ESCUELA SUPERIOR POLITÉCNICA DEL LITORAL

Instituto de Ciencias Matemáticas

"Diseño e Implementación de un Algoritmo de tipo Recocido Simulado para la resolución del Problema del Agente Viajero"

TESIS DE GRADO

Previo a la obtención del Título de:

INGENIERO EN ESTADÍSTICA INFORMÁTICA

Presentada por:

Elías David Araujo Carrión

GUAYAQUIL – ECUADOR

Año: 2006

TRIBUNAL DE GRADUACIÓN

Ing. Robert Toledo E.
PRESIDENTE DEL TRIBUNAL

Mat. Johny Bustamante R.
DIRECTOR DE TESIS

Ing. Soraya Solís G.
VOCAL

Mat. César Guerrero L.
VOCAL

DECLARACIÓN EXPRESA

"La responsabilidad del contenido de esta Tesis de Grado, me corresponden exclusivamente; y el patrimonio intelectual de la misma a la ESCUELA SUPERIOR POLITÉCNICA DEL LITORAL"

(Reglamento de Graduación de la ESPOL).

DAVID ARAUJO C.

Elías David Araujo Carrión
CI: 070339184-7

AGRADECIMIENTO.

A las Familias: Marín Ríos, Sanchez Araujo,.

A mis amigos: Ligia, Betsy, Sonia, Adriana,

Jenny, Nadia, Denny, Jazmin, Mave, Mariela

A mis familiares: Hector, Ceilán, Ximena, Katty

A Daniela por todo su apoyo.

DEDICATORIA.

A Dios, A Mis Hermanos, A Mis Cuñado(a)s,

A Mis Sobrino(a)s, Y, A Las Personas Que

Siempre Estuvieron En Todo Momento

Apoyándome Y Dandome Todo El Amor

Posible, Mis Padres: Jorge y Fanny.

RESUMEN

En el presente trabajo resolveremos el problema del Agente Viajero haciendo uso del Algoritmo del Recocido Simulado, para determinar las rutas válidas entre dos o más nodos. El algoritmo fue aplicado en el sector céntrico de Guayaquil considerado como el sector 44 en el plano General de Guayaquil.

El algoritmo de Recocido Simulado fue diseñado para el área de termodinámica. Dada la dificultad práctica para resolver de forma exacta (simplex, "ramificación y acotación", teoría de grafos, etc) toda una serie de importantes problemas combinatorios complejos ya sea por que utilizan un tiempo no polinomial de resolución o por que no necesitan llagar a un óptimo, comenzaron aparecer algoritmos que proporcionan soluciones factibles (es decir, que satisfacen las restricciones del problema), las cuales aunque no optimicen la solución objetivo, se supone que al menos se acercan al valor

óptimo en un tiempo de cálculos razonables. Podríamos llamarlas en lugar de óptimas, "satisfactorias", pues al menos es de suponer que son lo suficientemente buenas como para servirnos. Este tipo de algoritmo se denominan heurísticos.

Esta tesis está dividida en 6 capítulos además de las conclusiones y recomendaciones. Capítulo 1, llamado Teoría de Grafos que nos muestra antecedente necesario para la compresión de esta Tesis. En el Capítulo 2 se presenta una amplia descripción de la Heurística y sus formas de resolver problemas difíciles. En el siguiente Capítulo, se analiza la teoría relacionada con la implementación computacional de este algoritmo, por lo cual se le ha llamado al capítulo Complejidad Computacional.

Los capítulos 4,5 y 6 son el desarrollo de mi tesis, en el Capítulo 4 presento el diseño del algoritmo de tipo Recocido Simulado, su analogía paramétrica, el Modelamiento, el algoritmo y como resuelve el Problema del Agente Viajero. En el Capítulo 5 se realizó la implementación del algoritmo de Recocido Simulado V.2, donde se comenzó el levantamiento de la información de campo, se determinó rutas y nodos, georeferenciación de planos y puntos, determinación de la matriz de distancia y la implementación computacional en sí, considerando importante dedicar el capítulo 6 únicamente al análisis de resultados.

ÍNDICE GENERAL

Pág.

ÍNDICE DE FIGURAS.

Pág.

ÍNDICE DE TABLAS.

Pág.

ABREVIATURAS.

α	Alfa (valor < 1).
e	Aristas.
β	Beta (real positivo cercano a cero).
ΔE	Cambio de energía.
S	Conjunto de Soluciones.
K	Constante de Boltzman.
c(i)	Costo de la solución.
ε	Excilon.
e	Exponencial.
FRZN	Frozen (Parámetro de congelamiento).
$f(x)$	Función de la variable x.
Z	Función Objetivo.
t(n)	Función tiempo.
G	Grafo.
log n	Logaritmo de n.
ln	Logaritmo natural.
A(G)	Matriz de Adyacencia de un Grafo.
M(G)	Matriz de Incidencia de un Grafo.

Min	Minimizar.
N	Nodos
Np.	No polinomial.
m_{ij}	Número de veces que v y e son incidentes.
a_{ij}	Números de aristas que unen dos vértices.
N	Número entero
Ω	Omega (Rango)
$\forall$	Para todo.
$\in$	Pertenece a.
PAV (TSP).	Problema del Agente Viajero.
P(j)	Probabilidad de Boltzman.
Pi	Puntos
RS.	Recocido Simulado
$\sum$	Sumatoria.
T	Temperatura del sistema.
T_0	Temperatura inicial.
i_0	Solución inicial.
V	Vértices.

INTRODUCCION

Hoy en día, en el mundo cambiante y competitivo en que vivimos, la toma de decisiones incorrectas puede traer pérdidas de recursos tanto tangibles como intangibles, por tal motivo debemos basarnos en características que determinen que las decisiones tomadas sean las mejores

Esto hemos observado en el caso de optimización de rutas, tomemos en consideración el caso de Distribución de Agua Potable, que depende de características como el relieve del suelo, la precisión del agua, kilómetros a recorrer, etc. Del tal forma que para tener un buen servicio hubo la necesidad de medir los costos de llevar el agua a los hogares por diferentes rutas, donde la responsabilidad de tomar una buena decisión para la transportación de este líquido vital era primordial. De igual forma, con grandes y pequeñas redes entre las que podemos citar: la red de distribución de Energía, el Servicio Telefónico e incluso para la transmisión de datos es necesario valuar los medios y vías para que la información llegue a tiempo.

Actualmente en países desarrollados se está ahorrando recursos por la optimización de rutas, el saber escoger la ruta es importante a la hora de

brindar un servicio, existen problemas en empresas que son más pequeños pero no menos importantes, como por ejemplo transporte de personas, de mercadería, de información, etc. Uno de los problemas combinatorios más famosos es el Problema del Agente Viajero (PAV).

El PAV trata de determinar, dado un mapa de carreteras en que orden debe visitarse n ciudades de forma que partiendo de una cualquiera se recorra el menor número de kilómetros y se vuelva al punto de partida tras visitar una sola vez cada una de ellas.

El propósito de esta investigación es llevar a cabo la implementación del algoritmo heurístico de Recocido Simulado (RS) para la solución del Problema del Agente Viajero. El algoritmo RS es un método global de optimización muy hábil para escapar de óptimos locales que efectúa una búsqueda estocástica sobre el espacio de soluciones. De los resultados obtenidos puede verse que el algoritmo RS mejora la calidad de la solución reportada por otros métodos y permite, además, establecer un balance entre eficacia y eficiencia. Este método se lo diseñó e implementó para poder obtener soluciones satisfactorias en problemas complejos que involucra la optimización en diversos campos de la vida, ya que tiene la propiedad de que, con una propia calibración en un tiempo polinomial produce buenas soluciones.

CAPÍTULO I

1. TEORÍA DE GRAFOS.

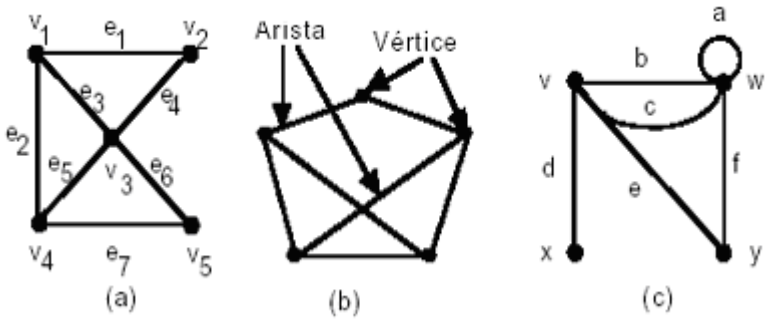
1.1. Grafos No Dirigidos

En un sentido amplio, un grafo G es un diagrama que, si se interpreta en forma adecuada proporciona información. Cuando los grafos no tienen dirección alguna, se dice que son grafos no dirigidos, por lo que en esta sección se discutirá sobre ellos mencionándolos simplemente como grafos.

Lo más importante de un grafo son sus aristas y vértices.

Se denotan: Grafo (G), los vértices (v1, v2,….., vv), y las aristas por (e1, e2,….., ee).

Figura 1.1.
Vértices y Aristas.



Fuente: Castañeda Roldán, C. Y. 2000.
Elaborado por: Elías David Araujo Carrión

Las aristas adyacentes en un camino tienen un vértice común. Por lo tanto, un camino determina una sucesión de vértices. Las sucesiones de vértices correspondientes a los caminos de la figura 1.1.c están representados en la siguiente tabla:

Tabla 1.1.
Caminos de la Figura 1.1.c

El camino	Su Sucesión de vértices
d-b-f-e	x-v-w-y-v
c-f-e-b-a	v-w-y-v-w-w
b-a-f-e-c	v-w-w-y-v-w
c-a-f-e-b	v-w-w-y-v-w
b-a-b-e-f-a-a-b	v-w-w-v-y-w-w-w-v

Fuente: Castañeda Roldán, C. Y. 2000.
Elaborado por: Elías David Araujo Carrión

Es interesante detectar ciertos detalles como que el número de vértices en una sucesión de vértices supera en uno al de las aristas en un camino.

1.2. Camino Cerrado

Un camino es cerrado si el primero y el último vértice de la sucesión de vértices son el mismo. Un ciclo es un camino cerrado eficiente en el sentido de que no repite aristas y en la sucesión de vértices todos son distintos, excepto el primero y el último.

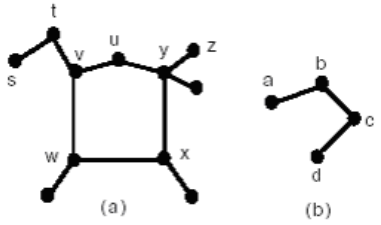
1.3. Grafos Cíclicos y Acíclicos

Grafo cíclico es aquel camino que recorre todos los vértices (nodos) de un grafo pasando una y sólo una vez por cada arco (arista) del grafo, siendo condición necesaria que regrese al vértice inicial de salida (ciclo = camino en un grafo donde coinciden vértice inicial o de salida y vértice final o meta). Una definición más formal lo define como: "aquel ciclo que contiene todas las arístas de un grafo solamente una vez".

Un grafo es acíclico si no tiene ciclos. Un camino es acíclico si el subgrafo que contiene las aristas y los vértices del camino es acíclico. El grafo de la figura 1.2.a no es acíclico ya que v-u-y-x-w es un ciclo.

El grafo de la figura 1.2.b es acíclico ya que no contiene ciclos .

Figura 1.2.
(a) Grafo No Acíclico; (b) Grafo Acíclico



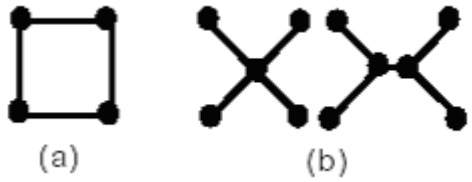
Fuente: Castañeda Roldán, C. Y. 2000.
Elaborado por: Elías David Araujo Carrión

1.4. Grafos Conexo y no Conexo

Un grafo no dirigido es conexo si todos sus pares de vértices distintos están conectados por un camino en el grafo.

Ver figura 1.3.a donde se muestra un grafo conexo y en la figura 1.3.b un grafo no Conexo.

Figura 1.3.
(a) Grafico Conexo; (b) grafico no Conexo.



Fuente: Castañeda Roldán, C. Y. 2000.
Elaborado Por: Elías David Araujo Carrión

1.5 Matriz de Incidencia y Matriz de Adyacencia.

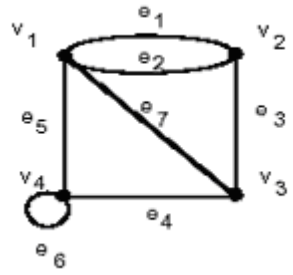
Cada grafo puede ser representado con estructuras matemáticas, una de estas estructuras consiste en que cada grafo puede tener una matriz de incidencia y una matriz de adyacencia, así :

La matriz de incidencia de un Grafo es la matriz M(G) = [m_{ij}], donde m_{ij} es el número de veces (0, 1 o 2) que v_i y e_j son incidentes. La matriz incidente de un grafo es justo un camino diferente de especificar al grafo.

Otra matriz asociada con G es la **matriz de adyacencia**, esta es la matriz v x v A(G) = [a_{ij}], en donde a_{ij} es el número de aristas que unen a v_i con v_j. Se muestra a continuación en la Figura 1.4 un ejemplo de un grafo, en la tabla 1.2 su matriz de incidencia, y en la tabla 1.3 su matriz de adyacencia.

La matriz de adyacencia de un Grafo es generalmente considerada más pequeña que su matriz de incidencia, y es en esta forma que un grafo se almacena comúnmente en las computadoras.

