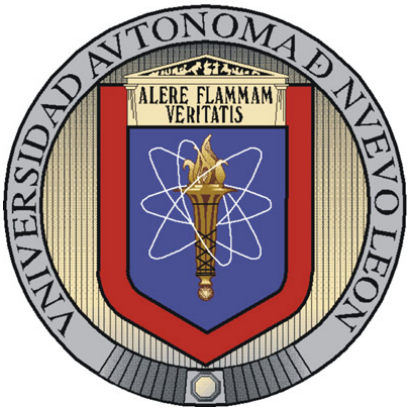
Universidad Autónoma de Nuevo León

Facultad de Ingeniería Mecánica y Eléctrica

División de Estudios de Posgrado



Optimización de la trayectoria de un agente con medidas de desempeño asociadas

por

Ing. David Roberto Valenzuela Vega

en opción al grado de

Maestría en Ciencias

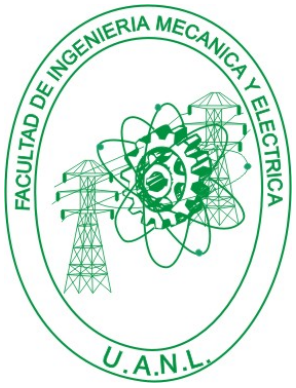
en Ingeniería de Sistemas

San Nicolás de los Garza, Nuevo León marzo 2010

Universidad Autónoma de Nuevo León

Facultad de Ingeniería Mecánica y Eléctrica

División de Estudios de Posgrado



Optimización de la trayectoria de un agente con medidas de desempeño asociadas

por

Ing. David Roberto Valenzuela Vega

en opción al grado de

Maestría en Ciencias

en Ingeniería de Sistemas

San Nicolás de los Garza, Nuevo León marzo 2010

Universidad Autónoma de Nuevo León

Facultad de Ingeniería Mecánica y Eléctrica

División de Estudios de Posgrado

Los miembros del Comité de Tesis recomendamos que la Tesis Optimización de la trayectoria de un agente con medidas de desempeño asociadas, realizada por el alumno Ing. David Roberto Valenzuela Vega, con número de matrícula 01474028, sea aceptada para su defensa como opción al grado de Maestría en Ciencias en Ingeniería de Sistemas.

El Comité de Tesis

Dr. J. Arturo Berrones Santos
Asesor

Dra. Satu Elisa Schaeffer
Revisor

Dr. Luis Torres Treviño
Revisor

Vo. Bo.

Dr. Moisés Hinojosa Rivera
División de Estudios de Posgrado

San Nicolás de los Garza, Nuevo León, marzo 2010

Índice general

Lista de figuras

Lista de tablas

Agradecimientos

Gracias a la coordinación del Programa de Posgrado en Ingeniería de Sistemas (PISIS) por darme la oportunidad de formar parte de éste. A la Universidad Autónoma de Nuevo León y a la Facultad de Ingeniería Mecánica y Eléctrica por el apoyo que me han dado durante la realización de la maestría. Al Consejo Nacional de Ciencia y Tecnología (CONACyT) por la beca de manutención otorgada durante mis estudios de maestría.

A los miembros del comité de tesis, Dr. J. Arturo Berrones Santos, Dra. Satu Elisa Schaeffer y Dr. Luis Torres Treviño, por sus recomendaciones para el mejoramiento de este trabajo.

A todos los profesores del PISIS por su ejemplo y dedicación. A mis compañeros M. C. Efraín Soto Apolinar y Lic. Jabneel Rocío Maldonado Flores por las incontables horas de trabajo, las oportunas críticas y correcciones y la inmejorable compañia.

Personalmente, agradezco a la M. I. Margarita Loredo Cancino por su apoyo incondicional durante el desarrollo de mis estudios de posgrado.

Resumen

Ing. David Roberto Valenzuela Vega.

Candidato para el grado de Maestro en Ingeniería
con especialidad en Ingeniería de Sistemas.

Universidad Autónoma de Nuevo León.
Facultad de Ingeniería Mecánica y Eléctrica.

Título del estudio:

Optimización de la trayectoria de un agente con medidas de desempeño asociadas

Número de páginas: 74.

Objetivos y método de estudio: El presente trabajo trata sobre la formulación de un modelo en el que se busca definir la trayectoria de un agente sobre una superficie conocida; la cual está sujeta a restricciones de paso estáticas (obstáculos) y a criterios de funcionalidad. Con el fin de representar el problema de una manera abstracta y manejable se genera un grafo tal que contenga los elementos que se desean controlar, es decir, la trayectoria y los costos asociados a la misma. Estos costos están relacionados directamente con los criterios de funcionalidad y por lo tanto, se buscará minimizarlos. Además se desea abordar la problemática asociada a la transmisión de información entre el ambiente, el agente y el método de solución.

Contribuciones y conclusiones: Se propone el uso de una variante del problema del camino hamiltoniano de menor costo para modelar el problema de cobertura. El problema se modela a partir del grafo asociado al área de interés. El modelo generado puede resolver escenarios en los que los obstáculos están dispuestos de forma irregular al permitir el paso sobre regiones ya visitadas. Dado que la modificación del problema del camino hamiltoniano de menor costo y la estructura de las instancias producen una mayor dificultad a la hora de resolver el problema, se propone una técnica para disminuir el tiempo de cómputo a través de la adición de un componente aleatorio.

Los principales resultados fueron presentados en la XIV Escuela Latinoamericana de Verano de Investigación de Operaciones, en El Fuerte, Sinaloa, México, en Agosto de 2009. En el seminario de TI & Software del Centro de Innovación, Investigación y Desarrollo en Ingeniería y Tecnología el 13 de Mayo de 2009. Y en los Seminarios de Investigación del Programa de Posgrado en Ingeniería de Sistemas el 19 de Marzo y el 3 de Septiembre de 2009.

Firma del asesor: _____

Dr. J. Arturo Berrones Santos

Capítulo 1

Introducción

1.1 Relevancia

El presente proyecto nace de la colaboración entre el Posgrado de Ingeniería de Sistemas (PISIS) y el área de mecatrónica del Centro de Innovación, Investigación y Desarrollo en Ingeniería y Tecnología (CIIDIT) de la Universidad Autónoma de Nuevo León (UANL). El trabajo que se lleva a cabo en el área de mecatrónica está relacionado con el desarrollo de proyectos de robótica, autotrónica y domótica entre otros, siendo el primero de ellos, la robótica, el punto de encuentro entre dicha área y el trabajo realizado en el PISIS.

El trabajo que se realiza dentro del PISIS se puede definir en general como el uso de técnicas cuantitativas para la solución de problemas de decisión. Así dentro de la robótica y, más específicamente, dentro del área de control y la inteligencia artificial se vislumbra el potencial del trabajo en conjunto entre ambas disciplinas.

Dentro del área relativamente nueva de la inteligencia artificial existen un sinnúmero de campos con problemas por resolver y a su vez problemas resueltos pero con un potencial de mejora considerable. Uno de estos campos es el del llamado *planning* o *covering*, el cual busca definir la trayectoria de un robot sobre una superficie dada. Dicha trayectoria está sujeta a restricciones de paso (obstáculos).

Sin embargo, aunque existen algoritmos diseñados para cubrir un área conocida, el interés de este proyecto es abordar el problema de *covering* cuando éste tiene

asociado criterios de funcionalidad medibles y, que además, resulte de interés llevar un control de ellos.

Un punto importante de remarcar es que el presente proyecto busca desarrollar y/o utilizar técnicas que resulten económicas tanto desde el punto de vista computacional como desde el monetario.

1.2 Descripción del problema

El problema que se plantea es un problema de construcción, en este caso, de la trayectoria que ha de seguir un robot para cubrir completamente o la mayor parte posible de una superficie dada. Sin embargo, no sólo es de interés que el robot cubra con su trayectoria total o parcialmente el área, sino también que sea susceptible a una especie de control sobre la trayectoria generada, respondiendo ésta a necesidades específicas del usuario, tales como tiempo de operación, acceso preferencial a áreas de interés, o por el contrario, el control sobre el acceso a áreas restringidas. Por otra parte el control sobre la trayectoria puede obedecer también a necesidades del robot, tal como la economía de movimientos para el ahorro de energía.

Así, es necesaria la construcción de un modelo que refleje las restricciones de paso y nos permita actuar sobre las restricciones de funcionalidad. Además debe contar con un modelado que permita controlar ciertas características, tales como su costo total, pasos dentro de la trayectoria, o cualquier otra medida de desempeño que se quiera optimizar. Lo anterior deriva en la necesidad de plantearlo por medio de un modelo matemático, para que sea susceptible a la optimización.

El uso de las técnicas adecuadas de optimización tiene como fin tanto incrementar los alcances de los procedimientos utilizados en el campo de la inteligencia artificial, como ofrecer técnicas alternativas para la toma de decisiones al enfrentar en el futuro otros problemas, como por ejemplo, trayectorias dentro de ambientes desconocidos y dinámicos.

Capítulo 2

Antecedentes

En este capítulo se abordan los temas que servirán de base para el desarrollo del presente trabajo. En la primera sección se presenta un resumen de las principales metodologías utilizadas al abordar el problema de cobertura en robótica. A lo largo de la sección 2.2 se presentan las definiciones y conceptos referentes a la teoría de grafos, tales que serán fundamentales al momento de modelar el problema. En las dos secciones siguientes se presentan las herramientas que serán utilizadas en la solución del problema. Por último se presenta una introducción a la teoría de la complejidad computacional, que servirá para definir la dificultad del problema abordado.

2.1 Problema de cobertura en la robótica

El problema de cobertura en robótica [3] surge en diversas aplicaciones, tales como detección de minas, remoción de nieve, procesos de pintura automatizados y búsquedas de imperfecciones sobre una superficie, entre otros. El determinar este tipo de trayectoria no puede ser definido por el enfoque tradicional de 'punto a punto', ya que este tipo de problemas requieren no solo llegar a un punto final, sino que además es necesario que hayan pasado por todos los puntos antes de llegar a su meta. Este problema está relacionado con el *problema de cobertura del vendedor* (*covering salesman problem*), que es una variante del *problema del agente viajero* en la que en lugar de tener que visitar cada ciudad, el agente debe visitar a un vecindario de cada una de las ciudades tal que la distancia recorrida por el agente sea mínima. En este caso,

se desea que todos los vecindarios sean visitados tal que la distancia recorrida por el agente sea mínima.

En un principio se abordó el problema por medio de heurísticas que involucraban una serie de reglas simples. Desafortunadamente estas metodologías no ofrecen alguna garantía respecto a que la operación de cobertura sea realizada en su totalidad. De hecho, uno de los mayores logros de algunos de los trabajos recientes relacionados con el problema de cobertura es el garantizar que la trayectoria generada cubrirá completamente el espacio libre.

Con el fin de garantizar de algún modo la total cobertura, varios enfoques para la planeación de trayectorias utilizan, ya sea implícita o explícitamente, una **descomposición celular** del espacio disponible. La descomposición celular divide la región o área de interés en celdas, las cuales son facilmente visitadas en lo individual. Así, la total cobertura del área es alcanzada al asegurarse que el robot ha visitado todas las celdas generadas en la descomposición.

Los esquemas de cobertura completa tienen la ventaja de que no dejan duda alguna de que ha sido cubierta totalmente el área de interés. Sin embargo, estos enfoques requieren más poder computacional y mejores sensores que los que regularmente un robot simple puede tener. Es por esto que desde el punto de vista costo/beneficio los métodos aleatorizados siguen teniendo aceptación.

Los algoritmos de cobertura para robots móviles pueden clasificarse en **heurísticos** y **completos**. A su vez, los algoritmos completos pueden dividirse respecto a la descomposición que hacen del espacio libre. De esta manera se tienen los algoritmos de **descomposición celular aproximada**, los algoritmos de **descomposición celular semi-aproximada**, y los algoritmos de **descomposición celular exacta**.

Los algoritmos heurísticos son aquellos en los que el robot es equipado con una serie de reglas simples. La combinación y jerarquización de dichas reglas resultan en acciones más complejas, como la exploración por ejemplo. Como ya se ha mencionado, este enfoque no garantiza la cobertura total, aunque tienen la ventaja de

ser muy económicos al implementarlos, sobre todo porque no requieren sensores de posicionamiento.

La **descomposición celular aproximada** se basa en la representación del espacio libre por medio de una cuadrícula. En este esquema todas las celdas tienen el mismo tamaño y forma. De esta forma la unión de todas las celdas aproxima el área libre de interés. Esta idea fue desarrollada inicialmente por Elfes [8] y Moravac [23]. Aquí se asume normalmente que una celda es cubierta cuando el robot visita dicha celda; típicamente la celda es del tamaño del robot o de la herramienta que ejecuta la acción. Cuando el robot ha visitado cada una de las celdas entonces ha sido cubierta toda el área.

Zelinsky et al. [27] usaron un algoritmo de frente de onda (*wavefront algorithm*) para determinar la trayectoria de cobertura. El **algoritmo de frente de onda** asigna inicialmente un 0 a la celda objetivo y un 1 a todas las celdas que la rodean. En seguida se marcan con 2 todas las celdas sin marcar que rodean a las celdas con etiqueta 1. Este proceso se repite hasta que se alcanza la celda de inicio (previamente identificada). Una vez hecho esto, el robot usa un descenso de gradiente a lo largo de esta función de potencial numérico [14] para encontrar el camino. Una ventaja con la que no cuentan otros esquemas es que, además de obtener la trayectoria, es posible indicar un punto inicial y uno final para la trayectoria. El algoritmo puede también generar trayectorias seguras a través de sus funciones de costo y así optimizar el proceso de cobertura.

En cuanto a la generación de trayectorias óptimas Gabriely y Rimon [10] tienen en desarrollo el algoritmo *STC* (*Spanning tree covering*), además del híbrido en el que se utiliza el concepto de **colonia de hormigas** [6]. En esta última versión el robot no tiene conocimiento previo del medio y necesita dejar marcas (al estilo de feromonas) durante el proceso de cobertura. Una suposición que hace este método es que el espacio libre nunca será más angosto que el doble del diámetro del robot o de la herramienta que éste utiliza.

La **descomposición celular semi-aproximada** se basa en una discretización parcial del espacio libre, donde las celdas están fijas en lo ancho, pero la parte superior e inferior pueden tener cualquier forma. Una ventaja de este método es que puede funcionar en medios conectados de manera simple y de forma no trivial.

La **descomposición celular exacta** utiliza el conjunto de regiones que no se intersectan entre sí, cada una de ellas llamadas celdas, tal que la unión de todas las celdas generan el área de interés. Típicamente el robot puede cubrir cada celda con movimientos de adelanto y retroceso simples, así la cobertura completa se realiza al visitar y cubrir cada una de las celdas. Una técnica de descomposición comúnmente utilizada es la descomposición trapezoidal, en la que, como su nombre lo dice, el área es dividida en celdas trapezoidales. Este método supone el conocimiento del área de interés.

2.2 Teoría de grafos

Como se menciona en Lawler [17], se dice que la teoría de grafos fue fundada en 1736 cuando Euler modeló el famoso problema, hasta ese momento no resuelto, conocido como el **Problema de los puentes de Königsberg**, dos islas que estaban unidas entre sí y hacia las orillas del Rio Pregel por medio de siete puentes. La pregunta fue, si era posible empezar en cualquiera de las cuatro áreas de tierra, caminar a través de cada uno de los siete puentes exactamente una vez y regresar al punto de inicio.

La pregunta general, para un grafo dado G, es si existe un **camino cerrado** que contenga cada arco exactamente una vez. A ese camino, si existe, se le llama un **camino euleriano**, y se dice que el grafo es un **grafo de euler** o **euleriano**. Euler fue capaz de contestar a esta pregunta, siendo una respuesta negativa para el grafo de Königsberg, que se muestra en la figura 2.1 y resolvió el problema para todos los grafos de la siguiente manera.

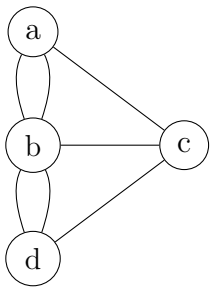


Figura 2.1: Grafo de Königsberg.

Teorema 2.1 *Un grafo (o multigrafo) G es Euleriano si y solo si G es conexo y cada nodo de G tiene un grado par.*

Por otra parte Hamilton quien inventó el juego icosiano (figura 2.2), ahora conocido como juego de Hamilton, que involucraba encontrar un ciclo hamiltoniano en las aristas de un grafo de un dodecaedro, planteó este nuevo problema en la

teoría de grafos. Así, en su honor, se le llama a un ciclo que pasa por cada nodo del grafo exactamente una vez, **ciclo hamiltoniano**, y al grafo que lo contiene **grafo de hamilton** o **hamiltoniano**. Hamilton resolvió el problema para el juego icosiano (figura 2.3); sin embargo, esta solución no se generaliza para todos los grafos. En contraste con las sencillas condiciones suficientes y necesarias para un **grafo de euler**, para un **grafo de hamilton**, es aún un desafío encontrar una caracterización efectiva.

