total de las 30 ejecuciones para cada instancia, La quinta columna etiquetada con representa el cálculo de la desviación estándar, permite conocer que tan dispersas están las soluciones con respecto a la media.

Además de hacer un análisis de la calidad de las soluciones, otro punto importante a evaluar es el tiempo requerido por cada una de las instancias, esto determina que tan eficiente es el algoritmo propuesto en este trabajo de

investigación. En la figura 4-15 se muestra el tiempo de ejecución del algoritmo de Recocido Simulado aplicando la estructura de vecindad híbrida para cada tamaño de instancia de prueba con un total de 30 ejecuciones para cada una. Donde se puede observar que entre mayor sea el tamaño de la instancia el tiempo de ejecución también crece. En esta gráfica se observa que en promedio el tiempo de cada ejecución es considerablemente grande, pasando de 30 minutos para la instancia de 200 vértices a más de 11 horas para la instancia de 500 vértices. Esto indica que a pesar de que el problema del árbol de expansión mínima es clasificado por la teoría de la complejidad como un problema de tipo P, el tiempo de ejecución del algoritmo de Recocido Simulado tiene un comportamiento en tiempo de ejecución casi de forma exponencial al aumentar el tamaño de la instancia. Lo cual indica que para instancias grandes arriba de 500 vértices se requiere de equipo de cómputo especializado de alto desempeño o bien la paralelización del algoritmo.

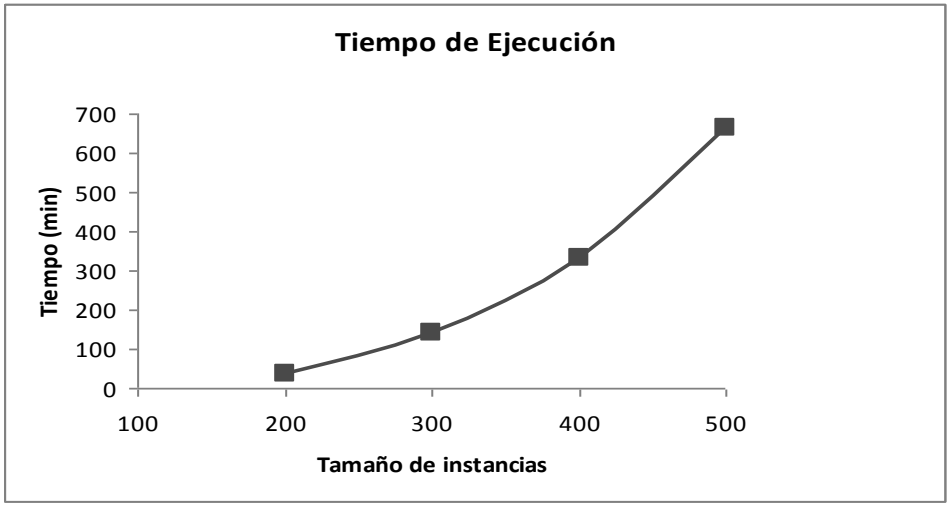


Figura 4-15 Tiempo de ejecución para cada tamaño de instancia con 30 ejecuciones

4.5 Análisis de Eficacia y Eficiencia del Algoritmo

El análisis de eficacia y eficiencia del algoritmo propuesto se lleva a cabo para evaluar la calidad de las soluciones obtenidas, como los recursos y tiempo necesarios para su ejecución.

Se hace una comparación de los resultados obtenidos en 30 pruebas por el algoritmo de Recocido Simulado y el algoritmo de búsqueda local iterada aplicando en ambos una estructura de vecindad híbrida, cabe mencionar que sólo se compara con un tamaño de instancia de 200 vértices.

Tabla 4-8 Comparación ILS contra SA para MST con una instancia de 200 vértices y 30 pruebas

	Mejor Solución	**Promedio**
ILS	7987	8418.5
SA	7198	7816.8

En la tabla 4-8 se puede observar que el algoritmo de Recocido Simulado da mejores resultados en cuanto a eficacia tanto en la mejor solución como en promedio de todas las soluciones encontradas.

El algoritmo de Recocido Simulado propuesto en esta trabajo de investigación, aplicado al problema del árbol de expansión mínima es competitivo en cuanto a eficacia, debido a que se obtuvieron buenos resultados para dicho problema.

