Universidad de Valladolid

ESCUELA DE INGENIERÍAS INDUSTRIALES

UNIVERSIDAD DE VALLADOLID

ESCUELA DE INGENIERIAS INDUSTRIALES

Grado en Ingeniería de Organización Industrial

Resolución del Problema del Viajante de Comercio (TSP) y su variante con Ventanas de Tiempo (TSPTW) usando métodos heurísticos de búsqueda local

Autor:

Pérez de Vargas Moreno, Beatriz

Tutor:

Sáez Aguado, Jesús
Departamento

Valladolid, junio de 2015.

Agradecimientos

A Jesús Sáez por su interés y ayuda durante todo el proyecto para que esto saliera adelante.
A mi familia y a Carlos, por su apoyo incondicional.

Resumen:

En el presente proyecto se llevará a cabo el estudio del Problema del Viajante de Comercio (TSP) y su variante con Ventanas de Tiempo (TSPTW). Para cada uno de estos problemas se presentará una explicación del mismo, aplicaciones, formulación matemática y resolución. Dicha resolución se llevará a cabo mediante métodos exactos y heurísticas de búsqueda local. Entre las heurísticas se incluyen dos novedosas propuestas: *Threshold Acceptance* y *Late Acceptance Hill-Climbing*, y dos ampliamente conocidas y estudiadas: *Simulated Annealing* y *Hill-Climbing*. Se pretende realizar una comparativa entre ellas, tomando como base los resultados obtenidos por el método exacto y los mejores resultados conocidos, de cara a conocer el algoritmo que mejor rendimiento presenta para estos problemas. Para el caso del TSPTW veremos una heurística más, variante de las anteriores, llamada *Compressed-Annealing Simulated*.

Palabras clave: "Problema del Viajante de Comercio", "Ventanas de Tiempo", "Búsqueda local", "Recocido Simulado", "*Late Acceptance Hill-Climbing*"

Abstract:

The aim of this project is to analyze the Traveling Salesman Problem (TSP) and its variant with Time Windows (TSPTW). For each of these problems, it will be presented an explanation, its applications, mathematic formulation and resolution. These resolutions will be undertaken through exact methods and local search heuristics. Between these later are included two novel approaches: Threshold Acceptance and Late Acceptance Hill-Climbing, and two widely known and studied methods: Simulated Annealing and Hill-Climbing. A comparative will be made between them, taking as a basis the results achieved with the exact method and the best known values, with the aim to determine the algorithm with the best performance at solving this type of problems. For the TSPTW another heuristic will be presented, variant of the previously mentioned, called Compressed-Annealing Simulated.

Key words: “Traveling Salesman Problem”, “Time Windows”, “Local Search”, “Simulated-Annealing”, “Late Acceptance Hill-Climbing”

Índice general

Índice de figuras

Índice de tablas

Introducción

En el presente proyecto se tratará el Problema del Viajante Comercio (TSP, Traveling Salesman Problem) y su variante con ventanas de tiempo (TSPTW, Traveling Salesman Problem with Time Windows), con el fin de estudiar diferentes métodos para su resolución, entre los que encontraremos métodos exactos y métodos heurísticos. Para comparar estos métodos no sólo atenderemos a la función objetivo que obtengan, sino también a sus tiempos computacionales y la robustez que presenten.

El documento se estructura en cuatro capítulos diferentes, a excepción de este apartado en el que hablaremos también de la motivación que ha llevado al estudio de este tema particular, los objetivos que se plantean y las herramientas que se emplearán. En el capítulo uno se procede a una introducción del TSP, donde hablaremos de su historia, dificultad, aplicaciones, formulación matemática y variantes, entre otras. Además, se procederá a resolver este problema mediante el método exacto que emplea el "*solver"* elegido, basándonos en dos modelos matemáticos distintos. En el capítulo 2 se procede al estudio de cuatro métodos heurísticos de búsqueda local, dos de ellos muy conocidos como son el *Simulated Annealing* (Recocido Simulado) y *Hill-Climbing*, y otros dos algoritmos notoriamente más recientes: *Late Acceptance Hill-Climbing y Threshold Accepting*. Al final de éste capítulo se procederá a realizar un análisis de éstos cuatro métodos comparándolos con los resultados del método exacto que obteníamos en el capítulo uno. Por otro lado, en el capítulo tres y cuatro nos centraremos en el TSPTW, y al igual que hacíamos para el TSP, el capítulo tres servirá para introducir este problema y obtener resultados computacionales con el método exacto y el capítulo 4 se centrará en la resolución del TSPTW con heurísticas de búsqueda local, entre la que aparece un nuevo algoritmo: *Compressed-Annealing Simulated*. Al final de este capítulo se procederá a realizar un análisis similar a la que hacíamos para el TSP, aunque en este caso no sólo tomaremos como base de comparación los resultados obtenidos por el método exacto, sino también lo compararemos con los mejores valores conocidos hasta el momento. Por último, en el capítulo 5 se dan las conclusiones de este proyecto, aportando algunas ideas para posibles proyectos futuros.

I.1. Motivación y objetivos

Los problemas de optimización combinatoria están cada vez más presentes en nuestro día a día. Estos surgen debido a las necesidades que muestran ciertos sectores empresariales, como los relacionados con la logística, los procesos de fabricación, la programación de tareas o problemas de localización, entre otras. Como veremos más adelante, estos problemas normalmente no pueden resolverse evaluando las distintas soluciones posibles "a mano", y en función de la complejidad de los mismos, incluso potentes CPUs no son capaces dar una solución a dichos problemas. Por lo que la implementación de algoritmos de resolución que simplifiquen esta búsqueda de soluciones óptimas se hace cada vez más necesaria.

En el presente proyecto, y como ya se ha mencionado anteriormente, trabajaremos con el TSP y el TSPTW. Como veremos más adelante, El TSP está catalogado dentro de los problemas de mayor complejidad; de hecho aún no existe ningún método de resolución capaz de resolver este tipo de problemas en un tiempo polinomial. Si éste último se conoce por su dificultad de resolución, el TSPTW no sólo presenta la dificultad del problema del TSP si no que ésta se ve aumentada de forma exponencial al incorporar la componente de las Ventanas de Tiempo. El desafío de su resolución junto con la posibilidad de encontrar nuevas heurísticas que aún no hayan sido aplicadas a estos problemas y que puedan mejorar el rendimiento de las ya conocidas y estudiadas, han sido las principales motivaciones para el desarrollo del presente proyecto.

Por tanto, el objetivo principal del proyecto será la comparación de nuevos algoritmos con otros ampliamente conocidos, de cara a obtener el "mejor algoritmo" para la resolución de estos problemas; que será aquel que muestre un mejor rendimiento, el cual será medido a través de tres componentes: proximidad de la solución obtenida a la solución óptima o mejor valor conocido, tiempo computacional requerido y robustez.

I.2. Herramientas: Xpress-Mosel

Entre la amplia gama de compiladores y softwares disponibles en el mercado, en este proyecto se decidió utilizar Xpress-MP. Se trata de un entorno de modelización matemática y optimización de problemas, los cuales pueden ser proporcionados tanto en forma de librería como en un programa independiente.

Los dos elementos más importantes de Xpress-MP son el lenguaje de formulación de modelos (Xpress-Mosel) y los módulos para optimización (Xpress-Optimizer). Este último, será llamado dentro del programa a través del comando *"mmxprs"* y está confeccionado para aportar capacidad de resolución de alto rendimiento.

Además, hay disponibles otras herramientas que permiten, por ejemplo, dibujar gráficos, conectarse a bases de datos u hojas Excel, entre otras [36].

Otra de sus claras ventajas es la facilidad para trasladar estos códigos a otros lenguajes de alto nivel, como pueden ser C, C++ ó Java, dado que mosel utiliza sentencias *do*, *while*, *for*, etc., al igual que estos últimos.

Capítulo 1

El Problema del Viajante de Comercio

1.1. Introducción.

El Problema del Viajante de Comercio, conocido también con sus siglas inglesas TSP (*Traveling Salesman Problem*), trata de buscar la menor ruta posible que recorre una serie de ciudades (o nodos) de las que se conoce la distancia entre cada par de ellas. Dicha ruta debe visitar cada ciudad exactamente una vez y regresar a la ciudad origen. De una manera más informal: "Si un viajante parte de una ciudad, conociendo las distancias entre las distintas ciudades por las que ha de pasar ¿cuál es la ruta óptima que debe elegir para visitar todas las ciudades una sola vez y volver a la ciudad de partida?".

El problema del viajante es uno de los problemas más famosos y estudiados dentro de la optimización combinatoria[1]. A pesar de la aparente sencillez de su enunciado, el TSP es uno de los más complejos de resolver, lo que ha suscitado durante los últimos 60 años gran interés entre investigadores relacionados con las matemáticas, física, o biología entre otras.

1.2. Historia.

El primer uso del término "*Traveling Salesman Problem*" en la comunidad matemática puede datar del 1931-1932. No obstante, en 1832 se publicó un libro en Alemania titulado "*El viajante de comercio: cómo debe ser y qué debe hacer*

[1] La optimización combinatoria es un campo de las matemáticas aplicadas que combina técnicas de combinatoria, programación lineal y la teoría de algoritmos para resolver problemas de optimización donde el conjunto de posibles soluciones es discreto o se puede reducir a un conjunto discreto.

para conseguir comisiones y triunfar en su negocio. Por un viajante de comercio veterano" [26], que puede ser considerado como la primera referencia bibliográfica al TSP. Pese a que se trata de un libro que se centra principalmente en otros aspectos de la profesión, en el último capítulo se define, de manera explícita, el Problema del Viajante de Comercio.

Aunque el origen de éste término permanece aún desconocido, no hay duda de que Merrill Flood es el principal responsable de publicitarlo entre la comunidad matemática. Él recuerda haber oído hablar de ello a A. W. Tucker quien, a su vez, lo había oído de Hassler Whitney en la Universidad de Princeton (esta teoría no ha sido confirmada por el propio Tucker al no recordarlo, pero de haber ocurrido, sostiene que sería sobre 1931-1932).

En las décadas de los 50 y de los 60 el problema empezó a ser más popular. Gracias, entre otras cosas, a la campaña de publicidad llevada a cabo por "*Procter and Gamble*" (P&G) en 1962 [12]. La campaña anunciaba un concurso cuyo premio ascendería a 10.000$. Dicha recompensa sería para aquel que ayudara a *Toody y Muldoon* a decidir la ruta más corta entre 33 ciudades de Estados Unidos, como muestra la figura 1.1.

Figura.1.1. Concurso Car 54.

Los policías *Toody y Muldoon*, siempre conduciendo su famoso *Car 54*, eran personajes de una popular serie Americana. Y "su tarea" de visitar las 33 ciudades un claro ejemplo del Problema del viajante o TSP.

Mucha gente escribió a la compañía "Procter and Gamble" indicando que el problema era imposible, ya que no se podían comprobar todas la soluciones posibles. Sin embargo, 8 años antes (en 1954), el equipo de George Dantzing, Ray Fulkerson y Selmer Johnson ya había abordado un problema de rutas con 49 ciudades, encontrando el camino óptimo que minimizaba su coste.

Dantzing et al. no ganaron los 10.000$ del concurso propuesto por "Procter and Gamble", pero gracias a la implementación en ordenador de sus ideas ha sido posible trabajar de forma más fácil con el problema del TSP.

Durante los siguientes 17 años, los héroes de este campo siguieron siendo Danzting, Fulkerson y Johnson. Sin embargo, en 1971 los investigadores de IBM Michael Held y Richard Karp resolvieron un problema con 64 nodos. Dicho problema consistía en 64 puntos distribuidos aleatoriamente en una región cuadrada, cuyos costes de viaje eran considerados como la distancia en línea recta entre cada par de puntos.

Estos últimos fueron seguidos por los matemáticos Martin Grötschel and Manfred Padberg quienes se pusieron a la cabeza gracias a la construcción de una ruta óptima a través de 120 ciudades alemanas, publicada en 1977.

Tras ello, Padberg empezó a trabajar con el investigador de IBM Harlan Crowder, encontrando la solución óptima para un ejemplo de 318 nodos recogidos en una tarjeta de circuito impreso. Esto desembocaría en un año de bandera para el TSP, el 1987. Trabajando de forma independiente, Gröstschel and Padberg estaban al cargo de equipos que resolvieron de forma consecutiva la siguiente serie de instancias del TSP: 532 ciudades de Estados Unidos, 666 localizaciones en el mundo y 2392 nodos en un problema de perforación.

Siguiendo esta ola de éxitos el matemático Vasek Chvátal y William J.Cook entraron en el mundo del TSP, y ayudados por David Applegate y Robert Bixby, consiguieron en 1992 resolver un problema de perforado con 3038 nodos gracias a una larga red de ordenadores trabajando en paralelo. Tras este récord el equipo logró encontrar la ruta óptima para un problema de 13509 ciudades a través de los Estados Unidos en 1998, una ruta óptima de 24978 ciudades por Suecia en 2004 y la ruta mínima para una instancia de 85900 ciudades en 2006.

En la Figura 1.2 se recogen los hitos más importantes en la resolución del TSP desde 1954 hasta 2006.

Año	Autores	Nodos
1954	G. Dantzig, R. Fulkerson, S. Johnson	49
1971	M. Held, R.M. Karp	64
1975	P.M. Camerini, L. Fratta, F. Maffioli	67
1977	M. Grötschel	120
1980	H. Crowder and M. W. Padberg	318
1987	M. Padberg and G. Rinaldi	532
1987	M. Grötschel and O. Holland	666
1987	M. Padberg and G. Rinaldi	2392
1994	D. Applegate, R. Bixby, V. Chvátal, W. Cook	7397
1998	D. Applegate, R. Bixby, V. Chvátal, W. Cook	13509
2001	D. Applegate, R. Bixby, V. Chvátal, W. Cook	15112
2004	D. Applegate, R. Bixby, V. Chvátal, W. Cook	18512
2004	D. Applegate, R. Bixby, V. Chvátal, W. Cook, K. Helsgaun	24978
2004	W. Cook, Espinoza and Goycoolea	33810
2006	D. Applegate, R. Bixby, V. Chvátal, W. Cook	85900

Figura 1.2. Hitos históricos en la resolución del TSP

Los grandes avances en los últimos doce años que se recogen en la Figura 1.2 (1994-2006) fueron gracias al código de ordenador empleado a partir de 1992: El Concorde (vigente en la actualidad y consistente en más de 130000 líneas de código en C que en su día supuso una nueva etapa en lo que ha resolución del TSP respecta). Matemáticos y otros profesionales trabajan cada día en la mejora de este programa compuesto por las mejores técnicas disponibles hasta el momento.

En la Figura 1.2 se puede observar la progresión en el número de ciudades hasta el año 2006.

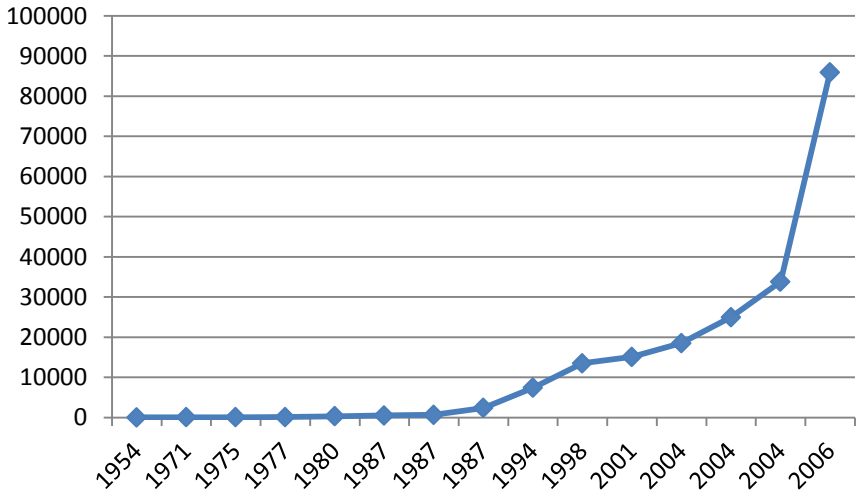

Fig.1.3. Progresos en la historia del TSP

1.3. Dificultad

El TSP es un problema NP-duro. Igual que ocurre con otros problemas en el ámbito de la Organización Industrial, el TSP se caracteriza por ser fácilmente descriptible pero difícil de resolver, por lo que ha despertado un gran interés a lo largo de la historia no sólo en su estudio sino también en la obtención de heurísticas para su resolución.

Su dificultad de resolución radica en que el número de posibles soluciones aumenta significativamente al aumentar el tamaño de la muestra (n). Si pensamos en el concurso propuesto por "*Procter and Gamble*" del que hemos hablado en el anterior apartado, en el cual se ofrecía 10.000$ para quien pudiera ayudar a *Toody y Muldoon* a decidir la ruta más corta entre 33 ciudades; a priori podemos pensar que en lugar de hacer complicados cálculos, uno podría haber comprobado los distintos caminos y luego elegir la mejor solución. Sin embargo, veamos como no resultaría sencillo. Como el origen viene determinado, restarían $(n-1)$ puntos para empezar, a continuación tendríamos que elegir cualquiera de los $(n-2)$ restantes y así sucesivamente. De esta forma, multiplicando todas estas cantidades obtenemos el número total de rutas o soluciones posibles que han de ser evaluadas: $(n-1)!$

En el caso particular de 33 ciudades esto significaría evaluar $32! = 2{,}63 * 10^{35}$; es decir, para un problema no excesivamente grande el número de pruebas aumenta considerablemente. Así, aunque el problema puede resolverse en un número finito de pasos, esto no es suficiente y se precisa buscar otros algoritmos que reduzcan este número de pruebas.

La única reducción que se puede hacer es en el caso que la dirección de ida sea igual a la de vuelta; es decir que no importe la dirección en la que viajemos. Sin embargo, el número de soluciones posibles seguiría siendo abrumador. Para nuestro caso particular:

$$\frac{32!}{2} = 131565418466846765053609080000000$$

Para esta cifra necesitaríamos alrededor de 28 trillones de años (y mucho tiempo libre) para comprobar todas las posibles soluciones.

Por esta razón, el problema del viajante ha alcanzado una importante posición en la teoría de la complejidad. Según esta teoría, los problemas para los que existe un buen algoritmo de resolución son clasificados como P, de tiempo polinómico. Por el contrario, los problemas para los que aún no se ha encontrado un algoritmo eficiente son clasificados como NP, de tiempo polinómico no determinista.

Además, se ha demostrado que muchos de los problemas denominados como "difíciles" son computacionalmente equivalentes. Es decir, un algoritmo polinómico que puede usarse para resolver uno de estos problemas sería válido para resolver el resto de los englobados en esta categoría. Estos problemas reciben el nombre de NP-duro (más conocidos por su nombre inglés *NP-hard*). Por esta razón no es de extrañar que el Instituto Clay [11] haya ofrecido un millón de dólares a quien descubra un algoritmo eficiente para el TSP o demuestre la no existencia del mismo. De existir, todos los problemas NP podrían resolverse en un tiempo polinómico y se concluiría que P y NP son iguales.

1.4. Aplicaciones

A pesar de que su nombre hace pensar que tiene una aplicación muy concreta: decidir la ruta para un vendedor o agente comercial, el TSP se puede emplear en cualquier situación que requiera seleccionar nodos en cierto orden de cara a reducir los costos. Hasta el día de hoy, éste se ha aplicado sobre una gran variedad de problemas. A continuación, se comentan brevemente algunas de las aplicaciones más importantes del Problema del Viajante de Comercio.

- **Logística:** Las aplicaciones más directas y más abundantes del TSP se centran en el campo de la logística. El flujo de personas, mercancías y vehículos en torno a una serie de ciudades o clientes se adapta perfectamente a la filosofía del TSP. Dentro de este campo encontramos:

  - Vendedores y turistas: La mayoría de las agencias de viajes o agentes comerciales utilizan algún planificador de rutas para determinar cuál es el mejor camino para visitar los puntos que desean y regresar al punto de origen, ya sea el hotel o la oficina. Estos planificadores generalmente incluyen algún algoritmo de resolución del TSP.

  - Horarios de transportes laborales y/o escolares. Actualmente, muchas empresas dedicadas al transporte de personas adquieren software de resolución de TSP que les permite reducir gastos de una manera significativa.

  - Empresas de paquetería o servicios de comida a domicilio: en ocasiones el reparto de mercancía puede modelizarse como un TSP. Se trata de los casos en los que las casas están muy alejadas unas de otras o cuando sólo se debe visitar algunas de ellas.

- **Industria:** Las aplicaciones en industria no son tan numerosas como las anteriores, pero la aplicación del TSP a este ámbito también ha dado lugar a una significativa reducción de los costes. Entre estas aplicaciones encontramos:

  - Producción de placas de circuito impreso: Estas placas normalmente contienen numerosos agujeros para hacer posteriormente las conexiones oportunas. Dichos agujeros son producidos por perforadoras automáticas que se mueven entre los puntos especificados creando un agujero tras otro. Por tanto, la aplicación del TSP en este campo consiste en, tomando como ciudades cada una de las posiciones donde debe realizarse una perforación y las distancias entre ellas como el tiempo que necesita la máquina en trasladarse de una a otra, minimizar el tiempo que pierde la taladradora en moverse de una posición a otra. Esto no sólo ocurre con las perforadoras, también cuando es necesario realizar soldaduras o grabados.

  - Microprocesadores personalizados: La misma clase de aplicación, pero a una escala mucho menor, fue la desarrollada en los Laboratorios Bell a mediados de los 80. Estos investigadores desarrollaron una técnica para la rápida producción de microprocesadores a medida. Este proceso comienza con un microprocesador básico, el cual está formado por una red de puertas lógicas sencillas. Después, porciones de esta red son cortadas con láser para crear grupos individuales de puertas que permiten al chip desarrollar las funciones especificadas por el cliente. En este caso, el TSP sirve para guiar al laser a través de los puntos que requieren ser cortados, lo que permite reducir el tiempo hasta un 50%.

  - Secuenciación de tareas. Se refiere al orden en el cual n trabajos tienen que ser procesados de tal forma que se minimice el costo total de producción. Supongamos que una maquina debe realizar una serie de tareas en el mínimo tiempo posible y sin importar el orden de las mismas. Supongamos que se tarda un tiempo t_{ij} en poner a punto la máquina para realizar la tarea j si la última tarea que realizó fue la i. En ese caso, podemos aplicar el TSP suponiendo que cada tarea es uno de los nodos a visitar y han de realizarse todas las tareas para producir el producto considerando que la distancia entre ellos es t_{ij}. El nodo origen y destino serán el estado de la maquina cuando empieza o termina el producto. Dado que el tiempo que se emplea en realizar cada tarea no depende del orden, no será necesario incluir estos tiempos en el modelo.

- **Búsqueda de planetas:** A pesar de que normalmente asociamos el TSP con aplicaciones que requieren la visita física a lugares remotos, éste también puede ser empleado cuando los sitios se pueden observar desde la distancia sin necesidad de viajar realmente. Un ejemplo de ello puede darse cuando los nodos considerados son planetas, estrellas o galaxias, y las observaciones se deben hacer con algún tipo de telescopio.
  Interesantes ejemplos del TSP han sido considerados en el trabajo de planificación de misiones de telescopios espaciales de la NASA. Éste es usado para determinar la secuencia de observaciones que deberá hacer el telescopio una vez esté en su posición.

- **Organización de datos:** La organización de datos en grupos (*clusters*) de elementos con propiedades similares es un problema básico en análisis de datos. El problema del viajante ha sido aplicado frecuentemente en problemas de este tipo cuando existe una buena medida de la similitud $s(a;b)$ entre cada pareja de datos $(a;b)$. La idea es que, usando $s(a;b)$ como distancias, un camino Hamiltoniano de coste máximo situaría las observaciones más parecidas cerca unas de otras y se podría, por tanto, utilizar intervalos del camino como *clusters*. Como ya hemos comentado el TSP busca siempre la ruta de coste mínimo. Sin embargo, aquí buscamos el camino de coste máximo, puesto que la medida de similitud toma un valor mayor cuanto más próximas estén las observaciones entre sí. Bastaría con multiplicar las distancias por (-1) obteniendo el camino de coste mínimo, de coste negativo, que sería equivalente al camino de coste máximo.
  Una vez obtenido el camino, la selección de los *clusters* puede hacerse a mano, buscando los puntos de corte naturales según la naturaleza de los datos, o puede hacerse de manera automática. Para esto último, y según William J.Cook [12], uno de los métodos más usados es el propuesto por S. Climer y W. Zhang: añadir k ciudades cuya distancia al resto de ciudades sea 0. Así, estas ciudades adicionales servirían para identificar los k *clusters*, puesto que un camino óptimo usaría las conexiones de coste cero.