Figura 1.4.
Grafo. (G).



Fuente: Castañeda Roldán, C. Y. 2000.
Elaborado por: Elías David Araujo Carrión

Tabla 1.2.
Matriz de Incidencia M(G)

	e_1	e_2	e_3	e_4	e_5	e_6	e_7
V_1	1	1	0	0	1	0	1
V_2	1	1	1	0	0	0	0
V_3	0	0	1	1	0	0	1
V_4	0	0	0	1	1	2	0

Fuente: Castañeda Roldán, C. Y. 2000.
Elaborado por: Elías David Araujo Carrión

Tabla 1.3.
Matriz de Adyacencia. A(G).

	V_1	V_2	V_3	V_4
V_1	0	2	1	1
V_2	2	0	1	0
V_3	1	1	0	1
V_4	1	0	1	1

Fuente: Castañeda Roldán, C. Y. 2000.
Elaborado por: Elías David Araujo Carrión

CAPÍTULO II

2. HEURÍSTICAS.

2.1. Clasificación.

Las heurísticas se clasifican de acuerdo a su tiempo de resolución, cuando necesitan un tiempo polinomial para su resolución se denominan de clase P y cuando se necesita un tiempo exponencial se denomina NP estos a su vez se subdividen en NP duros y NP no duros.

2.2. Problemas P, Np Y Np-Completos

Cuando nos enfrentamos a un problema concreto, habrá una serie de algoritmos aplicables. Se suele decir que el orden de complejidad de un problema es el del mejor algoritmo que se conozca para resolverlo. Así se clasifican los problemas, y los estudios sobre algoritmos que se aplican a la realidad. Estos estudios han llevado a la constatación de que existen problemas muy difíciles, problemas que desafían la utilización de los ordenadores para resolverlos. En lo que sigue esbozaremos las clases de problemas que hoy por hoy se escapan a un tratamiento informático.

Clase P. Los algoritmos de complejidad polinómica se dice que son tratables en el sentido que suelen ser abordables en la práctica. Los problemas para los que se conocen algoritmos con esta complejidad se dice que forman la clase P. Aquellos problemas para los que la mejor solución que se conoce es de complejidad superior a la polinómica, se dice que son problemas intratables. Sería muy interesante encontrar alguna solución polinómica (o mejor) que permitiera abordarlos.

Clase NP. Algunos de estos problemas intratables pueden caracterizarse por el curioso hecho de que puede aplicarse un algoritmo polinómico para

comprobar si una posible solución es válida o no. Esta característica lleva a un método de resolución no determinista consistente en aplicar heurísticos para obtener soluciones hipotéticas que se van desestimando (o aceptando) a ritmo polinómico. Los problemas de esta clase se denominan NP (la N de no - determinísticos y la P de polinómicos).

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

2.3. El Problema De Satisfactibilidad

Dada la dificultad de dar solución a problemas combinatorios de tipo NP (que no tienen un tiempo polinomial de respuesta), y ante la necesidad de ofrecer alguna solución dado su interés práctico, surgen diversos algoritmos que proporcionan soluciones factibles, es decir, que satisfacen las restricciones del problema, las cuales aunque no optimicen la función objetivo, al menos se acercan al valor óptimo en un tiempo de cálculo razonable, podríamos llamarlas en lugar de óptimas, “satisfactorias”, pues son suficientemente buenas como para servirnos.

Estos algoritmos son de tipo Heurísticos debido a que necesitan un tiempo polinomial para su resolución

Una posible manera de definir estos métodos es: como “procedimientos simples, a menudo basados en el sentido común; que se supone ofrecerán una buena solución (aunque no necesariamente la óptima) a problemas difíciles, de un modo fácil y rápido [Zanakis, Evans, 1981].

Es recomendable la utilización de algoritmos heurísticos para la resolución de un problema cuando:

- No existe un algoritmo exacto de resolución o este requiere mucho tiempo de cálculo o de memoria.
- Cuando no se necesita la solución óptima.

- Cuando los datos son poco fiables.
- Cuando limitaciones de tiempo, espacio (para almacenamiento de datos), etc. Obliguen al empleo de métodos de rápida respuesta, aún a costa de la precisión.
- Como paso intermedio en la aplicación de otro algoritmo.

El problema de satisfactibilidad consiste en:

Sea:

$Min\ Z = f\ \ x$ → función objetivo

$\forall x \in \Omega \rightarrow$ Restricción

Se define Ω como el rango donde se encuentra todos los puntos posibles x_i para que la solución sea satisfactoria. Pero para hacer más satisfactorio la solución del problema es necesario acotar al máximo el rango Ω a Ω_1, lo cual se lo puede hacer por varios métodos. Los cuales se detallan a continuación:

Métodos Constructivos.- Consisten en ir paulatinamente añadiendo componentes individuales a la solución e ir evaluando hasta que se obtiene una solución lo suficientemente buena.

Métodos de Descomposición.- se trata de dividir el problema en subproblemas más pequeños, siendo el output de su siguiente, de forma que

al resolverlos todos obtengamos una solución para el problema global (Divide y vencerás).

Métodos de reducción.- Tratan de identificar alguna característica que presumiblemente deba poseer la solución óptima y de ese modo simplificar el problema.

Manipulación del modelo.- Estas heurísticas modifican la estructura del modelo con el fin de hacerlo mas sencillo de resolver, deduciendo, a partir de su solución, la solución del problema original.

Métodos de búsqueda por entornos.- A esta categoría pertenecen la heurística en la que estamos interesados (RS). Estos métodos parten de una solución factible inicial (obtenida quizá mediante otra heurística) y, mediante alteraciones de esa solución, van pasando de forma iterativa, y mientras no se cumpla el criterio de parada, a otras soluciones factibles de su “entorno”, almacenando la mejor de las alternativas visitadas.

Los algoritmos tradicionales de búsqueda por entornos parten de una solución inicial, que de modo paulatino es transformada en otras que a su vez son mejoradas al introducirles pequeñas perturbaciones o cambios (tales como cambiar el valor de una variable o intercambiar los valores que tienen

dos variables). Si este cambio da lugar a una solución "mejor" que la actual, se sustituye ésta por la nueva, continuando el proceso hasta que no es posible ninguna nueva mejora. Esto significa que la búsqueda finaliza en la mejor solución visitada hasta ese momento.

Aproximación: Un algoritmo que rápidamente encuentra una solución no necesariamente óptima, pero dentro de un cierto rango de error. En algunos casos, encontrar una buena aproximación es suficiente para resolver el problema, pero no todos los problemas NP-completos tienen buenos algoritmos de aproximación.

Relajación: Una cuestión relevante al abordar un problema real es la obtención de un modelo que permita emplear una técnica de resolución apropiada. Si con este modelo el problema resulta difícil de resolver se acude a modelos modificados con lo que es más sencillo encontrar buenas soluciones o en los que los procedimientos son más eficientes. Una relajación de un problema es un modelo simplificado obtenido al eliminar, debilitar o modificar restricciones (u objetivos) del problema real. En cualquier formulación siempre existe algún grado de simplificación, lo que puede afectar en mayor o menor medida el ajuste a la realidad de los

procedimientos y de las soluciones propuestas. El modelo de relajación son estrategias para el empleo de relajaciones del problema en el diseño de heurísticas, las buenas relajaciones son las que simplifican el problema y hacen más eficiente los procedimientos de solución, pero cuya resolución proporciona muy buena solución del problema original. Por ejemplo, para un problema de programación lineal entera, su relajación lineal consiste en ignorar las restricciones de que las variables sean enteras. Se utilizan frecuentemente para aplicar procedimientos eficientes de programación lineal, como el método Simplex, a dicha relajación y proponer una solución entera muy próxima a la solución del problema relajado; el modelo de relajación más utilizado es: Métodos de Relajación Langraniana

Probabilístico: Una algoritmo probabilístico obtiene en promedio una buena solución al problema planteado, para una distribución de los datos de entrada dada.

Casos particulares: Cuando se reconocen casos particulares del problema para los cuales existen soluciones rápidas.

2.4. Problemas De Decisión

Un problema de decisión es aquel problema que se modela matemáticamente con variables binarias {0,1}; si o no, aceptado, no aceptado, etc.

Muchos de los problemas que estudiamos pueden ser descritos como un problema de decisión o como problemas de optimización; por ejemplo: ¿Cuál es la longitud del camino más corto de estas ciudades?, ¿Cuál es el tamaño del conjunto independiente más largo de vértices en G?

La teoría de NP-completitud es un intento, con un éxito parcial, para proveer que ciertos problemas son intrínsecamente difíciles de resolver. Por lo que se dará una breve, e informal introducción a esta área teórica tan importante de la ciencia de la computación.

Así, un problema de decisión es especificado por describir una instancia típica del problema, y haciendo la pregunta de ¿cuál es… ? y ser contestada con un "si" o "no".

Cualquier problema de optimización causa en forma natural una relación con un problema de decisión. Para un problema de minimización, se puede preguntar ¿es la solución óptima $\leq$ a un valor X especificado ?. El valor de X

debe ser especificado como parte de las instancias del problema de decisión. Para un problema de maximización, se debe reemplazar el "≤" por "≥".

2.5. Problemas De Optimización

En la resolución de problemas de optimización es frecuentemente importante encontrar máximos y mínimos globales. La programación lineal nos permite afrontar este tipo de problemas en el caso de funciones lineales. En general, la programación lineal permite encontrar máximos o mínimos globales para cualquier función lineal objetivo sujeta a un conjunto de restricciones definidas normalmente por un conjunto de desigualdades lineales.

Por ejemplo el Problema del agente Viajero (TSP) está entre los problemas denominados NP-Completos, esto es, los problemas que no se pueden resolver en tiempo polinomial en función del tamaño de la entrada (en este caso el número N de ciudades que el viajante debe recorrer). Sin embargo, algunos casos concretos del problema sí han sido resueltos hasta su optimización, lo que le convierte en un excelente banco de pruebas para algoritmos de optimización que pertenezcan a la misma familia; una formulación equivalente en términos de la teoría de grafos es la de encontrar en un grafo completamente conexo y con arcos ponderados el ciclo

hamiltoniano de menor coste. En esta formulación cada vértice del grafo representa una ciudad, cada arco representa una carretera y el peso asociado a cada arco representa la longitud de la carretera.

CAPÍTULO III

3. COMPLEJIDAD COMPUTACIONAL.

Prácticamente todas las áreas de las Ciencias Computacionales tratan, en mayor o menor grado con complejidad computacional. El tema es muy común entre expertos del área, sobre todo entre aquellos relacionados con NP-completo y entre quienes buscan algoritmos eficientes para diversos problemas de aplicación. La importancia de la complejidad computacional radica en que se ha convertido en una forma de clasificar buenos y malos algoritmos y de clasificar problemas computacionales como fáciles y difíciles. Aquí presentaremos una introducción a este tema.

El concepto intuitivo de algoritmo, lo tenemos prácticamente todos.
Un algoritmo es una serie finita de pasos para resolver un problema.

Hay que hacer énfasis en dos aspectos para que un algoritmo exista:

1) El número de pasos debe ser finito. De esta manera el algoritmo debe terminar en un tiempo finito con la solución del problema,

2) El algoritmo debe ser capaz de determinar la solución del problema.

Turing en 1936 mostró también que existen problemas matemáticos bien definidos para los cuales no hay un algoritmo. Hay muchos ejemplos de problemas para los cuales no existe un algoritmo.

Un problema de este tipo es el llamado problema de paro (halting problem): Dado un programa de computadora con sus entradas, ¿parará este alguna vez?, Turing probó que no hay un algoritmo que pueda resolver correctamente todas las instancias de este problema.

Alguien podría pensar en encontrar algunos métodos para detectar patrones que permitan examinar el programa para cualquier entrada. Sin embargo, siempre habrá sutilezas que escapen al análisis correspondiente.

Alguna persona más suspicaz, podría proponer simplemente correr el programa y reportar éxito si se alcanza una declaración de fin. Desgraciadamente, este esquema no garantiza por sí mismo un paro y en consecuencia el problema no puede ser resuelto con los pasos propuestos. Como consecuencia, de acuerdo con la definición anterior, ese último procedimiento no es un algoritmo, pues no se llega a una solución.

Muchas veces aunque exista un algoritmo que pueda solucionar correctamente cualquier instancia de un problema dado, no siempre dicho algoritmo es satisfactorio porque puede requerir de tiempos exageradamente excesivos para llegar a la solución.

Por ejemplo, considere de igual forma el problema del agente viajero.

3.1. Rendimiento de los Algoritmos.

Para propósitos de comparación del rendimiento de diferentes algoritmos, se requiere de una medida de rendimiento de algoritmos; la medida más ampliamente aceptada es el tiempo que el algoritmo en cuestión consume antes de producir la solución final. Esta cantidad de tiempo puede variar de una computadora a otra, debido a diferencias en su velocidad de procesamiento y en la forma como el programa es traducido en instrucciones de máquina. En el análisis de algoritmos, el tiempo siempre es expresado en términos del número de pasos elementales (operaciones aritméticas, comparaciones, lecturas, escrituras, etc.) requeridos para la ejecución del algoritmo sobre una computadora hipotética. Dichos pasos elementales tienen una duración unitaria de tiempo. De esta forma, el tiempo requerido para la ejecución de un algoritmo es establecido como el número de pasos elementales o instrucciones que efectúa antes de producir la solución final.