Además de hacer un análisis de la eficacia, también es necesario hacer una evaluación con respecto al tiempo requerido por cada algoritmo, esto determina que tan eficiente es el algoritmo. En la figura 4-16 muestra el tiempo de ejecución de cada algoritmo utilizando una estructura de vecindad híbrida para el problema del árbol de expansión mínima.

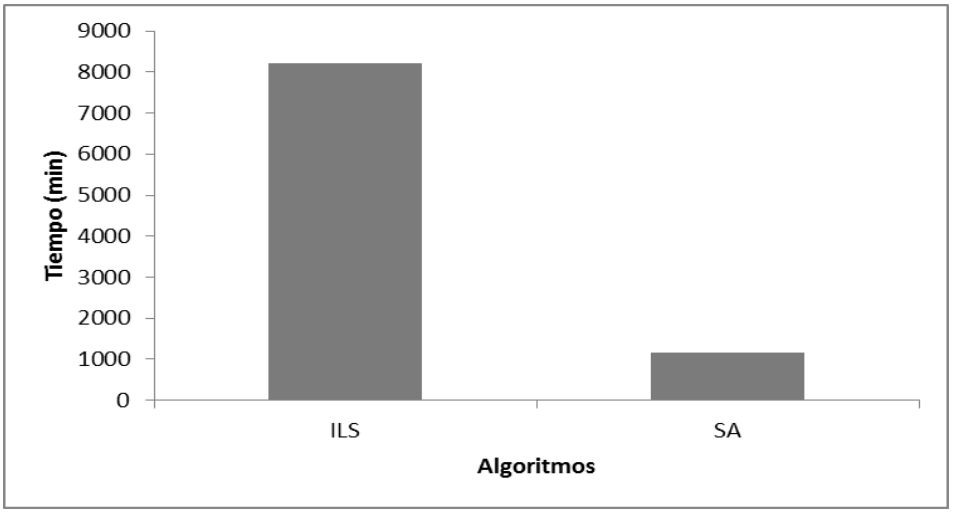


Figura 4-16 Tiempo de ejecución para el algoritmo de Recocido Simulado con Búsqueda Local Iterada

Podemos observar en la figura 4-16 que el tiempo de ejecución para instancia de 200 vertices en ILS es muy elevado en comparación con SA, cabe mencionar que para cada algoritmo se hicieron 30 pruebas y el instancia de entrada tiene las mismas caracteristicas esto con la finalidad de hacer una comparación adecuada.

También se evaluó la eficiencia de la lista tabú para la instancia de 200 vértices, esto determina que tan eficiente es el algoritmo propuesto de acuerdo al tamaño de lista tabú. En la figura 4-17 se muestra el tiempo de ejecución para cada tamaño de la lista tabú. Se puede observar que al incrementar el tamaño de la lista el tiempo de ejecución va decrementando.

Esto es lógico, ya que cada vez que el algoritmo de metrópolis entra a hacer un movimiento para encontrar una solución vecina primero se busca en la lista tabú, si existe dicho movimiento, se rechaza y entonces la solución vecina se queda con la solución anterior entra a evaluar la función objetivo, pero al evaluarse en la función de costo esta se cumple ya que son iguales, por lo tanto estamos ganando en eficiencia por no estar evaluando la función de Boltzmann, ya que al evaluar ésta implica un costo computacional.