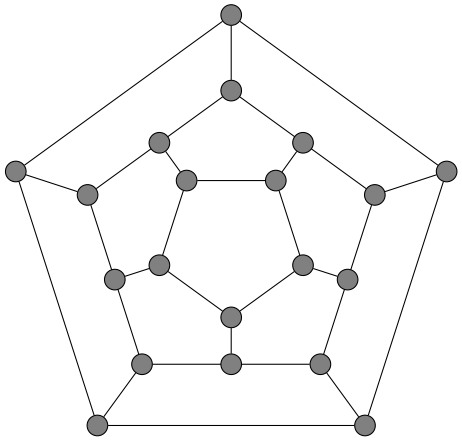


Figura 2.2: Juego icosiano.

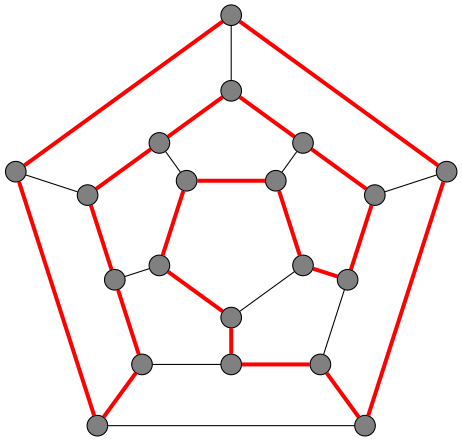


Figura 2.3: Solución del juego icosiano.

En este punto es necesario mostrar parte de la terminología que se usará. Tomando como referencia Lesniak[18] y Diestel[5] dado un grafo $G = (N, A)$, donde

N es el conjunto de nodos y A es el conjunto de arcos, se tiene que:

- **Caminata** (*walk*). Es la sucesión de nodos y arcos que unen el nodo u con el nodo v.
- **Caminata abierta** (*open walk*). Es una caminata en la cual el nodo inicial u es diferente del nodo final v.
- **Caminata cerrada** (*close walk*). Es una caminata en la cual el nodo inicial u es igual que el nodo final v.
- **Camino** (*path*). Es una caminata en la que no se repiten los nodos.
- **Ciclo**. Es un camino en el que el nodo inicial u es igual que el nodo final v.
- **Caminata de expansión** (*Spanning walk*). Es una caminata en la que se visitan todos los nodos del conjunto N por lo menos una vez.
- **Camino de expansión** (*Spanning path*). Es una caminata en la que se visitan todos los nodos del conjunto N exactamente una vez.

La teoría de grafos se refiere al estudio de las propiedades de las redes o grafos; en Orlin[1] se define un grafo o red dirigida de la siguiente forma:

Definición 2.2 *Un grafo dirigido $G = (N, A)$ consiste en el conjunto N de nodos y el conjunto A de arcos cuyos elementos son los pares ordenados de nodos distintos.*

Los arcos pueden tener valores asociados los cuales denotan algún tipo de atributo, como costo, capacidad o distancia, por citar algunos ejemplos.

Un grafo suele ser la representación abstracta de una serie de objetos en la que algunos de esos objetos están enlazados entre sí. Así, la forma gráfica de éste suele verse como se presenta en la figura 2.4 .

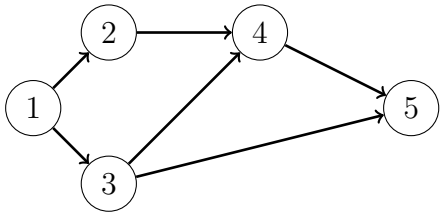


Figura 2.4: Ejemplo: grafo dirigido.

En el ejemplo que se muestra se tienen 5 nodos y 6 arcos, por lo que $N = 5$ y $A = 6$. Cada uno de estos arcos se puede identificar por medio del par de nodos asociados al mismo, es decir, su nodo de inicio y su nodo final. De esta forma podemos denominar a cada uno de los elementos del conjunto A con el parámetro a_{ij} donde i es el nodo de inicio y j el nodo final. Agregando la nomenclatura anterior el grafo se puede observar en la figura 2.5.

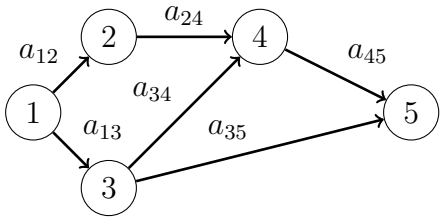


Figura 2.5: Ejemplo: grafo dirigido con aristas ponderadas.

Una vez que han sido identificadas las aristas es sencillo asignarles un valor a cado uno; asociando estos valores a un atributo en particular.

2.3 Optimización combinatoria

Optimización [24] significa maximizar (o minimizar) una función compuesta por un conjunto de variables sujetas a una serie de restricciones. En cuanto a la denominación *discreta*, *combinatoria* o *entera* se refiere a que una o todas las variables pertenecen a un conjunto discreto, típicamente a un subconjunto de enteros. Este tipo de restricciones permiten modelar fenómenos en los que las alternativas no forman un continuo.

Un problema en el que las variables en la función tienen permitido tomar valores continuos es llamado problema de **programación lineal (LP)** [2]. Por otra parte, si una o varias de las variables están forzadas a tomar valores enteros entonces es denominado **problema entero mixto (MIP)**. Cuando todas las variables dentro de la función están forzadas a tomar valores enteros entonces es un problema **lineal entero puro (IP)**.

En varios modelos las variables enteras son utilizadas para representar relaciones lógicas, de manera que éstas quedan sujetas a los valores 0 y 1. Así, resulta que se tiene un problema **0-1 MIP** ó **0-1 IP** según sea el caso llamado problema de **optimización combinatoria CP**. La forma general en la que se define un problema de optimización combinatoria es como sigue [24]. Sea $N = \{1, \ldots, n\}$ un conjunto finito y sea $c = (c_1, \ldots, c_n)$ con un n-vector. Para $F \subseteq N$ se define que $c(F) = \sum_{j \in F} c_j$. Supóngase que se es dada la colección de subconjuntos K de N. El problema de optimización combinatoria es:

$$\max\{c(F) : F \in K\}.$$

Los problemas de optimización discreta abundan en la vida diaria. Una importante y amplia gama de aplicaciones están relacionadas con el manejo y uso eficiente de los recursos con la finalidad de incrementar la productividad. Como es el caso de la distribución de mercancías, planeación de la producción, secuenciamiento de máquinas, localización de inmuebles, reparto de carteras, diseño de redes de comunicacíon y

transporte entre otras.

Un importante y muy común uso de las variables 0-1 es el de representar una opción **binaria**. Considérese un evento o acción que puede o no ocurrir, y supóngase que es parte del problema decidir entre estas dos opciones. Para modelar tal dicotomía es usada la variable binaria x y se tiene que

$$x = \begin{cases} 1, & \text{si el evento ocurre y} \\ 0, & \text{si el evento no ocurre.} \end{cases}$$

El evento puede representar varias cosas, dependiendo de la situación específica. Algunos ejemplos bien conocidos son el problema de la mochila (*knapsack problem*), el problema de asignación (*assignment and matching problems*), el problema de cobertura (*set-covering*) y el problema del agente viajero (*salesman problem*).

2.4 Método de ramificación

El método de ramificación y corte es un método utilizado en el área de la optimización combinatoria para resolver problemas lineales enteros, esto es, problemas de programación lineal en los que algunas o todas las variables de decisión están restringidas a tomar valores enteros. Este método es un híbrido entre el **método de ramificación y acotamiento** y los **métodos de planos cortantes** [15].

2.4.1 Método de ramificación y acotamiento

El **método de ramificación y acotamiento** es una técnica para la completa enumeración de todas las posibles soluciones de un problema sin tener que considerarlas una por una. El método de ramificación y acotamiento es uno de los mejores esquemas conocidos para obtener una solución óptima en problemas de optimización combinatoria pertenecientes a la clase NP–duro. Fue propuesto por Land y Doig [16] en 1960 y aplicado a un problema *TSP* por primera vez por Little et al. [19] en 1963.

Para aplicar el **método de ramificación y acotamiento** a un problema de optimización combinatoria, son necesarios dos pasos:

- *Ramificación* : dado un subconjunto de las soluciones posibles, éste debe poder ser dividido en al menos dos subconjuntos no vacíos.
- *Acotamiento* : para un subconjunto obtenido del paso anterior, para cualquier solución dentro del subconjunto debe ser posible calcular una cota inferior (en el caso de minimizar) del costo asociado a dicha solución.

Teniendo como entrada una instancia del problema, se obtiene como salida la solución óptima S^*. El procedimiento general del algoritmo se muestra en la tabla 2.1.

1. Se define el árbol inicial $T := (\{S\}, \emptyset)$, donde S es el conjunto de todas las soluciones.

   Se marca S como activo.

   Se define la cota superior $U := \infty$.

2. Se elige un nodo X de el árbol T (si no existe, **detener**).

   Se marca X como no activo.

   (ramificación) Se encuentra la partición $X = X_1 \cup \cdots \cup X_t$.

3. **Para** $i = 1, \ldots, t$ **hacer**: (acotamiento) Encuentra la cota inferior L en el costo de cualquiera de las soluciones en X_i.

   Si $|X_i| = 1$ y el costo$(S) < U$ **entonces**:

   Define $U :=$ costo(S) y $S^* := S$.

   Si $|X_i| > 1$ y $L < U$ **entonces**:

   Define $T := (V(T) \cup \{X_i\}, E(T) \cup \{\{X, X_i\}\})$ y se marca X_i como activo.

4. **Ir a** 2.

Tabla 2.1: Algoritmo de ramificación y acotamiento.

2.4.2 Método de planos cortantes

Un **poliedro** es la región limitada por **polígonos**. Los polígonos son generados por las restricciones en el **espacio de solución**. Para un poliedro cualquiera P se tiene que $P \supset P_1$. Si se desea resolver el problema de programación entera $\min[cx : x \in P_1]$, es natural pensar en cortar ciertas partes de P tal que el conjunto resultante sea nuevamente un poliedro P' tal que $P \supset P' \supset P_1$. Con suerte, $\min[cx : x \in P']$ es logrado por medio de un vector entero, de otra forma, se puede repetir el procedimiento de corte para P' tal que se obtenga P'' y así sucesivamente. Ésta es la idea básica detrás del **método de planos cortantes**, que fue propuesto inicialmente para el **TSP** por Dantzig et al. [4] en 1954.

2.5 Complejidad

La teoría de la **complejidad computacional** [1] es la rama de la teoría de la computación que estudia, de manera teórica, los recursos requeridos durante el cómputo de un algoritmo para resolver un problema. De esta forma permite describir, analizar y clasificar los problemas de optimización.

Por lo regular se desea obtener una solución a los problemas de optimización de manera inmediata o lo mas rápida posible, razón por la cual se hace uso de un computador. Qué tan rápido puede ser obtenida esta solución depende de varias circunstancias, por ejemplo, la velocidad de la computadora, la calidad del compilador y lo sofisticado del código. Estas circunstancias son hasta cierto punto controlables, por ejemplo, la velocidad de los procesadores va en incremento a la par de los avances tecnológicos, por lo menos hasta el año 2020 aproximadamente según la **ley de Moore**[22]. Por otra parte, hay ciertos límites que no pueden ser superados (o por lo menos aún no se conoce la forma). Es de hecho el propósito de la teoría de la computación el identificar estos límites y agrupar en clases los problemas de optimización combinatoria en base a sus limitaciones estructurales.

2.5.1 Análisis asintótico

Un algoritmo es un procedimiento paso a paso para resolver un problema. Por problema se entiende un modelo general como el problema del camino mas corto o un problema de minimización del uso de recursos. Los problemas pueden ser parte de otros problemas. Una instancia es un caso particular de un problema, en el cual todos los parámetros del problema están definidos. Se dice que un algoritmo puede resolver un problema P, si para cualquier instancia de P el algoritmo puede garantizar una solución al mismo [1]. Generalmente se está interesado en encontrar el algoritmo más 'eficiente' para resolver un problema; sin embargo la 'eficiencia' de un algoritmo puede resultar un término subjetivo. Aunque es muy común que el

tiempo sea el recurso de cómputo dominante, por lo que es utilizado como pauta para medir la eficiencia de un algoritmo.

En la literatura se han adoptado regularmente tres enfoques básicos para medir el desempeño de un algoritmo:

- Análisis empírico.
- Análisis del caso medio.
- Análisis del peor caso.

Es bien conocido que el enfoque de mayor difusión es el análisis del peor caso. Para expresar el tiempo requerido por cierto algoritmo es conveniente contar con una medida de 'complejidad' de las instancias del problema que se aborda. Por lo regular un problema se vuelve más difícil mientras mas grande es. Por esta razón se toma como medida el tamaño de la instancia a tratar. De esta manera se relacionan los parámetros del problema con una serie de operaciones básicas que ha de realizar el algoritmo. Esta forma de evaluar un algoritmo es llamada notación $\mathcal{O}(\ )$. Según Orlin[1] esta notación se puede definir de la siguiente manera:

Definición 2.3 *Se dice que un algoritmo corre en un tiempo de $\mathcal{O}(f(n))$ si para algún número c y n_0, el mayor tiempo utilizado por el algoritmo a largo plazo es $cf(n)$ para todo $n \geq n_0$.*

En otras palabras, es una función que denota la menor cota superior del número de operaciones a realizarse sobre los parámetros de la instancia que tienen mayor peso computacional en el largo plazo. De acuerdo a la función que los acota los algoritmos pueden ser clasificados en algoritmo polinomial, algoritmo débilmente polinomial, algoritmo pseudo-polinomial y algoritmo exponencial.

Bajo esta clasificación los algoritmos polinomiales tienen preferencia ya que la dificultad para resolver el problema no incrementará de forma drástica al crecer la instancia.

2.5.2 Clase P

Ahora es posible definir la clase a la que pertenece un problema en el que el tiempo usado por un algoritmo está acotado polinomialmente en función del tamaño de la entrada n. Para analizar un algoritmo independientemente de su implementación o la arquitectura computacional sobre la que corre se utiliza una descripción abstracta de un proceso de cómputo el cual permite probar los fundamentos y las propiedades universales del algoritmo. Esta representación abstracta fue presentada por Turing [25] en 1936 y es denominada máquina de Turing (TM).

En Hartmann[11] se explica que una TM consiste en una función de transición δ, que representa el código del programa. Q es el conjunto de estados. Matemáticamente, el programa es una máquina con estados finitos. La TM tiene una pequeña memoria en la cual guarda su estado actual. Finalmente la TM tiene una cinta que se extiende infinitamente hacia la derecha y hacia la izquierda. La cinta consiste en una secuencia de celdas, las cuales pueden contener exactamente un símbolo del alfabeto Γ, y que conserva el símbolo B en las celdas en blanco. La TM puede accesar a la cinta por medio de un terminal de lectura/escritura, que le permite acceder a una celda a la vez. La TM puede mover la terminal hacia la izquierda o hacia la derecha. Una definición formal de una TM es como sigue:

Definición 2.4 *Una máquina de Turing M se define como $M = (Q, \Sigma, q_0, \delta)$ donde:*

Q es el conjunto finito de estados,

Σ es el conjunto finito de símbolos de entrada,

$q_0 \in Q$ es el estado inicial,

$\delta : Q \times \Gamma \rightarrow Q \times \Gamma \times \{R, L, N\}$ es la función de transición o movimento y

Una vez que se ha definido a grandes rasgos una TM, es posible definir la clase P de problemas como sigue:

Definición 2.5 Clase P

P es la clase de problemas (de decisión) resueltos por una TM determinista donde el peor caso de tiempo de cómputo para una entrada n está acotado asintóticamente por una función polinomial en n.

Para muchos de los problemas de optimización más interesantes no se sabe si pertenecen a P o no. Por lo que fue creada una clase especial para ellos, la clase NP.

2.5.3 Clase NP

Para problemas pesados de optimización encontrar el óptimo o responder si existe una solución con $H(\sigma) \leq K$ toma mucho tiempo con los algoritmos actuales. Por otra parte, para ese mismo tipo de problemas es fácil verificar si una solución dada cumple con la restricción propuesta. En esta clase de problemas permanece la pregunta abierta: ¿cómo encontrar una buena solución de entre un gran número de posibilidades? En el TSP por ejemplo existen $(n-1)!$ posibles **ciclos**, de tal manera que evaluarlos uno a uno es imposible realizarlo en un tiempo polinomial.

En lugar de responder a esta pregunta, ha sido replanteada. De tal forma que se asume una solución tentativa disponible por un cómputo no convencional. Para formalizar este cómputo no trivial se planteó de forma teórica un máquina de Turing no determinística (NTM). Ésta permite desarrollar pruebas formales tal que prueben que dos problemas son equivalentes en cuanto a su complejidad computacional.