El número de pasos requeridos por un algoritmo no es siempre el mismo para todas las entradas y frecuentemente es difícil obtener una fórmula precisa que establezca la función de tiempo. Es por ello que se consideran tres medidas del tiempo de ejecución del algoritmo, las cuales dependen del tamaño de la entrada n:

a. El mínimo tiempo necesario(mejor caso),

b. El máximo tiempo necesario (peor caso) y,

c. El tiempo promedio necesario (caso promedio).

De esta forma la complejidad de un algoritmo es función del tamaño de la entrada (por ejemplo $10n^3$, 2^n, $n \log n$).

El interés principal en el comportamiento de un algoritmo es para entradas grandes, pues estas establecerán los límites de aplicabilidad de este. Por lo tanto, generalmente se tiene poco interés en el valor exacto del tiempo requerido para la ejecución de un algoritmo. Siendo el interés principal la razón de crecimiento del mejor caso,

Por ejemplo, supóngase que el tiempo del peor caso de un algoritmo es:
$t(n) = 60n^2 + 5n + 1$ para entradas de tamaño n.

Para n grandes, el término $60n^2$ es aproximadamente igual a t(n) (ver tabla 1).
En este sentido, t(n) crece como $60n^2$.

Tabla 3.1.
Razón de crecimiento del tiempo del peor caso de un algoritmo

n	t(n) = 60n2 + 5n + 1	60n2
10	6,051	6,000
100	600,501	600,000
1,000	60,005,001	60,000,000
10,000	6,000,050,001	6,000,000,000

Fuente: www.cs.umass.edu.com
Elaborado Por: Elías David Araujo Carrión

Si la medida del tiempo de la tabla 3,1, está en segundos, se podría dividir entre 60 para obtener una medida del tiempo del peor caso para entradas de tamaño n en minutos. Este cambio de unidades no afecta la forma como crece la función f(n) que representa la medida del tiempo del peor caso. De esta manera, se puede eliminar los coeficientes constantes. Con estas suposiciones t(n) crece como n^2 cuando se incrementa n.

En la actualidad hay un acuerdo general entre los investigadores de que un algoritmo es de utilidad práctica cuando su complejidad computacional crece polinomialmente con respecto del tamaño de la entrada, por ejemplo, algoritmos de complejidad O(n) o $O(n^3)$.

Naturalmente, algoritmos para los cuales la complejidad asintótica no es un polinomio en sí pero puede ser acotada por un polinomio también están incluidos ($n^{2.5}$, n log n)

Se le da el nombre de algoritmo exponencial a todo aquel que viola todo acotamiento polinomial para instancias suficientemente grandes. Se les da este nombre porque 2^n es una razón de crecimiento no polinomial. Otros ejemplos de razones exponenciales de crecimiento son k^n (para cualquier constante $k > 1$), n!, n^n y nlog n. Es obvio que, cuando el tamaño de la entrada crece, cualquier algoritmo polinomial eventualmente llegará a ser más eficiente que cualquier algoritmo exponencial.

Tabla 3.2.
Crecimiento de funciones polinomiales y exponenciales.

Función	**Valores Aproximados**		
n	10	100	1,000
n log n	33	664	9,966
n^3	1,000	1,000,000	109
106 n^8	1,014	1022	1030
2^n	1,024	1.27 x 1030	1.05 x 10301
n log n	2,099	1.93 x 1013	7.89 x 1029
n!	3,628,800	10158	4 x 102,567

Fuente: www.cs.umass.edu.com
Elaborado Por: Elías David Araujo Carrión

3.2. Programación heurística.

Muchos problemas de optimización no pueden ser abordados por métodos exactos, ya sea, por su alto grado combinatorio o por la dificultad de generar un modelo basado en programación matemática que represente exactamente una situación real. Para situaciones de ésta naturaleza se han venido generando desde la década de los sesenta, métodos conocidos como heurísticos, capaces de encontrar soluciones de buena calidad pero en muchos casos aproximada a la solución óptima.

En el primer tiempo se generaron métodos orientados específicamente a la resolución de cada problema, gran parte de estos métodos fueron generados inspirándose en la resolución de problemas de fácil representación pero de muy difícil solución como lo son: el Problema del Agente Viajero; el Problema de la Mochila; el Problema de los Conjuntos de Cobertura; etc.

Por la naturaleza diferente de estos problemas los métodos que se generaron eran útiles apenas para el problema en el cual habían sido inspirados, a partir de los años 80` se han generado una familia de métodos conocidos como meta-heurísticos que ahora tienen la capacidad de ser aplicables a problemas de diversa naturaleza. Es decir, una misma plantilla

algorítmica puede ser utilizada para resolver problemas que provienen de diversos sectores.

En el caso particular en que el número de variables que intervienen en el modelo sea muy elevado, el algoritmo simplex deja de ser eficiente, debido al tiempo necesario para encontrar la solución óptima del modelo. Estas ineficiencias dadas en los algoritmos para encontrar la solución óptima al problema, dio origen a los algoritmos que tratan de encontrar de modo eficiente una solución factible (que satisfaga las restricciones del problema) cercana al óptimo. Este tipo de algoritmos se denominan heurísticos (del griego heuriskein).

Estos pueden ser definidos como procedimientos simples, a menudo basados en el sentido común, que se supone ofrecerán una buena solución (aunque no necesariamente la óptima) a problemas difíciles, de un modo fácil y rápido (Zanakis y Evans, 1981).

Existen diferentes tipos de heurísticas, las cuales algunas se emplean conjuntamente, tales como los métodos constructivos, de descomposición, de reducción, de manipulación del modelo, de búsquedas por entornos, etc.. Uno de los mayores inconvenientes con los que se enfrentan estos métodos

es la existencia de óptimos locales que no sean absolutos, así como el conseguir hacerse independientes de la solución inicial de la que se parta.

CAPÍTULO IV

4. DISEÑO DEL ALGORITMO DE RECOCIDO SIMULADO.

4.1. Recocido Simulado.

El Recocido Simulado es el algoritmo que usaremos para resolver nuestro problema, se basa en los conceptos descritos originalmente por la mecánica estadística que describe el proceso físico sufrido por un sólido al ser sometido a un baño térmico (Kirkpatrick y Gelatt, 1983). Este puede ser simulado por el algoritmo de Metrópolis, basado en la técnica de Monte Carlo. Así pues, los estados del sistema se corresponden con las soluciones del problema, la energía de los estados con los criterios de evaluación de la calidad de la solución, el estado fundamental con la solución óptima del

problema, los estados metaestables serán los equivalentes a los óptimos locales, y la temperatura correspondería a una variable de control.

Con este artículo se ha querido presentar el concepto de la optimización, analizando las principales herramientas de las que se disponen para poder resolver los distintos tipos de problemas, en función del número de variables que lo componen y de su complejidad. Por último se presentan algunas las últimas técnicas más relevantes para resolver problemas de optimización, siendo éstas tema principal de trabajo en numerosas líneas de investigación.

4.2. Analogía del Recocido Simulado.

El método del recocido se utiliza en la industria para obtener materiales más resistentes, o más cristalinos, en general, para mejorar las cualidades de un material.

El proceso consiste en “derretir” el material (calentarlo a muy alta temperatura). En esa situación, los átomos adquieren una distribución “azarosa” dentro de la estructura del material y la energía del sistema es máxima. Luego se hace descender la temperatura muy lentamente por etapas, dejando que en cada una de esas etapas los átomos queden en equilibrio (es decir, que los átomos alcancen una configuración optima para

esa temperatura). Al final del proceso, los átomos forman una estructura cristalina altamente regular, el material alcanza así una máxima resistencia y la energía del sistema es mínima.

Experimentalmente se comprueba que si la temperatura se hace descender bruscamente o no se espera suficiente tiempo en cada etapa, al final la estructura del material no es la óptima.

La rama de la Física llamada Mecánica Estadística se encargó de desarrollar una serie de métodos para estudiar el comportamiento de grandes cantidades de átomos de un sistema. Debido a que en promedio, en un sistema hay 10^{23} átomos por cm^3, solamente puede estudiarse el comportamiento mas probable del sistema en equilibrio a una temperatura dada. La experimentación mostró que los átomos de un sistema en un proceso de recocido se comportan según el factor de probabilidad de Boltzman. En 1953 Metrópolis modeló el proceso de recocido: en cada paso del algoritmo se le da al átomo un desplazamiento azaroso y se mide el cambio de energía ΔE. Si $\Delta E \leq 0$ se acepta el desplazamiento. Si $\Delta E > 0$, se acepta el desplazamiento con probabilidad exp $(-\Delta E / T.K)$, donde T es la temperatura del sistema y K es la constante de Boltzman.

4.3. El método del Recocido Simulado.

Sea S el conjunto de soluciones posibles del sistema (a las que identificamos con los diferentes "estados del sistema") y tenemos una función costo $Z(t)$ sobre los elementos de S (a la que identificamos con la "energía del sistema"). Se quiere encontrar un elemento en S que minimice la función costo (análogamente, se trata de encontrar un estado en el cual la energía del sistema sea mínima). Asumimos que los estados del sistema tienen la función de distribución de probabilidad de Boltzman, i.e., la probabilidad de que el sistema se encuentre en el estado j es:

$$P(j) = \frac{1}{Z_T} e^{-\frac{c(j)}{t}}$$

$Z(t) = \sum e^{-\frac{c(i)}{t}}$ Donde (suma sobre todos los elementos i de S)

t es la temperatura del sistema y c(i) es el costo de la solución i.

Sea S* el subconjunto de S de las soluciones que minimizan c (globalmente, es decir soluciones óptimas del problema). Para t suficientemente chico:

de donde:

$$e^{\left|-\frac{c(j^*)}{t}\right|} >> e^{\left|-\frac{c(j)}{t}\right|}$$

$\forall$ **j! = j***

Entonces:

$$P(j) = \frac{e^{\left|-\frac{c(j)}{t}\right|}}{\left\{S^* e^{\left|-\frac{c(j^+)}{t}\right|}\right\}} = \begin{cases} 0 \text{ si } \mathbf{j! = j^*} \\ \frac{1}{S} \text{ si } \mathbf{j = j^*} \end{cases}$$

(Esto se obtiene de tomar t tendiendo a 0)

Por lo tanto: Σ P (j*) = 1 (suma sobre todos los j* en S*).

4.4. El Algoritmo de Recocido Simulado.

El algoritmo se divide en etapas. A cada etapa le corresponde una temperatura menor que la que tenía la etapa anterior (a esto hace referencia la monotonía: después de cada etapa la temperatura baja, se enfría el sistema). Por lo tanto hace falta un criterio de cambio de la temperatura ("cuanto tiempo" se espera en cada etapa para dar lugar a que el sistema alcance su "equilibrio térmico").

Datos iniciales y parámetro a ser definidos para poder inicializar el algoritmo:

Temperatura inicial (T_0)

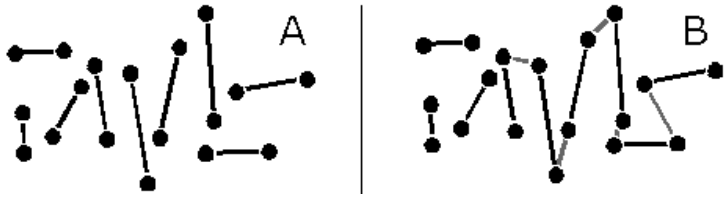
La temperatura inicial T_0 debe ser una temperatura que permita elegir el movimiento, es decir que la probabilidad de pasar del estado i al j (en N(i)) sea muy alta, sin importar la diferencia c(j) – c(i). Esto es que el sistema tenga un alto grado de libertad. En problemas como TSP, donde el input son los nodos de un grafo y las soluciones posibles son distintas formas de recorrer estos nodos, pude tomarse T_0 proporcional a la raíz cuadrada de la cantidad de nodos. En general se toma un valor T_0 que se cree suficientemente alto y se observa la primera etapa para verificar que el sistema tenga un grado de libertad y en función de esta observación se ajusta T_0.

Solución inicial (i_0)

En todas las versiones, el sistema debe ser "derretido" antes de implementar el algoritmo. Esto es que la solución factible inicial que llamamos i_0 debería ser una solución tomada al azar del conjunto de soluciones factibles. En algunos problemas esto puede hacerse utilizando pseudo-random numbers provistos por una máquina (ejemplo de esto es el bandwidth problem). Pero en muchos casos ya es problemático encontrar una solución, por lo que es imposible tomar una al azar. En estos casos se implementa un algoritmo "greedy" tipo local search para buscar una solución factible y se toma esta como i_0 (ejemplo de esto es el TSP).

Algoritmo de Greedy

Figura 4.1.
Vecino más cercano por Método de Greedy



Fuente: Castañeda Roldán, C. Y. 2000.
Elaborado por: Elías David Araujo Carrión

Este algoritmo busca la producción de un ciclo hamiltoniano uniendo una cantidad de bordes de grado dos(Ver Figura 9.A), luego anexa el borde más corto cada vez, siempre que no haya sido, previamente agregado (Ver Figura 9.B) y, cumpliendo con las condiciones de validez mencionadas en la Sección .

Factor de enfriamiento

$T_{next} = \alpha T$ (factor de enfriamiento geométrico, $\alpha < 1$, muy cercano a 1)

$T_{next} = 1 / (1 + \beta T)$ (donde β es un real positivo cercano a cero)

Criterio de cambio de la temperatura

Se usan dos parámetros: K = número de iteraciones que estamos dispuestos a hacer en cada etapa (equivalente a la cantidad de tiempo que vamos a esperar a que el sistema alcance su equilibrio térmico para una dada temperatura T; A = cantidad de aceptaciones que se permiten hacer en cada etapa.