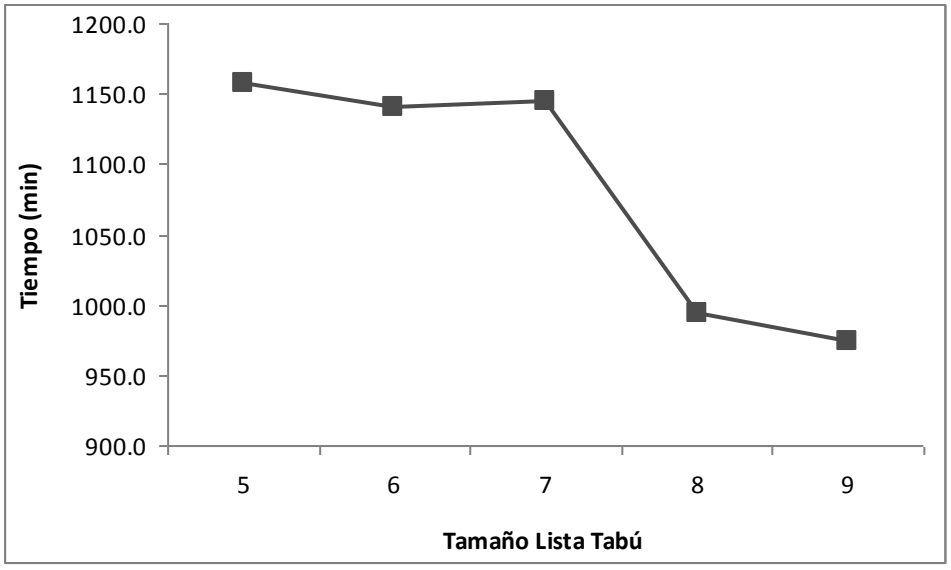


Figura 4-17 Tiempo de ejecución para cada tamaño de la lista tabú con 30 ejecuciones.

Capítulo 5

Conclusiones y Trabajos Futuros

5.1 Conclusiones

Se concluye que la estructura de vecindad híbrida es adecuada para el problema del árbol de expansión mínima, comparando los resultados obtenidos en las pruebas experimentales de estructuras de vecindad realizada para las instancias de 100 y 200. En cuanto a la eficiencia se encontró que la estructura híbrida resulta ser competitiva con respecto al resto de las estructuras analizadas.

De acuerdo a las pruebas experimentales realizadas al algoritmo de Recocido Simulado, utilizando una estructura de vecindad híbrida y una lista tabú, se puede concluir que el algoritmo propuesto es mejor en eficacia con respecto al algoritmo de búsqueda local iterada, debido a la calidad de soluciones encontradas y en cuanto a eficiencia también se demuestra que

es mejor en más del 80% en tiempo de ejecución para la instancia de 200 vértices.

La implementación de una lista tabú al Recocido Simulado permitió mejorar aún más la eficacia. También la eficiencia se mejoró ya que la lista tabú permite dentro del algoritmo Metrópolis un ahorro en la evaluación de la función de probabilidad de boltzmann cada vez que se tiene un movimiento tabú.

5.2 Trabajos Futuros

La propuesta de solución desarrollada durante este trabajo de investigación, servirá como base para las siguientes actividades a realizar como trabajo futuro:

1. Implementar el problema del árbol de expansión mínima a un área en específico por ejemplo para el sistema de redes hidráulicas, redes de telecomunicaciones, sistema de transporte u otra área de investigación.
2. Trabajar con estructuras híbridas de vecindad en metaheurísticas que realicen exploración del espacio de soluciones.
3. Propuesta de una metaheurística híbrida que pueda explorar y explotar el espacio de soluciones para el problema del árbol de

expansión mínima aplicado a un problema real. La explotación del espacio de soluciones se haría con Recocido Simulado.

Referencias

Assad, A., Xu, W. (1991).The Quadratic Minimum Spanning Tree Problem. Naval Research Logistics, 39, pp. 399-417.

Arajy, Y. Z., Abdullah, S. (2010). Hybrid Variable Neighbourhood Search Algorithm for Attribute Reduction in Rough Set Theory, Data Mining and Optimisation Research Group (DMO) Centre for Artificial Intelligence Technology Universiti Kebangsaan Malaysia, 43600 Bangi Selangor, Malaysia, ISSN: 978-1-4244-8136-1.

Ausiello, G., Crescenzi, P., Gambosi, G., Kann, V., Marchetti-Spaccamela, A., Protasi, M. (1999). Complexity and approximation: Combinatorial Optimization Problems and their Aproximability Properties. Springer-Verlag. by Simulated Annealing. IBM Research Report RC 9355.

Bertsimas, D. J. (1990).The Probabilistic Minimum Spanning Tree Problem, NETWORKS, Vol. 20, pp. 245-275.

Cerny, V. (1985). A Thermodynamical Approach to the Travelling Salesman Problem: an Efficient Simulation Algorithm. J. of Optimization Theory and Applic.,45: 41-55.

Yoon-Teck, B., Chin-Kuan, H., Hong-Tat, E. (2008). Ant Colony Optimization Approaches to the Degree-constrained Minimum Spanning Tree Problem. Journal of Information Science and Engineering, Vol. 24 No. 4, pp. 1081-1094.

Cook, W. J., Cunningham, W. H., Pulleyblank, W. R., Schrijver, A. (1997). Combinatorial Optimization; John Wiley & Sons; 1er edition, ISBN 047155894X. November 12, 1997.