Definición 2.6 Máquina de Turing no determinística (NTM)

Una NTM está definida como una TM (determinística) con la única excepción de que δ es una relación en $(Q \times \Gamma) \times (Q \times \Gamma \times \{L, R, N\})$ en lugar de una función.

Utilizando una TM, es posible definir un lenguaje L como un conjunto de palabras, que son aceptadas por la TM. Del mismo modo pueden ser definidos para la NTM un conjunto de lenguajes. Con la diferencia de que ahora la palabra w

pertenece a L si existe al menos una secuencia en las configuraciones tal que conlleve a ser aceptada. En este caso, pueden existir otras posibles configuraciones de secuencias con la entrada w que tienen como término una configuración no aceptada.

Definición 2.7 Clase NP

La clase NP de problemas polinomiales no determinísticos contiene a todos los problemas de decisión L en los que existe una NTM M que acepta L y su tiempo $t_M(n)$ está acotado por una función $p(n)$ polinomial.

Como no es posible construir una NTM, la idea básica es que una TM trate iterativamente todas las posibles secuencias de cálculos realizados por una NTM. Si una de estas es aceptada, la palabra entrante es aceptada por la TM, de otra manera es rechazada. El tiempo de cómputo al emular la NTM por medio de la TM crece considerablemente, como lo expone el siguiente teorema.

Teorema 2.8 *Para todo $L \in NP$ existe una función polinomial p y una TM M tal que M acepte a L en un tiempo de cómputo de $2^{p(n)}$.*

Dado que el tiempo de cómputo es exponencial y por ende finito para cada lenguaje $L \in NP$, la correspondiente TM tendrá un término para todas las entradas. Entonces cada $L \in NP$ es decidible. Aún más, desde que TM es también una NTM, se tiene que $P \subset NP$. Además existen dentro de esta clase los llamados NP-completos, término que describe una clase equivalente de problemas de decisión en la que se muestra que los más comunes y difíciles problemas de optimización combinatoria son computacionalmente equivalentes.

Capítulo 3

Formulación del problema

En este capítulo se presenta el problema de cobertura desde un enfoque de teoría de grafos, describiendo la metodología utilizada para obtener una representación abstracta del problema. Una vez obtenida la representación abstracta, se formula el modelo matemático asociado a ella.

3.1 Problema de cobertura desde un enfoque de teoría de grafos

El primer punto que es necesario abordar es el de representar el problema de cobertura de una forma abstracta, es decir, modelarlo. Al respecto se debe cuidar que dicho modelo conserve una representación tanto del entorno físico, como de los criterios de funcionalidad. El uso de un grafo para modelar el problema nos permite conservar las propiedades del entorno (área libre y obstáculos) y asociar a las aristas los criterios de funcionalidad. Además, la posibilidad de modelar el problema como una red (o grafo) nos brinda una serie de ventajas, tales como que nos permite tener una representación abstracta, clara y manejable computacionalmente del problema y encontrar analogías y/o aproximaciones del problema, como por ejemplo, el problema del agente viajero, que de hecho es tomado como una primera aproximación al problema y que puede derivar en el uso de otros modelos similares. Además es posible utilizar las herramientas de análisis y/o solución existentes en la teoría de grafos.

3.1.1 Construcción del modelo

El primer supuesto que se tiene es el conocimiento del área, es decir, que se conoce la posición y el tamaño de los obstáculos. Otro supuesto es que el área accesible resultante sea conexa, lo que se refiere a que no deben existir áreas sin obstáculos que resulten aisladas del resto del área sin obstáculos.

Dado que se conoce el área a ser cubierta, el primer paso para la construcción del modelo es identificar la posición de los obstáculos y el área libre con los que tendrá interacción el robot, el siguiente esquema muestra un posible caso:

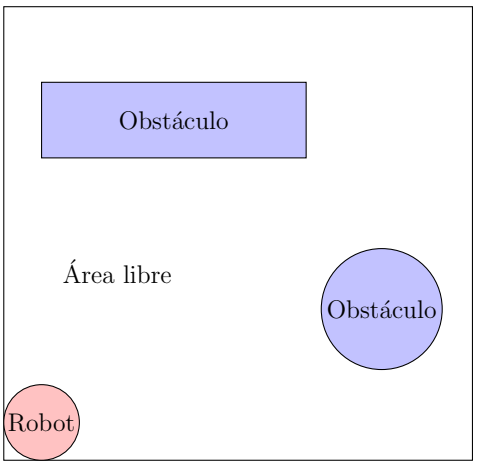


Figura 3.1: Área con obstáculos.

Una vez que se identificó el área se procede a cuadricular ésta para crear celdas (figura 3.2). Las dimensiones de las celdas estarán directamente relacionadas con las dimensiones del robot; esto con el fin de que sea cubierta la mayor superficie posible.

Cualquier celda que contenga total o parcialmente un obstáculo será tomado como un área no accesible; en la figura 3.3 se muestra una representación más abstracta del área, en la que ya se han numerado las celdas y definido las zonas inaccesibles (celdas en color azul).

Por último se representa cada celda etiquetada como un nodo y se generan arcos dirigidos entre cada nodo y sus nodos contiguos. La representación como red

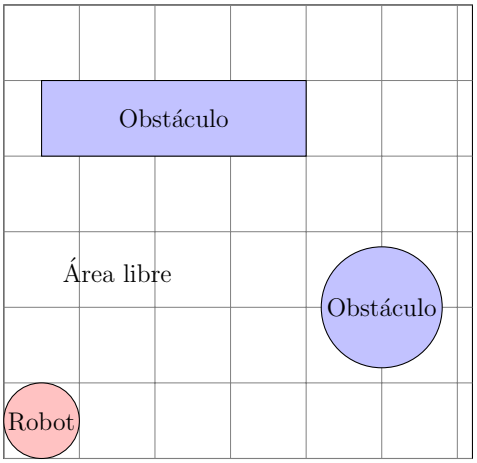


Figura 3.2: Construcción de celdas.

23	24	25	26	27	28
				21	22
15	16	17	18	19	20
11	12	13	14		
7	8	9	10		
1	2	3	4	5	6

Figura 3.3: Numeración de celdas accesibles.

del área mostrada tendría la forma mostrada en la figura 3.4.

De esta forma se ha generado una representación del problema por medio de un grafo, en donde las trayectorias posibles están representadas por la existencia de un nodo y donde los criterios de funcionalidad pueden ser asociados a los costos por utilizar un arco. Por lo tanto, la solución al problema resultaría de encontrar una caminata a lo largo del grafo tal que se visiten todos los nodos y se incurra en el menor costo total.

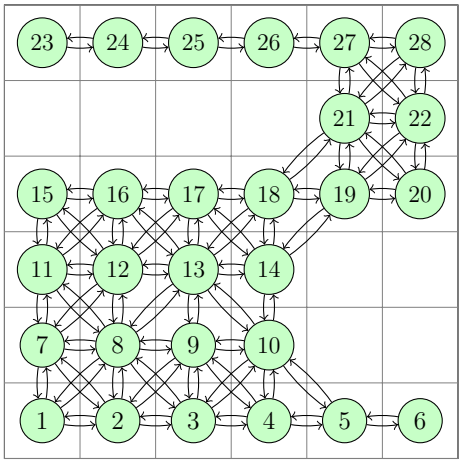


Figura 3.4: Representación como red.

3.2 Modelo matemático

Una forma de encontrar una solución, dado que ya se tiene el problema representado por medio de un grafo, es asociarle una función objetivo. La función objetivo asociada al problema busca minimizar el costo total en que se incurre debido al uso de los arcos que unen los nodos y que en conjunto definen la trayectoria del robot sobre el área. Para definir una función objetivo es necesario contar con el modelo matemático. El modelo matemático se formulará en base a el grafo inducido por el área de interés, sin embargo es conveniente utilizar una base para la contrucción del mismo, por lo que se aprovechará la similitud del problema con el *TSP* (*Traveling salesman problem*). El problema del agente viajero es un problema de optimización combinatoria estudiado en el área de la investigación de operaciones y en el de las ciencias computacionales teóricas. Dada una lista de ciudades y sus distancias entre ellas, el objetivo es encontrar el recorrido más corto posible tal que se visiten todas las ciudades exactamente una vez.

El problema se puede modelar de diversas maneras, en este caso se optó por utilizar un modelo basado en el utilizado en Orlin[1] en el que se conserva una estructura de flujo en redes. Además se utilizó como guía la formulación presentada

en Makhorin[21], utilizando el programa *tsp.mod*. Como primer paso es necesario definir las variables y los parámetros que han de usarse en la formulación; éstos se presentan a continuación:

- Variables:

  x_{ij}: camino del nodo i al nodo j.

$$x_{ij} = \begin{cases} 1 & \text{si el camino es utilizado} \\ 0 & \text{si el camino no es utilizado} \end{cases}$$

  y_{ij}: flujo del nodo i al nodo j

- Parámetros: c_{ij} = costo por usar el arco de i a j.

  b_i = Parámetro de balance de masa.

El modelo asociado al grafo es el siguiente:

$$\min \sum_{i,j=1}^{n} c_{ij} x_{ij} \tag{3.1}$$

La función objetivo mostrada por la ecuación (3.1) está sujeta a las siguientes restricciones:

$$y_{ij} \leq (n-1) x_{ij} \tag{3.2}$$

$$y_{ij} \geq x_{ij} \tag{3.3}$$

$$\sum_{i,j}^{n} y_{ij} - \sum_{i,j}^{n} y_{ji} = b_i \tag{3.4}$$

$$\sum_{ij}^{n} x_{ij} + \sum_{ij}^{n} x_{ji} \geq 2 \tag{3.5}$$

$$\sum_{ij}^{n} x_{ij} - \sum_{ij}^{n} x_{ji} = 0 \tag{3.6}$$

Donde la restricción (3.2) tiene como función evitar que haya flujo a través de un arco que no esté activado, además en el caso de que este arco esté activado no permite que se exceda la capacidad máxima de flujo. Por otra parte, la restricción (3.3) evita que haya arcos activos con flujo nulo.

La restricción (3.4) es llamada restricción de balance de masa en los problemas de flujo en redes. En dichos problemas suelen existir nodos que proporcionan flujo (llamados nodos fuente) y nodos que absorben flujo. El parámetro b_i indica dicha propiedad, siendo un número positivo en el primer caso, donde sale más flujo del que entra, y negativo en el segundo, donde entra más flujo del que sale. En particular para este problema esta restricción tiene como función evitar los *subtours*, dejando un 'paquete en cada ciudad', esto siguiendo el planteamiento del *TSP*. Dado que el parámetro b_i es igual a -1 (nodos que absorben flujo o 'paquetes') para todos los nodos excepto el primero, que es el nodo fuente y que tiene un valor igual a $n-1$ donde n es la cantidad total de nodos, se tiene que solo puede y debe dejar un paquete por nodo, lo que resulta en que tiene que visitar todos los nodos en una sola caminata. Lo que hace la restricción 3.5 es asegurar que todos los nodos dentro del conjunto K es decir, todos excepto el primero y el último, tengan activos por lo menos 2 arcos, que se suponen uno de entrada y otro salida. Por último la restricción (3.6) revisa que se haya entrado y salido el mismo número de veces a el nodo. Cabe mencionar que esta restricción corre sobre todos los nodos, excepto por el primero, en el que se sale sin haber entrado, y el último en el que se entra y no se sale más.

En base a lo anterior se ha logrado construir un modelo tal que represente el problema expuesto de forma abstracta, obteniendo también una forma de controlar la trayectoria por medio de la manipulación de los costos y su subsecuente minimización.

3.3 Características del problema

El problema que se aborda es el de encontrar la caminata de expansión que presente el menor costo. Esto nos lleva al problema de decisión que implica el escoger de entre el conjunto de soluciones, aquella que tenga el costo mínimo. Donde el conjunto de soluciones es aquel que contiene las caminatas de expansión posibles en el grafo. Sin embargo, en un inicio no se conocen dichas caminatas y surge la tarea de generarlas. Cabe mencionar que en un escenario en el que no se restrinja el número de visitas a un nodo, el número de caminatas posibles es, obviamente, infinito. Por otra parte si restringimos a una sola visita por nodo, como en el problema del agente viajero, es bien sabido que buscar todas las permutaciones lleva a una complejidad computacional de $\mathcal{O}(n!)$, lo que significa que si se cuenta con $n = 16$ nodos, es posible generar $20,922,789,888,000$ caminos diferentes, lo que es un número muy grande, aún para un numero pequeño de nodos.

El deteminar si existe un camino Hamiltoniano en un grafo dado es NP-completo y encontar un camino es FNP, es decir, la extención de un problema NP en el que se abordan las relaciones binarias asociadas al problema de desición como parte del problema conjunto.

Capítulo 4

Metodología para la solución

Una vez que se ha formulado el modelo el siguiente paso es encontrar una solución para éste. Una forma de hacerlo es utilizar las herramientas computacionales existentes, en este caso un *solver*. Cabe mencionar que uno de los objetivos del proyecto original, el cual inspiró el trabajo actual, es el de incurrir en la menor cantidad de costos, esto para que su implementación sea viable. Por esta razón se hace uso de una herramienta de distribución libre, en este caso, el solver *glpk* [20] de GNU.

4.1 Herramientas de solución

GLPK (GNU Linear Programming Kit)[20] es una serie de rutinas escritas en el lenguaje de programación ANSI C, organizadas en forma de librería. Estas fueron desarrolladas para resolver problemas de programación lineal (LP), programación entera mixta (MIP), y otros problemas relacionados. Los problemas pueden ser modelados en el lenguaje *MathProg* de GNU[21], el cual es similar a la sintaxis de AMPL [9] (A Mathematical Programming Language), que es un lenguaje de programación de alto nivel, desarrollado por los laboratorios Bell, para describir y resolver problemas complejos y de cómputo pesado. El problema modelado en *MathProg* es resuelto por el solver *GLPSOL*.

Una vez que se tiene el modelo matemático asociado al problema, es necesario traducirlo al lenguaje del solver. En la tabla 4.1 se muestra dicha traducción para las ecuaciones descritas en el capítulo anterior.

El problema que se resolverá por medio del solver consta de dos partes: una asociada al modelo y otra asociada a los datos o instancia. La parte asociada al modelo es la que se muestra en la tabla 4.1. La parte relacionada con la instancia se muestra en la sección siguiente.

4.2 Entradas

4.2.1 Nodos, aristas y costos

Ya que se cuenta con el modelo que representa la función objetivo y las restricciones, el siguiente paso es ingresar el problema o la instancia que se quiere resolver en particular. El solver en cuestión requiere que se ingresen los datos siguientes:

- El número n de nodos.
- El parámetro b_i para cada uno de los nodos.
- Cada uno de los arcos existentes.
- Los costos asociados a cada uno de los arcos.

En cuanto a los dos primeros puntos no resulta dificil ingresar los datos salvo para n grandes, sin embargo los datos referentes a los arcos son una tarea bastante grande, excepto para n pequeñas. Por ejemplo, si se tiene un problema con una estructura básica (área cuadrada) con $n = 16$ el número de arcos que se tiene que ingresar es de 74. Esto para un área de 4×4 celdas, que en realidad es muy pequeña, y no se compara con las áreas que se planean cubrir. Por esta razón se busca realizar esta operación de forma automatizada, como se muestra más adelante.

Linea en MathProg

```
param n, integer, >= 3;
set V := 1..n;
set K := 2..(n-1);
set E, within V cross V;
param c{(i,j) in E};
param b{i in V};
var x{(i,j) in E}, binary;
var y{(i,j) in E};
minimize total: sum{(i,j) in E} c[i,j] * x[i,j];
s.t. flujo{(i,j) in E}: y[i,j] <= (n-1)* x[i,j];
s.t. flujomin{(i,j) in E}: y[i,j] >= x[i,j];
s.t. balance{i in V}:
sum{(i,j) in E}y[i,j] - sum{(j,i) in E}y[j,i] = b[i];
s.t. visita{i in K}:
 sum{(i,j)inE}x[i,j] +  sum{(j,i) in E } x[j,i] >= 2;
s.t. flujo_acumulado{i in V}:
sum{(i,j) in E}y[i,j] <= n-1;
s.t. regresos{i in K}:
sum{(i,j) in E}x[i,j] - sum{(j,i) in E}x[j,i] = 0;
solve;
```

Tabla 4.1: Formulación del modelo en lenguaje MathProg.

4.2.2 Obstáculos

Otro dato que es importante agregar es la posición de los obstáculos dentro del área. Para visualizar los obstáculos dentro del área se generó una *matriz de obstáculos*, la cual representa en sus elementos las celdas que componen el área y les asigna un valor de cero si es un espacio libre y un valor de 1 si dicha celda está ocupada total o parcialmente por un obstáculo. La matriz de obstáculos para el área de la figura 3.2 se muestra en la figura .

$$\begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

Figura 4.1: Matriz de obstáculos.