A medida que T disminuye se supone que al sistema le resulta más difícil alcanzar un equilibrio porque es más dificultoso el movimiento, entonces hay que esperar más tiempo, esto se traduce en aumentar K.

Parámetro de aumento de K (ρ, se usan valores alrededor de 1,05)

Criterio de STOP

a) Lundy and Mees: si el algoritmo se detiene cuando $T < \varepsilon / [\ln (\#S - 1)/\theta]$

Donde #S es el cardinal del conjunto de soluciones (debe tenerse un método de estimar este valor).

Entonces, si i es la solución que da el algoritmo e i* en un optimo global,

$P(|c(i) - c(i^*)| < \varepsilon) = \theta$

b) En general se utiliza un parámetro de congelamiento (frozen: FRZN). Como a medida que disminuye la temperatura, aumenta el parámetro K y A permanece constante, la proporción A/K se hace pequeña. Asumimos que si A/K < FRZN el sistema esta congelado (la cantidad de aceptaciones respecto de la cantidad de iteraciones es muy chica, esto da la idea de que cambiar de configuración es muy difícil).

El Algoritmo:

1. $i = i_0$
2. $T = T_0$
3. $K = K_0$
4. while (condicion de STOP)
5. while (k < K && a < A)
6. generar j en N(i)
7. if (c(j) – c(i) < 0)

8. i = j

9. a = a + 1

10. else

11. generar un numero r al azar

12. if (r < exp [(c(i) – c(j))/T])

13. i = j

14. a = a + 1

15. k = k + 1

16. T = αT

17 K = ρK

18. k = 0

19. a = 0

20. mostrar i, c(i)

4.5. Problema Del Agente Viajero

Dado un grafo de N nodos, determinar la ruta más corta que parta de un nodo predeterminado Ni y visite todos los nodos solamente una vez y regrese al nodo de partida.

Este problema parece ser muy simple y puede ser que lo sea para ciertos tamaños de N.

Sin embargo existen instancias de este problema, con valores ni siquiera demasiado grandes de N, para los cuales encontrar la solución es un trabajo terrible. Tan solo basta pensar en una red de computadoras de unos 500 nodos con cierta topología para que sea muy difícil determinar la ruta más corta de un mensaje enviado desde un nodo i determinado y que debe visitar todos los nodos solamente una vez, trayendo de regreso el mensaje enviado y la traza de los nodos visitados.

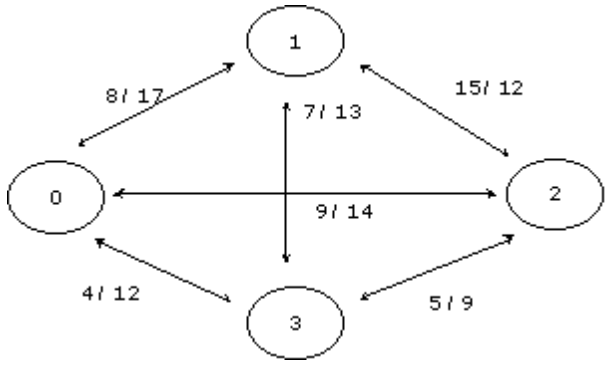
Para cualquier instancia del problema del agente viajero uno podría pensar en seleccionar todos y cada uno de los diferentes recorridos posibles y luego compararlos para encontrar el de menor longitud. Sin embargo, la ejecución de este algoritmo en una computadora requerirá de aproximadamente n! pasos, lo que para instancias relativamente pequeñas lo vuelve inútil, pero es posible realizarlo.

4.5.1. Modelamiento.

El problema del agente viajero se puede modelar fácilmente como un grafo completo dirigido, en donde los vértices del grafo son las ciudades y las aristas son los caminos, dichas aristas deben tener un peso, y este peso representa la distancia que hay entre dos vértices que están conectados por

medio de dicha arista, las aristas tendrán valores entre 1 y 100 Km. Una solución del problema del agente viajero, se puede representar como una secuencia de n+1 ciudades, en donde se comienza y se termina con la misma ciudad.

A continuación se presenta un problema ejemplo:



Como se podrá apreciar en el grafo anterior existen 4 ciudades, y los caminos entre cualquier par de ciudades son bidireccionales, sin embargo, los caminos no tiene la misma distancia en ambos sentidos, tomemos como ejemplo las ciudades 3 y 2, etiquetadas con un 5/9, el primer dígito corresponde a la distancia de ir de la ciudad 2 a la 3, es decir, 5 kilómetros; y el segundo dígito corresponde a la distancia de ir de la ciudad 3 a la 2, es decir, 9 kilómetros.

Dicho de otra manera, los pares de distancias que marcan las aristas en el grafo están asignados de la siguiente manera:

El primer número del par correspondiente a la distancia de ir de la ciudad menor a la ciudad mayor.

El segundo número corresponde a la distancia de ir de la ciudad mayor a la ciudad menor.

El problema de entrada estará dado en forma de una matriz de adyacencia, es decir:

Tabla 4.1
Matriz de Adyacencia del Ejemplo.

Ciudad	0	1	2	3
0	0	8	9	4
1	17	0	15	7
2	14	12	0	5
3	21	13	9	0

Fuente: www.ldc.usb.ve.com
Elaborado por: Elías David Araujo Carrión

No existe un camino entre una ciudad y ella misma, y se marca con un cero.

La columna 1 son las ciudades de origen.

La fila 1 son las ciudades de destino.

La intersección entre ciudad origen y destino es la distancia de ir de la ciudad origen a la ciudad destino.

La ciudad inicial-final siempre será la ciudad cero, en ella se iniciará y se terminará el recorrido total.

La solución al ejemplo anterior es:

Camino: 0 – 1 – 3 – 2 – 0

Recorrido: 38 Km

Los programas deberán resolver problemas de hasta 10 ciudades.

CAPÍTULO V

5. Implementación del Algoritmo de Recocido Simulado.

5.1. Levantamiento de información.

La recolección de datos partió con la selección de dos puntos estratégicos entre los cuales optimizaríamos las rutas válidas para llegar del punto 1 (P1) al punto 2 (P2).

Para esto se hizo necesario la obtención del plano georeferenciado de la Ciudad de Guayaquil, el mismo que fue facilitado por la empresa Busonil S.A. y que lo tenemos en la Figura 5.1.

Se ha seleccionado como P1 a la Piscina Olímpica Ubicada en las calles Hurtado y Avenida Del Ejército, la selección de este punto se realizó por ser un punto céntrico en cuyas rutas incluye calles como Clemente Ballén, Cristóbal Colon, Av. Machala, Av. Quito, Esmeraldas, Los Ríos, las más importantes calles del centro de la Ciudad de Guayaquil; como P2 se ha seleccionado a la Iglesia Catedral de la Ciudad de Guayaquil, por tratarse de un sitio bien visitado con dirección específica, vale recalcar esta información por cuanto en primera instancia se consideró al malecón 2000 como punto estratégico, pero varió esta consideración ya que al hablar de Malecón 2000 estamos hablando de varios puntos turísticos debido a su extensión, a la belleza y a la variedad del mismo que lo hacen visitado e interesante en cada punto geoespacial.

El siguiente paso fue determinar rutas válidas entre P1 y P2 las mismas que están detalladas en la Tabla 5.1.

Tabla 5.1
Rutas entre P1 y P2.

Ruta 1.- Hurtado y Av. Del Ejército por Av. Del Ejército sigo hasta Nueve de Octubre sigo hasta girar en Boyacá avanzo hasta Clemente Ballén llego a Chimborazo.

Ruta 2.- Hurtado y Av. Del Ejército por Av. Del Ejército sigo hasta Nueve de Octubre sigo hasta girar en Los Ríos avanzo hasta Clemente Ballén llego a Chimborazo.

Ruta 3.- Hurtado y Av. Del Ejército por Hurtado sigo hasta Los Ríos de allí hasta girar en Clemente Ballén hasta llegar a Chimborazo .

Ruta 4.- Hurtado y Av. Del Ejército por Av. Del Ejército sigo hasta Clemente Ballén llego a Chimborazo.

Ruta 5.- Hurtado y Av. Del Ejército por Hurtado sigo hasta Gabriel García Moreno giro en Clemente Ballén avanzo hasta Chimborazo.

Ruta 6.- Hurtado y Av. Del Ejército por Hurtado sigo hasta José Mascote giro en Clemente Ballén avanzo hasta Chimborazo.

Ruta 7.- Hurtado y Av. Del Ejército por Hurtado sigo hasta José Mascote sigo y giro en Cristóbal Colón avanzo hasta Boyacá giro en Clemente Ballén termino en Chimborazo.

Ruta 8.- Hurtado y Av. Del Ejército por Av. Del Ejército sigo hasta Nueve de Octubre sigo y giro en Machala avanzo hasta Clemente Ballén llego a Chimborazo.

Ruta 9 .-Hurtado y Av. Del Ejército por Hurtado sigo hasta José Mascote sigo y giro en Luque avanzo hasta Boyacá giro en Clemente Ballén termino en Chimborazo.

Ruta 10 Hurtado y Av. Del Ejército por Av. Del Ejército sigo hasta Nueve de Octubre sigo y giro en José de Antepara avanzo hasta Clemente Ballén llego a Chimborazo.

Ruta 11 Hurtado y Av. Del Ejército por Hurtado sigo hasta José Mascote luego giro en Luque avanzo hasta José de Antepara giro en Clemente Ballén y termino en Chimborazo.

Ruta 12 Hurtado y Av. Del Ejército por Hurtado sigo hasta José Mascote sigo y giro en Cristóbal Colón avanzo hasta Chimborazo llego a Clemente Ballén.

Ruta 13 Hurtado y Av. Del Ejército por Hurtado sigo hasta José Mascote sigo y giro en 10 de Agosto avanzo hasta Chimborazo llego a Clemente Ballén.

Ruta 14 Hurtado y Av. Del Ejército por Hurtado sigo hasta Los Ríos sigo y giro en 10 de Agosto avanzo hasta Chimborazo llego a Clemente Ballén

Fuente: Elías David Araujo Carrión
Elaborado por: Elías David Araujo Carrión

Una vez determinadas las rutas se ha procedido ha establecer los nodos generados de estas rutas, los que están detallados en la tabla 5.2.

Tabla 5.2.
Nodos de las rutas Preestablecidas.

NODOS	DIRECCION
1	PISCINA OLIMPICA
2	Av. Del Ejército y Nueve de Octubre
3	Hurtado y Los Ríos
4	Av. Del Ejército y Clemente Ballén
5	Hurtado y Gabriel García Moreno
6	Hurtado y José Mascote
7	Av. Del Ejército y Nueve de Octubre
8	Nueve de Octubre y Boyacá
9	Nueve de Octubre y Los Ríos
10	Los Ríos y Clemente Ballén
11	Clemente Ballén y Chimborazo
12	Gabriel García Moreno y Clemente Ballén
13	José Mascote y Clemente Ballén
14	José Mascote y Cristóbal Colon
15	Nueve de Octubre y Machala
16	José Mascote y Luque
17	Nueve de Octubre y José de Antepara
18	José Mascote y 10 de Agosto
19	Los Ríos y 10 de Agosto
20	Boyacá y Clemente Ballén
21	Cristóbal Colón y Boyacá
22	Machala y Clemente Ballén
23	Luque y Boyacá
24	José de Antepara y Clemente Ballén
25	Luque y José de Antepara
26	Cristóbal Colón y Chimborazo
27	10 de Agosto y Chimborazo
28	IGLESIA CATEDRAL

Fuente: Elías David Araujo Carrión
Elaborado por: Elías David Araujo Carrión

Tabla 5.3.
Georeferenciaciòn de los Nodos.