Cada día, las áreas donde trabaja el TSP son más dispares. Entre los últimos proyectos exitosos en referencia a la aplicación de éste último encontramos: planificación de caminos de senderismo, minimización de gasto de papel o patrones de corte en la industria cristalera.

1.5. Modelización y formulación matemática

1.5.1. Descripción formal

Existen múltiples formulaciones distintas del TSP; de hecho, el número de formulaciones propuestas asciende a 8 [32]. En este apartado se van a revisar algunas de las más utilizadas.

Previamente aclararemos los siguientes conceptos:

- Un *grafo* G = (N,A) consistente en un conjunto N de elementos llamados vértices o nodos y un conjunto A cuyos elementos llamaremos arcos o aristas. Cada arco o arista representa una conexión directa entre dos nodos o elementos de N.

- Un *grafo orientado o dirigido* es aquel en el que los arcos son pares ordenados; el arco (i,j) empieza en el nodo i y termina en el nodo j. (ver Figura 1.4.a)

- Un *grafo no orientado o no dirigido*, es un grafo en el que (i,j) y (j,i) representan el mismo arco. (ver Figura 1.4.b)

- Un *grafo completo* es un grafo (orientado o no) en el que todas las aristas posibles están presentes. (ver Figura 1.4.c)

- Un *camino* es una secuencia de aristas en las que todos los vértices son distintos.

- Un *circuito* es una secuencia de aristas en la que el primer vértice y el último son el mismo y además no hay más vértices coincidentes que estos dos.

- Un *camino hamiltoniano* es un camino que recorre todos los nodos del grafo una y sólo una vez.

- Un *circuito hamiltoniano* es un circuito que recorre todos los nodos del grafo una y sólo una vez.

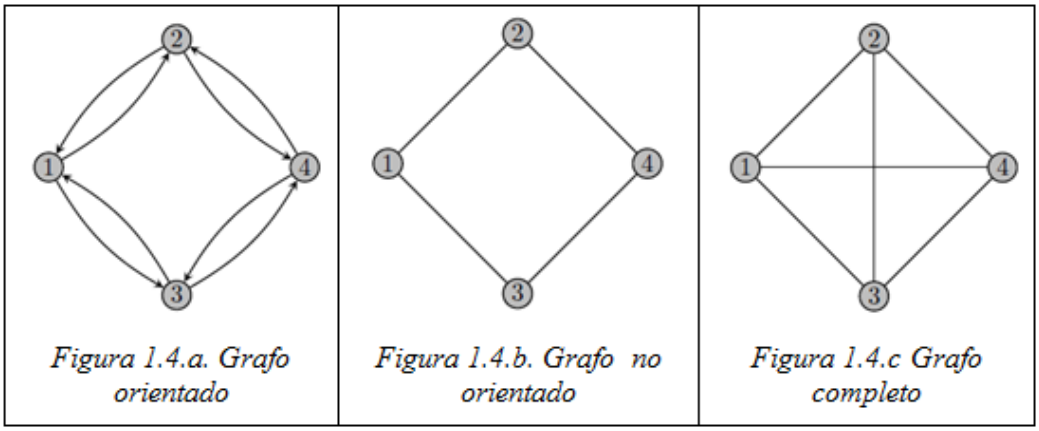

Figura 1.4.a. Grafo orientado

Figura 1.4.b. Grafo no orientado

Figura 1.4.c Grafo completo

Figura 1.4. Tipos de grafo

Conociendo estas definiciones, el problema del TSP puede ser descrito según la teoría de grafos de la siguiente manera: Sea $G = (N, A)$ un grafo completo, donde $N = 1, \dots, n$ es el conjunto de nodos o vértices y A es el conjunto de arcos. Los nodos $i = 2, \dots, n$ se corresponden con los clientes a visitar mientras que el nodo 1 es considerado la ciudad de origen y destino. A cada arco (i, j) se le asocia un valor no negativo d_{ij}, que representa la distancia del vértice i al j. El uso de los arcos (i, i) no está permitido, por lo que se impone que $d_{ii} = 1 \; \forall \, i \in N$.[2]

A modo de resumen podríamos concluir que el Problema del Viajante de Comercio se define sobre un grafo (generalmente completo) con costes asociados a los arcos y consiste en encontrar el circuito hamiltoniano de coste mínimo.

Se diferencian dos tipos de problema en función del tipo de grafo G:

- ATSP: (TSP asimétrico): Cuando G es un grafo dirigido. Lo que significa que la matriz de distancias es asimétrica; es decir, la distancia de ida no tiene porqué ser igual a la de vuelta.

- STSP: (TSP simétrico): Cuando G es un grafo no dirigido, es decir; $d_{ij} = d_{ji} \; \forall \, (i, j) \in A$, y por tanto la matriz de distancias es simétrica o lo que es igual, la distancia de ida es igual a la de vuelta.

[2] A la hora de describir o formular el problema del TSP, podemos encontrarnos la notación d_{ij} ó su equivalente c_{ij}, cuando en lugar de considerar la distancia se considera el coste.

1.5.2. Formulación matemática.

Como ya hemos comentado el objetivo del Problema del Viajante es encontrar una ruta que, comenzando y terminando en una misma ciudad, en este caso denotada como ciudad 1, pase una sola vez por cada una de las ciudades restantes y minimice la distancia total recorrida. Si definimos las variables binarias de decisión x_{ij} para todo $(i,j) \in A$, de forma que tomen el valor 1 si el arco (i,j) forma parte de la solución y 0 en otro caso; tenemos que el problema de programación lineal asociado al Problema del Viajante consiste en minimizar la siguiente función objetivo:

$$\sum_i \sum_j d_{ij}\, x_{ij}$$

Sujeto a las siguientes restricciones:

$$\sum_i x_{ij} = 1 \; \forall j$$

$$\sum_j x_{ij} = 1 \; \forall i$$

$$x_{ij} \in \{0,1\} \; \forall ij$$

La primera restricción asegura que sólo un arco puede entrar en cada vértice (a cada ciudad j solo se puede llegar desde una ciudad i). Mientras que la segunda asegura que solo un arco puede salir de cada ciudad (desde cada ciudad i solo se podrá llegar a una ciudad j).

Estas restricciones son necesarias pero no suficientes, pues pueden dar lugar a varios *subtours* o subcircuitos, como se puede observar en la Figura 1.5. Obsérvese que $x_{16} = x_{65} = x_{51} = x_{24} = x_{43} = x_{32} = 1$, por lo que no se viola ninguna de las restricciones.

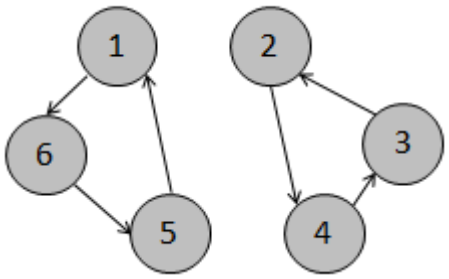

Figura 1.5. Formación de subcircuitos

Por tanto, para que el modelo nos proporcione una solución factible tenemos que añadir restricciones para eliminar los *subtours* o subcircuitos. Esto puede hacerse de tres formas:

- Restricciones de eliminación de *subtours*. Se trata de incluir la condición de que haya al menos un arco que salga de cualquier *subtour*:

$$\sum_{i\in S, j\notin S} x_{ij} \geq 1, \forall S \subset N \,/\, S \neq \Phi, S \neq N$$

  Esta restricción indica que cada subconjunto de nodos S es abandonado al menos una vez.

  El problema es que se generan demasiadas restricciones. Por ejemplo, si $|N|$ = n, se necesitarían $2^{\mathrm{n}} - 2$ restricciones del tipo anterior. (En total $2^{\mathrm{n}} + 2\mathrm{n} - 2$ restricciones: $2^{\mathrm{n}} - 2$ por las de ruptura de subcircuito que acabamos de explicar, n por la restricción de "entrada" y n por la restricción de "salida" de cada nodo).

  En la práctica se puede resolver el problema usando procedimientos de generación de restricciones o planos de corte. De esta forma, sólo se añaden las restricciones de eliminación de subtours que no cumple o viola la solución actual. La idea es no añadir todas las restricciones, sino ir añadiendo las que se necesiten.

- Restricciones de Tucker, Miller y Zemlin**.** Otra forma de evitar la formación de subcircuitos es a través de nuevas variables de decisión. De esta forma, definimos las variables $u_i, \forall_i \in N$, que representan el lugar de la secuencia en el que se visita el nodo i. Para el nodo 1 (origen), prefijamos el valor de u_0 en 1, pues es el primer nodo a visitar. A continuación se añaden $n - 1$ variables auxiliares continuas (y sin restricción de signo) u_1, quienes tomarán un valor entre 2 y n, y las $(n-1)^2$ restricciones siguientes:

$$\mathrm{C} - \mathrm{u}_j + n x_{ij} \leq n - 1$$
$$2 \leq i \neq j \leq n$$

  Es fácil ver que al añadir este número fijo de restricciones, es imposible que se forme ningún *subtour*. Por reducción al absurdo, si se formara un *subtour*, se formará uno que no contiene al nodo 1, por ejemplo $S = \{\mathrm{u}_1, \mathrm{u}_2, \ldots, \mathrm{u}_\mathrm{k}\}$ con k nodos. Escribiendo las restricciones anteriores a lo largo de este circuito, quedaría: $\mathrm{u_i} - \mathrm{u_{i+1}} + n \leq n -$

$1, i = 1, \dots, k$. Y ahora, sumando estas k desigualdades, se llega a $kn \leq k(n-1)$ lo que sería imposible.
Por lo tanto, todo *subtour* debe contener al nodo 1, lo cual es equivalente a decir que sólo puede haber un *subtour*.

- Problema auxiliar de redes**.** La última forma que proponemos es, además de las variables binarias del problema, considerar unas variables de flujo, y plantear un problema de flujo en redes, con la siguiente formulación:

$$Min \sum_i \sum_j d_{ij}\, x_{ij}$$

$$\sum_i x_{ij} = 1 \ \forall j$$

$$\sum_j x_{ij} = 1 \ \forall i$$

$$x_{ij} \in \{0,1\} \ \forall ij$$

$$\sum_j y_{ij} - \sum_j y_{ji} = b_i \qquad i = 1, \dots, n$$

$$0 \leq y_{ij} \leq (n-1)x_{ij}$$

En el nodo 1 elegido asignamos oferta $n-1$, es decir $b_1 = n-1$ y para el resto de los nodos tomamos demanda 1, es decir $b_1 = -1$, $i = 2, \dots, n$. De esta forma aseguramos que los nodos estarán conectados entre sí porque las unidades de flujo tienen que llegar a cada uno concreto. Y además, por las restricciones de asignación: de cada nodo debe salir un único arco y entrar un único arco, por lo que la solución debe ser un único tour**.**

1.6. Variantes del TSP

Numerosas variaciones del TSP que son estudiadas han surgido de una multitud de aplicaciones reales o potenciales. Algunas de las más importantes se recogen en el libro de Gutin y P.Punnen [22], de las cuales destacamos las siguientes:

- MAX TSP: El objetivo en este caso es encontrar un circuito hamiltoniano de coste máximo.

- TSP con cuello de botella: En este caso el objetivo no es minimizar el coste total sino encontrar un circuito hamiltoniano tal que el mayor coste entre todas las aristas del mismo sea lo menor posible.

- TSP con múltiples visitas (TSPM): Similar al TSP, pero en este caso buscamos encontrar el circuito hamiltoniano que minimice la distancia total pasando por cada nodo *al menos* una vez.

- TSP agrupado: Los nodos o ciudades están divididos en *clusters* o grupos, se busca encontrar un circuito hamiltoniano de coste mínimo de forma que las ciudades dentro del mismo grupo sean visitadas de manera consecutiva.

- TSP generalizado (GTSP): Como en el caso anterior, los nodos están divididos en *clusters*, pero ahora se busca encontrar un circuito hamiltoniano de coste mínimo que visite exactamente un nodo de cada grupo.

- TSP con múltiple viajantes (mTSP): existe un número determinado de viajantes, cada uno de ellos debe visitar ciertas ciudades, sin visitar las que ya han sido visitadas por el resto de viajantes.

- TSP periódico: Considera un período de planificación en función del día (k-día), para valores de k dados. Queremos encontrar k ciclos en G tal que cada nodo de G sea visitado un numero prescrito de veces durante el k - día período y el costo total del viaje para todo el conjunto sea mínimo.

- TSP con ventanas de tiempo (TSPTW): donde cada ciudad o nodo tiene un tiempo mínimo de llegada y máximo de salida. Este TSP lo abordaremos a fondo en el capítulo 3 y 4.

  Hay otras muchas variaciones que podrían haber sido descritas. Entre ellas encontramos el TSP blanco y negro, TSP dependiente del tiempo, TSP con entregas, TSP angular, TSP selectivo, TSP remoto, minmax TSP o TSP multicriterio.

1.7. Métodos de resolución

Desde los años 50, muchos métodos han sido aplicados para la resolución del TSP. En general estos métodos pueden ser clasificados en dos categorías: algoritmos aproximados y algoritmos exactos. Los algoritmos exactos usan modelos matemáticos, mientras que los algoritmos aproximados buscan soluciones usando heurísticas o metaheurísticas[3] y mejoras iterativas. Ejemplos de algoritmos exactos son: Ramificación y poda (*Branch and Bound*), Relajación Lagrangiana y Programación Entera. Dentro de los algoritmos aproximados tenemos, por ejemplo, la heurística del vecino más próximo, heurística de Greedy o heurísticas de inserción, entre otras. Por último, en cuanto a ejemplos de heurísticas de mejora tenemos *k-opt*, Recocido Simulado, Búsqueda Tabú o Algoritmos Genéticos [1].

1.7.1. Algoritmos exactos.

- Ramificación y Acotación (*Branch and Bound*).

Los orígenes de la idea de Ramificación y Acotación se remontan al trabajo de Dantzig, Fulkerson y Johnson (1954-1959) sobre el TSP, W.J.Cook et al. lo resumen de la siguiente forma [13]. Supongamos que partimos del grafo G = (V,E) con costes asociados a cada arco ($c_e : e \in E$), además consideramos un nuevo parámetro $\mathcal{T}$ que denota el conjunto de todas las posibles rutas de G. El límite inferior para el TSP es un número B tal que $C(T) \geq B$ para todo T $\in \mathcal{T}$. Ahora supongamos que dividimos $\mathcal{T}$ en dos conjuntos $\mathcal{T}_0$ y $\mathcal{T}_1$ tal que $\mathcal{T}_0 \cup \mathcal{T}_1 = \mathcal{T}$. Si podemos encontrar los números B_0 y B_1 tal que $C(T) \geq B_0$ para todo T $\in \mathcal{T}_0$ y $C(T) \geq B_1$ para todo T $\in \mathcal{T}_1$, entonces el mínimo entre B_0 y B_1 será el límite o cota inferior del TSP. La ruptura o división de $\mathcal{T}$ en $\mathcal{T}_0$ y $\mathcal{T}_1$ permite que nuestra técnica de delimitación o acotación inferior funcione mejor que todo el conjunto $\mathcal{T}$. Esto es la base de los métodos de ramificación y acotación: "romper" sucesivamente el conjunto de soluciones y aplicar la técnica de acotación inferior a cada parte. Ante la imposibilidad de resolver alguno de estos problemas o simplemente porque la solución obtenida no sea suficientemente buena, estos subproblemas pueden ser divididos a su vez en dos subproblemas, recibiendo el nombre de ramificaciones (generalmente estos métodos se representan por un árbol de decisión). Así, en cualquier punto podemos obtener una acotación inferior del problema original tomando el mínimo de todos los límites inferiores considerados correspondientes a las "ramas del árbol". El proceso continua hasta que esta acotación inferior es mayor o igual que el coste del recorrido considerado (en ese caso se prueba que es el óptimo).

[3] Las metaheurísticas están diseñadas para "escapar·" de mínimos locales permitiéndose aceptar soluciones peores con el objetivo de encontrar la mejor solución global.

Dentro de los métodos para resolver problemas de programación entera, los más exitosos están basados en la ramificación y acotación. Un método particular de ello es la Ramificación y Corte (*Branch and Cut*). Se trata de una combinación de planos de corte y ramificación y acotación. Estos métodos trabajan resolviendo una secuencia de relajaciones de programación lineal (LP). Los planos de corte mejoran la relajación del problema para acercarse con una mejor aproximación al problema de programación entera, y la ramificación y acotación, por su parte, continúa dividiendo el problema y acercándose a la solución final de éste. Dicho método es usado en la mayoría de los códigos comerciales de programación entera, como es el caso de Xpress-Optimizer, optimizador que hemos utilizado a lo largo del proyecto.

1.7.2. Algoritmos heurísticos de construcción.

- Heurística del vecino más próximo

Uno de las heurísticas más sencillas utilizadas para el TSP es la llamada heurística "del vecino más próximo". Según la cual, el "viajante" comenzaría en una ciudad y después visitaría la ciudad más cercana a esta ciudad origen. Desde allí, se visitaría la ciudad más próxima que aún no haya sido visitada y así sucesivamente hasta que se haya pasado por todas las ciudades, momento en el que el "viajante" ha de regresar a la ciudad origen [29].

De forma esquemática tendríamos:

- **Paso 1**. Seleccionar un nodo inicial.
- **Paso 2**. Identificar al nodo más cercano al último agregado, siempre que no haya sido agregado.
- **Paso 3.** Repetir el paso 2 hasta incluir todos los nodos.

Debemos considerar que a la hora de construir esta solución, el algoritmo suele comenzar muy bien, seleccionado aristas de bajo coste/distancia. Sin embargo, al final del proceso pueden quedar vértices cuya conexión obliga a introducir aristas de mucha distancia. Esto es lo que se conoce como "miopía" del procedimiento, ya que en cada momento se escoge la mejor opción disponible sin "ver" que esto puede ocasionar problemas en iteraciones posteriores.

Para implementar este algoritmo puede seguirse el siguiente esquema [35]:

Esquema básico heurística del vecino más próximo
Seleccionar un nodo arbitrario j, $l = j$ y $T = \{1,2,\dots,n\} \setminus \{j\}$ **While** $T \neq \emptyset$ **do** Seleccionar $j \in T$ tal que $c_{lj} = \min\{c_{lj} \mid i \; e \; T\}$ Conectar l a j y establecer $T = T \setminus \{j\}$ y $l = j$ **End-do** Conectar l al primer nodo que seleccionamos de forma arbitraria para formar la ruta.

Figura 1.6. Esquema básico heurística del vecino más próximo

1.7.3. Algoritmos heurísticos de mejora.

- k-opt

Se trata de una heurística de mejora, y por lo tanto, se ha de partir de una solución factible ya conocida. Se basa en la sustitución de k arcos de una ruta por otros k arcos de la misma. Se dice que un tour es k-optimo (k-opt) si es imposible obtener una ruta con menor distancia (o coste) remplazando k de sus arcos por otros k arcos distintos. Aunque la capacidad computacional ha aumentado mucho en los últimos años, en este proyecto sólo consideraremos el caso $k = 2$, por lo que en lo siguiente nos referiremos a ello como heurística 2-opt.

La heurística 2-opt está basada en el hecho de que para un problema Euclídeo, si dos aristas se cruzan, pueden ser sustituidas por otras dos que no se crucen, mejorando así el problema.

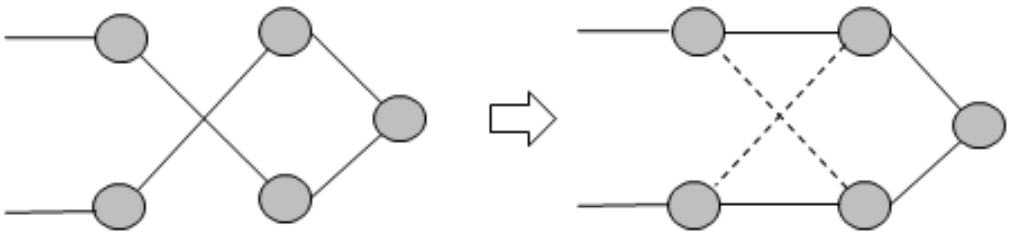

Figura 1.7. Mejora gracias al intercambio de aristas cruzadas

Teniendo en cuenta la descripción general, para el caso $k = 2$, es fácil intuir que se llevaría a cabo la sustitución de 2 de sus arcos, $(i, i+1)$ y $(j, j+1)$ con otros dos arcos, en este caso (i, j) y $(i+1, j+1)$ (ver Figura 1.8). Además, como podemos ver la orientación del camino $(i+1, \dots, j)$ cambia en la nueva ruta.

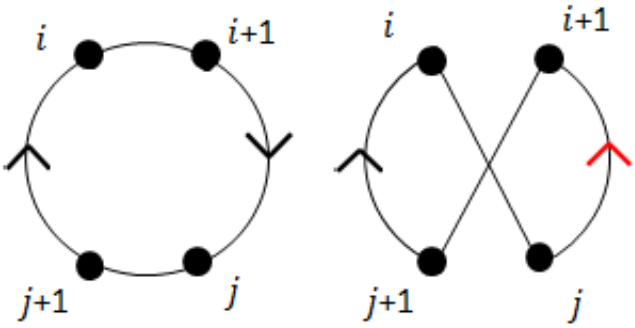

Figura 1.8. Intercambio 2-opt

Este intercambio sólo mejorará la actual ruta si y sólo si:

$$t_{i,i+1} + t_{j,j+1} > t_{i,j} \; t_{i+1,j+1}$$

Esquema básico heurística 2-opt

Generar una solución inicial s
Final = 0
While final = 0 **do**
Para todo nodo $i = 1, \dots, n$
Examinar todos los movimientos 2-opt que incluyan la arista de i a su sucesor en el ciclo.
If alguno de los movimientos examinados reduce la longitud de la ruta
Then realizar el mejor de todos total y actualizar s
final = 1
Else
final = 1
End-if
End-do

Figura 1.9. Esquema básico heurística 2-opt

1.7.4. Otros algoritmos

Los tres anteriores algoritmos han sido explicados debido a su uso durante este proyecto. Ramificación y Acotación, como ya se ha comentado previamente, es el método exacto en el que se basa el solver empleado (Xpress-Optimizer). La heurística del vecino más próximo será empleada en las heurísticas utilizadas en los capítulos 2 y 4 para generar soluciones iniciales. Mientras que para realizar el paso de una solución factible a otra se empleará el intercambio 2-opt.

Sin embargo, hay otras tres heurísticas muy conocidas que debido a su extendido uso en la resolución del TSP y de otros muchos problemas, merecen tener cabida en este apartado. Hablamos de las metaheurísticas Búsqueda Tabú, Algoritmos Genéticos y GRASP [14].

- Búsqueda Tabú

La Búsqueda Tabú es un tipo de búsqueda por entornos[4], que según su definidor, "guía un procedimiento de búsqueda local para explorar las soluciones más allá del óptimo local".

La Búsqueda Tabú permite moverse a una solución del entorno que no sea tan buena como la actual, de modo que se pueda escapar de óptimos locales y continuar estratégicamente la búsqueda de soluciones aún mejores. Para ello, se clasifica un número determinado de movimientos recientes como "movimientos tabú", los cuales no es posible repetir durante un horizonte temporal. Es decir, se mantiene una memoria de eventos (por ejemplo soluciones o algún otro elemento propio de las soluciones visitadas) que sean de interés en relación con el pasado. Así, al explorar el "entorno" de la solución actual, estará prohibido aceptar como la siguiente solución un elemento de la búsqueda tabú.

- Algoritmos Genéticos

Los Algoritmos Genéticos imitan el proceso darwiniano de selección natural. En cada iteración el algoritmo procesa no solo una solución sino en conjunto de ellas, llamado población.