Cruz-Chávez, M.A. (2005). Cooperación de Procesos para el Problema de Jop Shop Scheduling Aplicando Recocido Simulado, Tesis de Doctorado, Marzo 2005.

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.

Cruz-Chávez, M.A, Martínez-Oropeza, A., Serna Barquera, S. A. (2010). Neighborhood Hybrid Structure for Discrete Optimization Problems, Electronics, Robotics and Automotive Mechanics Conference, CERMA2010, IEEE Computer Society, ISBN 978-0-7695-4204-1, pp. 108-113.

Cruz-Chávez, M.A., Díaz-Parra, O., Juárez-Romero, D., Martínez-Rangel, M.G. (2008). Memetic Algorithm Based on a Constraint Satisfaction Technique for VRPTW, Lecture Notes in Artificial Intelligence, Springer Verlag Pub., Berlin Heidelberg, ISSN: 0302-9743, Vol. 5097, pp. 376-387.

Cruz-Chávez, M. A., Juárez-Pérez, F. (2010). 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.

Cruz-Chávez, M. A., Juárez-Pérez, F., Ávila-Melgar, E. Y., Martínez-Oropeza A.(2009). Simulated Annealing Algorithm for the Weighted Unrelated Parallel Machines Problem. Cerma 2009. ISBN-13:978-0-7695-3799-3. pp. 94.

Cruz-Chávez, M.A., Frausto-Solís, J., Zavala-Díaz, J.C., Sanvicente-Sánchez, H., Cruz-Rosales, M. H. (2006). A Simulated Annealing Algorithm with Cooperative Processes for Scheduling Problems. LNCS. Springer, Heidelberg, ISSN: 0302-9743.

Gilbert, E.N., Pollak, H.O. (1968). Steiner Minimal Trees. SIAM Journal on Applied Mathematics, 16(1): 129.

Dijkstra, E.W., (1959). A Note on Two Problems in Connexion with Graphs, Numer. Math.1, pp. 269-271.

Glover, F., Laguna, M. (1997). *Tabu Search*. Kluwer Academic Publisher.

Glover, F.(1989). Tabu Search. Part I. ORSA Journal of Computing. Vol. 1. No. 3. pp. 190 – 206.

Glover, F. (1990). Tabu Search: Part II. ORSA Journal on Computing.

Hansen, P., Mladenovic, N. (1997). Variable neighborhood search, Computers and Operations Research 24, pp 1097- 110.

Hansen, P., Mladenovic, N. (2001). Variable Neighborhood Search: Principles and Applications, Montréal, Canada, European Journal of Operational Research 130, pp 449-467.

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

Hochbaum, D. S., Moreno-Centeno, E. (2008). Network Flows and Graphs. Berkeley IEOR 266. Aug 28 - Sep 11, 2008.

Hu, B., Leitner, M., Raidl, G. R. (2005). Computing Generalized Minimum Spanning Trees with Variable Neighborhood Search, Institute of Computer Graphics and Algorithms Vienna University of Technology, Vienna, Austria, MEC-VNS: 18th Mini Euro Conference on VNS.

Ishii, H., Shiode, S., Nishida, T., Namasuya, Y. (1981). Stochastic Spanning Tree Problem, Discrete Applied Mathematics, Vol. 3, pp. 263-273.

Kirkpatrick, S., Gelatt, C.D., Vecchi, M. P. (1983). Optimization by Simulated Annealing. Science, Vol. 220, pp 671-680. 1983.

Kruskal, J.B. (1956). On the Shortest Spanning Tree of a Graph and the Traveling Salesman Problem, Proc. Amer. Math. Soc, vol. 7, 48-50.

Maleq, K. M. A. (2007). Distributed Approximation Algorithms for Minimum Spanning Trees and other Related Problems with Applications to Wireless Ad Hoc Networks, Thesis Doctored, December 2007.

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

Martínez, O. A. (2010). Solución al Problema de Máquinas en Paralelo no Relacionadas mediante un Algoritmo de Colonia de Hormigas, Tesis de Maestría, Agosto 2010.

Martínez, R. M. G. (2008). Algoritmo de Recocido Simulado Paralelizado, Aplicado al Problema de Asignación de Recursos en un Taller de Manufactura Flexible sujeto a Disposiciones de Tiempo, Tesis de Doctorado, Diciembre 2008.

Metrópolis, N., Rosenbluth, A. W., Rosenbluth, M., Teller, A.H., Teller, E. (1953). Equation of State Calculations by Fast Computating Machines. J.Chem.Phys.Vol.21, pp. 1087-1092, 1953.