NODOS	DIRECCION	LATITUD	LONGITUD
1	PISCINA OLIMPICA	2.11.36	79.53.53
2	Hurtado y Nueve de Octubre	2.11.58	79.53.39
3	Hurtado y Los Ríos	2.11.25	79.53.66
4	Av. Del Ejército y Clemente Ballén	2.11.53	79.53.58
5	Hurtado y Gabriel García Moreno	2.11.37	79.53.51
6	Hurtado y José Mascote	2.11.35	79.53.57
7	Av. Del Ejército y Nueve de Octubre	2.11.27	79.53.50
8	Nueve de Octubre y Boyacá	2.11.46	79.53.05
9	Nueve de Octubre y Los Ríos	2.11.25	79.53.06
10	Los Ríos y Clemente Ballén	2.11.51	79.53.72
11	Clemente Ballén y Chimborazo	2.11.65	79.53.01
12	Gabriel García Moreno y Clemente Ballén	2.11.54	79.53.55
13	José Mascote y Clemente Ballén	2.11.53	79.53.61
14	José Mascote y Cristóbal Colon	2.11.66	79.53.65
15	Nueve de octubre y Machala	2.11.34	79.53.39
16	José Mascote y Luque	2.11.44	79.53.52
17	Nueve de octubre y José de Antepara	2.11.32	79.53.44
18	José Mascote y 10 de Agosto	2.11.59	79.53.56
19	Los Ríos y 10 de Agosto	2.11.56	79.53.73
20	Boyacá y Clemente Ballén	2.11.63	79.53.08
21	Cristóbal Colón y Boyacá	2.11.77	79.53.10
22	Machala y Clemente Ballén	2.11.56	79.53.45
23	Luque y Boyacá	2.11.49	79.53.05
24	José de Antepara y Clemente Ballén	2.11.55	79.53.50
25	Luque y José de Antepara	2.11.45	79.53.50
26	Cristóbal Colón y Chimborazo	2.11.79	79.53.04
27	10 de Agosto y Chimborazo	2.11.68	79.53.02
28	IGLESIA CATEDRAL	2.11.69	79.53.02

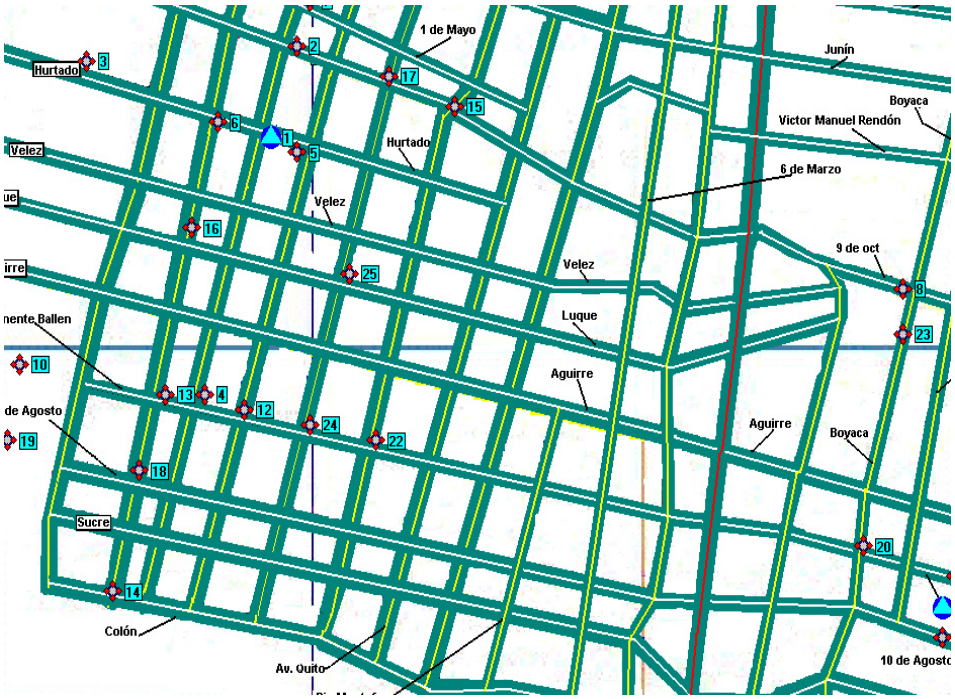
Fuente: Elías David Araujo Carrión

Elaborado por: Elías David Araujo Carrión

Con las dirección de los nodos, procedemos a levantar la información georeferenciada con la ayuda de un receptor GPS (Global Position System), esta recolección de datos se la realizo desde el 11 de Octubre del 2005 al 23 de Octubre del 2005, Esto se detalla en la Tabla 5.3.

Con las Georeferencias obtenidas desde el GPS, ubico cada nodo en el plano de Guayaquil, obteniendo la Figura 5.1.

Figura 5.1.
Ubicación Georeferenciada de los nodos en el Plano de Guayaquil.



Fuente: Busonil S.A.
Elaborado por: Ing. Eduardo Moyano

A partir del plano de la Figura 5.1 podemos obtener la Tabla 5.4 (Anexo 1.), que es la matriz de adyacencia entre nodos, en esta matriz las rutas entre nodo y nodo no probables ya sea por que no están conectadas o por ser una sola vía, están representados por valores de 1000 que es cuantificablemente alto e impide optimizar cualquier ruta garantizando que esa ruta no será tomada por el sistema. De igual forma la diagonal

Principal de la matriz de adyacencia esta representada por ceros, lo que significa que el costo de estar en el sobre su propio eje es 0. Las rutas validas y probables tienen la matriz de adyacencia asociado su propio costo.

5.2. Implementación Computacional

5.2.1. Análisis Paràmetrico.-

Solución Inicial (io).- Para nuestro problema la solución inicial esta dada por una combinación aleatoria de los nodos previamente establecidos, sin tener en consideración si esta eso no factible ya que los costos se encargaran de hacer automáticamente este filtro y según nuestro diseño la ruta no factible tendrá un costo superior a 1000 como mínimo, esto esta establecido como un proceso interno del software.

Temperatura Inicial (To).- Es bueno ser realista a la hora de determinar la temperatura inicial que para mi investigación es el costo de llegar dela piscina olímpica a la catedral sobre el cual va a pivotear el software.

Factor de Enfriamiento.- Como observamos en la línea 16 de nuestro algoritmo el software utiliza el factor de enfriamiento geométrico

Número de Iteraciones.- tiene que ser de acuerdo a los recursos que dispongamos, sabiendo que el software esta implementado en Visual

Basic 6.0 y es lo suficientemente bueno como para soportar un alto número de iteraciones.

Condición de Parada.- En cuento a la condición de parada la de Lundy y Mees, es la que usa el software que resuelve nuestro problema.

5.2.2. Actualización del software.

El sistema que decidí utilizar es el RECOCIDO SIMULADO, diseñado en Visual Basic 6.0 y tomado de la Tesis de Grado de la Ing. Daniela Yumbulema Rea, que resuelve satisfactoriamente nuestro problema para el cual se ha realizado el análisis Paràmetrico respectivo explicado en la sección 5.2.1.

La interfaz de este sistema fue modificada de tal manera que obtenemos un sistema personalizado “RECOCIDO SIMULADO V.2”

CAPITULO VI

6. ANÁLISIS DE RESULTADOS

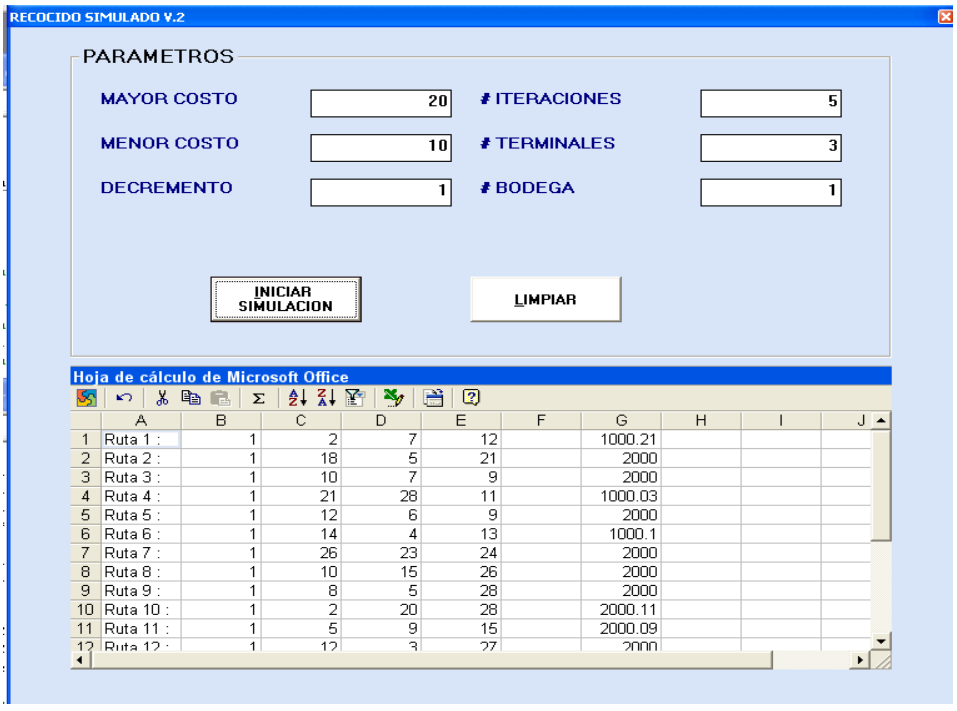
6.1. Interfaz de Recocido Simulado V2

En la Figura 6.1, podemos observar como se presentan las rutas generadas y en la parte inferior de la hoja de Excel se muestra la mejor ruta con su respectivo costo, de acuerdo a los parámetros ingresados que se encuentran visibles en la parte superior.

Mediante varias corridas se ha observado que el mejor tiempo obtenido al recorrer las 28 puntos esta alrededor de 1.81 Km. Manifiesto alrededor por que el valor varía a conforme disminuyo el decremento y aumento las iteraciones, Si como ejemplo tomamos de centro de distribución al nodo 1 y como terminales de distribución a los 27 nodos restantes, la solución que

obtendremos, asignando los siguientes valores a los parámetros: MAYOR COSTO de 20 Km. Y MENOR COSTO de 30 Km., con un decremento en 1 Km. y con 5 iteraciones. Esta es la forma en la que el recocido simulado muestras las varias opciones que puede tener el usuario solamente manipulando los parámetros, y al mismo tiempo proporciona una ruta lo suficientemente aceptable.

Figura 6.1.
Interfaz del Sistema RECOCIDO SIMULADO V.2



	A	B	C	D	E	F	G	H	I	J
1	Ruta 1 :	1	2	7	12		1000.21			
2	Ruta 2 :	1	18	5	21		2000			
3	Ruta 3 :	1	10	7	9		2000			
4	Ruta 4 :	1	21	28	11		1000.03			
5	Ruta 5 :	1	12	6	9		2000			
6	Ruta 6 :	1	14	4	13		1000.1			
7	Ruta 7 :	1	26	23	24		2000			
8	Ruta 8 :	1	10	15	26		2000			
9	Ruta 9 :	1	8	5	28		2000			
10	Ruta 10 :	1	2	20	28		2000.11			
11	Ruta 11 :	1	5	9	15		2000.09			
12	Ruta 12 :	1	12	3	27		2000			

Fuente: Elías David Araujo Carrión
Elaborado por: Elías David Araujo Carrión

Este sistema posee cuadros de texto donde podemos ingresar los parámetros elegidos y una hoja electrónica en la parte inferior donde se presentan los resultados.

Mediante varias corridas hemos calibrado el decremento (T_α) y obtenemos el siguiente resultado que se muestra en la tabla 5.5

Tabla 6.1.

Tabla de Resultados (calibrando Decremento T_α)

# Calibraciones	Mayor Costo	Menor Costo	Decremento	# Interacciones	# Terminales	# Bodegas	Resultado (Km)
1	20	10	0.5	5	3	1	1.81
2	20	10	1	5	3	1	1064
3	20	10	0.8	5	3	1	1001
4	20	10	0.7	5	3	1	1000.53
5	20	10	0.3	5	3	1	1000.21

Fuente: Elías David Araujo Carrión
Elaborado por: Elías David Araujo Carrrión

Podemos observar que con el parámetro de decremento 0.5 tenemos un valor más satisfactorio alrededor de decremento $T_{0.5}$.

RECOCIDO SIMULADO V.2 es de fácil manejo y se puede adaptar a cualquier situación lo único que se debe cambiar la matriz de datos generada en Microsoft Access que se encuentra en el directorio que nosotros hayamos guardado con el nombre de “matriz” este nombre es importante ejemplo: c:/RECOCIDOSIMULADO/matriz.mdb.

La conexión a la “matriz.mdb”, esta previamente establecida, en caso de producir algún problema se debe ir a panel de control, herramientas administrativas, orígenes de datos ODBC , en la ventana de administrador de origen datos ODBC elegir agregar, y realizar la respectiva conexión a Driver Do Microsoft Access, con doble clic vamos a otra ventana que nos permite ingresar el nombre del origen del la base de datos donde demos escribir matriz, luego en base de datos seleccionamos nuestra base y aceptar.

6.2. Resultados del Sistema.

Luego de varias corridas, y de verificar con los parámetros estimados podemos analizarlo siguiente:

- A medida de que se aumente el rango en el costo aumenta el número de iteraciones.
- A medida de que se disminuye el decremento aumenta el número de iteraciones
- Mientras mayor sea el número de iteraciones mayor será el tiempo requerido para que el sistema termine el proceso.
- En este caso se ha comprobado que el número de terminales juega un papel importante ya que existe manera de llegar del P1 al P2 con 2 y 3 terminales, haciendo que el mayor número de terminales provoca un mayor número de combinaciones disminuyendo la probabilidad de llegar al satisfactorio e incluyendo nodos intermedios que son innecesarios, por esta razón se recomienda que el número de terminales no exceda mas de 5.
- La bodega siempre debe ser 1, de acuerdo con la definición de la misma; se refiere a la cantidad de puntos de partida.
- Porque algoritmo usa números aleatorios cada corrida es diferente a las demás, en nuestro caso el resultado depende de los nodos y el costo; por ejemplo el sistema puede dar un resultado que no sea el

mejor, esto se produce de acuerdo a como está llenada la matriz. Los valores imposibles generan rutas con costos superiores a 1000.

6.3. Resultados Reales.

El sistema comprobado realmente funciona y es bueno a pesar de sobreestimar el costo satisfactorio en un valor difícil de determinar por ser muy variable, esto se ha determinado recorriendo cada ruta proporcionada por el sistema. Sin embargo, en pocos casos excede de 300 metros.

Mentalmente podemos establecer una ruta óptima debido a nuestra racionalidad y conocimiento de la ciudad, pero por tratarse de un sistema que no piensa y únicamente desarrolla instrucciones usando combinaciones aleatoriamente generadas, puede darse el caso que la ruta por nosotros considerada la mejor ni si quiera se haya generado, recordemos que el algoritmo nos muestra las mejores rutas dándonos la opción de escoger la que mas nos conviene de acuerdo a nuestro criterio o a condiciones externas.