Esta población inicial se genera de manera aleatoria, y se trataría de un conjunto de individuos (formados por cromosomas) que representan posibles soluciones del problema. Después, en cada iteración y utilizando tres elementos clave (operador de selección, operador de cruce y operador de mutación), la población de soluciones es transformada en la siguiente población de la siguiente manera:

[4] La búsqueda por entornos son métodos que parten de una solución factible y, mediante alteración de esa solución, van pasando de forma iterativa (y mientras no se cumpla algún criterio de parada) a otras factibles de su "entorno", almacenando como óptima la mejor de las soluciones visitadas. (Véase Introducción Capítulo 2)

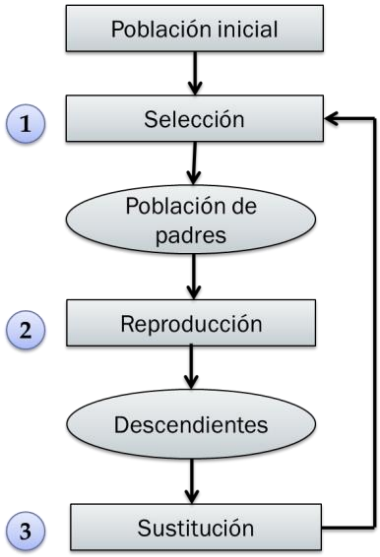

Figura 1.10. Esquema Algoritmos Genéticos

En primer lugar, las soluciones de la población inicial se agrupan en parejas. De esos pares de individuos se eligen a los mejores mediante el operador de selección, dando lugar a la "población de padres". El operador de selección que se usa normalmente es una selección mediante torneo. Dicho método de selección compara los valores objetivos de cada par de soluciones escogiendo la mejor de ellas.

Después pasaríamos a la fase de reproducción. En la que de nuevo agrupamos la población de padres en parejas. Cada par de padres es transformado en dos nuevas soluciones llamados descendientes. Para ello se usa el operador de cruce. El operador de cruce más utilizando es la permutación de 2 puntos (PMX), que consiste en elegir un subsegmento de uno de los progenitores y cruzarlos preservando el orden y la posición de la mayor cantidad de genes posible del otro manteniendo la coherencia. (Ver figura 1.11)

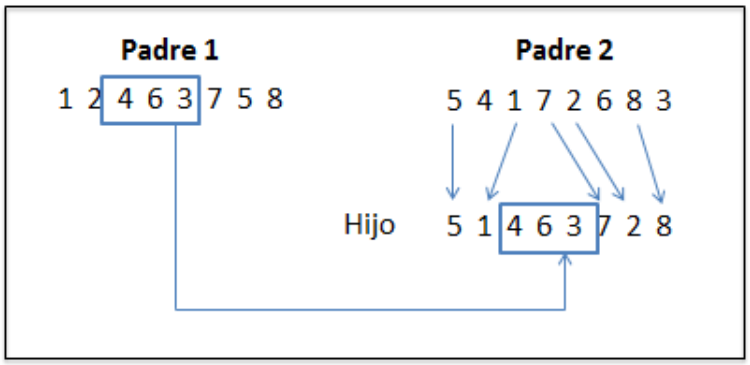

Figura 1.11. Ejemplo aplicación del operador PMX

Tras realizar el cruce de las parejas, cada uno de los individuos de la población de descendientes podrá sufrir mutación con una pequeña probabilidad p (operador de mutación). Se trata de un operador de cambio estándar, que al azar escoge dos vértices en la solución y cambia sus posiciones.

El último paso será quedarnos con las mejores soluciones. Diferentes criterios pueden seguirse en este punto. Ejemplos de ello tendríamos intercambiar el peor hijo con el mejor individuo de la población inicial de cada iteración o bien quedarnos con las mejores soluciones del total de soluciones exploradas (padres y descendientes).

- GRASP

Este algoritmo fue desarrollado originalmente por Feo y Resende (1989), su nombre proviene del acrónimo de *Greedy Randomized Adaptative Search Procedure,* cuya tradición literal podría ser Procedimientos de Búsqueda Ávidos, Aleatorizados y Adaptativos. Cada uno de estos adjetivos caracteriza una de las componentes de esta metaheurística.

Se trata de un método *multi-start* en el cual cada iteración consta de dos pasos: la fase de construcción y la fase de búsqueda local. En el primero se construye una solución inicial factible mediante un método greedy aleatorizado, cuyo entorno es investigado hasta que, gracias a un procedimiento de intercambio, se encuentra un óptimo local. La forma en la que se lleva a cabo la segunda fase puede ir desde sencillos métodos de búsqueda local hasta procedimientos muy sofisticados.

Una vez que se han ejecutado los dos pasos, la solución obtenida se almacena y se procede a efectuar una nueva iteración, guardando cada vez la mejor solución que se haya encontrado hasta el momento. En definitiva, GRASP dirige la mayor parte de su esfuerzo a construir soluciones de alta calidad que posteriormente son procesadas con el fin de conseguir otras aún mejor.

1.8. Resultados computacionales método exacto

Para conocer las soluciones óptimas, y poder posteriormente compararlas con las soluciones obtenidas mediante el empleo de heurísticas, nuestra primera opción será la resolución con el *solver* que incorpora "Xpress".

Para obtener los resultados se han empleado dos modelos: el modelo de Tucker, Miller y Zemlin y el modelo auxiliar de redes.

Antes de introducir los datos computacionales se hace conveniente exponer las características principales del sistema en el que se ha ejecutado. En nuestro caso se trata de un ordenador con un sistema operativo Windows7 Professional de 64 bits, un procesador Intel CORE con velocidad 3.60GHz y una memoria instalada (RAM) de 16GB.

En nuestro caso, se van a probar ambos métodos para las instancias propuestas por Dumas et al. [16], de las cuales hemos escogido 5 grupos: 20, 40, 60, 80 y 100 nodos.

En las siguientes tablas aparecen los resultados obtenidos para cada grupo de instancias junto con el tiempo requerido para obtener dichos valores, el cual se ha limitado a un máximo de 200 segundos. Aquella solución que destaque sobre la otra será marcada en negrita. Además, en caso de que el método no encuentre ningún ruta factible se pondrá un guión (-) en la casilla correspondiente.

- **20 nodos:**

	Tucker et al.		Redes	
Archivo	Resultado	Tiempo	Resultado	Tiempo
n20w20.001	198	0,172	198	0,094
n20w20.002	174	0,608	174	0,515
n20w20.003	213	2,295	213	1,186
n20w20.004	189	0,686	189	1,592
n20w20.005	193	0,641	193	2,184
n20w40.001	157	0,375	157	0,141
n20w40.002	189	0,468	189	0,109
n20w40.003	196	0,265	196	0,249
n20w40.004	180	1,902	180	0,124
n20w40.005	175	0,297	175	0,344
n20w60.001	183	0,483	183	0,672
n20w60.002	164	0,548	164	0,234
n20w60.003	191	1,463	191	0,219
n20w60.004	157	0,452	157	0,234
n20w60.005	193	0,609	193	0,250
n20w80.001	180	0,406	180	0,140
n20w80.002	188	0,951	188	0,641
n20w80.003	188	0,609	188	0,218
n20w80.004	189	17,50	189	0,188
n20w80.005	158	0,437	158	0,359
n20w100.001	191	0,218	191	0,265
n20w100.002	179	0,468	179	0,343
n20w100.003	188	0,608	188	0,608
n20w100.004	215	0,686	215	0,234
n20w100.005	201	0,515	201	0,297

Tabla 1.1.Valores objetivos obtenidos mediante el método exacto para n = 20 (TSP)

- **40 nodos:**

	Tucker et al.		Redes	
Archivo	Resultado	Tiempo	Resultado	Tiempo
n40w20.001	272	200,0	**259**	24,48
n40w20.002	243	46,03	243	5,789
n40w20.003	263	1,887	263	1,108
n40w20.004	243	200,0	**228**	14,18
n40w20.005	247	32,05	247	1,562
n40w40.001	251	3,634	251	1,421
n40w40.002	232	5,554	232	8,909
n40w40.003	258	200,0	**257**	1,404
n40w40.004	244	4,743	244	2,106
n40w40.005	241	2,872	241	1,747
n40w60.001	232	17,22	232	1,982
n40w60.002	256	2,122	256	1,513
n40w60.003	225	200,0	225	1,326
n40w60.004	228	200,0	228	4,727
n40w60.005	244	200,0	224	3,416
n40w80.001	257	2,293	257	1,951
n40w80.002	254	4,804	254	7,021
n40w80.003	264	200,0	**258**	3,416
n40w80.004	250	34,69	250	8,691
n40w80.005	268	200,0	**253**	3,432
n40w100.001	240	2,137	240	1,076
n40w100.002	255	128,1	255	14,81
n40w100.003	214	3,481	214	1,965
n40w100.004	243	3,923	243	1,888
n40w100.005	223	50,32	223	4,009

Tabla 1.2.Valores objetivos obtenidos mediante el método exacto para n = 40 (TSP)

- **60 nodos:**

	Tucker et al.		Redes	
Archivo	Resultado	Tiempo	Resultado	Tiempo
n60w20.001	303	200,0	303	200,0
n60w20.002	325	200,0	**298**	200,0
n60w20.003	311	200,0	**310**	200,0
n60w20.004	369	200,0	**295**	200,0
n60w20.005	306	200,0	**295**	200,0
n60w40.001	359	200,0	**300**	99,52
n60w40.002	302	51,02	302	3,262
n60w40.003	308	200,0	**306**	3,589
n60w40.004	315	200,0	**291**	200,0
n60w40.005	304	200,0	**297**	50,18

n60w60.001	329	200,0	**327**	200,0
n60w60.002	338	200,0	**314**	5,912
n60w60.003	312	200,0	290	180,5
n60w60.004	333	200,0	**309**	200,0
n60w60.005	346	200,0	**315**	200,0
n60w80.001	354	200,0	**290**	129,7
n60w80.002	309	200,0	309	200,0
n60w80.003	352	200,0	**299**	200,0
n60w80.004	345	200,0	**306**	200,0
n60w80.005	300	200,0	286	200,0
n60w100.001	310	200,0	**271**	200,0
n60w100.002	318	200,0	**296**	200,0
n60w100.003	318	200,0	**280**	200,0
n60w100.004	328	200,0	**301**	200,0
n60w100.005	293	200,0	**278**	4,494

Tabla 1.3.Valores objetivos obtenidos mediante el método exacto para n = 60 (TSP)

- **80 nodos:**

	Tucker et al.		Redes	
Archivo	Resultado	Tiempo	Resultado	Tiempo
n80w20.001	407	200	**333**	200
n80w20.002	362	200	**326**	200
n80w20.003	353	200	**326**	200
n80w20.004	395	200	**331**	200
n80w20.005	336	200	**328**	200
n80w40.001	401	200	**348**	200
n80w40.002	358	200	**327**	200
n80w40.003	373	200	**354**	200
n80w40.004	388	200	**341**	200
n80w40.005	**361**	200	399	200
n80w60.001	304	200	**292**	200
n80w60.002	393	200	**318**	200
n80w60.003	355	200	**325**	200
n80w60.004	411	200	**355**	200
n80w60.005	359	200	**343**	200
n80w80.001	420	200	**325**	200
n80w80.002	362	200	**346**	200
n80w80.003	362	200	**334**	200
n80w80.004	367	200	**327**	200
n80w80.005	380	200	**367**	200

Tabla 1.4.Valores objetivos obtenidos mediante el método exacto para n = 80 (TSP)

- **100 nodos**:

	Tucker et al.		Redes	
Archivo	Resultado	Tiempo	Resultado	Tiempo
n100w20.001	**397**	200	418	200
n100w20.002	432	200	**370**	200
n100w20.003	**408**	200	429	200
n100w20.004	418	200	418	200
n100w20.005	452	200	**416**	200
n100w40.001	438	200	**428**	200
n100w40.002	**400**	200	416	200
n100w40.003	429	200	**369**	200
n100w40.004	**392**	200	450	200
n100w40.005	537	200	**381**	200
n100w60.001	**426**	200	435	200
n100w60.002	484	200	**431**	200
n100w60.003	**490**	200	516	200
n100w60.004	526	200	**387**	200
n100w60.005	505	200	**373**	200

Tabla 1.5.Valores objetivos obtenidos mediante el método exacto para n = 100 (TSP)

Como vemos, a medida que aumenta el número de nodos, encontrar el óptimo se hace más complicado. Incluso para un número relativamente pequeño, como es 40 nodos, el *solver* no consigue encontrar la solución óptima en el tiempo máximo permitido. Por ello, resulta necesario el uso de heurísticas en este tipo de problemas.

En cuanto a los modelos, el modelo auxiliar de redes ofrece mejores soluciones para casi todos los nodos, salvo el caso de n = 100 donde hay un mayor "rivalidad". En el siguiente capítulo, para comprobar la eficacia de las distintas heurísticas cogeremos la mejor solución de las obtenidas entre ambos modelos (marcadas con en negrita) y el tiempo medio requerido por ambos modelos.

1.9. Otras consideraciones

La elección de implementar estos métodos (y los que siguen en el resto de capítulos) en lenguaje Mosel, en lugar de otros conocidos como C, C++ o Java, de los que existe una amplia bibliografía y programas ejemplo para la resolución del TSP, ha sido gracias a la consideración de otra serie de ventajas, a mayores de las comentadas en la Introducción.

Además de su interfaz para modelar problemas de forma sencilla, la facilidad para usar datos externos y su lenguaje "amigable" de alto nivel, tenemos la posibilidad de dibujar soluciones y/o imprimir y/o exportarlas en el formato deseado.

El resultado de estos problemas, aunque quede reflejado en el coste o distancia total, es una ruta. Si hubiésemos empleado estos códigos o programas para una empresa particular; los comerciales, operarios, astronautas o cualquier profesional dentro de un entorno en el que pueda ser aplicado el TSP (véase apartado 1.4), hubiesen agradecido la visualización de la ruta para poder efectuar su trabajo. Por ello, y gracias al procedimiento *draw* [18], Xpress-MP no sólo nos ofrece resultados numéricos, sino que, en caso de trabajar con coordenadas podemos visualizar la ruta de forma gráfica.

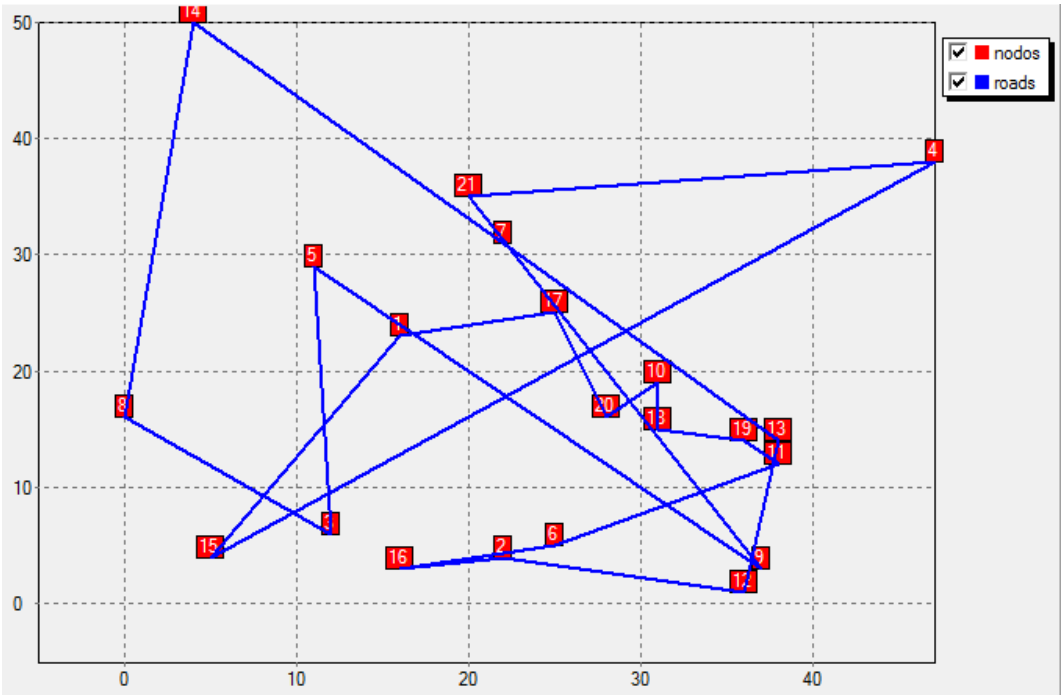

Figura 1.12. Ejemplo gráfico obtenido con "Xpress-MP"

Capítulo 2

Comparativa de diferentes heurísticas dentro de la búsqueda local para la resolución del TSP

2.1. Introducción

El objetivo de este capítulo es realizar la comparativa de cuatro heurísticas pertenecientes a la familia de la búsqueda local: *Hill-Climbing (HI), Simulated Annealing (SA), Threshold Accepting (TA) y Late Acceptance Hill-Climbing (LAHC).* La búsqueda local es un método de mejora heurística para resolver problemas computacionalmente duros o difíciles. Estos algoritmos parten de una solución factible inicial S, y posteriormente tratan de mejorarla. Para ello, se explora el entorno o vecindad de la solución, mediante una operación básica llamada movimiento que, a partir de una solución dada es capaz de proporcionar otras mediante alteraciones de dicha solución. Es decir, se va pasando de forma iterativa (y mientras no se cumpla un determinado criterio de parada), a otras soluciones factibles del entorno, almacenando como óptima la mejor de las soluciones visitadas [14].

Formalmente:

Se considera un problema general de optimización, como el de minimizar una función $f(x)$ sujeto a $x \in X$, donde $f(x)$ puede ser lineal o no lineal, y el conjunto X representa el conjunto de soluciones factibles. Cada solución $x \in X$ tiene asociada un conjunto de soluciones $N(x) \subseteq X$, que denominaremos *entorno* de x. Además, dada una solución $x \in X$, cada solución de su entorno $x' \in N(x)$ puede obtenerse directamente a partir de x mediante una operación llamada *movimiento*.

Por tanto, puede decirse que un procedimiento de búsqueda local parte de una solución x_0, examina su entorno $N(x_0)$ y escoge una nueva solución $x_1 \in N(x_0)$. Dicho de otro modo, realiza un movimiento que aplicado a x_0 da como resultado x_1. Este proceso puede ser aplicado reiteradamente, obteniéndose una trayectoria.

$$x_0 \rightarrow x_1 \rightarrow x_2 \rightarrow \cdots \rightarrow x_n$$

En este proyecto, para generar la solución inicial se ha empleado la heurística del vecino más próximo, el cual empieza seleccionando aleatoriamente la ciudad de inicio y después visitando la ciudad más próxima que no haya sido visitada todavía del conjunto de ciudades o nodos.

Para realizar el paso de una solución factible a otra, usaremos el intercambio 2-opt remplazando dos arcos del recorrido por otros dos diferentes tal como se explicó en el capítulo 1.

Estos métodos se basan, por tanto, en buscar de entre los elementos del entorno de la solución actual aquel que tenga un mejor valor de acuerdo con algún criterio predefinido, moverse a él y repetir la operación hasta que se considere que no es posible hallar una mejor solución, bien porque no haya ningún elemento en el entorno de la solución actual, o bien porque se verifique algún criterio de parada.

Uno de los mayores inconvenientes con los que se enfrentan estas técnicas es la existencia de óptimos locales que no sean absolutos. Si a lo largo de la búsqueda se cae en un óptimo local, en principio la heurística se quedaría "pegada" en ese punto, sin poder escapar del mínimo local. Esto está simbólicamente ilustrado en la Figura 2.1: todas las soluciones en el entorno $N(x_n)$ son peores que x_n aunque más allá de este entorno existe un mínimo global. *Simulated Annealing (SA), Threshold Accepting (TA) y Late Acceptance Hill-Climbing (LAHC)* son procedimientos metaheurísticos de búsqueda local especialmente diseñados para evitar esta situación, permitiendo la aceptación de soluciones peores. En cambio, *Hill-Climbing (HC)* sólo permite movimientos en el entorno de la solución actual que mejoran el valor objetivo

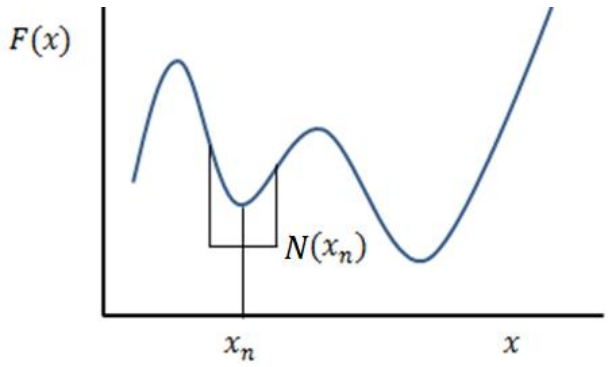

Figura 2.1.Atrapado en un mínimo local

2.2. Hill-Climbing

El algoritmo *Hill-Climbing* (también conocido como método del descenso, *steepest descent* o *best improvement*) es uno de los primeros métodos que fueron estudiados dentro de la familia de los algoritmos de búsqueda local. Dicho algoritmo empieza con una solución inicial considerada como la solución actual en la primera iteración. Después, en cada iteración, se modifica la solución actual usando el cambio 2-opt, en busca de una solución candidata. Esta solución candidata es comparada con la actual. Si es mejor se acepta y se toma como la actual solución para la siguiente iteración; en caso contrario, la solución candidata es rechazada y mantenemos la solución actual para la siguiente iteración. La búsqueda continúa hasta que el número máximo de iteraciones es alcanzado.

Este algoritmo es eficiente en cuanto a tiempo requerido se refiere ya que es muy rápido. Sin embargo, en términos de calidad la solución normalmente queda atrapada en un óptimo local devolviendo una solución que suele quedar lejos del óptimo local. Esto es debido a que no acepta soluciones peores. No obstante, por su simplicidad, y sobre todo en ocasiones en las que el tiempo juega un papel importante, esta heurística sigue siendo muy utilizada en la actualidad.

La figura 2.2 describe el pseudocódigo de este algoritmo [1].

Esquema básico del algoritmo Hill-Climbing

Generar una solución inicial s
Calcular la función de coste inicial $C(s)$
Primera iteración $I = 0$;
While la condición de parada no se satisfaga **do**
  Construir una solución candidata s^*
  Calcular su función de coste $C(s*)$
  if $C(s^*) \leq C(s)$
  Then aceptamos el candidato $(s = s^*)$
  Else rechazamos el candidato $(s = s)$
  End-if
  Incrementamos el número de iteraciones $I = I + 1$
End-do

Figura 2.2. Esquema básico algoritmo Hill-Climbing

2.3. Simulated Annealing *(Recocido simulado)*

2.3.1. Introducción

El algoritmo *Simulated Annealing* (Recocido Simulado) es una de las heurísticas de búsqueda local más estudiadas y conocidas. Kirkpatric, Gellat and Vecci (1983) y, por otra parte y de modo independiente, Cerny (1985) introdujeron los conceptos de recocido dentro del mundo de la optimización combinatoria.

Este algoritmo no busca la mejor solución en el entorno $N(x_n)$ de la actual solución x_n; simplemente genera aleatoriamente una solución x en $N(x_n)$. Si $F(x) \leq F(x_n)$, x pasa a ser la actual solución. En caso contrario, se sigue una de las dos alternativas siguientes: o bien x pasa a ser la solución actual con probabilidad $p(n)$ o bien x_n continua siendo la actual solución con probabilidad 1-$p(n)$. Típicamente, $p(n)$ decrece con el tiempo y con el deterioro de F ($= F(x) - F(x_n)$).

La idea de este algoritmo tiene su origen en la metalurgia y la termodinámica: cuando el hierro fundido es enfriado muy despacio tiende a solidificar en una estructura de energía mínima. Este proceso de recocido es imitado por esta estrategia de búsqueda local; al principio, casi cualquier movimiento es aceptado lo que permite explorar el espacio en el que nos movemos. Después, conforme disminuye la "temperatura", disminuye la probabilidad de aceptar una solución peor que la anterior. Por tanto, la estrategia que seguiremos en el Recocido Simulado será partir de una temperatura "alta" (con la cual permitiremos cambios a soluciones peores en los primeros pasos, cuando aún estamos lejos del óptimo global), y posteriormente ir reduciendo dicha temperatura, disminuyendo la probabilidad de cambios a soluciones peores cuando ya nos hayamos acercado al óptimo buscado [34].