Michiels, W., Aarts, E., Korst, J. (1998). Theoretical Aspects of Local Search, Philips Research High Tech Campus 345656 AE Eindhoven The Netherlands and Eindhoven University of Technology P.O. Box 513 5600MB Eindhoven The Netherlands.

Mitsuo, G., Runwei, C. (1944). Genetic Algorithms & Engineering Design, ISBN 0-471-12741-8.

Myung, Y.S., Lee, C.H., Tcha, D.W. (1995). On the Generalized Minimum Spanning Tree Problem, Networks, Vol. 26, pp. 231-241.

Narula, S.C., Ho, C.A. (1980). Degree-Constrained Minimum Spanning Tree, Computer and Operation Research, vol. 7, pp.239-249.

Boruvka, O. (1926). jistém problému minimálnín. Práce Mor. Prírodovêd. Spol. v Brnê (Acta Societ. Scient Natur. Moravicae)3(1926), 37-58.

Osman, I.H., Kelly, J.P. (1996). Meta-Heuristics: Theory and Applications, 39 Kluwer Academic Publishers. ISBN: 0792397002. USA.

Pannell, D. J. (2009).Sensitivity analysis: strategies, methods, concepts, examples.

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

Van Laarhoven P. J. M., Aarts, E. H. L., Lenstra, J. K. (1992).Job Shop Scheduling by Simulated Annealing, Operations Research, Vol. 40, No. 1, pp. 113-125.

Pinedo, M. (2008). Scheduling Theory, Algorithms and Systems.Vol.1 3rd Edition, ISBN 9780387789347.

Pop, P., Sabo, C., Pop, S. C., Craciun, M. V. (2007). Solving the generalized minimum spanning tree problem with simulated annealing. Mathematics Subject Classification. 90B10, 90C10, 90C27, 90C39.2007, CREATIVE MATH. & INF., pp. 42 – 53.

Prim, R.C., (1957). Shortest Connection Networks and Some Generalizations, Bell System Technical Journal, vol. 36, pp.1389-1401.

Reimann, M., Laumanns, M. (2009). A hybrid ACO algorithm for the Capacitated Minimum Spanning Tree Problem, Institute for Operations Research, Swiss Federal Institute of Technology Zurich, Clausiusstrasse 47, CH-8092 Zurich, Switzerland.

Sollin, M. (1965). "Le tracé de canalisation" (in French).Programming Games and Transportation Networks.

Talbi, El-Ghazali. (2009). Metaheuristics: from design to implementation. ISBN 978-0-470-27858-1.

Torres, D. J. F., Velez, G. M. C. (2007). Algoritmo de Recocido Simulado para la Descomposición Robusta del Horizonte de Tiempo en Problemas de Planeación de Producción, Ingeniería y Ciencias, ISSN 1794-9165, vol. 3, Num. 5, pp.7-27. Vol. 2, pp. 4-32.

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

Apéndice A

De acuerdo a Aart y Van Laarhoven (1985), la complejidad computacional del algoritmo de Recocido Simulado se presenta en la siguiente expresión (1)

$$O(\tau L \ln|R|) \qquad (1)$$

donde τ es el tiempo para generar y evaluar una posible solución, L es la longitud de la cadena de Markov y R es el espacio de soluciones del problema que se evalúa.

Para el caso del problema del árbol de expansión mínima, el número máximo de pasos requeridos para generar y evaluar una solución tiene la complejidad de $O(n^5)$ donde n representa el número de vértices en el grafo que simboliza al problema. En el caso del problema MST L es (n + 1) es decir el tamaño de la vecindad y el espacio de soluciones para este problema es n!, entonces R tiene una complejidad de $R \in O(n!)$. Por lo tanto la complejidad Asintótica del algoritmo de Recocido Simulado para el problema del árbol de expansión mínima haciendo las sustituciones de τ, L y R en la expresión (1), se presenta en (2).

$$O\left((n^5)(n+1)\ln(n!)\right) \qquad (2)$$

Si se toma $\ln(n!)$ y pasamos al límite cuando *n* tiende al infinito, entonces el límite es $n \ln n$ y se tiene la expresión (3).

$$O\left((n^6 + n^5)n\ln n)\right) \qquad (3)$$

Por lo tanto la complejidad asintótica del algoritmo de Recosido Simulado para el problema del árbol de mínima expansión se presenta en la expresión (4)

$$O\left((n^7 + n^6)\ln n\right) \qquad (4)$$

En la figura A-1 muestra la complejidad de ejecución del algoritmo, donde el eje de n representa en número de vértices que contiene un grafo y f(n) representa la complejidad asintótica del problema tratado es decir cuando n tiende a infinito número de instrucciones durante la ejecución del algoritmo. Donde podemos observar que al aumentar el tamaño de n el número de instrucciones también aumenta.