CONCLUSIONES

1. La implementación del Recocido Simulado V.2 es una versión fácil de utilizar, sencilla en codificación, ocupa poco espacio en disco; haciéndolo un software manejable y adaptable a cualquier necesidad.

2. El algoritmo RS muestra ser un método eficaz para solucionar el problema de determinación óptima del usuario. El algoritmo mejora la calidad de la solución reportada por otros métodos y permite establecer un balance entre eficacia y eficiencia lo que lo hace una alternativa viable en la solución de problemas complejos.

3. El algoritmo funciona apropiadamente, dando como respuesta resultados válidos en cuanto a recursos computacionales, tiempo, etc.

4. De acuerdo a los resultados obtenidos se puede concluir que mediante el uso del algoritmo se logra obtener rutas válidas y no válidas, además de crear un criterio adecuado de una manara rápida y automática.

5. Se pudo probar que el algoritmo de Recocido Simulado representa una buena opción como técnica de optimización para la selección de la ruta más corta en el PAV. Además, fue disminuido el tiempo en el que se realizaba este proceso anteriormente, de semanas, a minutos en la actualidad.

6. El Recocido Simulado V.2 arroja en una hoja de Excel varias rutas, haciendo posible la elección de alguna de ellas de acuerdo a criterio del usuario, atendiendo a requerimiento externo que no estén considerado en el desarrollo del software, por ejemplo tiempo de congestionamiento en las rutas, etc.

7. En este caso hemos podido comprobar que al tratar con rutas de hasta 5 terminales el resultado no suele ser muy satisfactorio, sin embargo, lo resultados arrojan la ruta más corta entre los nodos extremos.

8. Usando el algoritmo de recocido simulado, hemos obtenido un plan de visitar sectores turísticos a través de rutas validas y con buenos resultados, comprobada su confiabilidad se puede asegurar que son satisfactorias.

9. Se ha establecido una correcta analogía paramétrica para resolver problemas de optimización combinatoria exclusivamente para caso del agente viajero considerando cada requisito que este debe cumplir, por ejemplo: no visitar dos veces el mismo nodo y partir de una bodega y regresar a la misma.

10. Se ha comprobado que al tratarse con 3 terminales, el número de combinaciones que debe generarse es C(28,5), haciendo poco probable que aleatoriamente se obtenga nuestra ruta preestablecidas.

11. El algoritmo en si, nos permite interactuar en cuanto a sus parámetros como por ejemplo: si el factor de decremento es exponencial o geométrico, estos sin dejar de ser heurística.

12. Se ha podido concluir que a clave para un buen algoritmo de recocido simulado está íntimamente ligada a la manera como se selecciona inicialmente la distancia y como ésta es reducida luego.

13. Para calibrar el decremento depende del rango (mayor costo – menor costo) si el rango es de orden de decenas se consideró un decremento (0,1) porque dará una cantidad de interacciones aceptables sin llevarnos a desperdiciar recurso, en caso de tratar en orden de centenas el decremento deberá considerarse mayor que 1

RECOMENDACIONES

1. Se recomienda usar como base el programa de Recocido Simulado V.2 para optimizar el Problema del agente Viajero, cuando este tenga que visitar más de 100 nodos, debido a que, la solución pude ser más satisfactoria.

2. Se puede hacer uso del Recosido Simulado V.2 para obtener beneficios que sirvan a los diferentes sectores industriales, y del transporte de nuestro país, de esta manera se contribuye al uso de la tecnología y las ganancias que esta nos ofrece.

3. El algoritmo se implementó en Visual Basic 6.0 en una computadora Pentium 4 de 200 Mhz. Cada prueba tomó cerca de 20 segundos en obtener a solución satisfactoria, por lo cual es recomendable implementar el algoritmo otros tipos de equipos y software más avanzados.

4. Además, se recomienda el uso del programa del RS para mejorar el problema de congestionamiento vehicular que se produce en ciertas horas en diversos sectores de la ciudad.

BIBLIOGRAFÍA

- **CASTAÑEDA ROLDAN CAROLINA YOLANDA. (1990) "**Teoría de Grafos"

- **COLECCIÓN L.N.S. (1992)**, "Metodología De La Investigación Científica", 3ra Edición.

- **COLECCIÓN NOVEDADES SOCIALES. (2003)**, "Sobre Tesis y Tesistas."

- **DÍAZ FERNÁNDEZ ADENSO, GONZÁLES JOSÉ LUÍS GONZÁLES, LAGUNA MANUEL LAGUNA, MOSCATO PABLO, T. TSENG FAN T. TSENG, FRED GLOVER; HASSAN M. (1996)**, "Optimización Heurísticas y Redes Neuronales ", 1era Edición, España

- **EDUARDO MORALES MANZANARES, "Algoritmo de Metrópolis",** http://dns1.mor.itesm.mx/~emorales/Cursos/Busqueda04/node78.html#tmetropolis.

- **GONZALES JORGE LUIS GONZALES, Z. RIOS ROGER. INVESTIGACION D EOPERACIONES EN ACCION. “** Aplicación del TSP en problemas de Manufactura y Logística”

- **MELIÁN BELÉN, MORENO PÉREZ JOSÉ A, MORENO VEGA J. MARCOS. “**Metaheurísticas: una visión Global”

G

Georeferencia.- Valores de Latitud, Longitud y Altitud que permiten localizar geográficamente un punto

Georeferenciaciòn.- Ubicar Geográficamente un Plano.

H

Heurísticas.- Algoritmos mediante los cuales se pueden resolver problemas no polinomiales.

O

Output.- La información de salida de un programa sirve como información de entrada del siguiente.

P

Problema del Agente Viajero.- Determina, dado un mapa de carreteras en que orden debe visitarse n ciudades de forma que se obtenga menor costo tras visitar una sola vez cada una de ellas.

R

Recocido Simulado.- Algoritmo usado en termodinámica para obtener estados de baja energía.

T

Termodinámica.- Parte de la mecánica que estudia las transformaciones e intercambio de energía.

ANEXOS

ANEXOS.

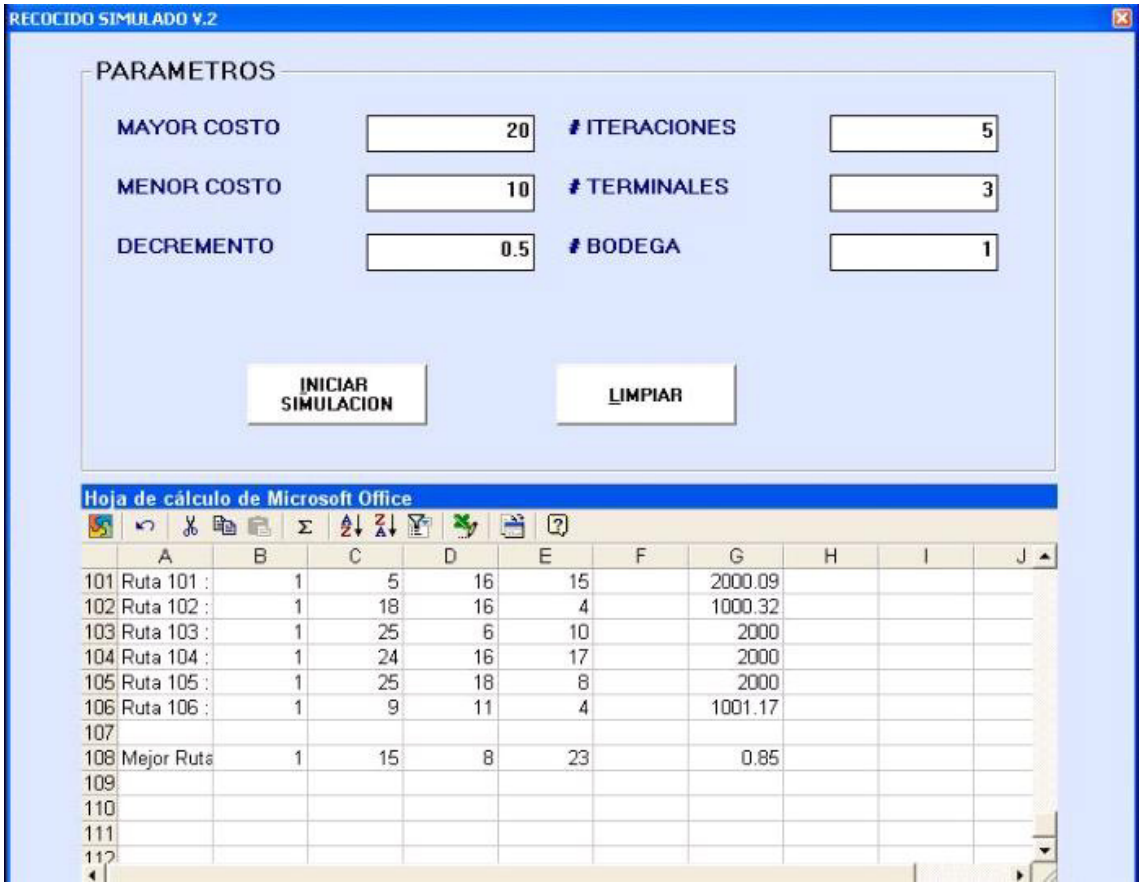
ANEXO I. Matriz de Adyacencia

Tabla 5.4.
Matriz de Adyacencia.

NODOS	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28
1	0	0.11	0.32	0.32	1000	0.1	1000	1000	1000	1000	1000	1000	1000	1000	1000	1000	1000	1000	1000	1000	1000	1000	1000	1000	1000	1000	1000
2	0.11	0	1000	0.53	1000	1000	0.1	0.96	0.32	1000	1000	1000	1000	1000	0.22	1000	1000	1000	1000	1000	1000	1000	1000	1000	1000	1000	1000	1000
3	1000	1000	0	1000	1000	0.21	1000	1000	1000	1000	1000	1000	1000	1000	1000	1000	1000	1000	1000	1000	1000	1000	1000	1000	1000	1000	1000	1000
4	1000	1000	1000	0	1000	1000	1000	1000	1000	1000	1.17	1	1000	1000	1000	1000	1000	1000	1000	1000	1000	1000	1000	1000	1000	1000	1000	1000
5	0.09	0.32	0.32	1000	0	0.11	1000	1000	1000	1000	1000	0.32	1000	1000	1000	1000	1000	1000	1000	1000	1000	1000	1000	1000	1000	1000	1000	1000
6	1000	1000	0.21	1000	1000	0	1000	1000	1000	1000	1000	1000	0.32	0.64	1000	0.21	1000	0.32	1000	1000	1000	1000	1000	1000	1000	1000	1000	1000
7	0.21	0.1	1000	0.53	1000	1000	0	1000	1000	1000	1000	1000	1000	1000	1000	1000	1000	1000	1000	1000	1000	1000	1000	1000	1000	1000	1000	1000
8	1000	0.96	1000	1000	1000	1000	1000	1000	1.27	1000	1000	1000	1000	1000	0.75	1000	0.85	1000	1000	0.32	1000	1000	1	1000	1000	1000	1000	1000
9	1000	0.32	1000	1000	1000	1000	1000	1.27	0	1000	1000	1000	1000	1000	0.53	1000	0.43	1000	1000	1000	1000	1000	1000	1000	1000	1000	1000	1000
10	1000	1000	1000	0.32	1000	1000	1000	1000	1000	0	1.38	0.32	0.21	1000	1000	1000	1000	1000	1000	1.27	1000	0.53	1000	0.43	1000	1000	1000	1000
11	1000	1000	1000	1000	1000	1000	1000	1000	1000	1000	0	1000	1000	1000	1000	1000	1000	1000	1000	1000	1000	1000	1000	1000	1000	1000	1000	0.03
12	1000	1000	1000	1000	1000	1000	1000	1000	1000	1000	1.06	0	1000	1000	1000	1000	1000	1000	1000	0.96	1000	2.1	1000	1.1	1000	1000	1000	1000
13	1000	1000	1000	0.1	1000	1000	1000	1000	1000	1000	1.17	0.11	0	0.32	1000	1000	1000	0.11	1000	1.06	0.21	0.32	1000	1000	1000	1000	1000	1000
14	1000	1000	1000	1000	1000	1000	1000	1000	1000	1000	1000	1000	1000	0	1000	1000	1000	1000	1000	1000	1.06	1000	1000	1000	1000	1.27	1000	1000
15	1000	1000	1000	1000	1000	1000	1000	0.74	1000	1000	1000	1000	1000	1000	0	1000	1000	1000	1000	1000	1000	0.43	1000	1000	1000	1000	1000	1000
16	1000	1000	1000	1000	1000	1000	1000	1000	1000	1000	1000	1000	0.21	0.53	1000	0	1000	0.32	1000	1000	1000	1000	1000	1000	0.21	1000	1000	1000
17	1000	0.21	1000	1000	1000	1000	1000	0.85	0.43	1000	1000	1000	1000	1000	0.11	1000	0	1000	1000	1000	1000	1000	1000	0.43	0.32	1000	1000	1000
18	1000	1000	1000	1000	1000	1000	1000	1000	1000	1000	1000	1000	1000	0.21	1000	1000	1000	0	1000	1000	1000	1000	1000	1000	1000	1000	1.27	1000
19	1000	1000	1000	1000	1000	1000	1000	1000	1000	1000	1000	1000	1000	1000	1000	1000	1000	0.21	0	1000	1000	1000	1000	1000	1000	1000	1.38	1000
20	1000	1000	1000	1000	1000	1000	1000	1000	1000	1000	0.21	1000	1000	1000	1000	1000	1000	1000	1000	0	21	1000	1000	1000	1000	1000	1000	1000
21	1000	1000	1000	1000	1000	1000	1000	0.64	1000	1000	1000	1000	1000	1000	1000	1000	1000	1000	1000	0.32	0	1000	0.53	1000	1000	0.11	1000	1000
22	1000	1000	1000	1000	1000	1000	1000	1000	1000	1000	0.85	1000	1000	1000	1000	1000	1000	1000	1000	0.74	1000	0	1000	1000	1000	1000	1000	1000
23	1000	1000	1000	1000	1000	1000	1000	0.1	1000	1000	1000	1000	1000	1000	1000	1000	1000	1000	1000	0.32		1000	0	1000	1000	1000	1000	1000
24	1000	1000	1000	1000	1000	1000	1000	1000	1000	1000	0.96	1000	1000	1000	1000	1000	1000	1000	1000	0.85	1000	0.11	1000	0	1000	1000	1000	1000
25	1000	1000	1000	1000	1000	1000	1000	1000	1000	1000	1000	1000	1000	1000	1000	1000	1000	1000	1000	1000	1000	1000	1000	0.21	0	1000	1000	1000
26	1000	1000	1000	1000	1000	1000	1000	1000	1000	1000	1000	1000	1000	1000	1000	1000	1000	1000	1000	1000	1000	1000	1000	1000	1000	0	0.21	1000
27	1000	1000	1000	1000	1000	1000	1000	1000	1000	1000	1000	1000	1000	1000	1000	1000	1000	1000	1000	1000	1000	1000	1000	1000	1000	1000	0	0.02
28	1000	1000	1000	1000	1000	1000	1000	1000	1000	1000	1000	1000	1000	1000	1000	1000	1000	1000	1000	1000	1000	1000	1000	1000	1000	1000	1000	0