El algoritmo *Simulated Annealing* podría representarse de la siguiente forma:

Esquema básico Simulated Annealing

$T \leftarrow T_0$
Generar una solución inicial x_1 en X;
$F^* \leftarrow F(x_1)$
$x^* \leftarrow x_1$
While la condición de parada no se satisfaga **do**
Generar aleatoriamente un x en el entorno $V(x_n)$ de x_n
if $F(x) \leq F(x_n)$, **then** $x_{n+1} \leftarrow x$
if $F(x) \leq F^*$, **then** $F^* \leftarrow F(x)$ y $x^* \leftarrow x$
else, generamos un número p aleatorio entre [0,1]
end if
if $p \leq p(n)$ **then** $x_{n+1} \leftarrow x$
end if
Se desminuye la temperatura según el programa de enfriamiento
end do

Figura 2.3. Esquema básico algoritmo Simulated Annealing

2.3.2. Decisiones genéricas

Entre las decisiones genéricas que hay que abordar a la hora de implementar este algoritmo tenemos la elección de la probabilidad de aceptación, del programa de enfriamiento y los criterios de parada.

a) Elección de la probabilidad de aceptación $p(n)$

Por analogía con la termodinámica, usualmente se coge la distribución de Boltzamann:

$$p(n) = \exp(-\frac{1}{T(n)}\Delta F_n)$$

Donde $\Delta F_n = F(x) - F(x_n)$ y $T(n)$ es la "temperatura" en el paso n.

b) Elección del programa de enfriamiento

No hay reglas generales para elegir el mejor "programa de enfriamiento" (*Cooling Schedule* en inglés), más bien podría decirse que es un arte [23]. Sin embargo, se ha de tener en cuenta que la temperatura inicial debe ser suficiente alta como para "fundir" el sistema por completo y debe reducirse a su punto de congelación a medida que la búsqueda progresa.

Por tanto, deben especificarse los siguientes parámetros:

- **Temperatura inicial:**

Podríamos decir que, en general, una de las características que debe cumplir toda heurística de búsqueda es la de no ser dependiente de la solución inicial de partida. Esto lo consigue el Recocido Simulado partiendo de una temperatura inicial alta, con lo cual al principio irá erráticamente recorriendo soluciones lejanas de la óptima. Sin embargo, no parece conveniente considerar para T_0 valores fijos independientes del problema. Esta temperatura puede ser estimada mediante la realización de una búsqueda inicial en la que se aceptan todos los aumentos (es decir, el número fijo de iteraciones de Recocido Simulado en el que todas las soluciones son aceptadas) y calculando la máxima diferencia en coste entre dos soluciones del entorno δf. Así, la temperatura inicial viene dada por:

$$T_0 = -\frac{\delta f}{\ln(p)}$$

Donde p es una probabilidad próxima a 1 (0.8-0.9). El cálculo exacto de esta máxima diferencia es bastante lento en muchos casos. Sin embargo, a menudo se pueden dar sencillas estimaciones de su valor.

Otra opción sería la propuesta por los autores de Optimización Heurística y Redes Neuronales [14] los cuales sugieren la siguiente fórmula:

$$T_0 = -\frac{\mu}{\ln(\Phi)} * C(S_{act})$$

Donde se considera que pudiera ser aceptable con un tanto por uno Φ de probabilidad una solución que sea un μ por uno peor que la inicial S_0. Por ejemplo, en el caso de considerar aceptar aproximadamente un $\Phi = 13\%$ de las veces (en la primera iteración) una solución que sea un $\mu = 1\%$ peor que la actual, tendríamos

$$T_0 = -\frac{0.01}{\ln(0.13)} * C(S_{act}) \approx 0.005 * C(S_{act})$$

Podemos usar cualquiera de las anteriores fórmulas. Sin embargo, a efectos prácticos, basta con considerar una temperatura lo suficientemente alta como para aceptar, al principio de la búsqueda, casi libremente las soluciones del entorno.

- Regla de decrecimiento de la temperatura o velocidad de enfriamiento

En este algoritmo la temperatura decrece de forma gradual tal que:

$$T_i > 0, \forall i$$

Y

$$\lim_{i \to \infty} T_i = 0$$

Donde i denota la iteración del algoritmo.

La calidad de las soluciones obtenidas y la velocidad del sistema de enfriamiento suelen ser inversamente proporcionales. Si se disminuye la temperatura lentamente, se obtienen mejores soluciones, pero con un aumento considerable del tiempo de cálculo.

La temperatura T se puede actualizar de diferentes maneras [23]:

- ***Geométrica***: Es el esquema geométrico, la temperatura es actualizada usando la siguiente fórmula:

  $$T_{i+1} = \alpha T_i$$

  Donde $\alpha \in [0,1]$, aunque este rango podría acotarse aún más ya que sus valores típicos se encuentran entre 0.8 y 0.99. Esta es la velocidad de enfriamiento más usada y queda reflejada en la Figura 2.4, donde vemos que empezando en T_0, la temperatura se mantiene constante durante N pasos consecutivos. Después, la temperatura disminuye a través de la multiplicación del factor fijo α. Por tanto, la anterior formula también puede plantearse como sigue:

  $$T(kN) = T_k = \alpha^k * T_0$$

  Lo cual implica la configuración de dos parámetros además de la temperatura inicial que veíamos anteriormente, el ratio de enfriamiento α y el valor de N.

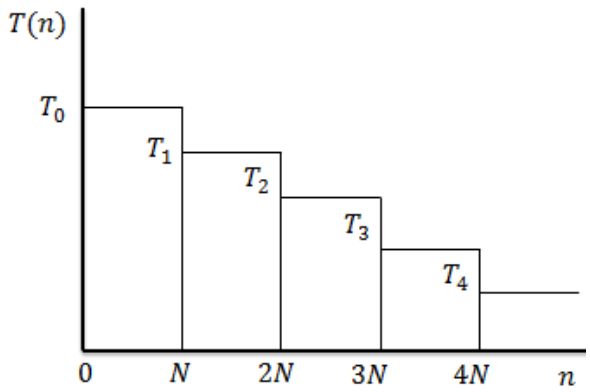

Figura 2.4. Esquema velocidad enfriamiento geométrica

- ***Lineal***: En el esquema lineal, la temperatura T se actualiza como sigue :

$$T_i = T_0 - i\alpha$$

Donde T_i representa la temperatura en la iteración i.

- ***Logarítmica*:** La temperatura en la iteración i es calculada usando la siguiente fórmula:

$$T_i = \frac{T_0}{\log(i)}$$

Este esquema es demasiado lento para ser aplicado en la práctica, pero se ha demostrado que tiene la propiedad de convergir a un óptimo global.

- ***Logarítmica modificada*:** La principal disyuntiva de los esquemas de enfriamiento es el uso de un gran número de iteraciones en pocas temperaturas o un pequeño número de iteraciones en muchas temperaturas. Esquemas logarítmicos modificados tales como

$$T_i = \frac{T_{i-1}}{1 + \alpha T_{i-1}}$$

pueden ser usados cuando α es un parámetro constante. Para cada temperatura sólo se permite una iteración.

c) Elección de los criterios de parada

En algunas implementaciones simples del *Simulated Annealing* el algoritmo termina su ejecución cuando se llega a una temperatura final especificada o bien cuando se llega a un máximo de iteraciones. Nótese que debido a ello necesitamos otros dos parámetros de entrada: temperatura final y número máximo de iteraciones.

Alternativamente o conjuntamente, la búsqueda se puede detener cuando la solución no ha mejorado al menos $\in_1$ % después de K_1 series consecutivas de N pasos. En nuestro caso, pararemos la ejecución si el programa no mejora en K iteraciones. Este valor dependerá del número de nodos que manejemos.

2.3.3. Parámetros recomendados

	n ≤ 20	20< n ≤ 40	40< n ≤ 60	60< n ≤ 80	n > 80
Temp.ini	400	10000	100000	100000	100000
N	80	1000	8000	15000	18000
α	0.995	0.961	0.931	0.916	0.91
K	450000	580000	650000	2000000	2000000
Iter max	950000000				
Tolerancia	0.00000000000000001				

Figura.2.5. Parámetros recomendados algoritmo Simulated Annealing (TSP)

2.4. Threshold Accepting

2.4.1. Introducción

El método *Threshold Accepting* (TA) es una variación del *Simulated Annealing* del que previamente hablábamos, introducido por primera vez por Dueck y Scheuer [15].

TA simplifica el algoritmo del Recocido Simulado dejando fuera el elemento probabilístico usado para aceptar soluciones peores. Para ello, introduce un umbral determinista de forma que una solución peor será aceptada si su diferencia con la solución actual es menor o igual a este umbral. Dentro de los componentes clave del *Threshold Accepting* encontramos la determinación de cómo va ir reduciéndose este umbral durante el transcurso del procedimiento, el criterio de parada, así como los métodos utilizados para crear soluciones iniciales y del entorno.

Suponiendo que X es el conjunto de todas las soluciones factibles del problema, TA comienza con una solución inicial $x_0 \in X$, que puede ser generada al azar, o como en nuestro caso, empleando la heurística del vecino más próximo.

Tras ello, el procedimiento continúa de manera iterativa. En cada iteración, el algoritmo decide si la nueva solución x' es mejor que la solución actual x_i gracias a un umbral (T_i) que define el "máximo empeoramiento" $f(x') - f(x_i)$, aceptable. Si esta diferencia es mayor que el umbral T_i la solución es rechazada, mientras que si es menor la solución actual será sustituida por la nueva. Además si la solución actual es menor que la mayor solución alcanzada hasta ahora x_{best}, entonces ésta será reemplazada por x_i.

Esquema básico algoritmo Threshold Accepting
Generar una solución inicial x_0, $x_{best} = x_0$ Especificar el umbral inicial T_0 **While** la condición de parada no se satisfaga **do** Construir una solución candidata x' **if** $f(x') \leq f(x_i) + T_i$ **Then** aceptamos el candidato ($x_{i+1} = x'$) **Else** rechazamos el candidato ($x_{i+1} = x_i$) **End-if** **If** $f(x_{best}) \leq f(x_i)$ **Then** $x_{best} = x_i$ **End-if** $T_{i+1} = next(Ti)$ Incrementamos el número de iteraciones $I = I + 1$ **End-do** Salida: x_{best}

Figura.2.6. Esquema básico algoritmo Threshold Accepting

Wang y Ting [41] sostienen que la principal ventaja de este método es su simplicidad conceptual y sus excelentes resultados en diferentes problemas de optimización combinatoria en el cual se incluye el TSP. Por el contrario, otros autores afirman que este método no asegura el mínimo global.

2.4.2. Decisiones genéricas

a) Determinación del umbral

En el caso del algoritmo *Simulated Annealing* hemos visto la importancia de proporcionar un buen y cuidadosamente elegido programa de recocido o programa de enfriamiento. Por lo tanto, es obvia la importancia de analizar la sensibilidad de este algoritmo en relación a la elección de la secuencia del umbral de aceptación.

Dueck y Scheueren [15], los padres de este algoritmo, operaban con la siguiente secuencia en las primeras versiones que utilizaron para resolver un problema de 442 nodos:

> [0.13, 0.12, 0.11, 0.10, 0.095, 0.09, 0.085, 0.08, 0.075, 0.075, 0.075, 0.07, 0.07, 0.07, 0.065, 0.065, 0.065, 0.065, 0.06, 0.06, 0.055, 0.055, 0.05, 0.05, 0.05, 0.04, 0.04, 0.03, 0.02, 0]

Intuitivamente, tenían la sensación de que este umbral sería el mejor. Sin embargo, para comprobar su eficacia realizaron algunos experimentos para encontrar mejores frecuencias de umbral. Dichos experimentos llevaron a determinar que esta secuencia era sensiblemente mejor al resto de las probadas y lo que es más importante, suscitaron la cierta insensibilidad del algoritmo con respecto a este umbral.

2.4.3. Modificación del Threshold Accepting original

Para hacerlo más eficaz se introdujo un nuevo parámetro R, siendo el número de veces para el cual se va a repetir el algoritmo anterior. Además, con esta metodología se propuso como secuencia óptima:

[0.099, 0.098, 0.097,…, 0.003, 0.002, 0.001, 0]

La razón de cambiar esta secuencia es que en esta versión hay cierto peligro de "circular" sin ninguna mejora si el algoritmo se ejecuta un largo tiempo con el mismo umbral. Por lo tanto, es preferible ejecutar el algoritmo con un mayor número de umbrales, disminuyendo el número de iteraciones dentro de cada

umbral. Sin embargo, una vez más, los experimentos llevados a cabo por Dueck y Scheueren suscitaron la insensibilidad del algoritmo a la secuencia del umbral elegido. Por lo tanto, la elección de la misma es trivial, dentro de unos límites.

Los experimentos que realizaban estos investigadores para la elección del umbral óptimo estaban centrados en un problema particular de n = 442. Por lo que la secuencia recomendada, tanto para esta versión como para la anterior, no es necesariamente la mejor. Aunque hemos comentado anteriormente que la elección de la secuencia es relativamente trivial, el valor máximo de ésta ha de ser aproximadamente menor que 0,5.

Aclarado el problema del umbral, hemos de decir que los resultados computacionales obtenidos gracias a esta última propuesta fueron notablemente mejores, incluso con valores bajos de R. Por ello, los resultados que presentaremos más adelante serán los obtenidos con esta versión del algoritmo.

Esquema básico del algoritmo Threshold Accepting modificado
Fijar un número de re-arranques R Para todo R **do** Generar una solución inicial x_0, $x_{best} = x_0$ Especificar el umbral inicial T_0 **While** la condición de parada no se satisfaga **do** Construir una solución candidata x' **if** $f(x') \leq f(x_i) + T_i$ **Then** aceptamos el candidato ($x_{i+1} = x'$) **Else** rechazamos el candidato ($x_{i+1} = x_i$) **End-if** **If** $f(x_{best}) \leq f(x_i)$ **Then** $x_{best} = x_i$ **End-if** $T_{i+1} = next(Ti)$ Incrementamos el número de iteraciones $I = I + 1$ **End-do** **End-do** Salida: x_{best}

Figura.2.7. Esquema básico algoritmo Threshold Accepting modificado

2.4.4. Parámetros recomendados

Umbral	[0.2, 0.0.198, 0.196, 0.194, … , 0.004, 0.002, 0]
N: Número fijo de iteraciones para cada valor del umbral	100
R: Número de re-arranques aleatorios	90

Figura.2.8. Parámetros recomendados algoritmo Threshold Accepting

2.5. Late Acceptance Hill-Climbing

2.5.1. Introducción

El algoritmo *Late Acceptance Hill-Climbing* (LAHC), versión mejorada del algoritmo *Hill-Climbing* previamente explicado, fue propuesto por Burke y Bykov (2008), quienes a la hora de implementarlo perseguían los tres siguientes objetivos. Primero, encontrar un algoritmo de búsqueda local que no requiriera de programa de enfriamiento. El segundo objetivo era usar eficazmente la información recogida en las iteraciones previas. Por último, pretendían emplear un nuevo mecanismo de aceptación que no fuera complicado (al menos, casi tan simple como el del algoritmo *Hill-Climbing*).

Así, la principal idea de este algoritmo es retrasar la comparación de la solución actual con otra del entorno; en lugar de compararla con la inmediatamente anterior, se compara con la solución obtenida varias iteraciones antes. De ahí su nombre *Late Acceptance Hill-Climbing*, ya que sigue la misma idea que el *Hill-Climbing* pero "retrasa" la comparación.

De un modo similar a las demás heurísticas tratadas en este capítulo (como HC, SA y TA), LAHC parte de una solución inicial y en cada iteración se evalúa a un nuevo candidato para aceptarlo o rechazarlo. Para emplear su regla de aceptación, LAHC mantiene una lista (de longitud fija) con los valores de la función objetivo obtenidos con anterioridad. El candidato actual es comparado con el último elemento de la lista y si no es peor, se acepta. Después del procedimiento de aceptación, el valor objetivo de la nueva solución se inserta al inicio de la lista y el último elemento de la misma es eliminado del final. Nótese que el valor insertado es igual que el valor de la solución candidata sólo en el caso que haya sido aceptado, en caso de haberlo rechazado éste sería igual al valor previo.

Sin embargo, se quiso hacer el procesamiento del LAHC independiente de la longitud de la lista eliminando así el "desplazamiento" de ésta en cada iteración. Para evitarlo Edmund K. Burke y Yuri Bykov propusieron las siguientes mejoras [8]:

La primera mejora propuesta se trata de realizar el desplazamiento "virtual" de la lista (ver figura 2.7). Ahora los elementos de la lista son inmóviles y la lista aparece como un array F_a de longitud Lf_a ($F_a = \{f_o, f_1, f_2, \dots, f_{LFa-1}\}$. Su principio virtual v, en la iteración ith, es calculada como:

$$v = i \ mod \ Lf_a$$

donde "mod" representa el resto de la división entre i y Lf_a . En cada iteración, el valor de f_v es comparado con el coste de la solución candidata y después de aceptarlo o rechazarlo, el nuevo valor del coste actual es asignado a f_v.

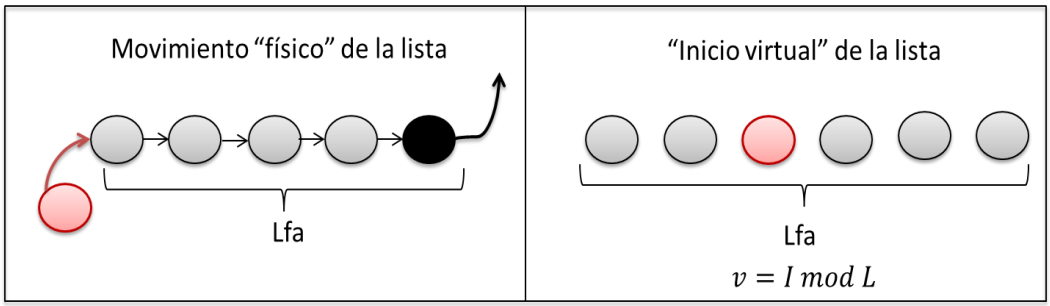

Figura.2.9. "Inicio virtual" de la lista

La longitud Lf_a es el único parámetro de entrada de este algoritmo (a excepción de los relativos al criterio de parada). La efectividad del LAHC no se ve afectada por los valores iniciales del array. Al principio de la búsqueda, los valores iniciales de la lista pueden contener cualquier valor tomado de forma arbitraria.

Nótese que LAHC emplea un criterio de aceptación tipo "greedy" (rechaza a todos los peores candidatos) sólo en el sentido de esta comparación "retrasada" o tardía. Sin embargo, si consideramos una solución y su candidato inmediato, LAHC (de un modo similar a SA y TA) permite aceptar soluciones peores. Teniendo en cuenta que la aceptación de peores soluciones por lo general aumenta la eficacia de cualquier procedimiento de búsqueda, cabe esperar que LAHC obtenga un mejor rendimiento que el ambicioso *Hill-Climbing*. Por otro lado, puede haber situaciones donde el coste actual sea peor que el valor de la lista. Aquí (basándonos en idea inicial de LAHC), incluso un movimiento que no empeore la solución puede ser rechazado. Tal comportamiento no es deseable en cualquier búsqueda computacional. Para ser coherente con ello, se propone una segunda mejora de la idea inicial: utilizar una regla de "aceptación tardía" no solo para los movimientos de empeoramiento, sino también para aceptar todas las soluciones que no empeoren. Por lo tanto, la condición de aceptación final que vamos a emplear en cada iteración se puede expresar con la siguiente fórmula:

$$C_i^* \leq C_{i-Lfa} \; o \; C_i^* \leq C_{i-1}$$

Esquema básico algoritmo Late Acceptance Hill-Climbing
Generar una solución inicial s Calcular la función de coste inicial $C(s)$ Especificar Lfa Para todo $k \in \{0 \dots Lfa-1\}$ $f_k = C(s)$ Primera iteración $I = 0$; **While** la condición de parada no se satisfaga **do** Construir una solución candidata s^* Calcular su función de costes $C(s*)$ $v = I \ mod \ Lfa$ **if** $C(s^*) \leq f_v$ ó $C(s^*) \leq C(s)$ **Then** aceptamos el candidato $(s = s^*)$ **Else** rechazamos el candidato $(s = s)$ **End-if** Insertamos el coste actual en el array $f_v = C(s)$ Incrementamos el número de iteraciones $I = I + 1$ **End-do**

Figura.2.10. Esquema básico algoritmo Late Acceptance Hill-Climbing

2.5.2. Decisiones genéricas

a) Longitud Lfa

Como ya hemos comentado, una de las ventajas de este algoritmo es que sólo es necesario configurar un parámetro: la longitud de Lfa. Edmund K.Burke y Yuri Bykov, llevaron a cabo una serie de experimentos para determinar cómo afectaba esta longitud al rendimiento del algoritmo.

En una primera serie de experimentos, se investigaron las propiedades básicas de LAHC utilizando diagramas tiempo - coste. Tal diagrama ilustra cómo decrece la función de costes a lo largo del proceso de búsqueda. Para ello, el algoritmo se ejecuta varias veces mientras se va variando Lfa. Durante la búsqueda, el coste actual se registró para cada segundo. Todos estos puntos se representan en la Figura 2.11, donde se han tomado Lfa = 1 (que sería equivalente al algoritmo HC), 5000, 15000 y 30000. Cabe señalar que los diagramas de tiempo-coste que realizaron para otros casos de referencia presentaron curvas similares a estas, lo que demuestra que el comportamiento de LAHC es bastante estable.

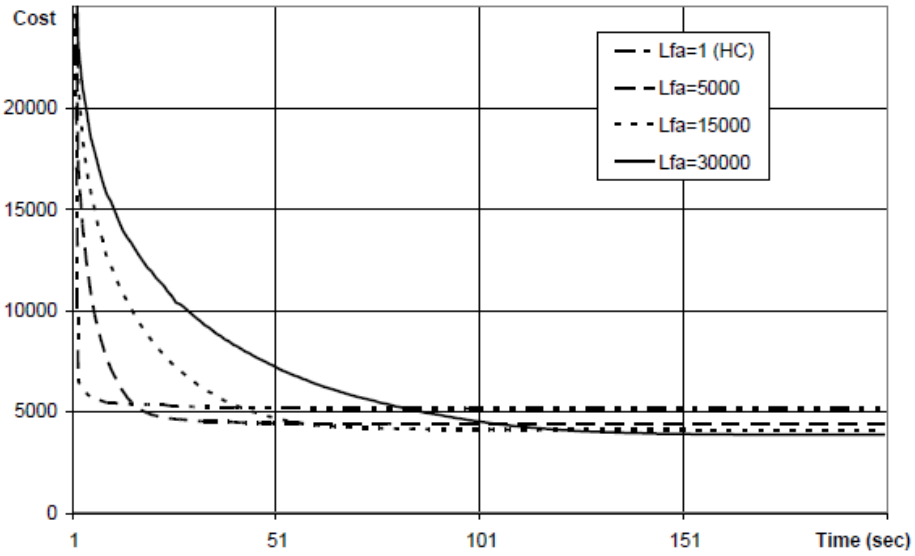

Figura.2.11. Diagrama coste-tiempo con diferentes Lfa

En esta figura, todos los diagramas se inician desde la misma solución (coste~ 25.000). Como vemos, el HC mejora el coste casi de inmediato. El LAHC con Lfa = 5.000 mejora el costo más lentamente; con Lfa = 15.000 aún más lentamente y Lfa = 30.000 es el más lento de los cuatro. Esta observación sugiere una de las principales propiedades de LAHC: cuanto mayor sea Lfa, más lento es el descenso de costes. Este comportamiento se ha observado en todos los experimentos realizados con este algoritmo.

Para explicar los beneficios de incrementar Lfa recurrimos a la Figura 2.12, que se trata de una parte de la Figura 2.11 con mayor detalle.