4.3 Traductores

Como se acaba de mostrar, se cuenta con dos entradas, una para definir la estructura del grafo y otra para asignar los parámetros asociados al mismo. Por otro lado se desea tener como salida una trayectoria, o más bien, una serie de instrucciones que, al ser ejecutadas por el robot, generen una trayectoria. El proceso necesario se presenta en el diagrama 4.2.

Siguiendo el diagrama de proceso mostrado en la figura 4.2, el primer paso necesario es la construcción de las entradas. En el caso de la matriz de obstáculos la construcción tendrá que ser por medio de la observación del entorno y la consiguiente abstracción de éste para generar las celdas y el numerado de las mismas con $\{0, 1\}$. Una alternativa posible es generar la matriz por medio de una caminata

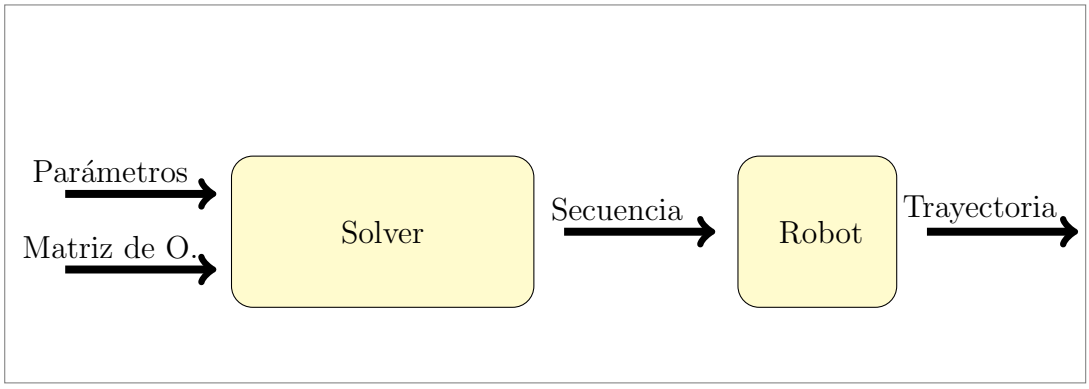


Figura 4.2: Diagrama de proceso de la metodología de solución.

de reconocimiento realizada por el mismo robot, aunque dicho algoritmo e implementación no se abarcan el el trabajo actual.

Como en cualquier aplicación, el programa que se utiliza como solver requiere que se ingresen los datos con un determinado formato y a su vez, el formato de salida del solver no es directamente asimilable por el agente que ha de ejecutar la trayectoria, por lo que nace la necesidad de construir un traductor que genere los formatos adecuados.

4.3.1 Traductor: Matriz de obstáculos-solver

Tomando como punto de partida que se cuenta con la matriz de obstáculos, el objetivo es utilizar ésta para generar la entrada asociada a los nodos, arcos y costos con el formato indicado. El hecho de capturar los datos de forma manual para una instancia en particular es una tarea que se puede volver sumamente larga y tediosa, razón por la cual se creó un programa traductor que tiene como función generar de forma automática esta entrada. El programa en cuestión etiqueta los nodos de forma ascendente iniciando en la esquina inferior izquierda y sigue hacia la derecha hasta acabar el renglón, para después iniciar nuevamente en el extremo izquierdo hasta numerar cada uno de los elementos de los renglones de la matriz, cabe señalar que la numeración sólo corre sobre las celdas libres, es decir las que están identificadas

con un cero. De esta forma identifica los n nodos existentes. Una vez que se cuenta con los nodos se generan los arcos que los unen. Para generar los arcos se toma como referencia el nodo i y se genera un arco de salida a cada uno de los elementos existentes dentro de la vecindad de Moore de dicho nodo. Esto se realiza para $i = 1 : n$. El programa reconoce aquellos nodos que se encuentran en la frontera, así como aquellos que cuentan con un nodo inaccesible como vecino; en estos casos no se generan los arcos hacia los nodos con estas características. Por último asigna los costos a cada arco, si es que se pueden asignar de forma recursiva. En el caso que se desea encontrar el camino de menor longitud, esto es posible, y dado que se toma este enfoque como base, el programa lo hace de forma predeterminada.

Al haber generado todos los datos necesarios el programa les da un formato afín al solver y los imprime en un archivo de texto llamado *modelo.txt*.

4.3.2 Traductor: Solver-robot

Una vez que se ha resuelto el problema por medio del solver, éste genera como salida un archivo de texto llamado *solucion.txt*, el cual consta de dos columnas de números. Dichos números representan en cada renglón los índices de los arcos utilizados, mostrándose el nodo fuente en primer lugar y el nodo destino en segundo.

El conjunto de los arcos mostrados forman la trayectoria que se desea obtener, sin embargo, en el archivo de salida del *solver* sólo se mencionan cuáles arcos han de estar activos, pero no presentan una secuencia coherente con la trayectoria que se desea obtener. Por lo tanto es necesario darles un orden, tal que se presenten como una secuencia de instrucciones que guien al robot a lo largo del área, de tal forma que sea cubierta en su totalidad. Para darle dicho orden a los datos, es necesario crear un programa, el cual lea los datos directamente del archivo de texto generado por el solver, ordene los datos y los imprima.

El ordenamiento de los arcos representados por sus índices es trivial en el caso en que no existen ciclos, ya que el simplemente seguir una cadena *'nodo fuente* -

nodo destino - nodo fuente' llevaría a un buen resultado. Sin embargo para el caso que contiene ciclos esto no es así. Un ejemplo sencillo de esto se puede observar en la figura 4.3, en la que se muestra el ciclo formado por los nodos $\{12, 15, 14\}$.

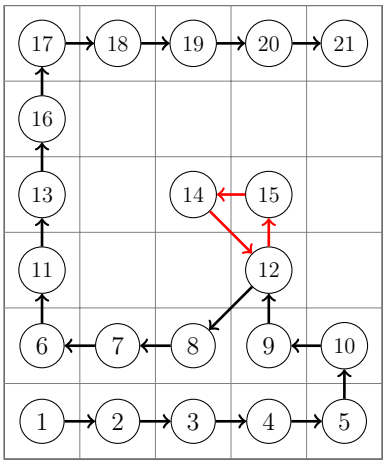


Figura 4.3: Ciclo dentro de la trayectoria.

El problema al momento de generar la secuencia se encuentra al llegar al nodo 12, ya que este tiene 2 arcos salientes activados y no solo uno como el resto. Así, se encuentra en una situación en la que tiene que 'decidir' cual arco tomar en primer lugar, en donde uno implica generar una trayectoria que visita todos los nodos y la otra genera una trayectoria factible (en la que alcanza el nodo final) pero incompleta. Debido a esto el programa debe ser capaz de identificar los ciclos y asignarlos de tal manera que la trayectoria resultante visite todos los nodos.

Capítulo 5

Implementación

Una parte importante del trabajo realizado en este estudio fue la implementación de la metodología propuesta en un robot real. En trabajos como Yang [26] importantes conclusiones son obtenidas en base a la simulación del algoritmo propuesto, sin embargo, la sencillez de los ambientes y el total control sobre el movimiento del robot son suposiciones que rara vez se encuentran en la práctica.

El hecho de llevar a cabo un proceso de experimentación conlleva un conocimiento profundo de las fortalezas y debilidades de la metodología que se propone. Así abre también nuevas interrogantes y trabajos a ser desarrollados.

La implementación en el robot corrió a cargo del Dr. Luis Torres Treviño del CIIDIT de la UANL. Parte del trabajo realizado por el Dr. Torres relacionado con el presente trabajo, además de ser parte de sus propias líneas de investigación, consistió en adaptar un dispositivo de comunicación remota a un robot prototipo. De tal forma que contara con un centro de procesamiento remoto, en este caso una PC (computadora personal por sus siglas en inglés). El hecho de poder hacer uso de una herramienta común, como una PC, utilizando recursos con los que ya cuenta y realizando modificaciones menores y de bajo costo son ventajas significativas a la hora de pensar en una incursión comercial del producto. Por esta razón, los bajos costos de implementación son un punto focal en el proyecto del que forma parte la investigación desarrollada en este trabajo.

Los detalles sobre la implentación de la interface en MatLab no se abordarán,

puesto que forman parte de los trabajos realizados por el Dr. Torres. Por otra parte sí resultan relevantes las particularidades relacionadas con el posicionamiento espacial y la secuencia de instrucciones generadas por el algoritmo de solución. El hecho de que el robot tenga conocimiento de su posición resulta en un problema no trivial y que conlleva comunmente a la inversión en sensores que lleven a cabo esta tarea, no hace falta mencionar que dicha inversión incrementa significativamente los costos de manufactura del robot.

La salida generada por el algoritmo de solución entrega una secuencia que consta de la sucesión de nodos que han de ser visitados. Dado que a los nodos se les puede asociar una posición cartesiana en el plano, se tradujo la secuencia de nodos en una secuencia de coordenadas. El robot puede recibir dicha secuencia como entrada, pero para llevar a cabo las instrucciones sería necesario que el robot supiera en que coordenada se encuentra actualmente para saber hacia donde debe dirigirse. Dicho conocimiento cae en el problema antes descrito y en la necesidad de más sensores. Para enfrentar este problema fue necesario modificar nuevamente la salida generada por el algoritmo de solución. De tal manera que la secuencia de coordenadas fue traducida en una secuencia de instrucciones básicas de dirección. Estas instrucciones básicas son:

- $\uparrow$ un paso al frente **N**.
- $\downarrow$ un paso atrás **S**.
- $\leftarrow$ un paso a la izquierda **O**.
- $\rightarrow$ un paso a la derecha **E**.
- $\nearrow$ un paso **NE**.
- $\nwarrow$ un paso **SE**.
- $\searrow$ un paso **SE**.
- $\swarrow$ un paso **SO**.

Estas instrucciones tienen la peculiaridad de que están orientadas con respecto al último movimiento realizado y no al entorno en general. De esta forma se toma en cuenta la orientación actual del frente del robot como punto de referencia. Para que dichas instrucciones tengan sentido se tiene que cumplir que el robot inicie el recorrido posicionado en el nodo uno, que es el de la esquina inferior izquierda de la matriz de obstáculos, orientado hacia el norte.

Por otra parte las instrucciones **N**, **S**, **SE**, etc. fueron traducidas a movimientos básicos del robot, tales como avanzar 10 cm (un paso), girar 90^o, girar 45^o etc.

De esta manera fue posible que el robot siguiera la trayectoria definida por el algoritmo de solución sin hacer uso de sensores adicionales.

Cabe mencionar que durante la implementación resultó evidente la posibilidad de formular el problema como uno de optimización dinámica, con el fin de reducir los giros en la medida de lo posible. Además surgieron pequeñas variaciones en el recorrido del robot, atribuidas a la superficie de desplazamiento, de aquí queda la interrogante de si es posible detectar y compensar estas variaciones con el fin de reducir las variaciones en la trayectoria. Si bien no se abordan las cuestiones mencionadas anteriormente en este trabajo, sí quedan como temas abiertos para trabajos sucesivos.

Capítulo 6

Resultados computacionales

Una vez que se cuenta con la implementación del algoritmo de solución es de interés conocer su comportamiento. Dicho análisis se desarrolla en el presente capítulo.

En términos prácticos la complejidad computacional está directamente relacionada con el tiempo de cómputo. Razón por la cual es de interés identificar los elementos que tienen influencia sobre dicho tiempo. Dichos elementos pueden ser de diversa naturaleza y dependen en gran medida del problema particular, es decir de la instancia. El tipo de elementos que se tratarán en el presente análisis están relacionados con posibles escenarios en la configuración de los costos, en la conformación del área de interés y cómo se comparan estos con una formulación conocida, el problema del camino hamiltoniano de menor costo.

6.1 Experimentos

Se busca que los experimentos brinden información respecto a la naturaleza del problema, teniendo como objetivos:

- Definir si la forma de los obstáculos afecta el tiempo de cómputo.
- Definir si la correlación entre los costos estructurados tiene un impacto en la complejidad del problema.
- Definir si, en el caso de tener costos aleatorios, diferentes costos llevan a difer-

entes tiempos de cómputo.

- Definir el impacto en el tiempo de cómputo de la transformación a un problema de camino Hamiltoniano puro.

6.1.1 Formas

Dado que el algoritmo de cobertura tiene aplicación en distintos ambientes, es de interés identificar si la forma en que se presentan los obstáculos tiene una impacto en el tiempo de cómputo. Resulta relevante dicho análisis puesto que permite conocer si el algoritmo se desarrollará mejor en ambientes con pequeños obstáculos bien distribuidos, como en el caso de la detección de minas dado que el terreno pudiera tener vegetación como obstácuos. O se desarrollará mejor en ambientes en los que los obstáculos se encuentran concentrados en ciertas áreas, como en el caso de limpieza automatizada.

Para determinar si la forma en que se presentan los obstáculos impacta en el tiempo de cómputo se generaron diversos escenarios de forma aleatoria, variando el tamaño de los obstáculos pero conservando el número de casillas libres. Los resultados para dichos experimentos se muestran en la tabla 6.1.

El análisis de los datos se realizó siguiendo la metodología propuesta en Hines [12] y utilizando como herramienta de cálculo el software *Octave* [7] y el paquete *Statistics*. La salida generada se muestra en la figura 6.1.

Con los datos arrojados por *Octave* podemos concluir que no existe evidencia suficiente para suponer que existe una diferencia estadísticamente significativa entre las medias de los 6 grupos. Ya que el estadístico $F_{0.01,5,54} \approx 3.34$ es mayor que $F_0 = 3.1601$.

En la figura 6.2 se muestran los tiempos de cómputo para las diversas configuraciones de obstáculos generadas.

	Tamaño de obstáculo					
Prueba	1	2	4	5	10	20
1	36.4	11.5	48.2	139.8	3.8	10.7
2	174.0	17.2	4.6	4.8	110.6	1.5
3	33.1	26.2	10.8	6.3	11.7	4.1
4	366.5	1.7	4.0	20.0	4.6	7.6
5	38.1	4.9	4.9	13.4	6.1	8.9
6	28.0	31.4	14.2	5.8	3.5	63.8
7	35.5	76.9	4.2	4.6	2.6	14.0
8	416.5	5.6	2.6	12.1	2.8	4.0
9	77.2	13.4	25.2	47.0	186.8	9.3
10	12	16.6	108.6	68.6	3.2	44.0
Media	121.73	20.54	22.73	32.24	33.57	16.79

Tabla 6.1: Tiempos de cómputo para diversas formas de obstáculos.

```
One-way ANOVA Table:

Source of Variation   Sum of Squares    df  Empirical Var
*******************************************************
Between Groups            79873.7733     5     15974.7547
Within Groups            272975.6400    54      5055.1044
-------------------------------------------------------
Total                    352849.4133    59
Test Statistic f              3.1601
p-value                       0.0142
ans =  0.014199
```

Figura 6.1: Análisis de varianza: impacto en el tiempo debido a la forma.

6.1.2 Costos estructurados

Uno de los usos inmediatos que puede tener el programa es la de encontrar el camino que cubra un área con la distancia como única medida de desempeño, lo cual lleva a una determinada asignación de costos que puede ser considerada como una formulación "base". Esta formulación da al grafo una estructura de costos tal que se estos están fuertemente correlacionados. La correlación de los costos pudiera no parecer un tema relevante en lo que respecta al estudio de los tiempos de cómputo, sin embargo parece relevante debido al algoritmo de búsqueda que se utiliza, *branch and cut*, en el que se crea un árbol de soluciones y se discrimina entre las 'ramas' a fin de conservar solo aquellas que tengan un valor atractivo en cuanto su evaluación. El problema surge en la posibilidad de no estar eliminando ramas, esto es, debido a que los costos asignados son iguales, se estarían generando soluciones que no dominan a las otras, llevando al algoritmo a una situación en la que no puede decidir entre dos (o más) ramas y por ende las conserva a ambas. Este comportamiento lleva a que el árbol de soluciones crezca bastante lo cual genera un tiempo de cómputo elevado.