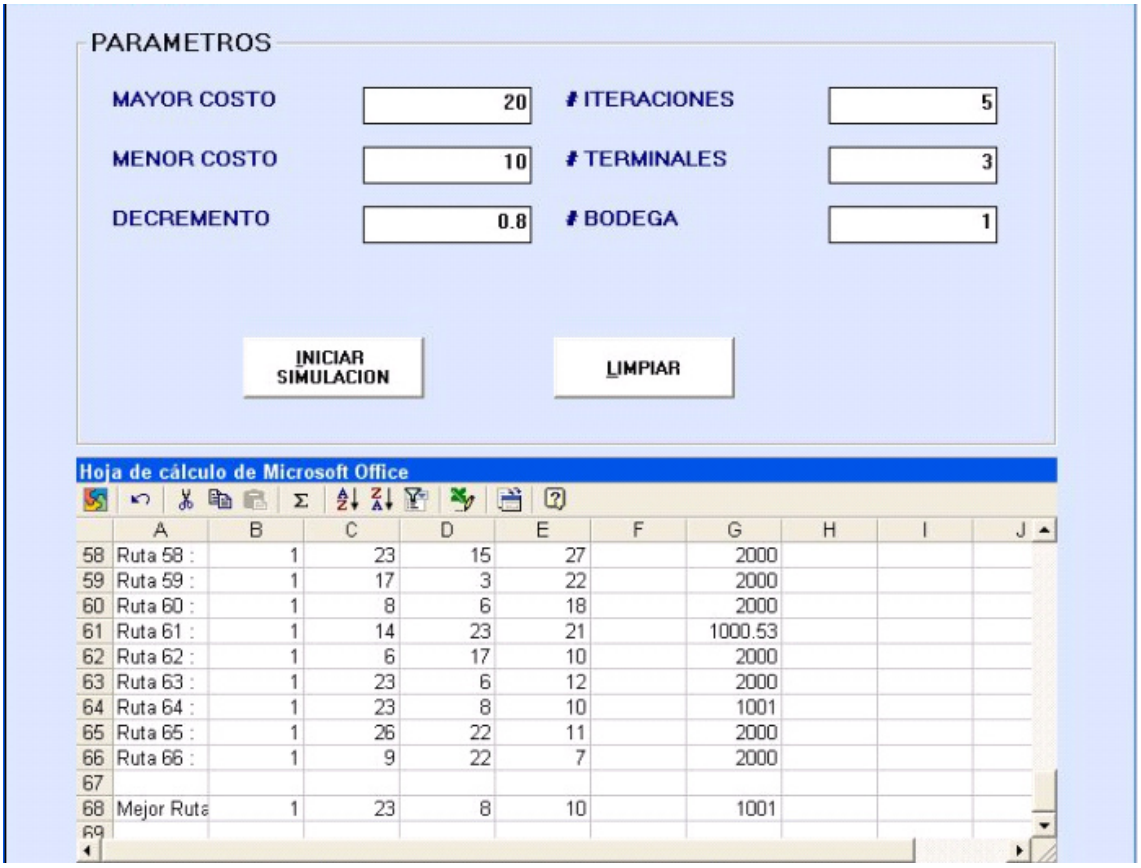
ANEXO II. Pantallas de resultados calibrando el decremento (T_α)

$\alpha = 0.5$



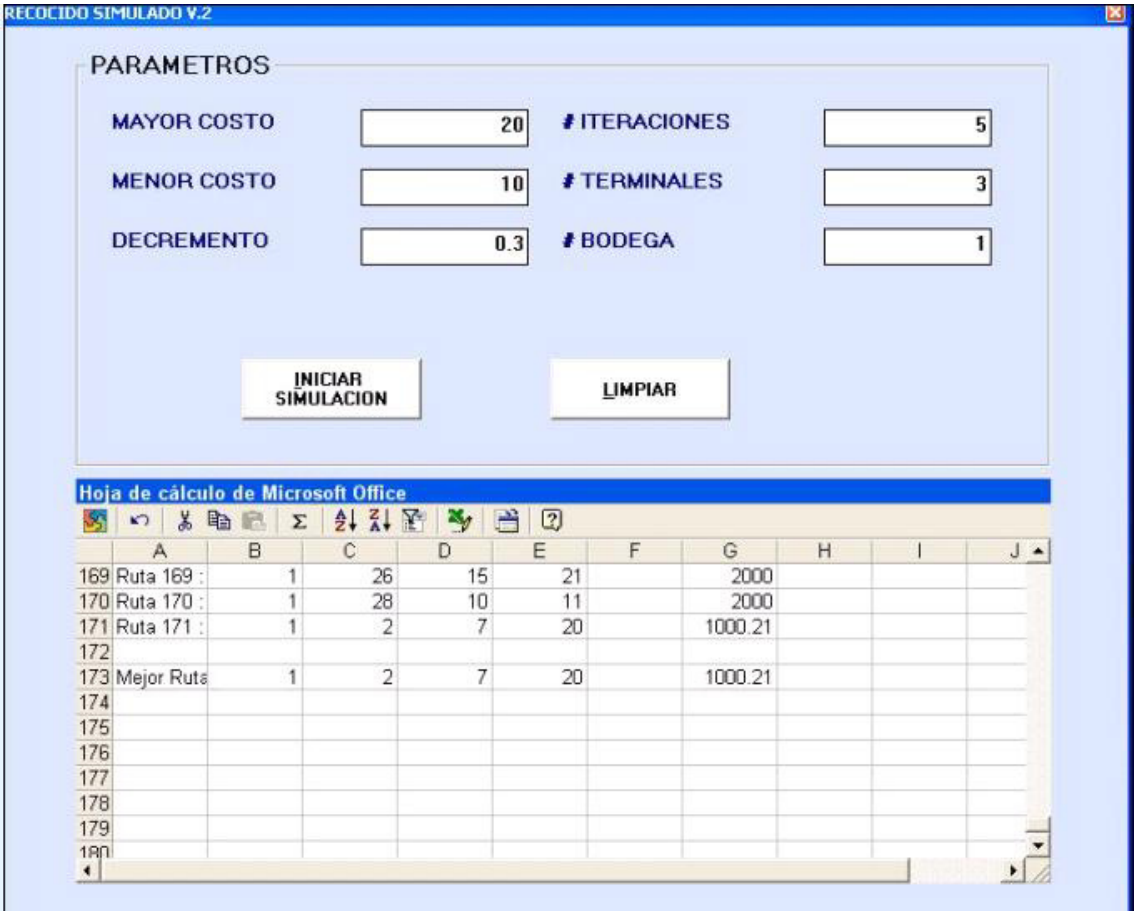
	A	B	C	D	E	F	G	H	I	J
101	Ruta 101 :	1	5	16	15		2000.09			
102	Ruta 102 :	1	18	16	4		1000.32			
103	Ruta 103 :	1	25	6	10		2000			
104	Ruta 104 :	1	24	16	17		2000			
105	Ruta 105 :	1	25	18	8		2000			
106	Ruta 106 :	1	9	11	4		1001.17			
107										
108	Mejor Ruta	1	15	8	23		0.85			
109										
110										
111										
112										

$\alpha = 0.8$



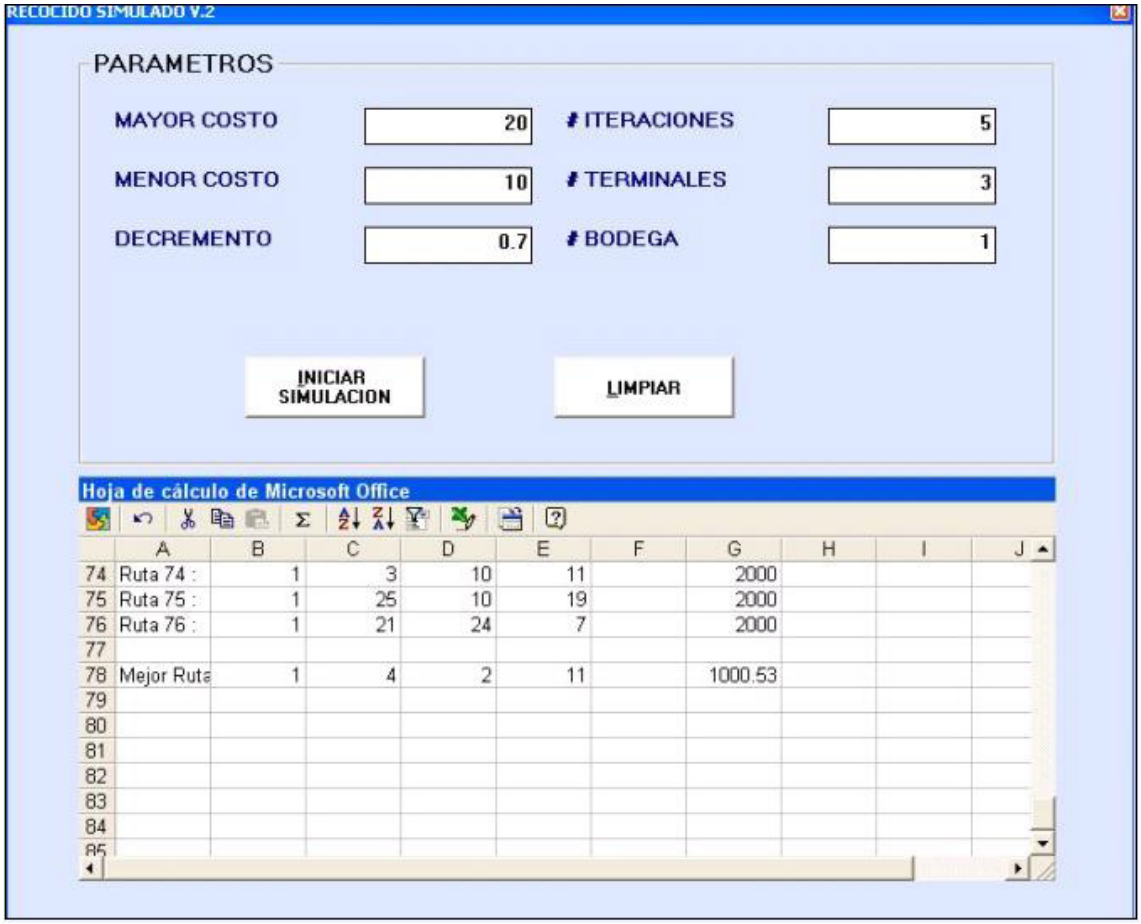
	A	B	C	D	E	F	G	H	I	J
58	Ruta 58 :	1	23	15	27		2000			
59	Ruta 59 :	1	17	3	22		2000			
60	Ruta 60 :	1	8	6	18		2000			
61	Ruta 61 :	1	14	23	21		1000.53			
62	Ruta 62 :	1	6	17	10		2000			
63	Ruta 63 :	1	23	6	12		2000			
64	Ruta 64 :	1	23	8	10		1001			
65	Ruta 65 :	1	26	22	11		2000			
66	Ruta 66 :	1	9	22	7		2000			
67										
68	Mejor Ruta	1	23	8	10		1001			
69										