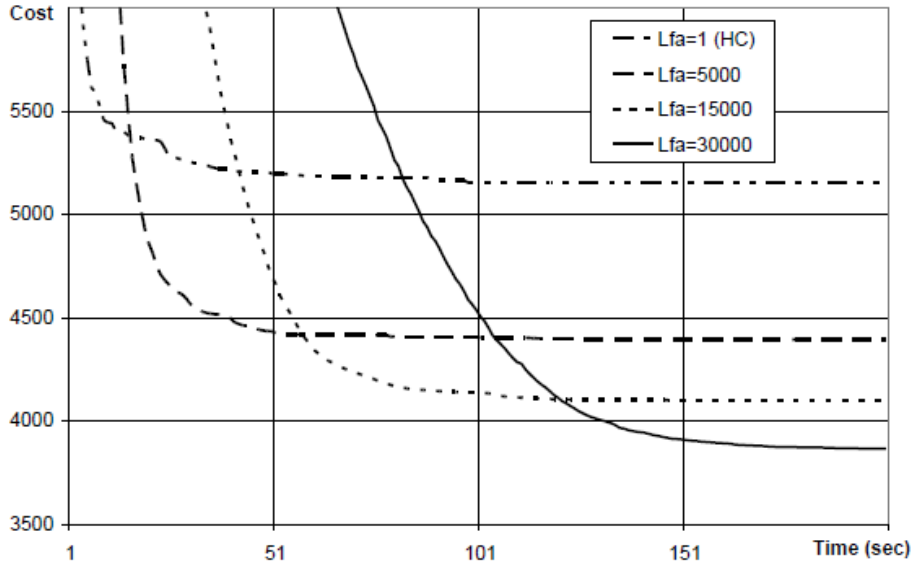

Figura.2.12. Detalle diagrama coste-tiempo

En primer lugar, los diagramas de las Figuras 2.11 y 2.12 demuestran una segunda propiedad importante del LAHC: su convergencia es bastante clara. Es decir, después de alcanzar un cierto valor, la reducción del coste se reduce drásticamente y el diagrama se hace plano. Se supone que después de converger, es muy difícil encontrar una mejora adicional, o al menos una que sea significativa. Cabe señalar que una vez más, esto no ha sido sólo observado en este ejemplo concreto, sino en una gran variedad de ellos.

En segundo lugar, la Figura 2.12 muestra una tendencia interesante: aquellas curvas que descienden de forma más lenta tienden a converger a un mejor resultado. Por tanto, podemos concluir que un aumento de Lfa puede ayudar a obtener mejores resultados.

b) Criterio de parada

La investigación de los diagramas de tiempo-coste en la figura 2.12 también puede sugerir las condiciones de parada adecuadas para este algoritmo.

Una primera variante, sería detener la búsqueda al alcanzar alguna solución objetivo. Esta condición de parada es común para problemas en los que la solución objetivo se conoce de antemano. Por ejemplo, cuando tenemos que encontrar una solución de coste cero. Sin embargo, en el TSP el coste objetivo no es tan claro, dado que se busca minimizar, no alcanzar el 0.

La segunda variante sugiere detener la búsqueda después de algún tiempo dado (o un número dado de iteraciones). Esta condición de parada es habitual para aquellas técnicas de búsqueda, que no muestran una clara convergencia, y que por tanto siguen mejorado de forma constante con el tiempo. Por el contrario, si tenemos un algoritmo con un criterio de convergencia claro, entonces la decisión de darlo por terminado en un punto dado en el tiempo podría llegar a ser ineficaz. En este caso, el criterio de parada debe aplicarse con especial cuidado a fin de invertir el tiempo disponible de la manera más eficaz posible. Volvamos al ejemplo de la Figura 2.12; si decidimos detener la búsqueda después de 50 segundos, el mejor resultado se logrará mediante la búsqueda con Lfa = 5.000. HC Greedy converge antes, pero con un resultado peor; LAHC con el mayor Lfa todavía no alcanza la convergencia y su resultado en este punto también es peor. El mismo razonamiento puede revelar que si la búsqueda tiene una duración de 100 segundos, el mejor valor de Lfa es 15.000 y para una búsqueda de 150 segundos, el mejor Lfa = 30.000.

El análisis de estas dos variantes de condición de parada demuestra que, en ambos casos, el enfoque más eficaz para terminar la búsqueda es en el momento de la convergencia. Si terminamos antes de que la solución converja, no estamos aprovechando toda la potencia de este método. Si dejamos que el ciclo de búsqueda siga ejecutándose tras la convergencia, entonces estamos perdiendo tiempo que podría haber sido aprovechado con un Lfa más grande.

Por lo tanto, la tercera variante de la condición de parada para LAHC es terminar lo más cerca posible del punto de convergencia. En la práctica, se detecta la convergencia cuando la solución actual no mejora significativamente durante un cierto tiempo. Este tiempo puede ser fijo o puede ser calculado como un porcentaje del tiempo total de búsqueda. En nuestro caso, y como ya se ha explicado en métodos anteriores, el procedimiento de búsqueda se termina después de pasar un número de iteraciones consecutivas sin mejorar (K). Esta cantidad de iteraciones varía en función del número de nodos.

2.5.3. Parámetros recomendados

	$n \leq 20$	$20 < n \leq 40$	$40 < n \leq 60$	$60 < n \leq 80$	$n > 80$
K	200000	400000	800000	1600000	3200000
Iter max	200000000				
Lfa	8000				

Figura.2.13. Parámetros recomendados algoritmo Late Acceptance Hill-Climbing (TSP)

2.6. Análisis de resultados obtenidos

Los anteriores algoritmos, y como ya comentábamos al principio, se han implementado en lenguaje Mosel y ejecutados en un ordenador Intel CORE con velocidad 3.60GHz y una memoria instalada (RAM) de 16GB. Cada algoritmo se ha probado para un total de 110 instancias propuestas por Dumas et al. [16], las cuales abarcan problemas de 20, 40, 60, 80 y 100 nodos. Además, para comprobar la robustez de dichas heurísticas, cada instancia se ha ejecutado 5 veces.

Para interpretar con precisión el rendimiento de estos algoritmos, para cada grupo (20, 40, 60, 80 y 100 nodos) se presentarán dos tablas.

En la primera de ellas, para cada instancia se recogerá la mejor solución obtenida y la solución media (de las 5 ejecuciones). Además, estos datos se compararán entre sí y con los resultados obtenidos con los resultados del método exacto. De forma que al final de dicha tabla, se presentará el porcentaje medio de mejora (+)

o empeoramiento (-) del método con respecto a la solución obtenida con el método exacto. Éste es calculado como:

$$\Delta = \frac{(solución\ método\ exacto - solución\ media\ heurística)}{solución\ \ media\ heurística} * 100$$

En la segunda tabla, se recogerá el tiempo medio empleado por la CPU para obtener el resultado (en segundos) y la desviación típica. Para ésta última, nos basaremos en la siguiente fórmula: $\sigma = \sqrt{\sum_{i=1}^{5}(x_i - \bar{x})^2/n}$, donde x_i es el valor objetivo obtenido en la ejecución i para una instancia dada, $\bar{x}$ es la media del valor objetivo para dicha instancia y n el número de ejecuciones llevada a cabo; en nuestro caso 5.

Lo que se pretende demostrar es cuál de éstos podría ser descrito como un "buen" algoritmo, entendiendo por ello que cumpla las siguientes propiedades:

- **Eficiente**. Un esfuerzo computacional realista para obtener la solución.
- **Eficaz.** La solución debe de estar, en promedio, cerca del óptimo.
- **Robusto.** La probabilidad de obtener una mala solución (lejos del óptimo) debe ser baja.

En cada tabla se destacarán en negrita aquellos valores que sean superiores al resto de las heurísticas. Además, si se supera el resultado obtenido por el método exacto pondremos junto con el valor un asterisco *.

- **20 nodos:**

- Valores objetivos obtenidos

	M. exacto	**HC**		**SA**		**TA**		**LAHC**	
Archivo	Coste	Coste medio	Mejor valor	Coste medio	Mejor valor	Coste medio	Mejor valor	Coste medio	Mejor valor
n20w20.001	198	205,2	198	198	198	198	198	198	198
n20w20.002	174	176,8	174	174	174	175	174	174	174
n20w20.003	213	215,4	214	213,2	213	213	213	213	213
n20w20.004	189	199,2	189	189	189	190	189	189	189
n20w20.005	193	196	194	193	193	193,8	193	193	193
n20w40.001	157	157	157	157	157	157	157	157	157
n20w40.002	189	191,8	189	189	189	190,4	189	189	189
n20w40.003	196	198	196	196	196	196	196	196	196
n20w40.004	180	187,8	183	180	180	180,4	180	180	180
n20w40.005	175	175	175	175	175	175	175	175	175
n20w60.001	183	184,6	184	183	183	183,8	183	183	183

n20w60.002	164	164	164	164	164	164,2	164	164	164
n20w60.003	191	200	198	191	191	191,6	191	191	191
n20w60.004	157	161	157	157	157	158,6	157	157	157
n20w60.005	193	191	194	193	193	193,6	193	193	193
n20w80.001	180	183,6	180	180	180	180	180	180	180
n20w80.002	188	188,4	188	188	188	188,4	188	188	188
n20w80.003	188	197,4	189	188	188	188	188	188	188
n20w80.004	189	194	192	189,2	189	190,6	189	**189**	189
n20w80.005	158	158	158	158	158	158	158	158	158
n20w100.001	191	192,8	191	191	191	191,6	191	191	191
n20w100.002	179	180,8	179	179	179	181,2	179	179	179
n20w100.003	188	190	190	**188**	188	189,6	188	188,8	188
n20w100.004	215	218,2	215	215	215	215	215	215	215
n20w100.005	201	207,6	206	201	201	201,6	201	201	201
Δ(%)		-1.840		**-0.007**		-0.3351		-0.0167	

Tabla 2.1.Comparativa de resultados obtenidos por las diferentes heurísticas para n =20 (TSP)

- Tiempo y desviación

	M. exacto	**HC**		**SA**		**TA**		**LAHC**	
Archivo	Tiempo	Tiemp. medio	σ	Tiemp. medio	σ	Tiemp. medio	σ	Tiemp. medio	σ
n20w20.001	0,133	0,001	6,735	2,118	0,000	3,890	0,000	0,908	0,000
n20w20.002	0,562	0,000	4,665	2,050	0,000	3,894	2,000	1,184	0,000
n20w20.003	1,741	0,000	1,200	2,084	0,400	3,902	0,000	0,89	0,000
n20w20.004	1,139	0,003	8,886	2,126	0,000	3,898	2,000	1,128	0,000
n20w20.005	1,412	0,000	3,033	1,884	0,000	3,896	0,400	1,02	0,000
n20w40.001	0,258	0,000	0,000	2,054	0,000	3,888	0,000	0,86	0,000
n20w40.002	0,289	0,000	4,261	2,102	0,000	3,972	1,744	0,98	0,000
n20w40.003	0,257	0,003	2,449	2,062	0,000	3,898	0,000	0,822	0,000
n20w40.004	1,013	0,000	5,115	2,092	0,000	3,898	0,490	1,26	0,000
n20w40.005	0,321	0,000	0,000	2,018	0,000	3,898	0,000	0,756	0,000
n20w60.001	0,578	0,003	1,200	2,064	0,000	3,898	1,600	1,044	0,000
n20w60.002	0,391	0,000	0,000	2,076	0,000	3,906	0,400	0,772	0,000
n20w60.003	0,841	0,000	1,265	2,034	0,000	3,912	1,200	1,044	0,000
n20w60.004	0,343	0,000	2,000	2,086	0,000	3,890	1,356	1,11	0,000
n20w60.005	0,430	0,003	1,789	2,098	0,000	3,912	0,490	0,848	0,000
n20w80.001	0,273	0,003	2,939	2,054	0,000	3,900	0,000	0,896	0,000
n20w80.002	0,796	0,000	0,800	2,084	0,000	3,888	0,800	0,848	0,000
n20w80.003	0,414	0,000	16,305	2,102	0,000	3,890	0,000	1,18	0,000
n20w80.004	0,969	0,003	3,033	2,104	0,400	3,894	2,059	0,922	0,000
n20w80.005	0,398	0,003	0,000	2,092	0,000	3,904	0,000	1,194	0,000
n20w100.001	0,242	0,003	2,227	2,080	0,000	3,896	0,800	1,502	0,000
n20w100.002	0,406	0,003	2,713	2,044	0,000	3,890	2,561	0,944	0,000
n20w100.003	0,608	0,003	0,000	2,042	0,000	3,902	1,020	0,85	0,980

n20w100.004	0,460	0,000	2,638	2,094	0,000	3,900	0,000	0,886	0,000
n20w100.005	0,406	0,003	1,960	2,096	0,000	3,896	1,200	1,12	0,000
Tiemp. medio	0,587	**0,0014***		2,0695		3,9005		0,9987	
Desviación (σ)	-	3,0081		**0,0321**		0,8048		0,0390	

Tabla 2.2.Comparativa de tiempo y desviación obtenidos por las diferentes heurísticas para n =20 (TSP)

Como cabía esperar, el algoritmo *Hill-Climbing* es el más rápido, pero al no aceptar soluciones peores en su búsqueda hace que suela quedar atrapado en mínimos locales, por lo que los resultados varían de una ejecución a otra, lo que queda reflejado en una desviación típica notoriamente superior al resto. Le seguiría la heurística *Threshold Accepting*, que aunque para todas las instancias alcanza el óptimo al menos una vez, no es un método que asegure la convergencia al óptimo en todos los casos. Aunque la diferencia entre métodos es mínima para este tamaño de problema, parece que el algoritmo de *Simulated Annealing* y *Late Acceptance Hill-Climbing* (en lo que sigue LAHC) serán los que mejores soluciones reporten. Para este caso particular de 20 nodos, mientras que LAHC es más rápido, también es un poco menos robusto que el *Simulated Annealing* (en lo que sigue SA).

- **40 nodos:**

- Valores objetivos obtenidos

	M. exacto	**HC**		**SA**		**TA**		**LAHC**	
Archivo	Coste	Coste medio	Mejor valor	Coste medio	Mejor valor	Coste medio	Mejor valor	Coste medio	Mejor valor
n40w20.001	259	279,8	279	259	259	264	259	259	259
n40w20.002	243	245,4	245	243,4	243	245,8	243	244,6	243
n40w20.003	263	270	263	263	263	263,6	263	263	263
n40w20.004	228	234,2	228	228	228	230,6	228	228	228
n40w20.005	247	251,4	247	247	247	252	248	247	247
n40w40.001	251	260,2	252	251	251	257,8	255	251	251
n40w40.002	232	240	232	232,2	232	237,4	235	**232**	232
n40w40.003	257	261,8	259	257	257	260,4	257	257	257
n40w40.004	244	258,6	251	244,4	244	254,4	251	**244**	244
n40w40.005	241	250,6	246	241	241	245,8	244	241	241
n40w60.001	232	245	233	232,4	232	237,2	234	**232**	232
n40w60.002	256	264,6	258	**256**	256	258,8	256	258,2	256
n40w60.003	225	233,6	225	225	225	229,4	225	225	225
n40w60.004	228	235,4	232	**228,2**	228	233,8	230	229	228
n40w60.005	224	228,4	225	224,4	224	227,2	227	224,4	224

n40w80.001	257	276	264	257,2	257	263,8	260	**257**	257
n40w80.002	254	266	259	**254**	254	257,8	254	255,6	254
n40w80.003	258	262,8	260	258,8	258	265,4	260	258,8	258
n40w80.004	250	262,4	250	**250**	250	254	250	250,6	250
n40w80.005	253	257,2	253	253,6	253	255	253	253,6	253
n40w100.001	240	247	245	**240**	240	244,6	243	240,6	240
n40w100.002	255	263,6	257	**255,4**	255	259,2	258	256	255
n40w100.003	214	214	214	214	214	215,2	214	214	214
n40w100.004	243	261,4	252	243,4	243	249	243	**243,2**	243
n40w100.005	223	237,8	233	223	223	228,4	225	223	223
Δ(%)		-3,593		-0,071		-1,829		-0,173	

Tabla 2.3.Comparativa de resultados obtenidos por las diferentes heurísticas para n =40 (TSP)

- Tiempo y desviación

	M. exacto	**HC**		**SA**		**TA**		**LAHC**	
Archivo	Tiempo	Tiemp. medio	σ	Tiemp. medio	σ	Tiemp. medio	σ	Tiemp. medio	σ
n40w20.001	111,8	0,020	0,401	5,732	0,000	7,214	5,514	5,100	0,000
n40w20.002	25,91	0,010	0,492	5,686	0,490	7,192	1,939	5,334	1,625
n40w20.003	1,498	0,010	10,85	5,750	0,000	7,336	0,800	7,556	0,000
n40w20.004	107,4	0,010	5,741	5,856	0,000	7,258	2,245	7,266	0,000
n40w20.005	16,85	0,010	3,611	5,988	0,000	7,222	3,464	6,536	0,000
n40w40.001	2,528	0,010	6,828	5,342	0,000	7,202	3,059	5,836	0,000
n40w40.002	7,232	0,020	4,154	6,074	0,400	7,272	2,245	6,038	0,000
n40w40.003	102,2	0,010	4,625	5,838	0,000	7,320	2,417	6,862	0,000
n40w40.004	3,425	0,010	5,082	5,886	0,490	7,230	1,744	6,548	0,000
n40w40.005	2,310	0,040	2,420	5,844	0,000	7,292	2,227	5,848	0,400
n40w60.001	9,603	0,010	7,724	5,872	0,490	7,298	2,561	6,172	0,000
n40w60.002	1,818	0,010	7,062	5,848	0,000	7,428	2,926	6,468	4,400
n40w60.003	100,4	0,010	7,502	5,696	0,000	7,664	3,929	5,326	0,000
n40w60.004	101,9	0,010	4,361	5,874	0,400	7,350	2,315	8,704	1,265
n40w60.005	101,4	0,010	3,671	5,622	0,490	7,336	0,400	6,764	0,490
n40w80.001	2,122	0,010	9,425	5,942	0,400	7,198	3,816	6,434	0,000
n40w80.002	5,913	0,010	4,002	5,864	0,000	7,282	4,490	7,640	3,200
n40w80.003	101,8	0,010	1,602	6,524	0,980	7,476	2,939	5,384	1,600
n40w80.004	21,92	0,010	8,871	5,826	0,000	7,242	4,050	4,900	1,200
n40w80.005	101,6	0,010	3,196	5,804	0,800	7,324	1,673	7,710	0,800
n40w100.001	1,607	0,010	1,265	5,884	0,000	7,212	1,020	5,670	0,800
n40w100.002	71,18	0,010	3,385	5,922	0,490	7,196	1,470	6,676	1,265
n40w100.003	2,723	0,010	0,002	6,436	0,000	7,200	1,470	7,388	0,000
n40w100.004	2,906	0,010	6,742	5,914	0,490	7,342	4,427	7,358	0,400
n40w100.005	27,15	0,010	3,601	5,834	0,000	7,274	3,072	7,592	0,000
Tiemp. medio	41,402	**0,0109***		5,8743		7,2944		6,5244	
Desviación (σ)	-	4,6621		**0,2367**		2,6484		0,6977	

Tabla 2.4. Comparativa de tiempo y desviación obtenidos por las diferentes heurísticas para n =40 (TSP)

De nuevo vemos que tanto SA como LAHC tienen un rendimiento muy parecido y superior a *Hill-Climbing* y *Threshold Acceptance*. Aunque ahora es SA quien necesita menos tiempo para alcanzar la solución y además presenta menor desviación típica que LAHC.

- **60 nodos:**

- Valores objetivos obtenidos

	M. exacto	**HC**		**SA**		**TA**		**LAHC**	
Archivo	Coste	Coste medio	Mejor valor	Coste medio	Mejor valor	Coste medio	Mejor valor	Coste medio	Mejor valor
n60w20.001	303	314,8	314	307,2	303	312,8	307	**303,6**	303
n60w20.002	298	309,4	304	299,6	299	309,8	305	**298,4**	**298**
n60w20.003	310	319	311	312,4	310	320,4	315	**310,6**	310
n60w20.004	295	303,4	297	297,6	296	308,6	305	**295,6**	**295**
n60w20.005	295	306,8	303	**296,8**	**295**	311,8	306	297	296
n60w40.001	300	315	304	305	303	313,4	306	**303**	**300**
n60w40.002	302	307,6	304	305,2	302	313,2	311	**302**	302
n60w40.003	306	318,6	307	307,2	306	313	310	**306,6**	306
n60w40.004	291	305,6	298	297,2	296	307,4	302	**293,6**	**291**
n60w40.005	297	310	305	302	298	305	297	**301,8**	297
n60w60.001	327	341,8	339	331,2	330	338	334	**328,6**	**327**
n60w60.002	314	326,4	318	318	315	325,2	319	**315,2**	**314**
n60w60.003	290	305,4	300	292,2	291	294,8	290	**291,4**	290
n60w60.004	309	312,6	311	312,6	310	322,6	319	314	**309**
n60w60.005	315	322	321	323,8	322	333,2	330	**320**	**315**
n60w80.001	290	297,2	293	292*	**290**	299,6	297	294,6	292
n60w80.002	309	325,6	318	311,6	310	316,2	310	**309,4**	**309**
n60w80.003	299	305,6	302	303,6	301	311	304	**301,8**	**299**
n60w80.004	306	322,6	316	309,2	308	318,8	315	**306**	**306**
n60w80.005	286	299,4	295	287,8*	287	297	287	288,4	**286**
n60w100.001	271	280,4	271	274,8	271	279,2	273	**270,6***	**270***
n60w100.002	296	309,4	304	302,2	301	308,4	306	**299,2**	**296**
n60w100.003	280	291,6	285	282,6	280	293,4	289	**281,6**	280
n60w100.004	301	313,2	305	305,6	302	316,4	309	**301,2**	**301**
n60w100.005	278	283,4	279	279,0	279	282,6	279	**278,8**	**278**
Δ(%)		-3.5855		-1.1612		-3.6444		**-0.5944**	

Tabla 2.5.Comparativa de resultados obtenidos por las diferentes heurísticas para n =60 (TSP)

- Tiempo y desviación

	M. exacto	**HC**		**SA**		**TA**		**LAHC**	
Archivo	Tiempo	Tiemp. medio	σ	Tiemp. medio	σ	Tiemp. medio	σ	Tiemp. medio	σ
n60w20.001	200,0	0,015	0,748	13,054	2,713	10,898	3,187	22,384	0,800
n60w20.002	200,0	0,022	3,007	13,158	0,490	10,642	3,868	21,692	0,800
n60w20.003	200,0	0,022	4,980	13,006	2,245	10,498	3,611	24,302	0,800
n60w20.004	200,0	0,019	5,463	12,958	1,020	10,486	2,577	22,408	0,800
n60w20.005	200,0	0,025	2,561	12,908	1,470	10,490	4,118	21,226	0,894
n60w40.001	99,53	0,019	8,532	12,978	2,280	10,490	5,200	18,366	2,530
n60w40.002	3,262	0,031	4,409	11,626	4,534	10,500	1,939	24,642	0,000
n60w40.003	101,5	0,031	6,119	12,936	0,980	10,500	2,449	19,724	1,200
n60w40.004	200,0	0,028	5,004	13,082	1,470	10,492	3,499	22,028	1,855
n60w40.005	125,2	0,031	4,195	13,056	3,795	10,540	5,177	19,688	2,857
n60w60.001	200,0	0,025	2,713	12,964	1,166	10,666	3,162	24,608	1,020
n60w60.002	103,3	0,031	5,083	13,016	1,789	10,520	5,418	20,928	1,166
n60w60.003	200,0	0,031	3,980	12,918	0,748	10,490	3,763	27,322	1,744
n60w60.004	200,0	0,021	1,960	13,422	3,072	10,652	2,871	20,294	4,290
n60w60.005	200,0	0,025	0,632	13,030	1,720	10,502	3,429	22,150	4,858
n60w80.001	164,8	0,025	4,490	12,906	1,897	10,548	1,855	20,162	1,356
n60w80.002	200,0	0,019	5,571	12,994	1,356	10,882	4,118	23,252	0,800
n60w80.003	200,0	0,019	2,939	11,708	4,224	10,590	4,099	18,636	1,600
n60w80.004	200,0	0,016	5,083	13,086	1,166	10,498	2,315	21,112	0,000
n60w80.005	200,0	0,022	3,200	13,000	0,980	10,482	7,616	21,890	1,497
n60w100.001	200,0	0,031	4,841	11,908	4,792	10,512	3,370	23,812	0,490
n60w100.002	200,0	0,041	5,499	12,984	0,748	10,482	2,871	22,596	2,638
n60w100.003	200,0	0,025	6,621	13,004	1,855	10,488	3,929	22,008	1,356
n60w100.004	200,0	0,018	6,493	13,004	2,939	10,490	4,841	25,140	0,400
n60w100.005	102,1	0,025	4,630	13,030	0,000	10,628	3,007	17,806	0,400
Tiemp. medio	182,5	**0,0247***		12,8694		10,5510		21,9270	
Desviación (σ)	-	4,350		1,978		3,691		**1,446**	

Tabla 2.6. Comparativa de tiempo y desviación obtenidos por las diferentes heurísticas para n =60 (TSP)

Aunque para las anteriores instancias vaticinábamos un rendimiento muy parecido para SA y LAHC, en el caso de n = 60 nodos, LAHC "coge las riendas" en cuanto a la obtención de mejores resultados, tanto en conjunto como de forma individual en cada ejecución. Si bien es verdad que estos resultados superiores a SA es debido a una convergencia más lenta que este último, necesitando casi 10 segundos más en promedio para obtener soluciones relativamente parecidas. En este caso, tendríamos que hacer un balance de nuestras prioridades: calidad de la solución o tiempo disponible.