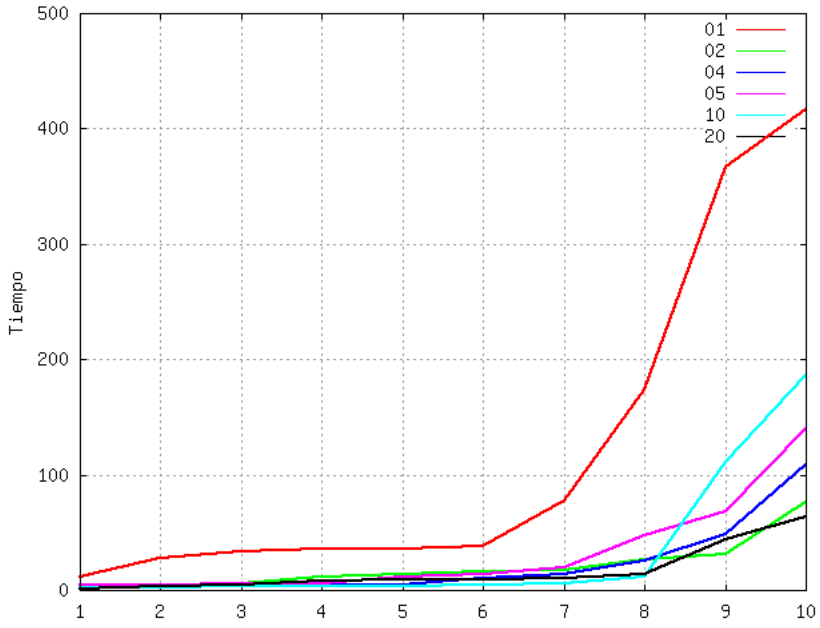


Figura 6.2: Tiempos de cómputo para diversas configuraciones de obstáculos.

Prueba	Modelo Estructurado	Modelo Aleatorio
1	314.6	16.0
2	1608.0	26.9
3	1116.6	144.9
4	3.5	73.8
5	1397.2	27.4
6	43.8	56.2
7	501.3	90.5
8	38.9	44.1
9	414.6	61.4
10	3.6	340.1
11	1190.4	103.3
12	444.7	99.2
13	79.8	87.7
14	934.1	164.2
15	3074.8	160.3
Media	744.39	99.733

Tabla 6.2: Tiempos de cómputo para la formulación estructurada y para la aleatoria.

Con el fin de determinar si el uso de costos estructurados tiene un efecto en el tiempo medio de cómputo, se desarrolló una prueba. Dicha prueba consiste en crear 15 áreas de 8×8 nodos en las que se asignan 12 obtáculos de forma aleatoria; posteriormente se le asigna a cada área los costos asociados a los arcos de la forma tradicional, es decir, asignando un costo a los arcos horizontales y verticales y otro mayor a los arcos diagonales. Después a esa misma área se asignan los costos de manera aleatoria; de esta forma se obtienen dos modelos, uno estructurado y uno aleatorio. Enseguida se resuelven por medio del solver y se captura el tiempo que llevó llegar a una solución. Los datos obtenidos son mostrados en la tabla 6.2.

Como se puede observar el tiempo medio de cómputo de ambos modelos difiere, sin embargo no está de más comprobarlo con bases estadísticas, por lo que se realizó la prueba ANOVA (análisis de varianzas) por medio del software *Octave* y el paquete *Statistics*. La impresión de pantalla de la salida del mismo se muestra en la figura 6.3.

```
One-way ANOVA Table:

Source of Variation   Sum of Squares    df  Empirical Var
*******************************************************
Between Groups          3156066.2679     1   3156066.2679
Within Groups          10035197.7841    28    358399.9209
-------------------------------------------------------
Total                  13191264.0521    29
Test Statistic f              8.8060
p-value                       0.0061
ans =  0.0060869
```

Figura 6.3: Análisis de varianza: impacto en el tiempo debido a la estructura de los datos.

Dado que el valor del estadístico $F_{0.01,1,28} = 7.64$ es menor a $F_0 = 8.6933$ se rechaza con una confianza del 99% la hipótesis nula de que las medias son iguales. Resultado que apoya el bajo valor del estadístico *p (0.0064)*.

En base a las pruebas realizadas estamos en posición de afirmar que los tiempos de cómputo para ambas formulaciones difieren significativamente, resultando evidente que el tiempo de cómputo necesario para el modelo aleatorio es, en general, menor que el necesario para el modelo estructurado. Este hecho lo podemos observar claramente en la gráfica 6.4, en la que se grafican del lado izquierdo los datos correspondientes al modelo estructurado y al derecho los del modelo aleatorio.

Es evidente que la distribución de los datos para el modelo estructurado es

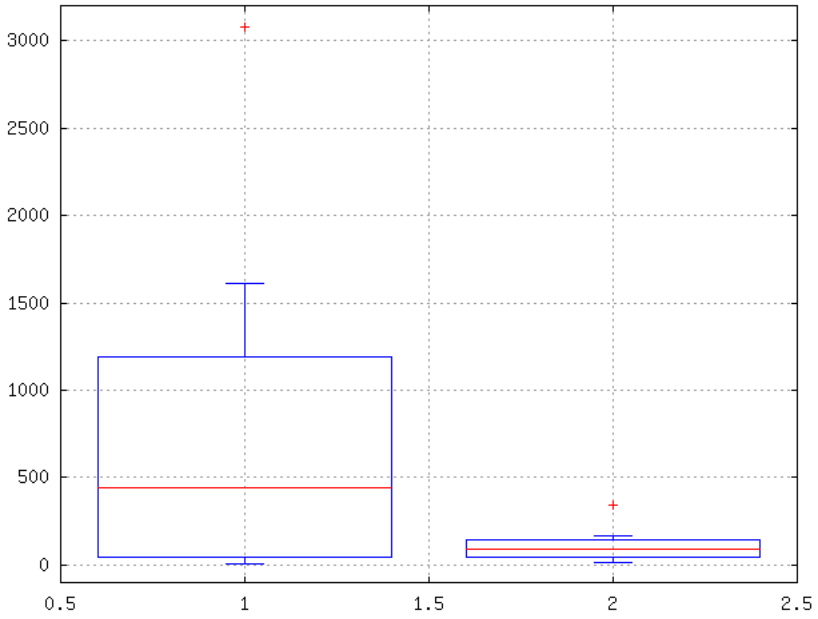


Figura 6.4: Diagrama de caja para el tiempo de cómputo: modelo estructurado vs. aleatorio.

mucho más amplia que para el modelo aleatorio, de hecho, en la gráfica podemos observar que el punto atípico (con símbolo '+') del modelo aleatorio queda aún por debajo del segundo cuartil del modelo estructurado, lo que nos habla de cómo afectan los costos correlacionados al tiempo de cómputo en el problema que se aborda en el presente trabajo.

6.1.3 Costos aleatorios

Ya se ha visto que la asignación de costos aleatorios produce cambios en el tiempo de cómputo, ahora resulta importante conocer la naturaleza de esos cambios y como se presentan en relación al tiempo de cómputo del modelo estructurado. Con el fin de identificar la variabilidad de los cambios producidos por la asignación de costos aleatorios se desarrolló la siguiente experimento. Se generó una matriz de obstáculos de 10×10 nodos, conteniendo 19 nodos marcados como obstáculos. Posteriormente se le asignaron los costos asociados a los arcos de forma aleatoria en 15 ocasiones.

La matriz generada de forma aleatoria se muestra en la figura 6.5.

0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	1	0	0	0
0	0	0	0	1	0	0	1	0	0
0	0	0	1	0	0	0	0	1	0
0	0	0	0	1	0	0	1	0	0
0	0	0	1	1	0	1	1	0	0
0	0	1	0	0	1	0	1	0	0
0	0	0	0	0	1	1	0	0	0
0	0	0	0	1	1	0	0	0	0

Figura 6.5: Matriz de obstáculos para prueba con costos aleatorios.

Los tiempos de cómputo generados por el solver glpk con el algoritmo de **ramificación y acotamiento** se muestran en la tabla 6.3.

Cabe mencionar que el tiempo generado en la novena prueba es una estimación a través de un modelo de regresión ya que dicha prueba se demoró bastante. En la gráfica 6.6 es posible observar como se distribuyen los tiempos de cómputo, con excepción del noveno que se tomó en cuenta para los cálculos pero que se eliminó de la gráfica con fines de legibilidad. Además, una vez que se obtuvieron los tiempos para los modelos aleatorios se generó el modelo estructurado y se resolvió, dando un tiempo de $10,887.8$ segundos, mismo que se incluye también en la gráfica.

Como se puede observar el valor para el tiempo estructurado cae por abajo de la media, pero por arriba de la mediana, y aunque es cierto que está lejos del peor caso encontrado, el tiempo necesario es mayor que el de la mayoría de los casos aleatorios. En la figura 6.7 es posible observar que aproximadamente el 73% de los valores generados por el modelo aleatorio caen por debajo del tiempo del modelo estructurado.[1]

[1]No se muestra cuando la línea toca el valor 1 ya que se recortó por motivos de legibilidad.

Prueba	Tiempo
1	56.4
2	768.2
3	587.8
4	1,910.0
5	973.8
6	10,262.1
7	121.2
8	1,569.6
9	252,575,860.7*
10	6,745.7
11	28,698.9
12	6,490.9
13	8,411.1
14	31,719.5
15	35,638.7

Tabla 6.3: Tiempo de cómputo para distintos modelos aleatorios.

De esta forma podemos concluir que la complejidad del problema con costos estructurados es generalmente mayor que con costos aleatorios.

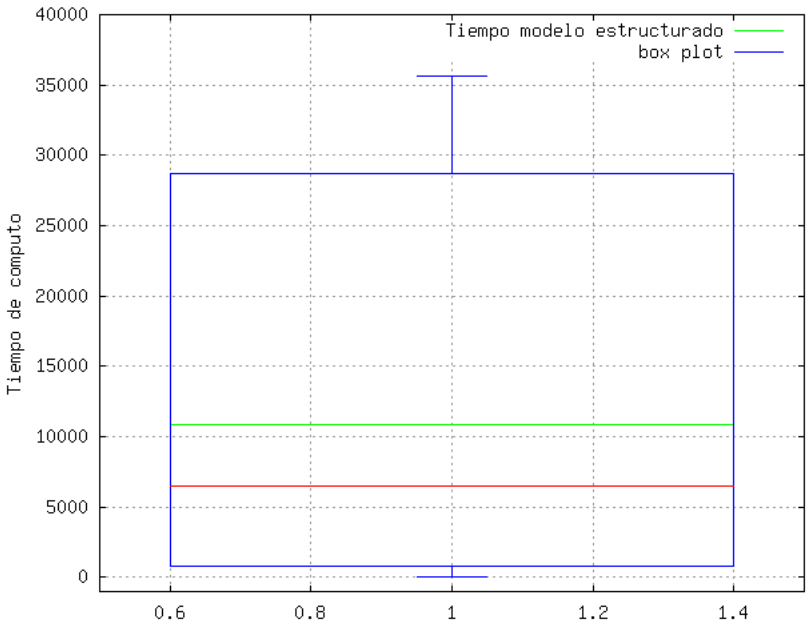


Figura 6.6: Diagrama de caja para los tiempos de cómputo de modelos aleatorios.

6.1.4 Como problema del camino hamiltoniano

Como ya se ha mencionado antes, el problema del camino hamiltoniano (*HPP* por sus siglas en inglés) está dentro de la clase **NP-Completo**, es decir, no existe un algoritmo determinista que resuelva el problema en un tiempo polinomial [13], dicho de otra manera, el problema es difícil. El interés ahora, es conocer como se comporta el tiempo de cómputo derivado de la formulación del problema como un *HPP* con respecto a la formulación propuesta. Y que sirva de punto de referencia para conocer la complejidad del problema que se está abordando.

Para responder dicha cuestión se generaron quince matrices aleatorias y se le asignaron costos aleatorios a las aristas asociadas a cada una, después se utilizó el solver glpk con el algoritmo *branch and cut* para resolver el modelo original, es decir, aquel que permite más de una visita a cada nodo, y se capturó el tiempo de cómputo utilizado, enseguida se resuelve la misma instancia con el modelo *HPP (Hamiltonian Path Problem)* , en el cual no se permite más de una visita a cada nodo.

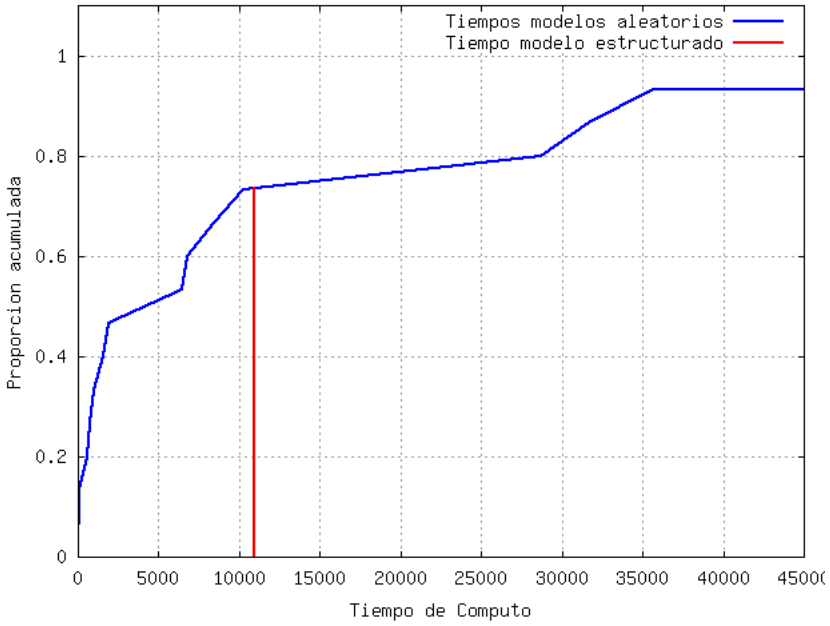


Figura 6.7: Diagrama acumulado de los tiempos de cómputo.

Cabe mencionar que se aseguró factibilidad para el *HPP* al momento de generar las instancias. Los resultados se muestran en la tabla 6.4.

Una vez que se obtuvieron los datos el siguiente paso es analizar si existe una diferencia significativa entre las medias de los tiempos de cómputo para ambos modelos. El análisis se realizó mediante la prueba ANOVA, utilizando el programa Octave con el paquete *Statistics*. El resultado que arroja se muestra en la figura 6.8.

Dado que el valor del estadístico $F_{0.01,1,28} = 7.64$ es menor que $F_0 = 7.6491$ la hipótesis nula de que las medias son iguales se rechaza. Sin embargo la diferencia es muy pequeña, como lo muestra también el estadístico p, en el que un valor de $p = 0.01$ aceptaría la hipótesis nula, siendo un valor de 0.0099 el que se presentó. En esta situación resulta conveniente analizar los datos de forma gráfica, para entender como se comportan y tener un mejor entendimiento del resultado de la prueba ANOVA. En la figura 6.9 se muestra el diagrama de caja para los datos generados por el modelo original del lado izquierdo y los del modelo *HPP* del lado derecho.

Como se puede observar en la gráfica los datos se distribuyen de forma diferente

Prueba	Modelo original	Modelo HPP
1	889.5	16.0
2	64.4	26.9
3	440.3	144.9
4	157.9	73.8
5	33.9	27.4
6	181.3	56.2
7	215.0	90.5
8	82.3	44.1
9	402.2	61.4
10	86.5	340.1
11	332.8	103.3
12	1.4	99.2
13	1386.2	87.7
14	994.8	164.2
15	869.5	160.3

Tabla 6.4: Tiempo de cómputo para el modelo original vs. *HPP*.

para ambos modelos. Lo que sugiere en este caso la gráfica es que en general los tiempos de cómputo para el modelo original son más grandes que los generados por el modelo del camino hamiltoniano. Si bien la prueba ANOVA no daba cuenta de a quién pertenecían los tiempos más grandes, si por lo menos daba por sentado que existía una diferencia entre ambas. Con el fin de analizar la relación existente entre el tiempo de cómputo con ambos modelos y determinar si existe evidencia suficiente para suponer que el tiempo de cómputo de el modelo original es mayor que el del modelo del camino hamiltoniano se realizó la prueba no paramétrica de signos para la igualdad de medianas. Los datos son los siguientes:

- μ_{mo} : la mediana del conjunto de los tiempos generados por el modelo original.
- μ_{hpp} : la mediana del conjunto de los tiempos generados por el modelo *HPP*.
- $H_0 : \mu_{mo} = \mu_{hpp}$
- $H_1 : \mu_{mo} > \mu_{hpp}$
- $\alpha = 0.01$

```
One-way ANOVA Table:

Source of Variation   Sum of Squares    df  Empirical Var
*******************************************************
Between Groups           718272.1333     1    718272.1333
Within Groups           2629278.0933    28     93902.7890
-------------------------------------------------------
Total                   3347550.2267    29
Test Statistic f              7.6491
p-value                       0.0099
ans =  0.0099417
```

Figura 6.8: Análisis de varianza: impacto debido al modelo.