$\alpha = 0.3$



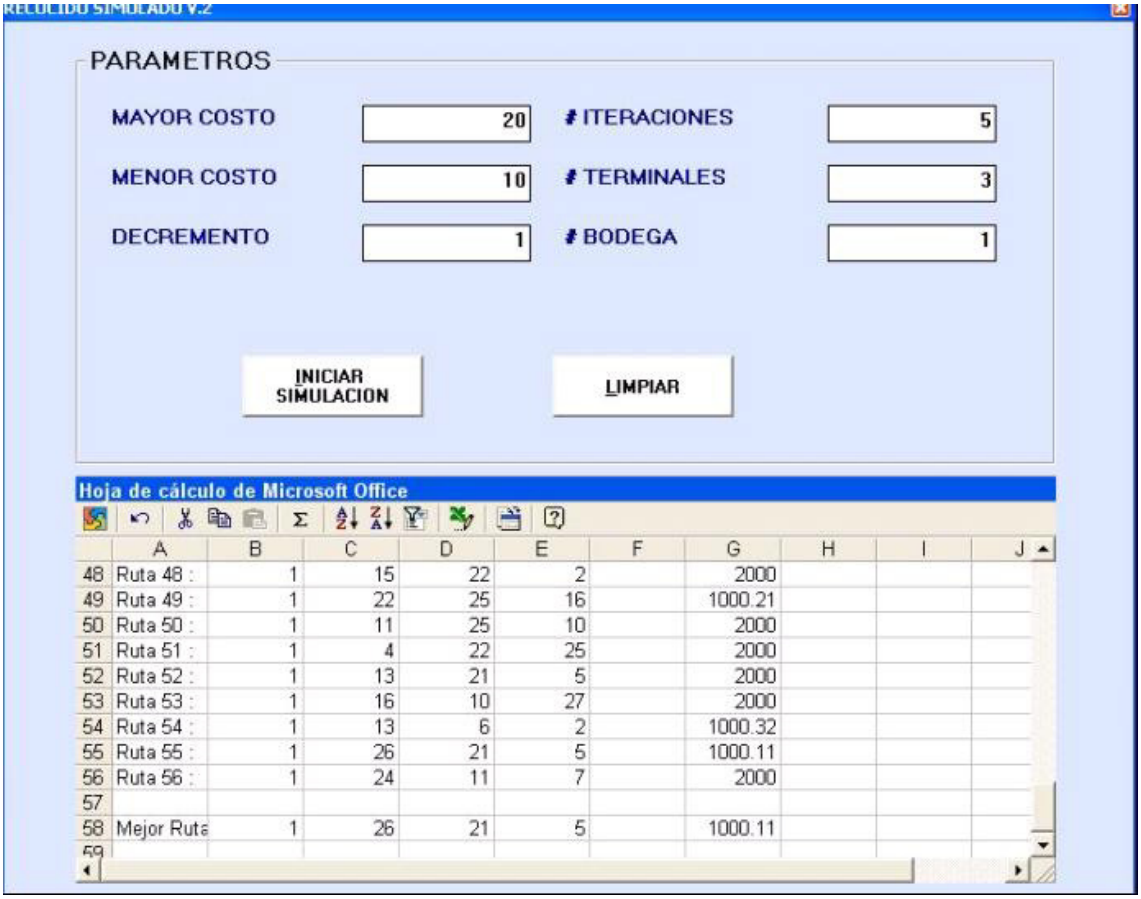
	A	B	C	D	E	F	G	H	I	J
169	Ruta 169 :	1	26	15	21		2000			
170	Ruta 170 :	1	28	10	11		2000			
171	Ruta 171 :	1	2	7	20		1000.21			
172										
173	Mejor Ruta	1	2	7	20		1000.21			
174										
175										
176										
177										
178										
179										
180										

$\alpha = 0.7$



	A	B	C	D	E	F	G	H	I	J
74	Ruta 74 :	1	3	10	11		2000			
75	Ruta 75 :	1	25	10	19		2000			
76	Ruta 76 :	1	21	24	7		2000			
77										
78	Mejor Ruta	1	4	2	11		1000.53			
79										
80										
81										
82										
83										
84										
85										

$\alpha = 1.0$



	A	B	C	D	E	F	G	H	I	J
48	Ruta 48 :	1	15	22	2		2000			
49	Ruta 49 :	1	22	25	16		1000.21			
50	Ruta 50 :	1	11	25	10		2000			
51	Ruta 51 :	1	4	22	25		2000			
52	Ruta 52 :	1	13	21	5		2000			
53	Ruta 53 :	1	16	10	27		2000			
54	Ruta 54 :	1	13	6	2		1000.32			
55	Ruta 55 :	1	26	21	5		1000.11			
56	Ruta 56 :	1	24	11	7		2000			
57										
58	Mejor Ruta	1	26	21	5		1000.11			
59										

ANEXO III. Codificación del algoritmo de Recocido Simulado en Visual Basic 6.0 usadas para la resolución del Agente Viajero.

Codificación del fmrecocido

Private Sub cmdcomenzar_Click()

```
If Len(txtmayor.Text) = 0 Or Len(txtmenor.Text) = 0 Or Len(txtalfa.Text) = 0
Or Len(txtiteraciones.Text) = 0 Or Len(txtterminales.Text) = 0 Or
Len(txtbodega.Text) = 0 Then
  MsgBox "Faltan datos para iniciar simulación?", vbCritical, "Mensaje del
Sistema"
  txtmayor.SetFocus
  Exit Sub
 Else
  If Val(txtterminales.Text) > 27 Then 'aqui ya no iria 79 sino el numero de
terminales posibles de acuerdo a la matriz c
   MsgBox "No puede haber más de 27 terminales", vbCritical, "Mensaje del
Sistema" 'aqui tambien cambie el mensaje en vez de 79 poner el tope de
terminales que acepta
   txtterminales.SetFocus
   Exit Sub
  End If
```

```
'Inicio de Simulación
sps.Cells.Clear
sps.Cells.Borders.Color = RGB(80, 150, 150)
ReDim Sact(1 To (Val(txtterminales.Text) + 1))
For i = 1 To (Val(txtterminales.Text) + 1)
 Sact(i) = 0
Next i
costeact = 0
t = Val(txtmayor.Text)
sps.Cells(1, 1) = "Ruta 1 :"
Sact(1) = Val(txtbodega.Text)
sps.Cells(1, 2) = Sact(1)
sps.Cells(1, Val(txtterminales.Text) + 4) = 0
Randomize Timer
For j = 1 To Val(txtterminales.Text)
 aleat = CInt(28 * Rnd) + 1
 Do While (verificar(aleat, j, 1) Or aleat = 59)
  aleat = CInt(28 * Rnd) + 1
 Loop
  Sact(j + 1) = aleat
  sps.Cells(1, j + 2) = Sact(j + 1)
  sps.Cells(1, Val(txtterminales.Text) + 4) = Val(sps.Cells(1,
Val(txtterminales.Text) + 4)) + C(Sact(j), Sact(j + 1))
```

```
  costeact = sps.Cells(1, Val(txtterminales.Text) + 4)
Next j
'guardo en un archivo rutas.txt metodo output para que en cada
'simulacion borre la informacion anterior y comience de nuevo
   Dim nFic%
   'Abrimos el fichero
   nFic = FreeFile
   Open "C:\ronnie\RecocidoSimulado\Rutas.txt" For Output As nFic
     For j = 2 To (Val(txtterminales.Text) + 4)
       Print #nFic, sps.Cells(1, j)
     Next j
   Close nFic
 'fin de proceso de guardar en un archivo
 m = 1
'Comienzo el algoritmo
For w = Val(txtmayor.Text) To Val(txtmenor.Text) Step -Val(txtalfa.Text)
 For i = 1 To Val(txtiteraciones.Text)
   m = m + 1
   costecand = 0
   ReDim Scand(1 To (Val(txtterminales.Text) + 1))
   Scand(1) = Val(txtbodega.Text)
   sps.Cells(m, 2) = Scand(1)
   sps.Cells(m, Val(txtterminales.Text) + 4) = 0
```

```
Randomize Timer
For k = 1 To Val(txtterminales.Text)
 sps.Cells(m, 1) = "Ruta " & m & " :"
 aleat = CInt(28 * Rnd) + 1
 Do While (verificar(aleat, k, 2) Or aleat = 29)
  aleat = CInt(28 * Rnd) + 1
 Loop
 Scand(k + 1) = aleat
 sps.Cells(m, k + 2) = Scand(k + 1)
 sps.Cells(m, Val(txtterminales.Text) + 4) = Val(sps.Cells(m,
Val(txtterminales.Text) + 4)) + C(Scand(k), Scand(k + 1))
 costecand = sps.Cells(m, Val(txtterminales.Text) + 4)
Next k
delta = costecand - costeact
If (Rnd < Exp(-(delta / t))) Or (delta < 0) Then
 For j = 1 To Val(txtterminales.Text) + 1
  Sact(j) = Scand(j)
 Next j
 costeact = costecand
End If
'guardo en un archivo rutas.txt y uso el metodo append para añadir
'informacion al fichero existente y no borre la informacion anterior
 'Abrimos el fichero
```

```
  nFic = FreeFile
  Open "C:\ronnie\RecocidoSimulado\Rutas.txt" For Append As nFic
    For j = 2 To (Val(txtterminales.Text) + 4)
      Print #nFic, sps.Cells(m, j)
    Next j
  Close nFic
'fin de proceso de guardar en un archivo
 Next i
Next w
For i = 1 To Val(txtterminales.Text) + 1
 sps.Cells(m + 2, i + 1) = Sact(i)
Next i
sps.Cells(m + 2, 1) = "Mejor Ruta Visitada:"
sps.Cells(m + 2, i + 2) = costeact
'guardo en un archivo rutas.txt y uso el metodo append para añadir
 'informacion de la mejor ruta visitada
  'Abrimos el fichero
  nFic = FreeFile
  Open "C:\ronnie\RecocidoSimulado\Rutas.txt" For Append As nFic
    For j = 2 To (Val(txtterminales.Text) + 4)
      Print #nFic, sps.Cells(m + 2, j)
    Next j
  Close nFic
```

```
  'fin de proceso de guardar en un archivo


 End If
End Sub

Private Sub cmdlimpiar_Click()
 txtmayor.Text = Empty
 txtmenor.Text = Empty
 txtalfa.Text = Empty
 txtiteraciones.Text = Empty
 txtterminales.Text = Empty
 txtbodega.Text = Empty
 sps.Cells.Clear
 txtmayor.SetFocus
End Sub

Private Sub Form_Load()
 'Me.Top = 0
 'Me.Left = 0
 txtmayor.Text = Empty
 txtmenor.Text = Empty
 txtalfa.Text = Empty
 txtiteraciones.Text = Empty
```

```
 txtbodega.Text = Empty
 txtterminales.Text = Empty
End Sub

Private Sub txtalfa_KeyPress(k As Integer)
If (k < 48 Or k > 57) And k <> 8 And k <> 13 And k <> 46 Then
 k = 0
Else
  If k = 13 Then
   txtiteraciones.SetFocus
  End If

End If
End Sub

Private Sub txtbodega_KeyPress(k As Integer)
If (k < 48 Or k > 57) And k <> 8 And k <> 13 Then
 k = 0
Else
  If k = 13 Then
   cmdcomenzar.SetFocus
  End If
End If
```

```
End Sub

Private Sub txtiteraciones_KeyPress(k As Integer)
If (k < 48 Or k > 57) And k <> 8 And k <> 13 Then
 k = 0
Else
  If k = 13 Then
   txtterminales.SetFocus
  End If
End If
End Sub

Private Sub txtmayor_KeyPress(k As Integer)
If (k < 48 Or k > 57) And k <> 8 And k <> 13 Then
 k = 0
Else
  If k = 13 Then
   txtmenor.SetFocus
  End If
End If
End Sub

Private Sub txtmenor_KeyPress(k As Integer)
```

```
If (k < 48 Or k > 57) And k <> 8 And k <> 13 Then
 k = 0
Else
  If k = 13 Then
   txtalfa.SetFocus
  End If
End If
End Sub

Private Sub txtterminales_KeyPress(k As Integer)
If (k < 48 Or k > 57) And k <> 8 And k <> 13 Then
 k = 0
Else
  If k = 13 Then
   txtbodega.SetFocus
  End If
End If
End Sub
```

Codificación del Module 1.

```
Public C(1 To 28, 1 To 28) As Double 'declaro la dimension de la matriz
asumo que es de 80*80 caso contrario poner la dimension real ahi usted
tiene que cambiar
Public i, j, k, m, aleat, h As Long
Public t, tf, costecand, costeact, delta, w As Double
Public Sact(), Scand() As Integer

Sub main()
'Lleno la matriz de costos asociados a cada ruta

'aqui en vez de esto amiga tiene que definir la matriz de cuanto * cuanto y
llenarla aqui le pongo un ejemplo
'si la matriz de costos es de 5*5 entonces tenemos
'1) cambiar la dimension de la matriz c eso esta arriba en la primera linea
'2) llenar la matriz así c(1,1)=20
'c(1,2)=25,.....,c(1,5)=30, y asi sucesivamente
'esto de los for de i y j ya no iria asi que hay que borrarlos
'For i = 1 To 80
 'For j = 1 To 80
```

```
  'If i = j Then
    'C(i, j) = 0
  'Else
    'C(i, j) = i + j
  'End If
' Next j
'Next i
Dim conec As New ADODB.Connection
Dim rs As New ADODB.Recordset

conec.Open "DSN=matriz"

For i = 1 To 28
j = 0
    Set rs = conec.Execute("select nodo" + "" & i & "  from matriz")

    While Not rs.EOF

     j = j + 1

        C(i, j) = rs.Fields("nodo" + "" & i & "")

        rs.MoveNext
```

```
    Wend
Next i

frmrecocido.Show
End Sub

Public Function verificar(ByVal v As Integer, ByVal tope As Integer, ByVal
vector As Integer) As Integer
verificar = 0
If vector = 1 Then
 For h = 1 To tope
  If v = Sact(h) Then
   verificar = 1
   h = tope + 1
  End If
 Next h
Else
 For h = 1 To tope
  If v = Scand(h) Then
   verificar = 1
   h = tope + 1
```

```
    End If
  Next h
End If
  End Function
```