- **80 nodos:**

- Valores objetivos obtenidos

	M. exacto	**HC**		**SA**		**TA**		**LAHC**	
Archivo	Coste	Coste medio	Mejor valor	Coste medio	Mejor valor	Coste medio	Mejor valor	Coste medio	Mejor valor
n80w20.001	333	331,4	325	317,6	**315***	332,2	323	**316,6***	316
n80w20.002	326	338	331	326,8	326	340,4	338	**326**	326
n80w20.003	326	330,4	326	325,4	322	337	330	**324,4***	**321***
n80w20.004	331	361	352	335,8	332	357,4	353	**331,8**	**331**
n80w20.005	328	339	332	330,6	329	336,2	330	**328**	**328**
n80w40.001	348	366,4	360	351,6	349	367,4	365	**348,2**	**348**
n80w40.002	327	326,2	319	319,4	316	330	326	**317***	**316***
n80w40.003	354	365,2	352	354,4	351	372,2	369	**352,8***	**350***
n80w40.004	341	343,4	341	**331,2***	329	343,2	336	332	329
n80w40.005	361	354,8	340	340	335	351,6	342	**335***	335
n80w60.001	292	297,2	294	**292,8**	292	300	297	294,6	292
n80w60.002	318	330,2	323	319,8	318	329,8	322	**317***	**316***
n80w60.003	325	339,8	336	328,8	326	345,4	335	**325,4**	**325**
n80w60.004	355	337,2	329	328,2	327	338,8	331	**325,6***	**325***
n80w60.005	343	354,4	349	347,4	345	356	351	**344,6**	**343**
n80w80.001	325	337	326	329,2	327	333,4	327	**328,4**	**328**
n80w80.002	346	358,4	349	346,2	344	359,4	355	**343***	**342***
n80w80.003	334	343,8	341	337	335	346,8	344	**334**	**332***
n80w80.004	327	347	336	331,2	329	345,4	343	**329**	**327**
n80w80.005	367	352,4	348	345,2	344	360,8	347	**344,6***	**343***
Δ(%)		-2.0989		+1.034		-2.534		**+1.647**	

Tabla 2.7.Comparativa de resultados obtenidos por las diferentes heurísticas para n =80 (TSP)

- Tiempo y desviación

	M. exacto	**HC**		**SA**		**TA**		**LAHC**	
Archivo	Tiempo	Tiemp. medio	σ	Tiemp. medio	σ	Tiemp. medio	σ	Tiemp. medio	σ
n80w20.001	200	0,056	8,476	49,718	2,417	14,068	5,706	57,90	1,200
n80w20.002	200	0,066	3,688	49,786	0,748	14,052	1,356	51,56	0,000
n80w20.003	200	0,038	6,974	45,716	3,826	14,082	5,099	54,18	3,137
n80w20.004	200	0,063	4,817	50,128	2,040	14,186	3,720	63,93	1,166
n80w20.005	200	0,025	3,578	50,032	1,020	14,288	4,445	46,77	0,000
n80w40.001	200	0,047	8,476	49,842	2,245	14,102	2,871	61,95	0,400
n80w40.002	200	0,047	5,115	50,108	3,072	14,058	2,757	66,37	1,265
n80w40.003	200	0,056	6,735	49,946	1,855	14,274	3,655	52,72	1,939
n80w40.004	200	0,053	1,855	49,898	1,720	14,054	6,400	64,67	2,098
n80w40.005	200	0,050	9,826	49,854	2,966	14,176	6,020	51,14	0,000
n80w60.001	200	0,047	4,118	49,896	1,600	14,544	2,449	53,21	2,059
n80w60.002	200	0,022	7,194	49,918	1,600	14,072	4,308	48,59	0,894

n80w60.003	200	0,037	5,307	49,698	1,939	14,216	5,678	56,52	0,490
n80w60.004	200	0,056	4,578	49,802	1,939	14,206	4,622	55,36	0,800
n80w60.005	200	0,047	4,176	49,832	2,059	14,322	3,847	57,97	1,625
n80w80.001	200	0,041	6,723	49,848	2,040	14,134	7,003	60,85	0,490
n80w80.002	200	0,044	4,800	50,042	2,135	14,306	3,137	61,74	0,632
n80w80.003	200	0,031	1,600	50,002	1,673	14,074	2,638	64,57	1,414
n80w80.004	200	0,044	7,321	49,826	2,315	14,076	2,728	73,68	1,095
n80w80.005	200	0,044	2,577	50,192	0,980	14,070	7,756	60,27	1,855
Tiemp. medio	200	**0,0456***		49,7042		14,168		58,1976	
Desviación (σ)	-	5,3967		2,0095		4,3097		**1,1279**	

Tabla 2.8. Comparativa de tiempo y desviación obtenidos por las diferentes heurísticas para n =80

Cada vez se acentúa más la eficacia del LAHC con respecto al resto; de hecho para el caso de n = 80, en 11 de 25 instancias mejora la solución obtenida por el método exacto. De nuevo, y análogo a la reflexión que hacíamos para n = 60, tendríamos que considerar nuestras prioridades a la hora de elegir entre SA y LAHC.

- **100 nodos:**

- Valores objetivos obtenidos

	M. exacto	**HC**		**SA**		**TA**		**LAHC**	
Archivo	Coste	Coste medio	Mejor valor	Coste medio	Mejor valor	Coste medio	Mejor valor	Coste medio	Mejor valor
n100w20.001	397	391,2	385	385,8	382	404	397	**381,2***	**380***
n100w20.002	370	388	375	368,8	367	391,6	387	**365,2***	**364***
n100w20.003	408	394,2	383	382,2	374	398,8	395	**375,6***	374
n100w20.004	418	398	393	388,6	386	409,6	405	**381,4***	**380***
n100w20.005	416	386,8	379	381,8	376	393	385	**375,4***	**374***
n100w40.001	428	390,4	383	377	373	399,8	393	**371,4***	**371***
n100w40.002	400	402,2	394	**365,8***	**363***	404,2	398	381,6	380
n100w40.003	369	374	369	365,8	361	383,4	375	**362,8***	**360***
n100w40.004	392	387,8	377	383	381	409,8	408	**377***	**373***
n100w40.005	381	395,8	386	376,4	372	397,4	390	**371***	**370***
n100w60.001	426	395,8	386	350,8	348	373,8	364	**346,8***	**346***
n100w60.002	431	382,8	371	365,8	361*	386	383	**365***	362
n100w60.003	490	384,2	376	374,6	369	397,2	388	**368***	**366***
n100w60.004	387	378,4	370	372,4	371	386,8	377	**367,8***	**366***
n100w60.005	373	389,8	376	386,4	383	395	387	**376,6***	**373***
Δ(%)		+4.248		+8.281		+2.681		**+9.418**	

Tabla 2.9.Comparativa de resultados obtenidos por las diferentes heurísticas para n =100 (TSP)

- Tiempo y varianza

	M. exacto	**HC**		**SA**		**TA**		**LAHC**	
Archivo	Tiempo	Tiemp. medio	σ	Tiemp. medio	σ	Tiemp. medio	σ	Tiemp. medio	σ
n100w20.001	200	0,081	5,528	63,782	3,059	17,404	5,099	137,7	1,600
n100w20.002	200	0,091	9,445	63,744	1,166	17,074	3,666	166,9	1,600
n100w20.003	200	0,088	6,940	64,130	5,600	17,194	3,187	139,7	1,855
n100w20.004	200	0,078	4,195	64,226	3,072	17,518	3,826	140,5	1,855
n100w20.005	200	0,069	4,118	64,024	4,167	17,188	5,514	158,5	1,356
n100w40.001	200	0,084	6,020	64,020	2,449	17,284	3,868	177,3	0,490
n100w40.002	200	0,195	6,242	63,684	2,315	17,710	3,868	129,0	1,200
n100w40.003	200	0,081	3,950	64,142	3,544	17,186	4,587	137,5	2,040
n100w40.004	200	0,100	10,49	63,796	1,265	17,080	2,400	169,4	2,280
n100w40.005	200	0,091	6,853	63,890	2,332	18,364	7,579	151,2	0,894
n100w60.001	200	0,091	6,853	63,888	1,600	17,364	8,035	150,3	0,980
n100w60.002	200	0,100	7,222	64,036	2,638	17,238	2,898	145,9	2,530
n100w60.003	200	0,094	4,956	63,624	3,611	17,138	5,418	152,5	1,265
n100w60.004	200	0,106	5,851	64,374	1,855	17,298	6,242	133,2	1,470
n100w60.005	200	0,066	10,08	63,868	3,382	17,208	5,967	127,9	2,577
Tiemp. medio	200	**0,0942***		63,9458		17,3494		147,83	
Desviación (σ)	-	6,5837		2,8037		4,8102		**1,5994**	

Tabla 2.10. Comparativa de tiempo y varianzas obtenidos por las diferentes heurísticas para n =100 (TSP)

Conclusiones similares a las de los otros grupos se obtienen también para n =100. Aquí, tanto SA como LAHC mejoran el resultado obtenido por el método exacto para las 15 instancias, aunque destaca la actuación de LAHC, quien supera a SA en 14 de estas 15 instancias.

Al comienzo de este apartado decíamos que un algoritmo tenía que tener tres cualidades para ser considerado un "buen" algoritmo: esfuerzo computacional realista, solución cercana al óptimo y robustez. Por ello, se resumen los datos recogidos durante todo el estudio en los siguientes tres gráficos que dan buena medida de estas tres cualidades. En ellos, se representa para cada modelo: el tiempo (Figura 2.14), la desviación típica (Figura 2.15) y el porcentaje de mejora o empeoramiento (Figura 2.16) en función del tamaño del problema.

- **Esfuerzo computacional realista**: cómo podemos ver en la Figura 2.14, las heurísticas son necesarias siempre que el tiempo sea un factor relevante. Existe un gran diferencia entre el tiempo necesario para calcular una solución con el método exacto y el tiempo necesario para obtenerlo con cualquiera de las heurísticas estudiadas, sobre todo en el caso del algoritmo *Hill-Climbing*, que como ya hemos comentado es el más rápido

de todos, necesitando menos de 0,2 segundos para obtener una solución en cualquiera de las instancias.

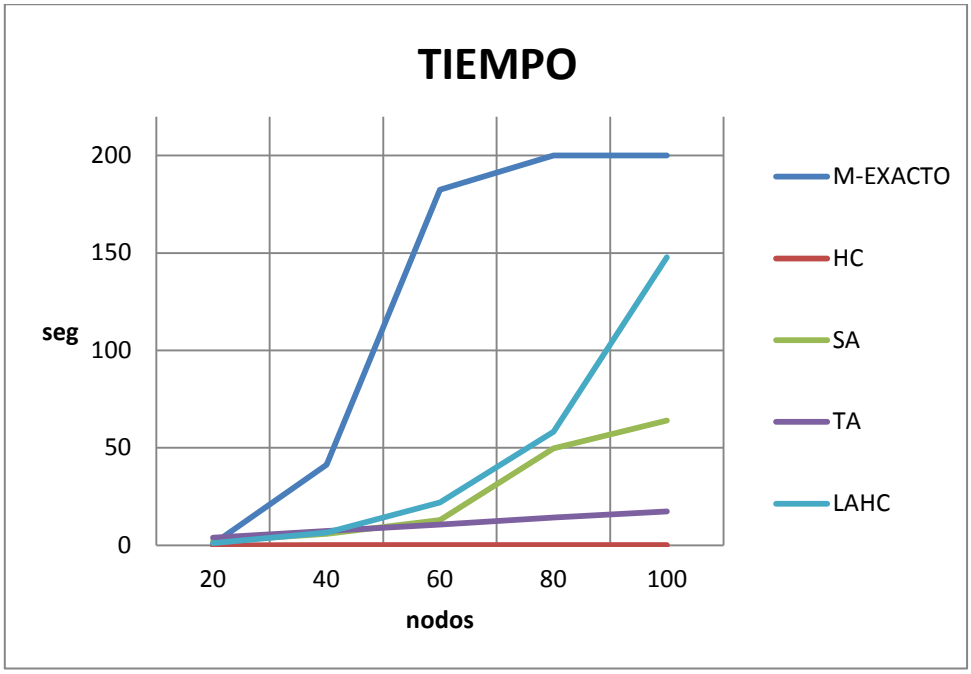

Figura 2.14. Tiempos obtenidos para los distintos métodos (TSP)

- **Robustez**: Para medir la robustez nos vamos a basar en la desviación típica. Se persigue obtener una desviación típica próxima a 0. Los modelos HC y TA se alejan bastante de este valor. Por tanto, podemos concluir que los más robustos son LAHC y SA.

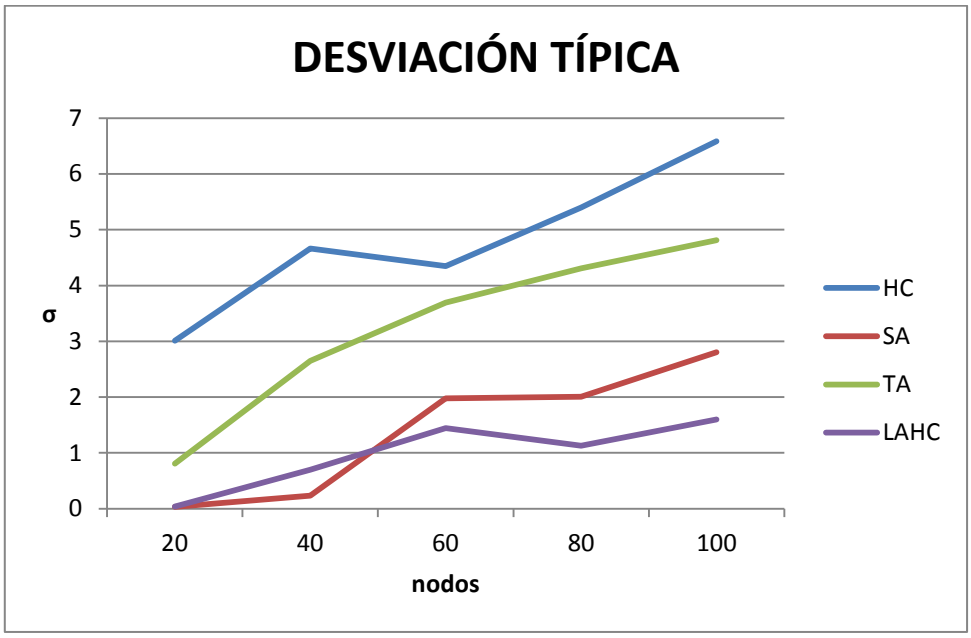

Figura 2.15. Desviación típica para los distintos métodos (TSP)

- **Solución cercana al óptimo**: Como vemos en la Figura 2.16. Para un tamaño pequeño de problema la mejora que puede observarse es irrelevante. Sin embargo, a media que el número de nodos aumenta, se obtienen soluciones hasta un 10% superior a las obtenidas con el método exacto.

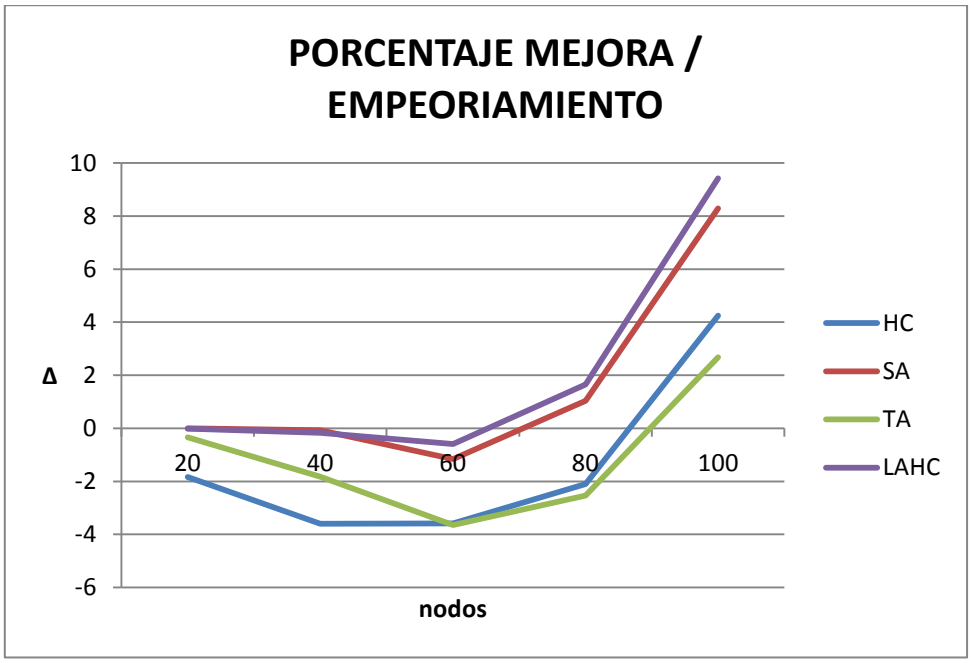

Figura 2.16. Porcentaje mejora/empeoramiento para los distintos métodos (TSP)

Los promedios finales de estos datos son:

	HC	SA	TA	LAHC
Tiempo	**0,03536**	26,893	10,662	47,095
σ	4,800	1,412	3,253	**0,982**
Δ	-1,374	1,615	-1,132	**2,05618**

Figura 2.17. Promedios finales (TSP)

Por tanto, si buscábamos el mejor algoritmo de estos cuatro, el más robusto y quien ofrece mejores soluciones es LAHC. Si lo que necesitamos es obtener una solución factible en el menor tiempo posible el algoritmo HC será la mejor opción.

2.7. Conclusiones

En este tema se han presentado 4 métodos para la resolución del TSP: Simulated *Annealing*, *Hill-Climbing, Threshold Acceptance y Late Acceptance Hill-Climbing*. Los dos primeros son muy conocidos y utilizados. Sin embargo, TA y LAHC son propuesta relativamente nuevas. Nuestro principal objetivo era comprobar si estas novedosas heurísticas ofrecían un rendimiento tan bueno como el que proponían sus autores. Tras los análisis efectuados en el apartado anterior, se puede afirmar con rotundidad que mientras *Threshold Acceptance* no es un método pésimo, ofrece un rendimiento notablemente inferior a lo que sostenían los padres de este algoritmo. En lo referente a LAHC, los resultados son incluso mejor de lo esperado, logrando superar el rendimiento de una de las heurísticas más efectivas y estudiadas para la resolución del TSP como es el Recocido Simulado (Google Académico muestra 856.000 publicaciones con la palabra clave "Simulated Annealing" y sólo 61 publicaciones sobre “Late Acceptance Hill-Climbing”).

Sin embargo, la principal ventaja que podemos corroborar de LAHC frente a SA es el reducido número de parámetros de entrada y la facilidad para “tunearlos”. Mientras que para el SA hemos tenido que ajustar 6 parámetros de entrada, que varían en función del tamaño del problema, para LAHC sólo hemos tenido que determinar 3, siendo la longitud Lfa uniforme para cualquier número de nodos.

Por eso no es de extrañar que, entre otra muchas noticias destacadas sobre este algoritmo, en diciembre de 2011 el LAHC ganase el primer premio en el Concurso Internacional de Optimización o que dos equipos de investigación participantes de *ROADEF / EURO Challenge 2012* presentasen algoritmos basados en LAHC y consiguieran el cuarto y séptimo lugar en sus respectivas categorías con más de 72 equipos participantes [40].

Capítulo 3

El Problema del Viajante de Comercio con Ventanas de Tiempo (TSPTW)

3.1. Introducción.

El Problema del Viajante con Ventanas de Tiempo (TSPTW) es una de las variantes del problema del TSP más estudiadas y conocidas. Aplicado a un viajante de comercio, consiste en buscar la ruta de coste mínimo que debe realizar éste último, empezando y volviendo a un mismo y único almacén, pasando por todos los nodos una sola vez en la franja horaria especificada por el cliente, conocida como ventana de tiempo. El servicio en cada nodo deberá comenzar dentro de la ventana de tiempo, definida como el intervalo entre la fecha más temprana y la más tardía en la que está permitido el comienzo del servicio en dicho nodo. No se permite llegar al nodo después de la fecha u hora más tardía. Sin embargo, si se llega antes de la fecha u hora más temprana, está permitido esperar hasta que el nodo esté listo para el comienzo del servicio. El TSPTW es un problema de rutas con limitación temporal que involucra un único vehículo del cual no se tiene en cuenta la capacidad.

Este tipo de problema puede ser encontrado en una gran variedad de aplicaciones relacionadas con sectores industriales y de servicios como es el caso de entregas bancarias, entregas postales o planificación de rutas de transporte escolar.

Además, tienen una importante aplicación en manufactura, como es el caso de los sistemas de manufactura automatizados, en los cuales un vehículo guiado de forma automática debe visitar un conjunto de máquinas flexibles. Para alcanzar una completa utilización de la máquina, el vehículo automatizado debe visitar cada máquina dentro de la ventana de tiempo determinada por el tiempo de procesamiento del trabajo que actualmente esté llevándose a cabo.

3.2. Formulación matemática

A la hora de formular el TSPTW se plantearán tres posibles modelos:

3.2.1. Modelo 1: Modelo de Dumas, Solomon y Soumis [5]

El TSPTW puede ser formulado como un grafo G = (V,A) donde V = $\{1, 2, \dots, n\}$ es el conjunto de clientes a visitar, 0 representa el almacén, y A =$\{(i,j): i,j \in V \cup \{0\}, i \neq j\}$ es el conjunto de arcos entre clientes. El coste de viajar desde i hasta j se representa por c_{ij} que incluye tanto el tiempo de servicio del cliente i como el tiempo necesario para viajar de i a j. Cada cliente o nodo tiene asociado una ventana de tiempo $[a_i, b_i]$ donde a_i y b_i representan el tiempo de inicio y final de servicio, respectivamente. El problema consiste en encontrar una ruta Hamiltoniana que empiece y termine en el almacén, satisfaga todas las restricciones de ventanas de tiempo y minimice la distancia total recorrida.

Siendo i igual a 1 si y sólo si el cliente j (o el almacén en caso de que $j = 0$) es visitado inmediatamente después del cliente i (o el almacén en caso de que $i = 0$) y 0 en caso de que no se cumplan esta condición, β_i el tiempo en el que el cliente i es visitado (tiempo inicio servicio) y M una constante muy grande; el TSPTW puede formularse como un sigue [17]:

$$Min \sum_{i \in V \cup \{0\}} \sum_{j \in V \cup \{0\}} c_{ij} x_{ij}$$

s.a:

$$\sum_{j \in V \cup \{0\}, i \neq j} x_{ij} = 1 \quad \forall j \in V \cup \{0\} \qquad (1)$$

$$\sum_{j \in V \cup \{0\}, j \neq i} x_{ij} = 1 \quad \forall i \in V \cup \{0\} \qquad (2)$$

$$\beta_j \geq \beta_i + c_{ij} - M(1 - x_{ij}) \quad \forall i,j \in V \cup \{0\}, j \neq 0 \qquad (3)$$

$$\alpha_i \geq \beta_i \leq b_i \quad \forall i \in V \cup \{0\} \qquad (4)$$

$$x_{ij} \in \{0,1\} \quad \forall i,j \in V \cup \{0\}$$

$$\beta_i \in \mathbb{R}_{\geq 0} \quad \forall i \in V \cup \{0\}$$

Como vemos la función objetivo es minimizar el coste de la ruta. Las restricciones (1) y (2) indican que cada cliente ha de ser visitados exactamente una vez. La restricción (3) asegura que el tiempo de llegada de un cliente no sea menor que el tiempo de visita del cliente anterior inmediato ($\beta_j = tiempo\ de\ llegada$). Además gracias a que todos los costes son positivos en la restricción (3) se evita la formación de subtours. Finalmente, la restricción (4) es la que impone que se cumplan las ventanas de tiempo.