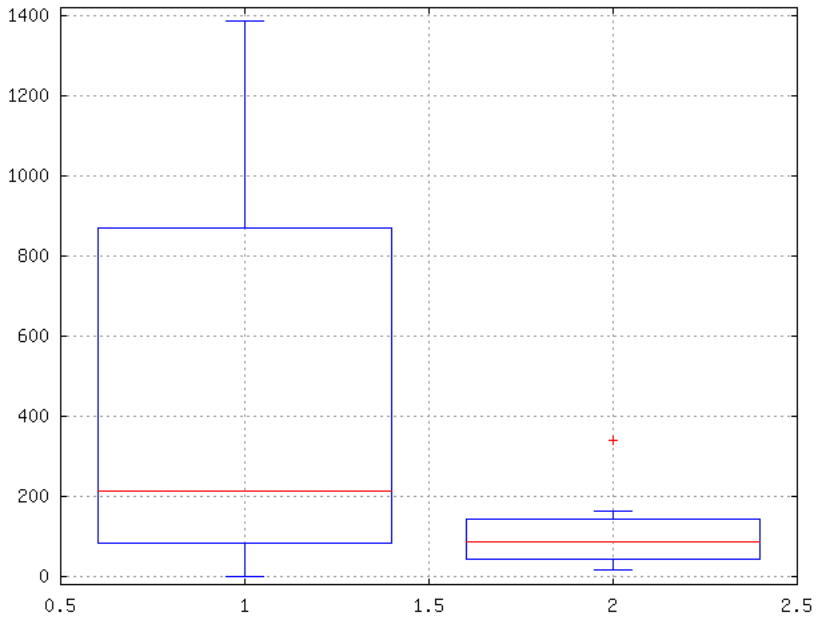


Figura 6.9: Diagrama de caja para los tiempos de cómputo del modelo original y el modelo *HPP*.

La tabla 6.5 muestra los signos generados por la diferencia entre los dos experimentos.

De la tabla 6.5 tenemos que $R^+ = 13$ y $R^- = 2$. Por lo tanto $R = \min(R^+, R^-) = \min(13, 2) = 2$. Dado que cuando $\mu_{mo} = \mu_{hpp}$, R se comporta como una variable aleatoria binomial $n = 15$ y $p = 0.5$, podemos calcular la probabilidad de que $R \leq 2$ de la siguiente manera:

$$P(R \leq 2) = \sum_{r=0}^{2} \binom{15}{r} (0.5)^r (0.5)^{15-r} = 0.004 \ .$$

Dado que $P < \alpha$ se rechaza la hipótesis nula de igualdad de medianas.

Dado que se desea conocer como se comporta el problema que se aborda en el peor de los casos, es interesante observar la aproximación que se muestra en el noveno dato de la tabla 6.3. Dicha estimación se realizó por medio de una regresión, aquella que mejor se ajustaba. En las gráficas 6.10 a 6.12 se muestran dichas regresiones.

Como se puede observar el modelo que mejor se ajusta es el de ley de potencias,

Prueba	MO	HPP	$t_{MO} - t_{HPP}$	Signo
1	889.5	16.0	873.5	+
2	64.4	26.9	37.5	+
3	440.3	144.9	295.4	+
4	157.9	73.8	84.1	+
5	33.9	27.4	6.5	+
6	181.3	56.2	125.1	+
7	215.0	90.5	124.5	+
8	82.3	44.1	38.2	+
9	402.2	61.4	340.8	+
10	86.5	340.1	-253.6	-
11	332.8	103.3	229.5	+
12	1.4	99.2	-97.8	-
13	1386.2	87.7	1298.5	+
14	994.8	164.2	830.6	+
15	869.5	160.3	709.2	+

Tabla 6.5: Tabla de signos para modelo original vs *HPP*.

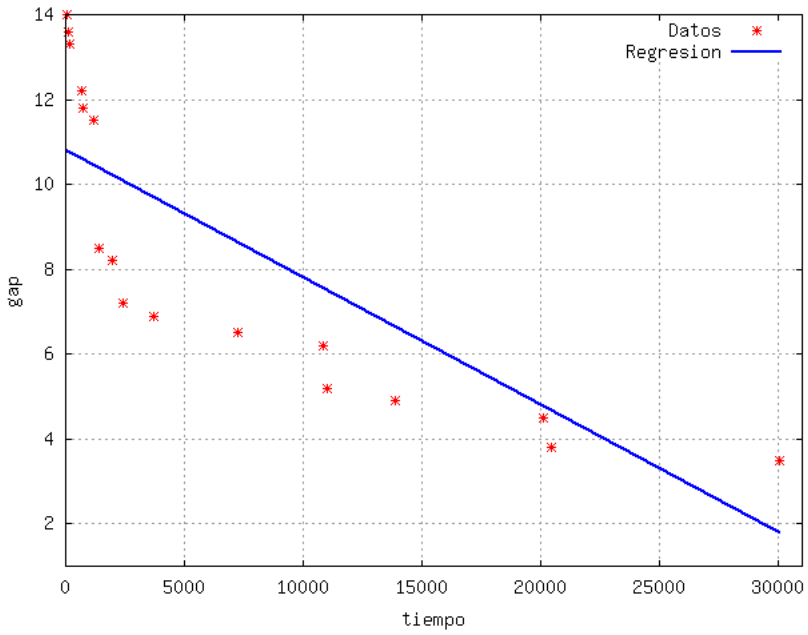


Figura 6.10: Regresión lineal, estadístico $R = 0.82$.

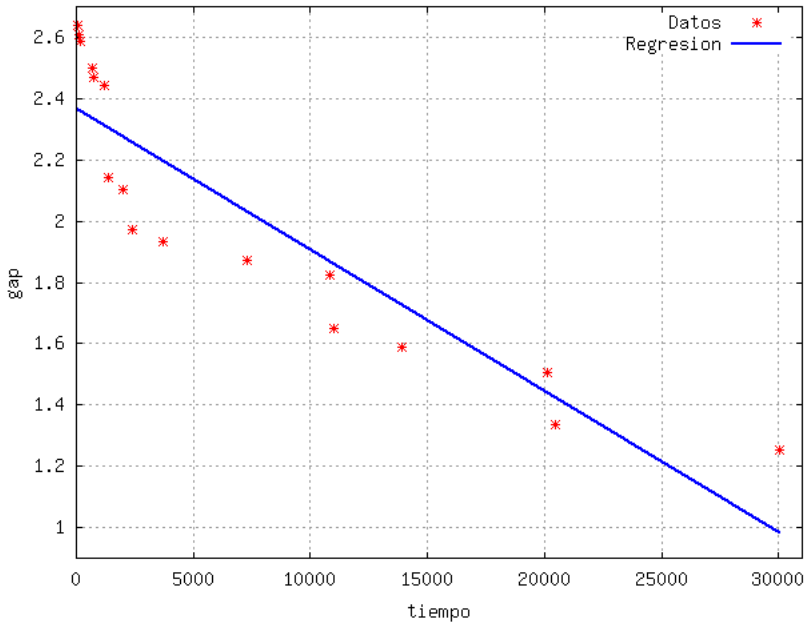


Figura 6.11: Regresión exponencial, estadístico $R = 0.90$.

lo que supone una disminución en el *gap* (diferencia entre la solución relajada y la mejor solución entera conocida) para luego no registrar cambios significativos por

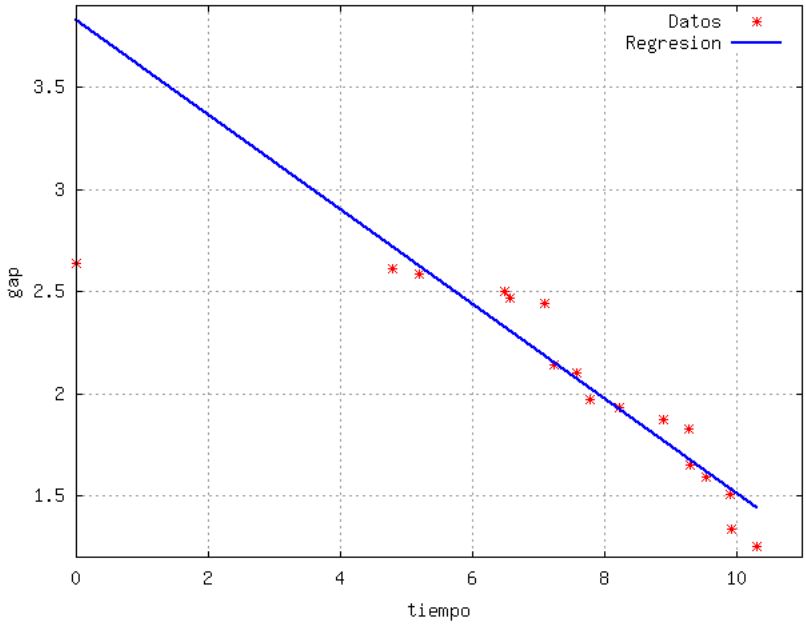


Figura 6.12: Regresión ley de potencias, estadístico $R = 0.95$.

largos periodos o definitivamente no cambiar. Esto habla de lo difícil que puede llegar a ser el problema, y la necesidad de desarrollar herramientas que ayuden a abordarlo de forma eficaz.

6.2 Modelo pseudo-aleatorio

Motivado en los resultados de la sección 6.1.2 en la que se observó que el hecho de que los problemas sin estructura se resuelven más fácilmente, se propone perturbar ligeramente los problemas estructurados, de tal suerte que la perturbación sea significativamente menor que la escala de valores de la función objetivo.

La forma de determinar el valor de las perturbaciones tiene que ser tal que no afecte a la función objetivo mas allá del desempeño en el algoritmo de ramificación y acotamiento. Esto es, que la solución del problema perturbado sea igual a la del problema no perturbado una vez que se eliminan las perturbaciones.

Tomando en cuenta que la función objetivo tiene un costo proporcional al

número de arcos utilizados, el ruido asociado a cada arco debe ser escogido tal que si ocurriese la perturbación máxima en cada arco, la perturbación total sea cuando menos un orden de magnitud menor que el mínimo de los costos asociados a los arcos. De tal forma que:

$$n\delta_{max} \leq \frac{\min(c)}{10}$$

donde n es el número de arcos utilizados, δ_{max} es el valor máximo que puede tomar la variable aleatoria δ y c es el vector de costos asociado al problema. De esta forma se garantiza que la acumulación de las perturbaciones pertenecientes a una solución no generen el efecto de uno o varios arcos adicionales, o que se tenga preferencia por un arco que en términos reales sea mayor que otro (en el caso de minimizar).

Sin embargo el término n no es conocido antes de iniciar la optimización por lo que no es posible utilizarlo para calcular la componente aleatoria. Por lo tanto se optó por utilizar el número total de arcos (N) en su lugar, que de hecho es conocido. De esta forma se tiene que cumplir que:

$$N\delta_{max} \leq \frac{\min(c)}{10}$$

lo que nos da el mismo resultado deseado. De esta manera se le agregaría a cada uno de los costos una variable aleatoria con distribución $U(0, \delta_{max})$ con $\delta_{max} = \min(c)/10N$.

Para saber si dicha modificación en los costos tienen una repercusión en el tiempo de cómputo se desarrolló el siguiente experimento. Se generaron 15 escenarios de forma aleatoria de 8 nodos por lado, con 12 nodos marcados como obstáculos. En seguida se generó un modelo tradicional y uno con una componente pseudo-aleatoria para cada uno de los escenarios. Los resultados se muestran el la tabla 6.6.

Como se puede observar en la tabla 6.6 la media de los tiempos generados por el modelo semi-aleatorio es mayor que el de los tiempos generados por el modelo estructurado. Sin embargo un análisis mas profundo muestra un comportamiento interesante, que se hace evidente al observar gráficamente los cambios en la función objetivo contra el número de iteraciones realizadas por el algoritmo. Las diferencias

Prueba	Estructurado	Semi-aleatorio
1	2930.4	9391.3
2	145.0	480.1
3	5380.7	1173.5
4	317.8	2197.2
5	78.8	719.8
6	484.2	322.7
7	491.6	205.4
8	124.2	126.1
9	23.2	1227.6
10	473.5	53.6
11	4968.4	1006.3
12	605.1	1297.5
13	1023.8	247.4
14	876.0	286.7
15	972.2	6343.0
Media	1259.7	1671.9

Tabla 6.6: Tabla de tiempos para modelo estructurado y modelo semi-aleatorio.

o cambios mostrados en las gráficas se definen como $u^* - u_i$, donde u^* es el valor de la función objetivo óptimo y u_i es el valor de la función objetivo en la i-ésima iteración.

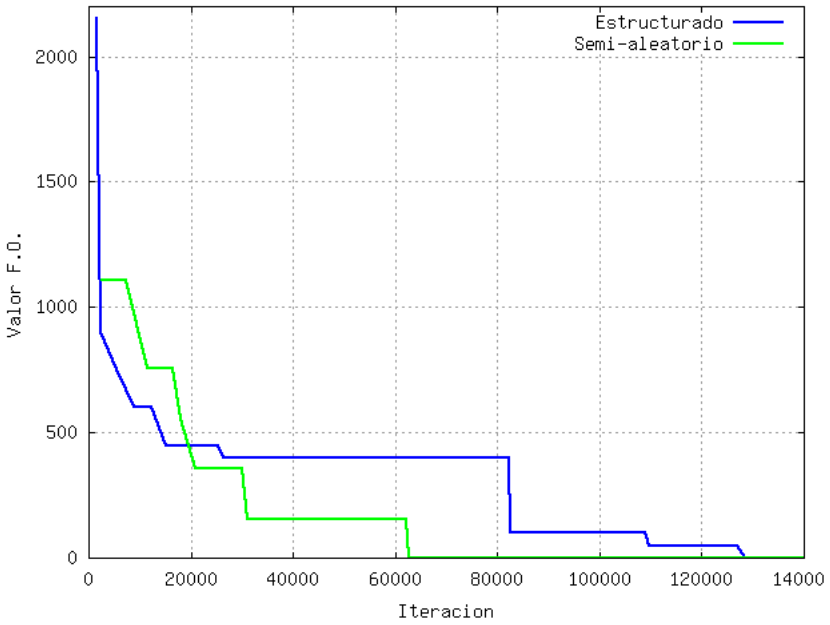


Figura 6.13: Cambios en la función objetivo vs. número de iteraciones, experimento 01.

En las gráficas es posible observar como los cambios en el algoritmo decrecen y entran posteriormente en un estado estable, el cual puede llegar a ser muy prolongado y que sugiere tiempo de cómputo desperdiciado. Al observar este comportamiento se propone un criterio de parada en el cual se identifique el momento en el que los cambios en la función son del orden de la componente aleatoria. Siguiendo este esquema aplicado a las pruebas realizadas anteriormente, el tiempo de cómputo resultante en los experimentos se reduce considerablemente. Dichos cambios se muestran en la tabla 6.7.

En la figura 6.28 se puede ver como se comparan los tiempos en la versión original del modelo contra el modelo semi-aleatorio, y al mismo tiempo éstos con y sin el criterio de parada.

Prueba	Original	Semi-aleatorio
1	300.1	180.0
2	145.0	480.1
3	4687.0	840.9
4	317.8	300.0
5	78.8	360.0
6	491.6	120
7	484.2	205.4
8	124.2	126.1
9	23.2	1227.6
10	473.5	53.6
11	4968.4	120.0
12	605.1	1260.9
13	1023.8	240.0
14	480.0	120.0
15	420.1	1802.7

Tabla 6.7: Tabla de tiempos para modelo estructurado y modelo semi-aleatorio con criterio de parada.

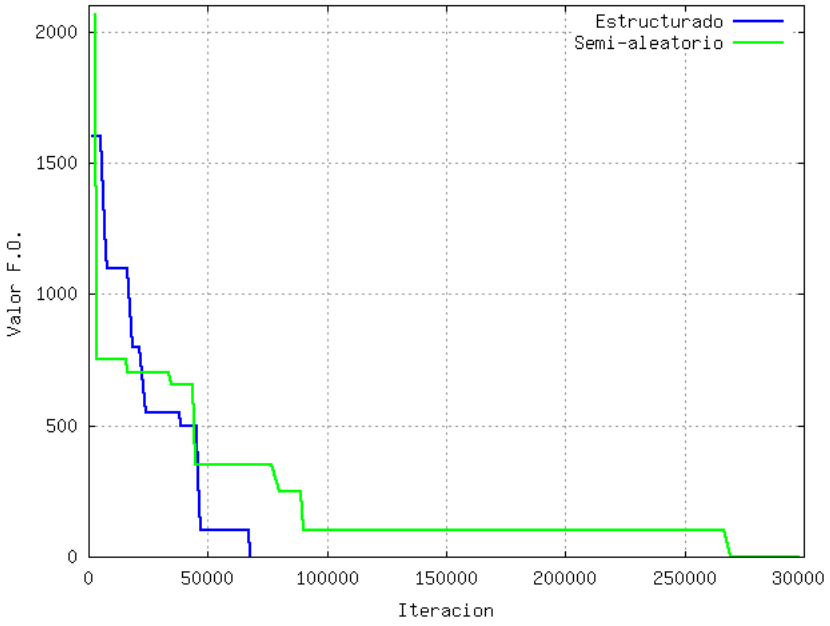


Figura 6.14: Cambios en la función objetivo vs. número de iteraciones, experimento 02.