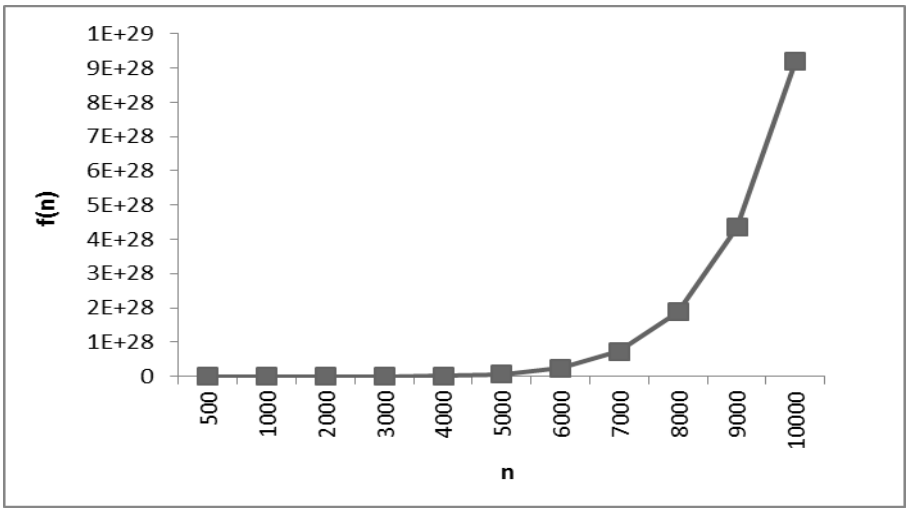


Figura A-1. Complejidad Asintótica para el algoritmo de Recocido Simulado

Glosario de Términos

Análisis de Sensibilidad: Es un componente importante en la construcción modelos matemáticos, computacionales y simulación.

Búsqueda Tabú: Es una de las técnica más utilizadas en algoritmos no determinísticos, proviene de la inteligencia artificial y está basada en una memoria adaptativa que le permite explorar un espacio de soluciones de manera eficiente a través de la estructura de vecindad [Glover, 1989]. TS considera dos tipos de memoria a largo y corto plazo.

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

Complejidad Temporal: Tiempo que necesita el algoritmo para ejecutarse.

Costo: En el caso del problema de MST, es el costo de cada arista.

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

Heurística: Procedimiento intuitivo bien definido que proporciona buenas soluciones aproximadas a problemas difíciles de resolver sin garantizar la optimalidad, en un tiempo computacional razonable [Wetzel, 1983].

Lista tabú: Utilizada contiene los movimientos que se realizan entre vértices para encontrar una solución vecina, esta lista tiene la función de evitar que una solución sea visitada nuevamente en base a una repetición del movimiento.

Metaheurísticas: Métodos aproximados que mejoran procedimientos heurísticos, los cuales son diseñados para ser aplicados a problemas considerados difíciles de resolver, donde las heurísticas no son eficientes [Osman y Nelly, 1996].

Recocido Simulado (SA): Es una metaheurística de búsqueda aleatoria utilizada en la solución de problemas de optimización combinatoria [Kirkpatrick et al., 1983]. Se basa en la analogía del proceso de recocido, es cuando un material se somete a un calentamiento a temperatura muy alta llegando al punto de fusión y después es enfriado gradualmente, sus

moléculas se acomodan de tal forma que la energía potencial de la configuración de las moléculas es mínima.

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.

Vecindad: Es el conjunto de soluciones que pueden ser alcanzadas desde una solución dada por medio de un movimiento.

Problema clase P: Es un conjunto de todos los problemas de decisión que pueden ser resueltos por un algoritmo determinístico en tiempo polinomial.

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

Tiempo polinomial: Se dice que un algoritmo puede ser resuelto en tiempo polinomial sí su función de complejidad es de orden *O(n)*, es decir, que puede ser representado por un polinomio.