3.2.2. Modelo 2: Time Indexed Formulation (TIF)

Se parte de la formulación del modelo 1, pero en lugar de tener el tiempo en el que el cliente i es visitado β_i, tenemos variables binarias z_i^t asociadas con cada $t \in W_i$ tal que $z_i^t = 1$ si y solo si el tiempo de comienzo del nodo i es t ($W_i = [a_i, b_i]$). Además tenemos las variables binarias y_{ij}^t asociadas con cada arco $(i,j) \in A$ y cada entero $t \in W_i$ tal que $y_{ij}^t = 1$ si y solo si el tiempo de comienzo en el nodo i es t y el arco (i,j) está presente en la ruta. Se asume que y_{ij}^t está presente en la formulación sólo si $t + \theta_{ij} \leq b_j$, es decir, uno puede empezar en el tiempo t en el nodo i , viajar a lo largo del arco (i,j) y llegar a j antes de la fecha máxima. Debemos tener en cuenta que θ_{ij} es equivalente a c_{ij}. Usamos $I_k(i,j)$ para denotar la colección de posibles tiempos de inicio en el nodo k asumiendo que el tiempo de inicio en el nodo i es t y el arco (k,i) es seleccionado:

$$I_k(i,t) = \{\tau \epsilon\, W_k : \max\{\tau + \theta_{ki,} a_i\} = t\,\}$$

En otras palabras, si el tiempo de comienzo t en el nodo i es a_i, entonces el tiempo de comienzo en el nodo k de una ruta factible es algún $\tau \in W_k$ que satisfaga $\tau \leq a_i - \theta_{ki}$. En caso contrario, el tiempo de inicio en el nodo k es exactamente $t - \theta_{ki}$ si este depende de W_k.

Nótese que esta definición del problema requiere dos nodos especiales de inicio y fin, que son respectivamente p y q.

Así, tenemos la formulación “Time Indexed” (TIF) para el TSTPW propuesta por Günlük y Tramontani [20]:

$$\min \sum_{(i,j)\in A} c_{ij}\, x_{ij}$$

s.a:

$$\sum_{t \in W_i} z_i^t = 1 \quad \forall\, i \in V \tag{1}$$

$$\sum_{j \in V+_{(i)}} y_{ij}^t = z_i^t \quad \forall\, i \in V \setminus \{q\}, \forall\ t \in W_i\,, \tag{2}$$

$$\sum_{k \in V-(i)} \sum_{\tau \in I_k(i,t)} y_{ki}^\tau = z_i^t \quad \forall\, i \in V \setminus \{p\}, \forall\ t \in W_i, \tag{3}$$

$$\sum_{t \in W_i} y_{ij}^t = x_{ij} \quad \forall\, (i,j) \in A, \tag{4}$$

$$x_{ij}, y_{ij}^t, z_i^t \in \{0,1\} \qquad \forall\, i \in V,\ \forall\, (i,j) \in A, \qquad \forall\, t \in W_i \tag{5}$$

Las restricciones (1) indican que una ruta debe comenzar en cada nodo i dentro de la ventana de tiempo W_i. La restricción (2) afirma que una ruta con tiempo de inicio o comienzo t en $i \neq q$ debe abandonar el nodo en un tiempo t lo largo de algún arco, mientras que las restricciones (3) afirman que para un nodo $i \neq p$, una ruta tiene una hora de inicio en el nodo anterior (por ejemplo k) contenida en $I_k(i,t)$. Es evidente que cualquier ruta factible se puede asignar a una solución entera de TIF "configurando" $x_{ij} = 1$ si el arco (i,j) es empleado en la ruta, y $z_i^t = 1$ si la ruta empieza en el instante t en el nodo i. Nótese que si $x_{ij} = 1$, y para algún t, t' tenemos que $z_i^t = 1\ y\ z_j^{t'} = 1$, entonces $t < t'$ y $\theta_{ij} > 0$ para todos los arcos de A. Por lo tanto, cada solución de este método define una ruta factible.

3.2.3. Modelo 3: Two-Commodity Flow Formulation

Langevin et al. [25] introdujeron la formulación conocida como flujo de dos productos: (más conocido por su nombre en inglés *Two-Commodity Flow Formulation*). En dicha formulación, vamos a tener dos "mercancías" o "productos": el flujo Y = (y_{ij}) y el flujo Z = (z_{ij}). Su fundamento teórico asume que mientras que una unidad del flujo Y es "entregada" en cada nodo, una unidad del flujo Z es quitada. Por lo que la cantidad total de Y y Z va a ser siempre la misma (ver figura 3.1).

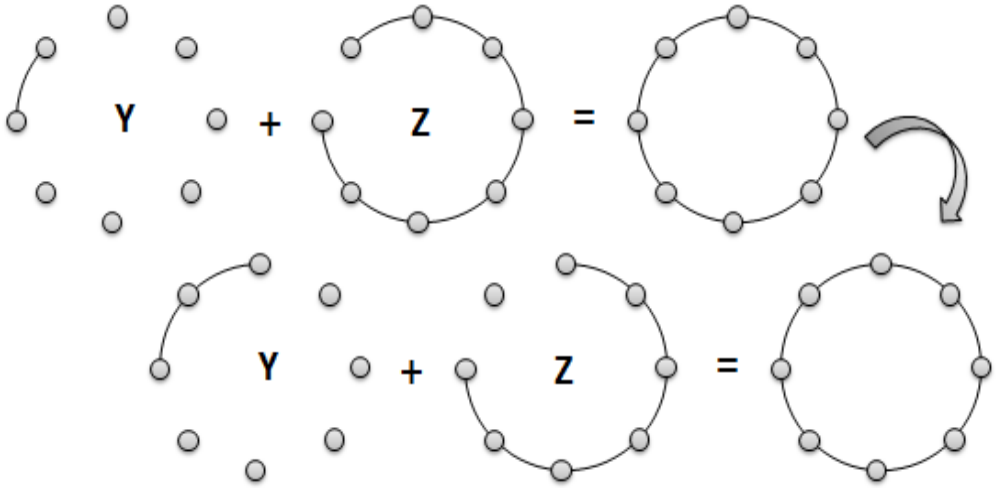

Figura 3.1 "Funcionamiento" flujo Y y flujo Z en el modelo Two Commodity Flow Formulation

Siendo [0,T] la ventana de tiempo en el nodo almacén, y siendo $[a_i, b_i]$, $i = 2, \dots, n$ el intervalo definido entre el tiempo de inicio y final de servicio de cada nodo, tal como veíamos en el anterior modelo, si el viajante llega antes del inicio del tiempo de servicio tendrá que esperar. Mientras que la llegada posterior a b_i no está permitida.

Aunque en este caso no lo abordaremos, con este modelo se puede tener en cuenta el coste de espera. Para ello, tenemos que definir w_i, $i = 2, \dots, n$ como el tiempo de espera antes de visitar el nodo i y $c_{i,} i = 2, \dots, n$ como el coste incurrido por unidad de espera. Como ya sabemos c_{ij} es el coste o distancia entre cada para de nodos.

La formulación sería entonces:

$$\min \sum_{i=1}^{n} \sum_{j=1}^{n} c_{ij}(y_{ij} + z_{ij})/T + \sum_{i=2}^{n} c_i w_i \qquad (14)$$

s.a:

$$\sum_{i=2}^{n} z_{i1} + \sum_{i=2}^{n} \frac{t_{i1}(y_{i1} + z_{i1})}{T} \leq T \qquad (9)$$

$$\sum_{j=1}^{n} (y_{ij} + z_{ij}) = T \qquad i \in N \backslash \{1\} \quad (10)$$

$$\sum_{i=1}^{n}(y_{ij} + z_{ij}) = T \qquad j \in N\backslash\{1\} \quad (11)$$

$$y_{ij} + z_{ij} \in \{0, T\} \qquad (i,j) \in A \quad (12)$$

$$y_{ij} \geq 0, \qquad z_{ij} \geq 0 \quad (i,j) \in A \qquad (13)$$

$$\sum_{j=1}^{n} z_{ij} - \sum_{j=1}^{n} z_{ji} - \sum_{j=1}^{n} \frac{t_{ij}(y_{ji} + z_{ji})}{T} \qquad i \in N\backslash\{1\} \quad (15)$$

$$a_i \leq \sum_{j=1}^{n} z_{ij} \leq b_i \qquad i \in N\backslash\{1\} \quad (16)$$

$$w_i \geq 0 \qquad i \in N\backslash\{1\} \quad (17)$$

Podemos hacer simplificaciones a este modelo. Por ejemplo, la relajación lineal de este modelo puede conseguirse remplazando las restricciones (12) por las inecuaciones $y_{ij} + z_{ij} \leq T$, $(i,j) \in A$. Sin embargo, serían redundantes ya que (10) u (11) podrían ser eliminadas.

Además, hemos dicho que en este caso no tendremos en cuenta el coste de espera, por lo que $(c_i = 0, i = 2, \dots, n)$ y por tanto las variables $w_i, i = 2, \dots, n$ pueden ser consideradas sobrantes.

Nótese que para aquellos arcos en los que el coste de viajar es idéntico al tiempo necesario para viajar (como hemos considerado a lo largo del proyecto) tendremos que sustituir en las fórmulas $c_{ij} = t_{ij}$ o viceversa.

3.3. Resultados computacionales método exacto

Para conocer las soluciones óptimas y poder posteriormente comparar con las heurísticas nuestra primera opción, como en el caso del TSP, será la resolución con "Xpress IVE". Para ello, nos hemos basado en los dos primeros modelos: el modelo de *Dumas et al.* y el modelo *Time Indexed Formulation*, ya que en pruebas iniciales el modelo *Two-Commodity Flow Formulation* ha requerido un tiempo notoriamente superior a los otros dos modelos para encontrar la solución óptima.

Como ya comentábamos en los anteriores capítulos, estos métodos se han implementado en lenguaje Mosel y ejecutados en un ordenador Intel CORE con velocidad 3.60GHz y una memoria instalada (RAM) de 16GB. Las instancias que se han ejecutado son las propuestas por Dumas et al. [16]. Sin embargo, en este caso sólo abarcaremos problemas de 20, 40, 60 y 80 nodos.

Nótese que ahora consideramos las ventanas de tiempo, por lo que las 95 instancias que se van a ejecutar para comprobar el rendimiento de estos métodos quedan divididas en 19 grupos, ya que dentro de cada grupo de nodos vamos a hacer una subdivisión en función de la amplitud de las ventanas de tiempo. Cada uno de los "lados" de la ventana de tiempo fue generado como una variable aleatoria uniforme dentro del rango [0, w/2] donde w = 20, 40, 60, 80 y 100. Por ejemplo, uno de los grupos sería el formado por las instancias n40w20.001, n40w20.002, n40w20.003, n40w20.004, n40w20.005 donde tenemos 40 clientes y las ventanas de tiempo están dentro del intervalo [0, 40/2] = [0, 20] unidades de tiempo.

La dificultad de resolución del TSPTW es mucho mayor que la del TSP. Nótese que para un determinado tamaño de problema, la dificultad del mismo aumenta a medida que la anchura de las ventanas de tiempo es mayor, debido a que hay una mayor probabilidad de que estas ventanas de tiempo se superpongan. Además, esto también aumenta con el incremento del número de nodos debido a que el área a considerar es el mismo para todas las instancias. Por lo tanto, a medida que n aumenta, también aumenta la densidad de puntos y como consecuencia se reduce la cantidad de rutas factibles.

Las siguientes tablas muestran para cada uno de los métodos mencionados anteriormente el resultado obtenido, así como el tiempo necesario para ello (limitado a 300 segundos). Para aquellos problemas en los que no se haya obtenido una solución factible aparecerá un guión en la casilla correspondiente a Resultado.

- **20 nodos:**

	Dumas et al.		TIF	
Archivo	Resultado	Tiempo	Resultado	Tiempo
n20w20.001	378	0,013	378	0,078
n20w20.002	286	0,012	286	0,156
n20w20.003	394	0,016	394	0,391
n20w20.004	396	0,001	396	0,359
n20w20.005	352	0,015	352	0,172
n20w40.001	254	0,112	254	2,496
n20w40.002	333	0,002	333	0,249

n20w40.003	317	0,016	317	0,889
n20w40.004	388	0,063	388	0,532
n20w40.005	288	0,062	288	1,013
n20w60.001	335	0,171	335	9,548
n20w60.002	244	0,031	244	2,293
n20w60.003	352	0,047	352	1,904
n20w60.004	280	0,125	280	15,47
n20w60.005	338	0,094	338	3,543
n20w80.001	329	0,251	329	16,18
n20w80.002	338	0,062	338	7,661
n20w80.003	320	0,421	320	1,376
n20w80.004	304	0,064	304	6,818
n20w80.005	264	0,936	264	119,3
n20w100.001	237	0,312	237	18,23
n20w100.002	222	0,018	222	17,49
n20w100.003	310	1,451	310	35,87
n20w100.004	349	0,047	349	15,94
n20w100.005	258	0,156	258	16,52

Tabla 3.1.Valores objetivos obtenidos mediante el método exacto para n = 20 (TSPTW)

- **40 nodos:**

	Dumas et al.		TIF	
Archivo	Resultado	Tiempo	Resultado	Tiempo
n40w20.001	497	0,125	497	0,795
n40w20.002	552	0,078	552	3,152
n40w20.003	478	0,047	478	2,636
n40w20.004	404	0,093	404	1,965
n40w20.005	499	0,047	499	2,995
n40w40.001	465	0,219	465	1,548
n40w40.002	461	0,265	461	14,838
n40w40.003	470	0,234	470	2,512
n40w40.004	452	0,843	452	13,684
n40w40.005	453	0,842	453	43,36
n40w60.001	470	27,19	494	72,392
n40w60.002	470	1,295	470	67,322
n40w60.003	393	7,318	393	121,367
n40w60.004	382	0,984	389	300
n40w60.005	328	0,624	328	28,317
n40w80.001	395	50,36	395	300,0
n40w80.002	454	300,0	-	300,0
n40w80.003	410	300,0	-	300,0
n40w80.004	417	1,81	-	300,0
n40w80.005	344	300,0	-	300,0

n40w100.001	453	300,0	-	300,0
n40w100.002	370	300,0	-	300,0
n40w100.003	361	300,0	-	300,0
n40w100.004	357	300,0	-	300,0
n40w100.005	393	300,0	-	300,0

Tabla 3.2.Valores objetivos obtenidos mediante el método exacto para n = 40

- **60 nodos**:

	Dumas et al.		TIF	
Archivo	Resultado	Tiempo	Resultado	Tiempo
n60w20.001	551	0,285	551	11,31
n60w20.002	605	0,207	605	8,922
n60w20.003	532	0,309	532	8,151
n60w20.004	616	0,165	616	9,034
n60w20.005	603	0,223	603	2,184
n60w40.001	591	1,340	591	44,82
n60w40.002	621	6,187	621	53,37
n60w40.003	604	300,0	603	56,03
n60w40.004	597	1,216	597	54,01
n60w40.005	539	1,114	539	26,01
n60w60.001	609	1,322	609	126,1
n60w60.002	568	300,0	589	300,0
n60w60.003	485	300,0	-	300,0
n60w60.004	574	300,0	-	300,0
n60w60.005	569	300,0	-	300,0
n60w80.001	474	300,0	-	300,0
n60w80.002	507	300,0	-	300,0
n60w80.003	568	300,0	-	300,0
n60w80.004	573	300,0	-	300,0
n60w80.005	470	300,0	-	300,0
n60w100.001	-	300,0	-	300,0
n60w100.002	-	300,0	-	300,0
n60w100.003	-	300,0	-	300,0
n60w100.004	-	300,0	-	300,0
n60w100.005	-	300,0	-	300,0

Tabla 3.3.Valores objetivos obtenidos mediante el método exacto para n = 60

- **80 nodos**:

	Da Silva		TIF	
Archivo	Resultado	Tiempo	Resultado	Tiempo
n80w20.001	616	0,543	616	31,32
n80w20.002	737	0,074	737	7,831
n80w20.003	667	0,781	667	7,055
n80w20.004	615	0,343	615	4,221
n80w20.005	748	0,748	748	17,03
n80w40.001	606	1,148	606	96,73
n80w40.002	-	300,0	618	300,0
n80w40.003	668	4,165	668	300,0
n80w40.004	557	300,0	-	300,0
n80w40.005	695	300,0	713	300,0
n80w60.001	563	300,0	-	300,0
n80w60.002	628	130,7	-	300,0
n80w60.003	649	100,7	-	300,0
n80w60.004	-	300,0	-	300,0
n80w60.005	-	300,0	-	300,0
n80w80.001	-	300,0	-	300,0
n80w80.002	-	300,0	-	300,0
n80w80.003	-	300,0	-	300,0
n80w80.004	-	300,0	-	300,0
n80w80.005	-	300,0	-	300,0

Tabla 3.4.Valores objetivos obtenidos mediante el método exacto para n = 80

La dificultad que comentábamos al inicio de éste análisis ha quedado probada. Mientras que en el capítulo 1, en el cual no se tenían en cuenta las ventanas de tiempo, Xpress siempre encontraba una solución, en el caso del TSPTW esto ocurre para todas las instancias. De hecho, solo encuentra solución con una probabilidad del 100% en el caso de 20 nodos. Veamos si con las heurísticas conseguimos dar solución estos problemas.

Capítulo 4

Heurísticas de búsqueda local para la resolución del Problema del Viajante de Comercio con Ventanas de Tiempo (TSPTW)

4.1. Introducción.

Debido a las limitaciones de los métodos exactos, la búsqueda ahora se centra en técnicas heurísticas adecuadas para la resolución de estos problemas. Diferentes métodos se han propuesto a lo largo de los últimos años para resolver el problema del TSPTW. Carlton y Barnes (1996), resolvieron este problema usando el método de la Búsqueda Tabú, el cual emplea una función de penalización estática para evitar considerar soluciones no factibles en la búsqueda del entorno. Gendreu et al. (1998) ofrecieron una heurística de construcción y post-optimización basada en la heurística usada para el TSP del vecino más próximo. Wolfer Calvo (2000) introdujo una heurística que primero construye una ruta inicial y después la mejora gracias a la búsqueda local. Por último, Jeffrey W.Ohlman y Barrett W. Thomas [31] presentaron la siguiente propuesta en 2007: *Compressed-Annealing Simulated* (Recocido Simulado con Compresión), que a diferencia de la Búsqueda Tabú, utiliza una función de penalización dinámica. Según los padres de este algoritmo, con el método *Compressed-Annealing Simulated* se superaron las mejores soluciones conocidas hasta el momento en un gran número de instancias. Además, demostraron que este método convergía hacia la mejor solución incluso considerando un elevado número de nodos con extendidas ventanas de tiempo.

El objetivo de este capítulo será comprobar la veracidad de esto último y la eficacia de dicho método. Además, y debido a los buenos resultados obtenidos para el TSP, se utilizarán métodos de penalización basados en la heurística *Simulated Annealing* y la heurística *Late Acceptance Hill-Climbing.*

4.2. Compressed-Annealing Simulated (Recocido simulado con compresión)

4.2.1. Formulación particular del modelo.

Para definir el TSPTW, Olhman y Thomas propusieron otra formulación distinta de las vistas en el capítulo anterior, la cual se presenta a continuación ya que será de ayuda para la compresión del algoritmo. Sea $G = (N, A)$ un grafo finito, donde $N = \{0,1, \dots, n\}$ es el conjunto finito de nodos o clientes y $A = N\ x\ N$ es el conjunto de arcos que conecta dichos clientes y asumiendo que existe un arco $(i, j) \in A$ para todo $i, j \in N$; una ruta será definida por el orden en que los clientes o nodos son visitados y se denota como $\zeta = \{ p_0, p_1, \dots, p_n, p_{n+1} \}$, donde p_i denota el índice del cliente en la posición i-ésima. Asumiendo que toda ruta ha de comenzar y terminar en el nodo $p_0 = p_{n+1} = 0$, el resto de los nodos ocupará una posición entre $p_1\ y\ p_n$, ambos inclusive.

Para $j = 0, \dots, n$, hay un coste $c(a_j)$ al pasar por el arco $a_j = (p_j, p_{j+1})$, que normalmente engloba el tiempo de servicio del cliente p_j más el tiempo de viaje entre $p_j\ y\ p_{j+1}$. Asociado a cada cliente, y como ya sabemos, tenemos una ventana de tiempo $[e_i, l_i]$ durante la cual el cliente i ha de ser visitado.

Perseguimos un doble objetivo: minimizar la suma de costes incurridos por viajar de un cliente a otro y volver al almacén o nodo origen lo antes posible. El primer objetivo será formulado como $f(\zeta) = \sum_{i=0}^{n} c(a_i)$. Además para comprobar la factibilidad del problema definiremos dos nuevas variables: el tiempo de llegada a cada nodo A_{pi} y el tiempo de inicio de servicio D_{pi} (que en el caso de un tiempo de servicio = 0 corresponde con el tiempo de salida)

El problema del TSPTW está compuesto de dos componentes principales: un Problema del Viajante de Comercio y un Problema de *scheduling* o programación. Como ya explicamos, el TSP es un problema NP-duro, si además añadimos la componente de las ventanas de tiempo la dificultad para encontrar una solución factible es aún mayor. Usando este método de penalización debemos descomponer parcialmente estas dos partes. Consideraremos soluciones no factibles relajando las restricciones correspondientes a las ventanas de tiempo $\{D_{pi} \leq l_i : i, \dots, n\}$ en la función objetivo con una función de penalización de la forma:

$$p(\zeta) = \sum_{i=1}^{n} [\max\{0, D_{pi} - l_{pi}\}]^{s},$$

Para cualquier $s > 0$. Así, esta versión relajada del TSPTW podría definirse como:

Minimizar $v(\zeta, \lambda)$

$$= \sum_{i=0}^{n} c(a_i) + \lambda \sum_{i=1}^{n} \max\{0,\ D_{pi} - l_{pi}\}$$

s.a:

$$A_{pi} = D_{pi-1} + c(a_{i-1}) \quad para\ i = 1,\ \ldots, n+1;$$

$$D_{pi} = \max\{A_{pi}, e_{pi}\}\ \ para\ i = 1, \ldots, n;$$

$$D_{p0} = 0$$

$$p_i \in \{1,2, \ldots, n\}\ para\ i = 1, \ldots, n;$$

$$p_i \neq p_j \quad para\ i, j = 1, \ldots, n\ , i \neq j$$

$$p_0 = 0$$

$$p_{n+1} = 0$$

Nótese que si consideramos la minimización del tiempo total de la ruta, podríamos conocer también el tiempo de espera en cada nodo, $W_{pi} = D_{pi} - A_{pi}$ para $i = 1, \ldots, n$. Entonces el término $\sum_{i=1}^{n} W_{pi}$ debería ser añadido a la función objetivo.

4.2.2. Método *Compressed-Annealing Simulated*

Una vez comprendido lo anterior, nos centramos en la resolución del TSPTW usando el algoritmo *Compressed-Annealing Simulated*, variante del Recocido Simulado que veíamos en profundidad en el capítulo 2, el cual es un método de búsqueda local inspirado en el proceso físico que lleva este nombre: fundir y más tarde enfriar lentamente un sólido de forma que éste alcance su estado de energía más bajo.

Theodoracatos y Grimsley (1995) y Morse (1997) introdujeron esta variable del Recocido Simulado gracias al uso de una nueva variable de penalización, el multiplicador λ que complementaba a la ya existente variable de temperatura. Manteniendo la analogía física del recocido, este nuevo parámetro recibe el nombre de "presión". De ahí el nombre de este algoritmo, ya que sería un recocido simulado comprimido. En el contexto del TSPTW, la temperatura controla la probabilidad de la transición a una ruta más costosa mientras que la

presión controla la probabilidad de la transición a una ruta no factible con respecto a las ventanas de tiempo.

Si miramos la figura 4.1, en la que se muestra el pseudocódigo de este método, podemos ver que sólo actualizamos la mejor solución cuando se encuentra una solución factible que mejora el coste $f(\zeta_{best})$. Esto refleja el principal objetivo, que es buscar una ruta que sea factible con el mínimo coste.