Si bien es evidente que cualquiera de las versiones con criterio de paro es mejor que su contraparte sin el, ahora resulta la cuestión de cuál de las dos versiones con criterio de paro utilizar, si la versión original o semi-aleatoria. Cabe aclarar que es poco probable encontrar un alternativa que funcione de forma idónea en todos los casos, por lo que se puede escoger aquella que en general se comporta mejor. Para tomar esta decisión se puede hacer uso nuevamente de un diagrama de caja, el cual es una herramienta cualitativa muy útil. En la figura 6.29 se muestra de izquierda a derecha el modelo original sin criterio de parada, el modelo original con criterio de parada, el modelo semi-aleatorio sin criterio de parada y el modelo semi-aleatorio con criterio de parada.

En la figura 6.29 se puede observar que el modelo semi-aleatorio con criterio de parada tiene un mejor comportamiento, ya que además de presentar la menor mediana, también presenta los *outlayers* más bajos. Este resultado es congruente con el hecho de que este enfoque tuvo un mejor desempeño en el 60% de los casos.

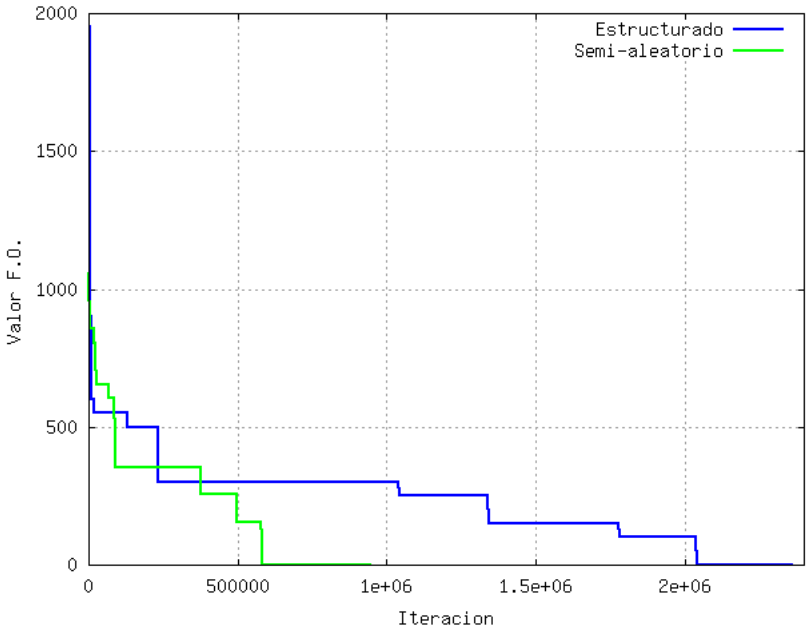


Figura 6.15: Cambios en la función objetivo vs. número de iteraciones, experimento 03.

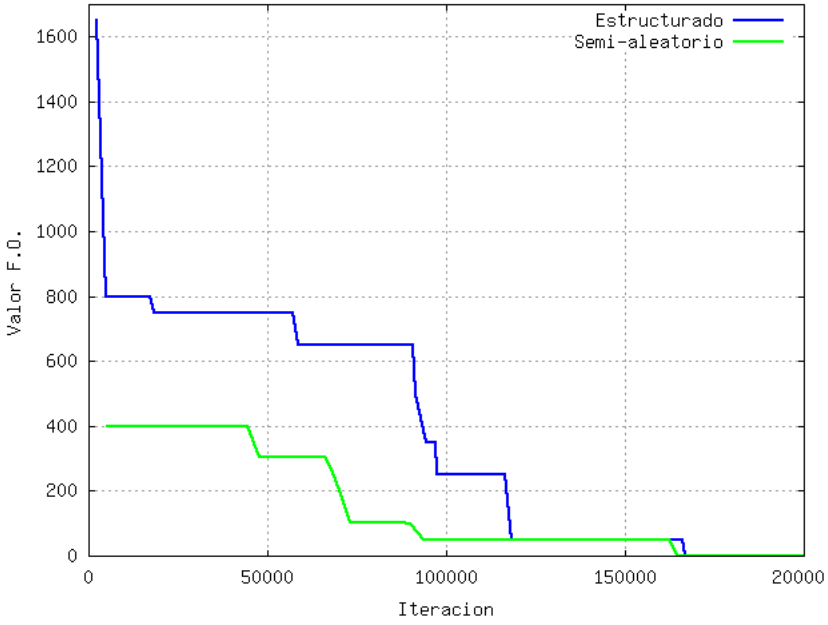


Figura 6.16: Cambios en la función objetivo vs. número de iteraciones, experimento 04.

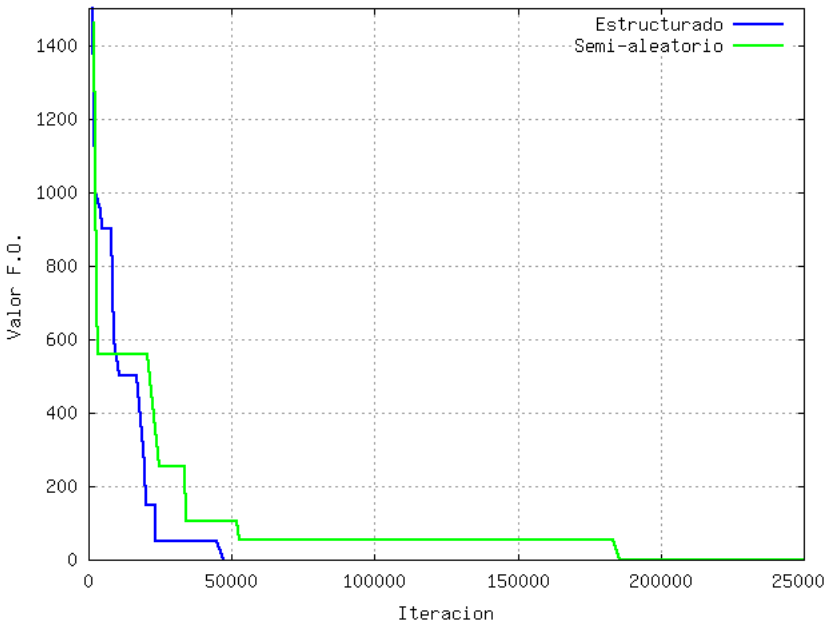


Figura 6.17: Cambios en la función objetivo vs. número de iteraciones, experimento 05.

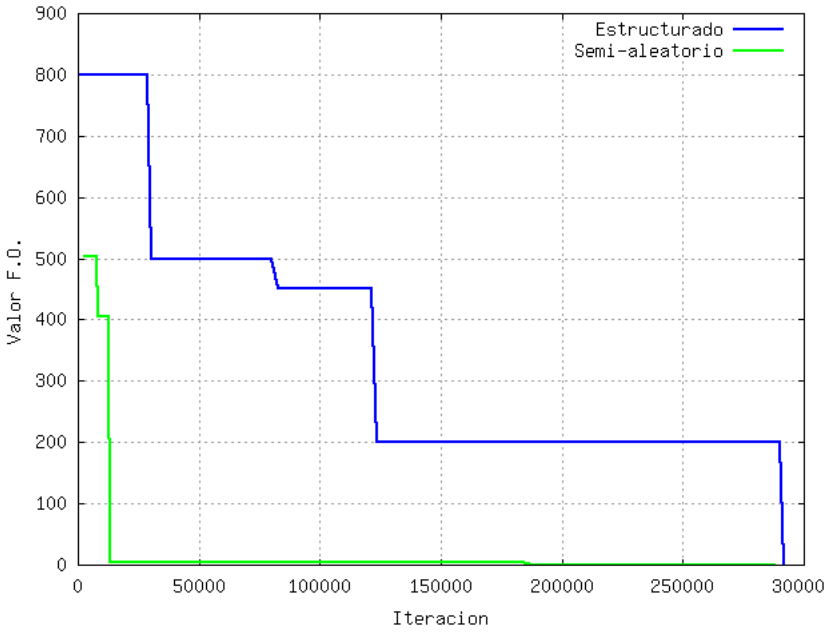


Figura 6.18: Cambios en la función objetivo vs. número de iteraciones, experimento 06

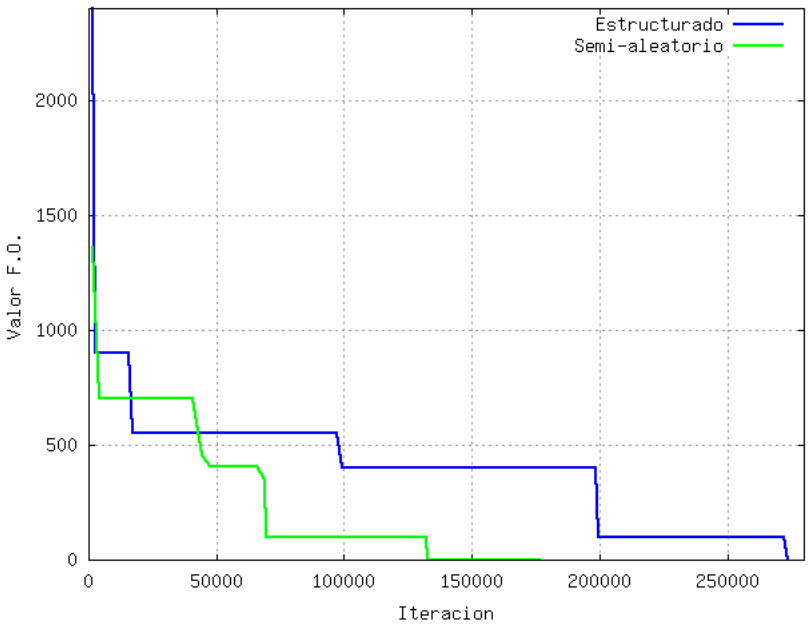


Figura 6.19: Cambios en la función objetivo vs. número de iteraciones, experimento 07.

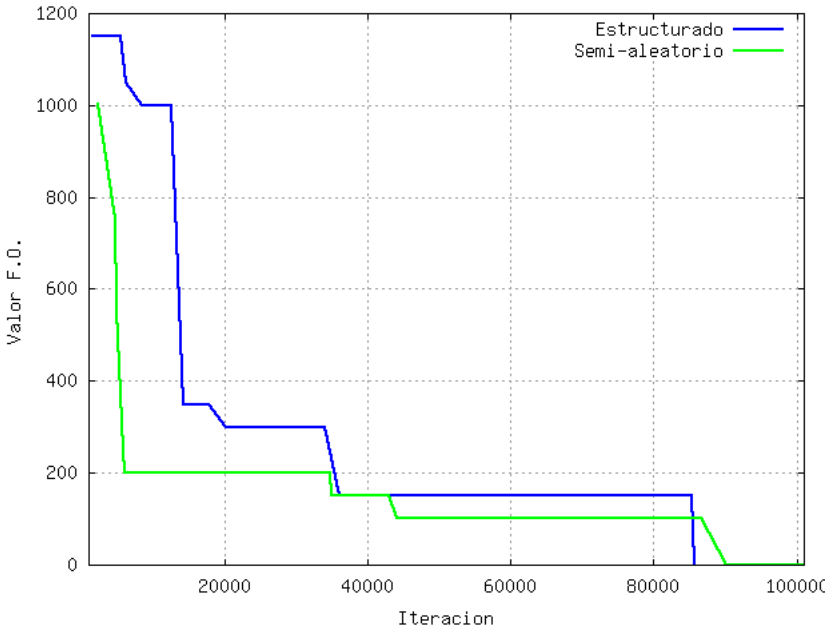


Figura 6.20: Cambios en la función objetivo vs. número de iteraciones, experimento 08.

Figura 6.21: Cambios en la función objetivo vs. número de iteraciones, experimento 09.

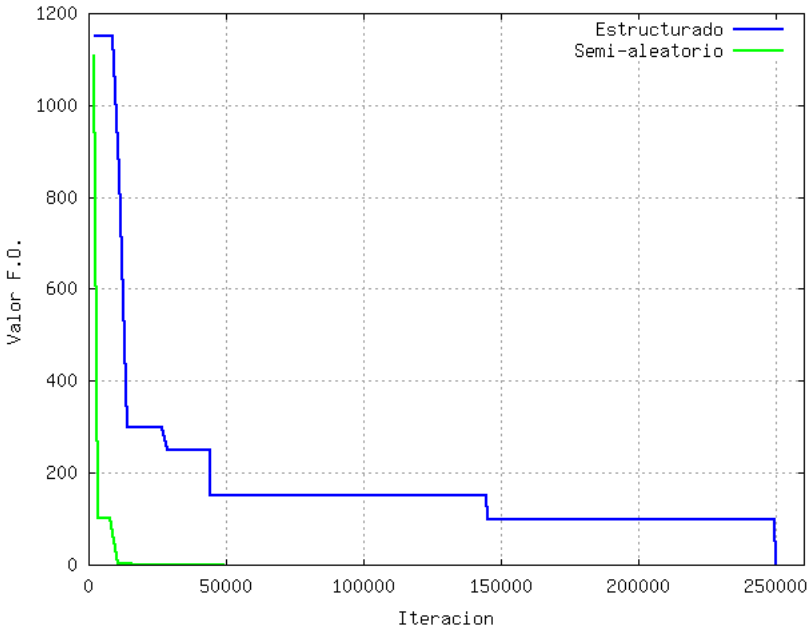


Figura 6.22: Cambios en la función objetivo vs. número de iteraciones, experimento 10.

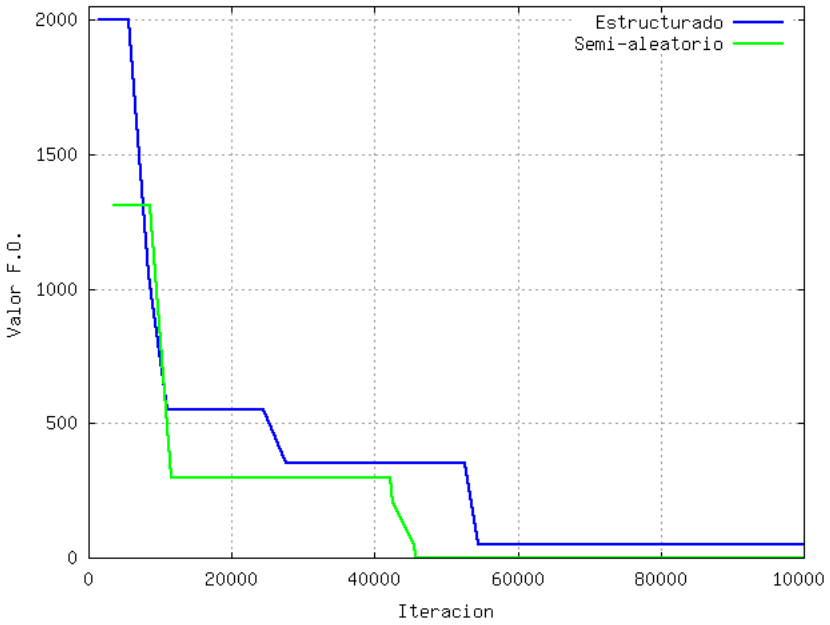


Figura 6.23: Cambios en la función objetivo vs. número de iteraciones, experimento 11.

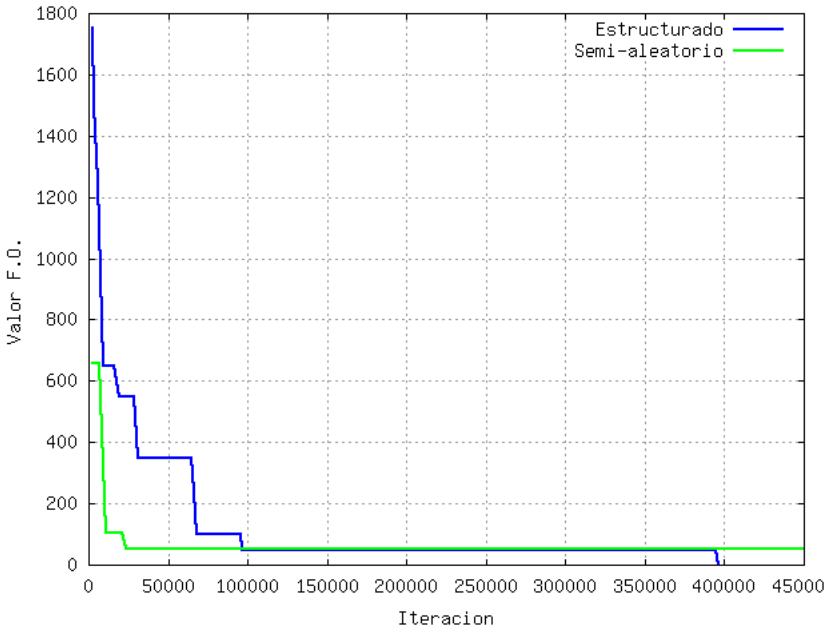


Figura 6.24: Cambios en la función objetivo vs. número de iteraciones, experimento 12.

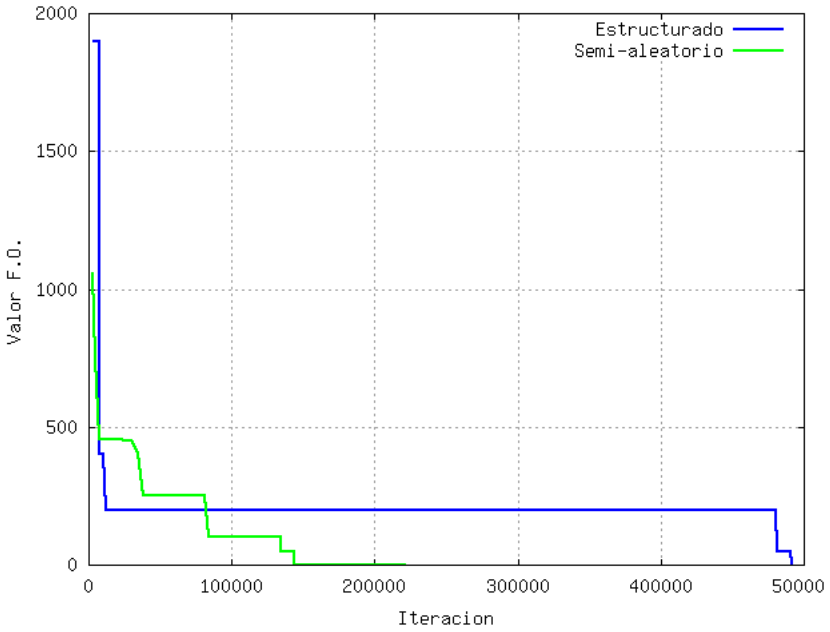


Figura 6.25: Cambios en la función objetivo vs. número de iteraciones, experimento 13.

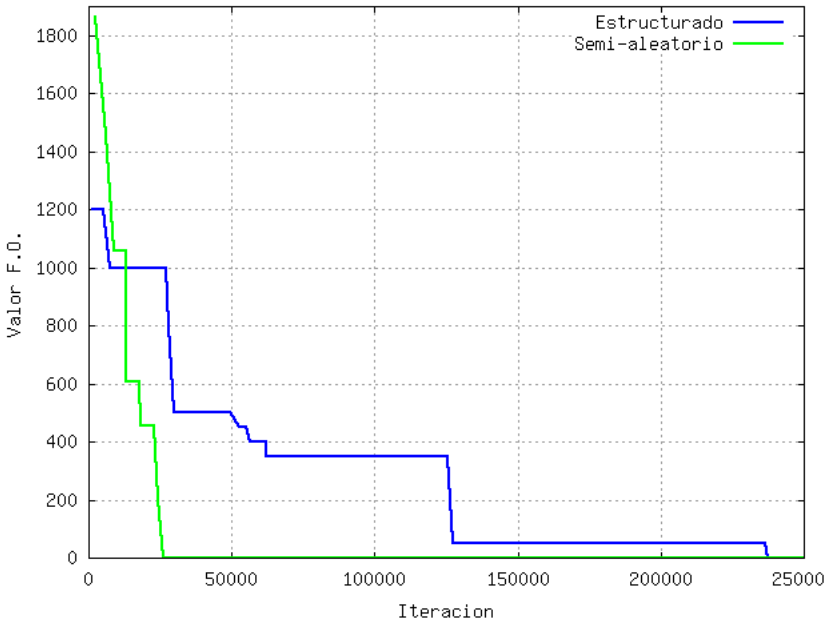


Figura 6.26: Cambios en la función objetivo vs. número de iteraciones, experimento 14.

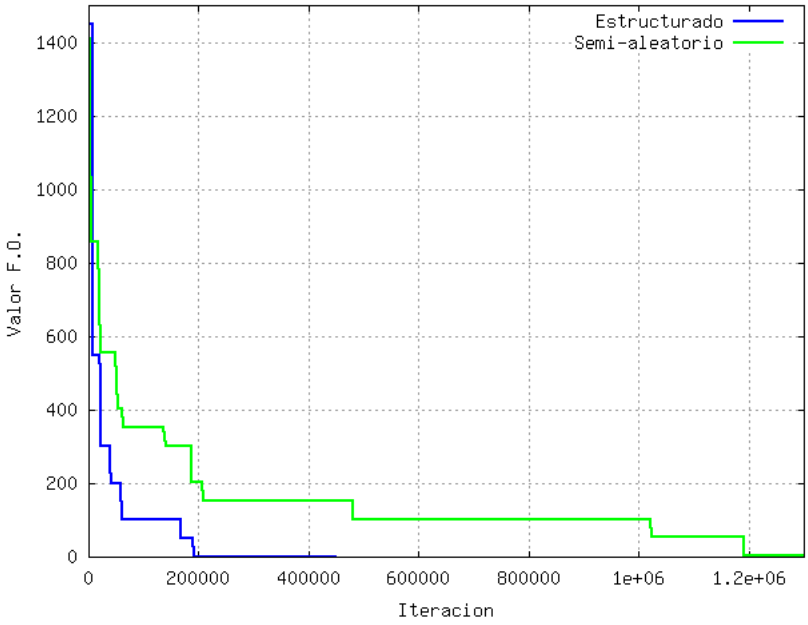


Figura 6.27: Cambios en la función objetivo vs. número de iteraciones, experimento 15.

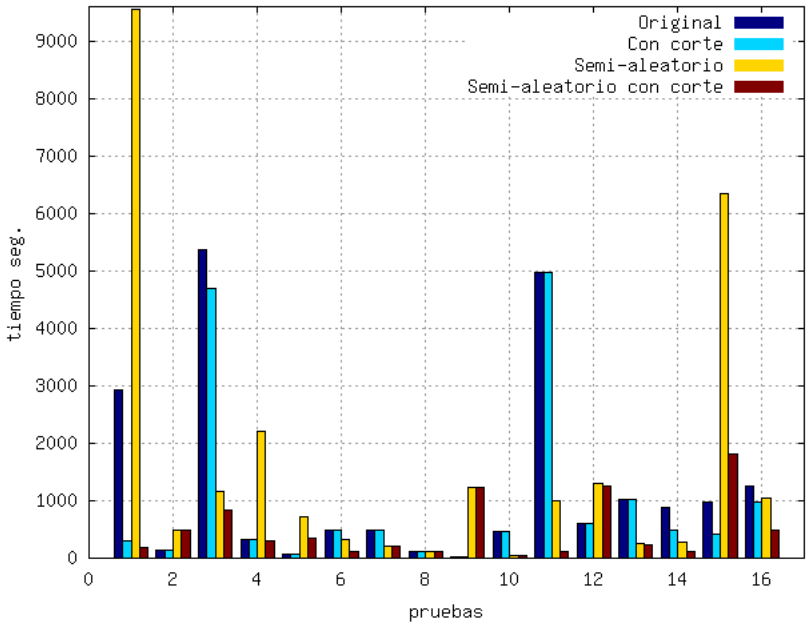


Figura 6.28: Gráfica de barras para los modelos original y semi-aleatorio con y sin criterio de parada.

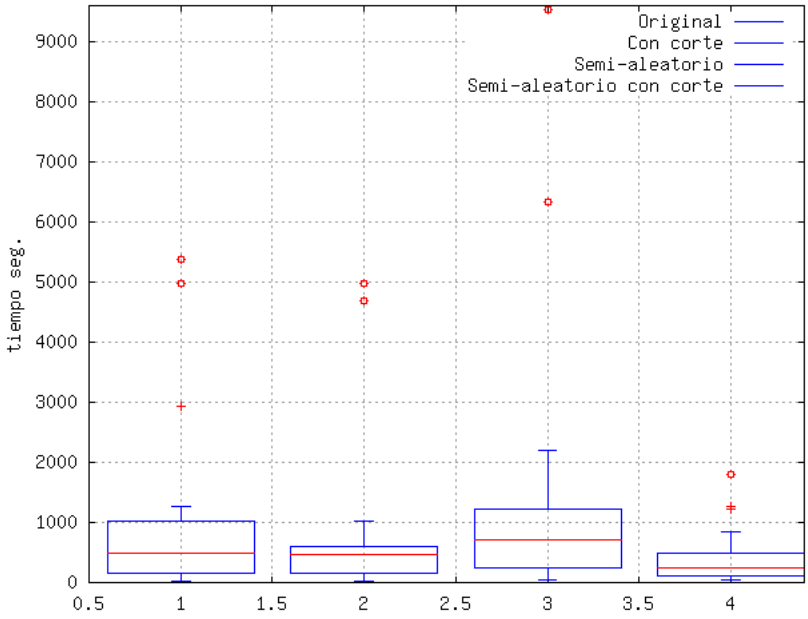


Figura 6.29: Gráfica de caja para los modelos original y semi-aleatorio con y sin criterio de parada.

Capítulo 7

Conclusiones

En el desarrollo de este trabajo se formuló una herramienta para enfrentar el problema de construción relacionado con la trayectoria que ha de seguir un robot para cubrir completa o mayormente una superficie dada.

Dicha herramienta modela el problema de tal manera que no sólo cubre con su trayectoria total o parcialmente el área, sino además es susceptible a un control sobre la trayectoria generada, respondiendo ésta a necesidades específicas del usuario, tales como tiempo de operación, acceso preferencial a áreas de interés, o por el contrario, el control sobre el acceso a áreas restringidas.

Por otra parte el modelo representa satisfactoriamente las restricciones de paso y además, debido a su formulación, facilita una futura implementación sobre ambientes desconocidos y dinámicos.

Un punto que es importante recalcar es la implementación de la herramienta en un robot real, emulando en pruebas de laboratorio aplicaciones que pueden ser observadas en el ámbito industrial.

7.1 Aportaciones

En cuanto a la implementación se tiene que:

- Se desarrolló una herramienta para traducir de forma automática el conocimiento

del entorno en el modelo que será optimizado.

- Se formuló y resolvió el problema por medio de una herramienta de distribución libre.
- Se desarrolló un traductor tal que la solución es dada como una serie de instrucciones.
- El modelo puede encontrar la trayectoria óptima sobre terrenos con obstáculos irregulares y de dificil acceso.

En cuanto al problema abordado:

- Se mostró que no existe suficiente evidencia para suponer cambios en la eficiencia del modelo atribuidos a la configuración de los obstáculos.
- Se mostró cómo un problema con costos estructurados resulta más difícil (tomando el tiempo de cómputo como referencia) que uno con costos aleatorios.
- Se mostró que la variación del problema del camino hamiltoniano de menor costo que se aborda en este trabajo supone una mayor dificultad que el original.
- Se detectó un comportamiento en el desarrollo del algoritmo el cual puede ser utilizado para disminuir drásticamente el tiempo de cómputo necesario para encontrar una solución.
- Se propuso una técnica para disminuir el tiempo de cómputo a través de la adición de una componente aleatoria.

7.2 Trabajo a futuro

Se propone realizar la formulación para el caso dinámico y desconocido utilizando como base la representación por medio de grafos y la abstracción del entorno por medio de la matriz de obstáculos.

Desarrollar completamente una interfaz amigable tal que no resulte complicado el implementar la metodología por parte de un usuario final. Una posible implementación en agentes que hagan uso de un mando central para operaciones complejas y que deleguen al agente solo tareas básicas como evitar obstáculos o regresar a su estación de recarga tienen una fuerte relación con el trabajo presentado.

Analizar la posibilidad de abordar el problema con el enfoque de la optimización dinámica y analizar si existen patrones en las variaciones que presenta el robot al desplazarse en diversas superficies.

Bibliografía

[1] Ahuja, Maganti y Orlin, *Network Flows. Theory, algorithms, and applications*, Prentice Hall, 1993.

[2] Bazaraa, M. S., J. J. Jarvis y H. D. Sherali, *Programación lineal y flujo en redes*, segunda edición, Limusa, México, DF, 2004.

[3] Choset, «Coverage for robotics. A survey of recent result», *Annals of Mathematics and Artificial Intelligence*, **31**, págs. 113 – 126, 2001.

[4] Dantzig, G. B., R. Fulkerson y S. Jhonson, «Solution of a large-scale traveling-salesman problem», *Operations Research*, **2**(4), págs. 393–410, Noviembre 1954.

[5] Diestel, R., *Graph Theory*, Springer, 2006.

[6] Dorigo, M., *Learning and natural algorithms*, Tesis Doctoral, Politecnico di Milano, 1992.

[7] Eaton, J. W., «GNU Octave», .

[8] Elfes, A., «Sonar-based real-world mapping and navigation», *IEEE Journal of robotics and automation*, **3**, págs. 249 – 265, 1987.

[9] Fourer, R., D. M. Gay y B. W. Kernighan, *AMPL: A modeling languaje for mathematical programing*, Brooks/Cole Publishing Company, 2002.

[10] GABRIELY, Y. y E. RIMON, «Spanning-tree based coverage of continuous areas by a mobile robot», *Annals of Mathematics and Artificial Intelligence*, **31**(1-4), págs. 77–98, Octubre 2001.

[11] HARTMANN, A. K. y M. WEIGT, *Phase transitions in combinatorial optimization problems. Basics, algorithms and statistical mechanics*, Wiley-VCH, 2005.

[12] HINES, W. W., D. C. MONTGOMERY, D. M. GOLDSMAN y C. M. BORROR, *Probabilidad y estadística para ingeniería*, cuarta edición, Compañia editorial continental, 2005.

[13] KARP, R. M., «Reducibility among combinatorial problems», *Complexity of computer computations*, 1972.

[14] KHATIB, O., «Real-time obstacle avoidance for manipulators and mobile robots», *International journal of robotics research*, **5**, págs. 90 –98, 1986.

[15] KORTE, B. y J. VYGEN, *Combinatorial Optimization, Theory and algorithms*, tercera edición, Springer, 2005.

[16] LAND, A. H. y A. G. DOIG, «An Automatic method of solving discrete programming problems», *Econometrica*, **28**(3), págs. 497–520, Julio 1960.

[17] LAWLER, E., *Combinatorial Optimization, Networks and Matroids*, DOVER Publications, 2001.

[18] LESNIAK-FOSTER, L. y J. E. WILLIAMSON, «On spanning and dominating circuits in graphs», *Canadian Mathematical Bulletin*, **20**, 1977.

[19] LITTLE, J. D. C., K. G. MURTY, D. W. SWEENEY y C. KAREL, «An algorthm for the traveling salesman rpoblem», *Operations Research*, **11**(6), págs. 972–989, Noviembre 1963.

[20] MAKHORIN, A., «Modeling Language GNU MathProg», .

[21] MAKHORIN, A., *Modeling Language GNU MathProg Language Reference*, draft edition for glpk version 4.34 edición, Diciembre 2008.

[22] MOORE, G. E., «Cramming more components onto integrated circuits», *Electronics Magazine*, **38**(8), abril 1965.

[23] MORAVEC, H. y A. ELFES, «Hight resolution maps for wide angles sonar», *IEEE International conference on robotics and Automation*, 1985.

[24] NEMHAUSER, G. y L. WOLSEY, *Integer and combinatorial optimization*, Wiley-Interscience, 1988.

[25] TURING, A. M., «On computable numbers, with an application to the Enscheidungsproblem.», *Proceedings of the London Mathematical Society*, **42**, 1936.

[26] YANG, S. X. y C. LUO, «A neural network approach to complete coverage path planning», *IEEE Transactions on sistems, man, and cibernetics*, **34**(1), págs. 718 – 724, Febrero 2004.

[27] ZELINSKY, A., R. JARVIS, J. BYRNE y S. YUTA, «Planning paths of complete coverage of an unstructured enviroment by a mobile robot», en *Proceedings of international conference on advanced robotics*, págs. 533 – 538, 1993.

Ficha autobiográfica

Ing. David Roberto Valenzuela Vega

Candidato para el grado de Maestro en Ciencias
con especialidad en Ingeniería de Sistemas

Universidad Autónoma de Nuevo León

Facultad de Ingeniería Mecánica y Eléctrica

Tesis:

Optimización de la trayectoria de un agente con medidas de desempeño asociadas

Nací el 17 de febrero de 1982. En el periodo 1999-2004 realicé mis estudios de licenciatura en el Instituto Tecnológico de San Luis Potosí obteniendo el título de Ingeniero Industrial en Calidad y Productividad. Presenté el estudio: Moore vs. Cemb, un análisis estadístico sobre la variabilidad en las mediciones de la oscilación axial en rines de acero como requisito para la Opción de titulación VI: Examen por áreas de conocimiento, bloque probabilidad y estadística. En el periodo 2004-2008 me desenvolví en el ámbito laboral en empresas del ramo automotriz y aeroespacial. A partir de enero de 2008 soy estudiante del Programa de Posgrado en Ingeniería de Sistemas.