Esquema básico del Compressed-Annealing Simulated

Generar una solución inicial ζ
Inicializar la mejor ruta encontrada, ζ_{best}, tal que $f(\zeta_{best}) = \infty$ y $p(\zeta_{best}) = 0$
$k = 0$
Especificar temperatura y presión inicial τ_k y λ_k
Definir i, el número de iteraciones para cada temperatura/presión
While (el criterio de parada no se satisfaga) **do**
    Contador = 0
    While (contador $\leq i$) **do**
    Contador = Contador + 1
    Generar aleatoriamente y, una solución del entorno de ζ
    Con probabilidad $\exp(\frac{-v(y,\lambda_k)-v(x,\lambda_k))^+}{\tau_k})$, $\zeta = y$
    If $p(\zeta) \leq 0$ y $f((\zeta)) \leq f(\zeta_{best})$ **then** $\zeta_{best} = \zeta$; **end-if**
    End-do
    $k = k + 1$
    Actualizar τ_k y λ_k
End-do

Figura.4.1. Esquema básico algoritmo Compressed-Annealing Simulated

4.2.3. Decisiones genéricas

a) Elección del programa de enfriamiento y compresión

El rendimiento de la búsqueda en este método se ve afectado directamente por la manera en que los parámetros de presión y temperatura son respectivamente aumentados y disminuidos durante la ejecución de dicho método. Cada i iteraciones, los valores de temperatura y presión son actualizados de acuerdo con los programas de enfriamiento $\{\tau_0, \tau_1, \dots\}$ y de compresión $\{\lambda_1, \lambda_2, \dots\}$, respectivamente.

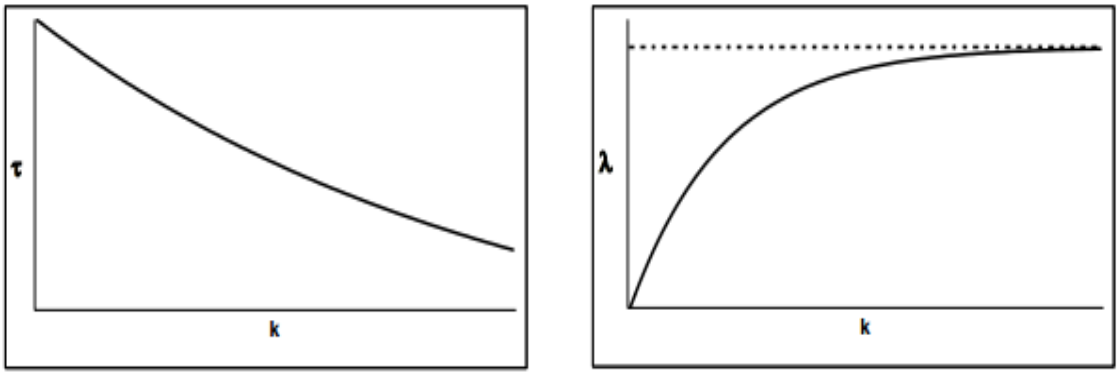

Figura 4.2. Programas de enfriamiento y compresión teóricos

Mientras que los valores teóricos de enfriamiento y compresión son demasiado lentos para ser utilizados en la práctica, estos nos proporcionan una visión sobre la "forma" que ha de seguir estos programas de enfriamiento y compresión (ver figura 4.2). La combinación de esta intuición, junto con experimentos llevados a cabo por distintos investigadores de este campo, respaldan el uso de un programa de enfriamiento geométrico (Dowsland 1993) y un programa de compresión exponencial limitado (Olhman et al 2004). Para parámetros $0 \leq \alpha \leq 1$, $\gamma \geq 0$, $\lambda_0 \geq 0$ y $\hat{\lambda} \geq 0$, estos programas pueden ser formalmente definidos como:

$$\tau_{k+1} = \alpha\tau_k$$

$$\lambda_{k+1} = \hat{\lambda}\left(1 - \frac{(\hat{\lambda} - \lambda_{0)}}{\hat{\lambda}} e^{-\gamma k}\right)$$

Los valores del ratio de enfriamiento suelen estar entre 0.8 y 0.99 y el valor del ratio de compresión γ normalmente varía entre 0.01 y 0.1. Para aplicar estos programas, también hemos de definir los valores de temperatura y presión inicial (τ_0 y λ_0), así como la presión máxima $\hat{\lambda}$. La presión inicial λ_0 puede ser 0 en todos los casos. Sin embargo, la elección de τ_0 y $\hat{\lambda}$ no es tan sencilla.

El valor de la temperatura inicial debe seleccionarse de manera que al principio la probabilidad de aceptar soluciones peores sea próxima a uno, lo que le permite al algoritmo suficiente movilidad para explorar el espacio. Sin embargo, si esta temperatura es excesivamente grande puede provocar tiempos computacionales muy grandes o una convergencia demasiado pobre. Además, el ajuste de la temperatura inicial se ve influido por la presencia de presión. Si elegimos un τ_0 excesivamente grande la búsqueda tenderá a ser arbitraria en lugar de "dirigida".

Cabe destacar también la dificultad de determinar un "buen" valor del multiplicador de penalización λ. Si este valor es muy grande impide la convergencia a soluciones no factibles, pero pueden retardar la búsqueda. Por otra parte, si el valor λ es "pequeño" puede concluir en una búsqueda del entorno robusta aunque el programa no podrá diferenciar de manera satisfactoria entre las soluciones factibles y no factibles.

Un primer paso a la hora de seleccionar los parámetros iniciales adecuados es generar R, un conjunto de 2r soluciones, generando aleatoriamente r pares de soluciones vecinas (con r = 5000 sería suficiente). Adaptando las técnicas de Laarhoven and Aarts (1987) and Dowsland (1993) y utilizando la información obtenida de R podemos especificar un valor de τ_0 apropiado. Para ello, primero tenemos que especificar χ_0, el porcentaje propuesto de soluciones que se van a aceptar en τ_0. Esta variable suele estar entre 0.8 y 0.99. Computando $|\overline{\Delta v}|$, la diferencia media absoluta de la función objetivo tras n pruebas, determinamos la temperatura inicial como sigue:

$$\tau_0 = \frac{|\overline{\Delta v}|}{\ln(1/\chi_0)}$$

Con este valor de temperatura inicial, el ratio de aceptación real debe ser supervisado durante un determinado número de iteraciones. Si la proporción real de la aceptación es menor que χ_0, entonces τ_0 es reinicializado en 1.5 veces su valor corriente. Este procedimiento es repetido hasta que el ratio de aceptación observado sea igual o superior a χ_0.

Como se muestra en el análisis teórico en Ohlmann et al. (2004), existe una presión máxima λ* a partir de la cual más allá de proporcionar una alta compresión sólo serviría para exagerar los rasgos de la solución encontrada. Por lo tanto, un programa ideal de compresión sería aumentar gradualmente λ desde cero hasta λ*, permitiendo así al algoritmo explorar el espacio de soluciones.

Por desgracia, la determinación de un límite superior λ* adecuado utilizando sólo la información contenida en *R* es muy difícil. No obstante, este valor λ* puede ser calibrado experimentalmente con la siguiente aproximación

$$\hat{\lambda} = max\left\{\frac{f(\zeta)}{p(\zeta)}\frac{k}{1-k}\right\}$$

Donde los valores de k varían desde 0.75 hasta 0.9.

Para reducir todo el proceso anterior y puesto que los resultados que esta parametrización ofrecía estaban lejos de dar una solución factible, y sobre todo para evitar la dificultad que supone encontrar el valor óptimo de estos parámetros, se optó por la siguiente aproximación: (ver Figura 4.3)

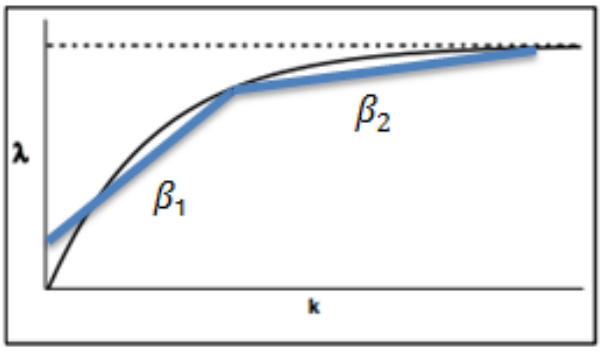

Figura 4.3. Aproximación del programa de compresión

Con esta aproximación, en lugar de tener que configurar $\gamma, \hat{\lambda}, \lambda^*$ y k, sólo tendremos que determinar la pendiente de estas dos rectas. Por lo que el programa de compresión quedará como sigue:

$$\lambda_{k+1} = \beta_1 \lambda_k$$

en el primer tramo, y

$$\lambda_{k+1} = \beta_2 \lambda_k$$

para el segundo. Se ha considerado como punto de inflexión o punto de división entre tramos la mitad del número máximo de iteraciones permitidas. En el caso del programa de enfriamiento bastaría con considerar una temperatura inicial lo suficientemente alta como para aceptar, al principio de la búsqueda, casi libremente las soluciones del entorno.

b) Número de iteraciones por temperatura/presión

Investigaciones teóricas del *Simulated Annealing* sugieren que el sistema debe permitir converger a su distribución estacionaria en cada ajuste de la temperatura. Desafortunadamente, el número de iteraciones necesarias para acercarse a la distribución estacionaria es exponencial al tamaño de problema según van Laarhoven y Aarts (1987). En la práctica, la longitud de la cadena de Markov a cada temperatura suele estar relacionada con el tamaño del problema o incluso con el espacio de soluciones posibles. Bonomi y Lutton (1984) propusieron establecer el número de iteraciones en cada la temperatura basándose en una función dependiente de forma polinómica del tamaño del problema. Un enfoque alternativo es determinar la longitud de la kth cadena de Markov no permitiendo una reducción de la temperatura hasta que un número mínimo de transiciones

hayan sido aceptadas o se lleven a cabo un número máximo de iteraciones. De esta forma Kirkpatrick et al. (1983) establecieron que la longitud de la k cadena de Markov tendría que depender directamente de k.

Podemos concluir que la mejor forma de establecer este número de iteraciones sería usar el tamaño del problema como una directriz inicial y después tratar de ajustar el número de iteraciones a través de pruebas experimentales.

c) Criterio de parada

A lo largo de los últimos años se han propuesto numerosos criterios de parada. Dentro de éstos, los más usados han sido los propuestos por Bonomi y Lutton o Jonhson et al. Bonomi y Lutton (1984) fijaban el número de valores de temperatura para el que se ejecutaba el algoritmo mientras Johnson et al. (1989) establecían la parada del algoritmo cuando el porcentaje de movimientos aceptados caía por debajo de un umbral para un número de iteraciones dado. Nosotros vamos a implementar un híbrido de estos dos enfoques mediante el control de la movilidad del algoritmo, mientras que un número mínimo de iteraciones será requerido. Precisamente, pararemos la ejecución del algoritmo cuando la mejor ruta encontrada no se haya actualizado en las últimos K iteraciones, por lo que se requiere un mínimo de ellas (K).

4.2.4. Parámetros recomendados

	$n \leq 20$	$n > 20$
Temp.ini	400	10000000
λ_0	6	10
N	90	2000
α	0.995	0.96
β_1	1.0001	1.0005
β_2	1.00001	1.00001
K	200000	400000
Iter max	20000000	
Tolerancia	0,000000001	

Figura 4.4. Parámetros recomendados algoritmo Compreseed-Annealing Simulated (TSPTW)

4.3. Métodos de penalización.

4.3.1. Introducción

En vista de los buenos resultados que proporcionaron las heurísticas *Simulated Annealing* y *Late Acceptance Hill-Climbing* para el problema del TSP, se pretende probar la eficacia de las mismas para esta variación del problema principal, el TSPTW.

A priori, puede pensarse que esto es inviable dado que estas heurísticas por sí solas no pueden tener en cuenta la función coste y la función desviación a un mismo tiempo. Por lo que, basándonos en los problemas de optimización multiobjetivo, se procederá a la introducción de un factor de penalización, que llamaremos ρ.

El problema del TSPTW, como explicábamos en el apartado anterior, está compuesto de dos componentes principales: un Problema del Viajante de Comercio y un Problema de *scheduling* o programación. Usando este método de penalización debemos descomponer estas dos partes. Así, utilizaremos la notación $f_1(x)$ para referirnos a la desviación y $f_2(x)$ para referirnos al coste.

Nuestro objetivo será minimizar $f_1(x)$ y $f_2(x)$ de forma simultánea, dado que si escogiéramos minimizar primero una de las dos la solución global podría no ser la mejor. A la vista de la Figura 4.3 es fácil comprobar esto último; si elegimos minimizar primero $f_1(x)$, el algoritmo podría elegir cualquiera de los puntos A, B o C (dado que presentan el mismo valor) sin tener en cuenta que el mejor de éstos es el A. Por ello, resulta indispensable que esta minimización se lleve a cabo de forma conjunta.

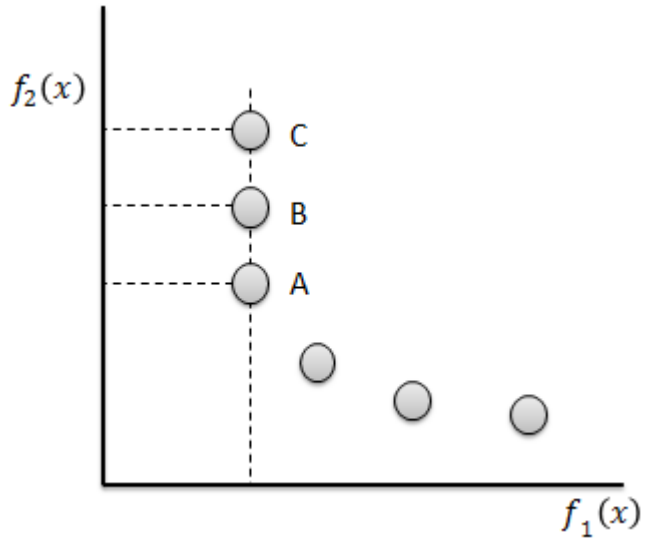

Figura 4.5. Importancia minimización simultánea

Para poder minimizar ambos objetivos de forma simultánea uno tendrá que prevalecer sobre el otro. En nuestro caso daremos prioridad a la desviación, pues sólo tendremos soluciones factibles si la desviación es 0. Para priorizar esta última multiplicaremos la función coste por un factor de penalización relativamente pequeño. De forma que, hasta que la desviación no sea 0, los cambios en el coste no tendrán un importancia excesivamente grande. Nuestra función objetivo será por tanto:

$$f_1(x) + \rho\, f_2(x) = \sum_{i=0}^{n} \max\{0,\, D_{pi} - l_{pi}\} + \rho \sum_{i=0}^{n} c(a_i) =$$

$$desviación + \rho\, coste$$

Nótese que esto sería equivalente a escoger un ρ relativamente grande y escoger una función objetivo de la forma:

$$f_1(x) + \frac{1}{\rho} f_2(x)$$

Como hemos comentado, probaremos este método con la heurística *Simulated Annealing* y *Late Acceptance Hill-Climbing*

4.3.2. Parámetros recomendados

- Simulated Annealing

	n ≤ 20	n > 20
Temp.ini	500	700
N	250	300
α	0.99	0.999
K	200000	2000000
ρ	0.01	
Iter max	6000000	
Tolerancia	0,0000001	

Figura 4.6. Parámetros recomendados algoritmo Simulated Annealing (TSPTW)

- Late Acceptance Hill-Climbing

	n ≤ 20	n > 20
LFA	6000	15000
K	200000	1200000
ρ	0.001	
Iter max	2000000	

Figura 4.7. Parámetros recomendados algoritmo Late Acceptance Hill-Climbing (TSPTW)

4.4. Análisis de resultados obtenidos

Los anteriores algoritmos, y como se ha comentado a lo largo del proyecto, se han implementado en lenguaje Mosel y ejecutados en un ordenador Intel CORE con velocidad 3.60GHz y una memoria instalada (RAM) de 16GB. En este caso el algoritmo fue probado para un total de 95 instancias propuestas y resueltas por Dumas et al. [16]. Aunque cómo veremos más adelante sólo quedarán reflejadas las primeras 50 (20 y 40 nodos), las cuales piden ser clasificadas en un total de 10 grupos, ya que dentro de cada grupo de nodos vamos a hacer una subdivisión en función de la amplitud de las ventanas de tiempo, como comentábamos en el capítulo 3.

Para interpretar con precisión el rendimiento de estos algoritmos, cada instancia se ejecutará 5 veces. Los resultados obtenidos se presentarán en dos tablas.

En la primera de ellas, para cada instancia se recogerá la mejor solución obtenida y la solución media (de las 5 ejecuciones). Además, estos datos se compararán también con los resultados del método exacto y los mejores valores conocidos hasta el momento. De forma que al final de dicha tabla, se presentará el porcentaje medio de mejora (+) o empeoramiento (-) del método con respecto a la solución obtenida con el método exacto. Éste es calculado como:

$$\Delta = \frac{(solución\ método\ exacto - solución\ media\ heurística)}{solución\ \ media\ heurística} * 100$$

En la segunda tabla, se recogerá el tiempo medio empleado por la CPU para obtener el resultado (en segundos) y la desviación típica. Para ésta última, nos basaremos en la siguiente fórmula: $\sigma = \sqrt{\sum_{i=1}^{5}(x_i - \bar{x})^2/n}$, donde x_i es el valor objetivo en la ejecución i para una instancia dada, $\bar{x}$ es la media del valor objetivo para dicha instancia y n el número de ejecuciones llevadas a cabo. Además, debido a la dificultad de estos problemas, se añade una nueva fila que indica la probabilidad de convergir hacía una solución factible para los distintos tamaños de problema.

Lo que se pretende demostrar, como hicimos en el caso del TSP, es cuál de éstos podría ser descrito como un "buen" algoritmo, entendiendo por ello que cumpla las siguientes propiedades:

- **Eficiente**. Un esfuerzo computacional realista para obtener la solución.
- **Eficaz.** La solución debe de estar, en promedio, cerca del óptimo.
- **Robusto.** La probabilidad de obtener una mala solución (lejos del óptimo) debe ser baja.

En cada tabla se destacarán en negrita aquellos valores que sean superiores al resto de las heurísticas. Además, si se supera el resultado obtenido por el método exacto pondremos junto con el valor un asterisco *. En caso de superar la mejor solución obtenida hasta el momento, esta será representada con dos asteriscos (**). Por último, para aquellos casos que no se encuentre una solución factible aparecerá un guión en la casilla correspondiente.

Antes de presentar las tablas, cabe recalcar que para cada grupo sólo se han considerado las soluciones factibles a la hora de realizar estos análisis. Por ello, podremos encontrarnos programas que converjan pocas veces a una solución factible pero en el caso de hacerlo son bastante robustos, mostrando siempre un resultado similar.

- **20 nodos:**

- Valores objetivos obtenidos

					M. Penalización			
	Mejor solución conocida	**M. exacto**	**CAS**		**SA**		**LAHC**	
Archivo		Coste	Coste medio	Mejor valor	Coste medio	Mejor valor	Coste medio	Mejor valor
n20w20.001	378	378	378	378	378	378	378	378
n20w20.002	286	286	286	286	286	286	286	286
n20w20.003	394	394	394	394	394	394	396	394
n20w20.004	396	396	396	396	396	396	398,4	396
n20w20.005	352	352	352	352	352	352	352	352
n20w40.001	254	254	254	254	254	254	254	254
n20w40.002	333	333	333	333	333	333	337,2	333
n20w40.003	317	317	317	317	317	317	317	317
n20w40.004	388	388	388,2	388	388	388	388	388
n20w40.005	288	288	288	288	288	288	288	288
n20w60.001	335	335	335	335	335	335	335	335
n20w60.002	244	244	244	244	244	244	244	244
n20w60.003	352	352	352	352	352	352	354,4	352
n20w60.004	280	280	-	-	280	280	280	280
n20w60.005	338	338	339	338	**338,6**	338	339,2	338
n20w80.001	329	329	329,2	329	329,2	329	329,6	329
n20w80.002	338	338	338	338	342,6	338	338	338
n20w80.003	320	320	320	320	320	320	320	320
n20w80.004	304	304	304	304	304	304	304	304
n20w80.005	264	264	264	264	265,4	264	264	264
n20w100.001	237	237	240	237	237	237	237	237
n20w100.002	222	222	222	222	222	222	222	222
n20w100.003	310	310	313	310	311,6	310	311,6	310

n20w100.004	349	349	349	349	349	349	349,8	349
n20w100.005	258	258	258,8	258	258	258	258	258
Probabilidad concurrencia			0,96		1		1	
Δ(%)			-0,123		**-0,105**		-0,172	

Tabla.4.1.Comparativa de resultados obtenidos por las diferentes heurísticas para n =20 (TSPTW)

- Tiempo y desviación

				M. Penalización			
	M. exacto	**CAS**		**SA**		**LAHC**	
Archivo	Tiempo	Tiemp. medio	σ	Tiemp. medio	σ	Tiemp. medio	σ
n20w20.001	0,046	3,326	0,000	4,464	0,000	6,084	0,000
n20w20.002	0,084	3,366	0,000	4,150	0,000	6,552	0,000
n20w20.003	0,203	3,400	0,000	3,958	0,000	6,440	4,211
n20w20.004	0,180	3,400	0,000	3,964	0,000	6,200	6,063
n20w20.005	0,094	3,264	0,000	4,016	0,000	5,708	0,000
n20w40.001	1,303	3,360	0,000	4,652	0,000	5,616	0,000
n20w40.002	0,125	3,316	0,000	4,658	0,000	5,488	18,56
n20w40.003	0,453	3,368	0,000	4,492	0,000	4,388	0,000
n20w40.004	0,282	3,414	0,042	3,862	0,000	6,246	0,000
n20w40.005	0,546	3,484	0,000	4,488	0,000	5,846	0,000
n20w60.001	4,860	3,526	0,000	4,944	0,000	4,944	0,000
n20w60.002	1,162	3,756	0,000	4,358	132,04	5,918	0,000
n20w60.003	0,976	3,400	0,000	4,302	0,000	5,850	6,063
n20w60.004	7,801	7,664	0,000	4,798	0,000	5,082	0,000
n20w60.005	1,819	3,598	0,421	5,108	0,379	5,014	0,358
n20w80.001	8,199	3,550	0,042	4,908	0,042	6,154	0,168
n20w80.002	3,861	3,758	0,000	4,456	9,642	5,170	0,000
n20w80.003	0,899	3,708	0,000	5,146	0,000	5,606	0,000
n20w80.004	3,441	3,498	0,000	4,516	0,000	5,630	0,000
n20w80.005	6,420	3,252	0,000	4,874	2,063	5,578	0,000
n20w100.001	9,268	4,146	2,947	6,694	0,000	6,272	0,000
n20w100.002	8,754	3,952	0,000	6,842	0,000	5,842	0,000
n20w100.003	18,644	4,444	34,358	4,758	1,432	5,484	1,432
n20w100.004	7,981	4,044	0,000	5,320	0,000	5,686	0,674
n20w100.005	8,354	4,146	0,674	7,028	0,000	5,920	0,000
Tiemp. medio	3,830	**3,766***		4,830		5,708	
Desviación(σ)	-	1,539		5,824		**1,501**	

Tabla 4.2.Comparativa de tiempo y varianzas por las diferentes heurísticas para n =20 (TSPTW)

A diferencia del TSP, en el que desde un principio unos métodos atisbaban un mejor rendimiento que otros, en este problema particular la elección del mejor método entre los analizados parece ser más complicada, ya que no hay un método superior al resto. El más cercano a la solución óptima es el método de *Simulated Annealing* (SA), el que menos tiempo tarda en encontrar la solución es el método de *Compressed-Annealing Simulated* (CAS) y por último, el más robusto es el *Late Acceptance Hill-Climbing* (LAHC).

- **40 nodos:**

- Valores objetivos obtenidos

					M. Penalización			
	Mejor solución conocida	**M. exacto**	**CAS**		**SA**		**LAHC**	
Archivo		Coste	Coste medio	Mejor valor	Coste medio	Mejor valor	Coste medio	Mejor valor
n40w20.001	500	500	-	-	500	500	-	-
n40w20.002	552	552	552	552	552	552	552	552
n40w20.003	478	478	478	478	478	478	478	478
n40w20.004	404	404	404	404	404	404	404	404
n40w20.005	499	499	499	499	499	499	499	499
n40w40.001	465	465	465,8	465	**465**	465	467	465
n40w40.002	461	461	461	461	461	461	461	461
n40w40.003	474	474	474	474	474	474	474	474
n40w40.004	452	452	452	452	452	452	458	452
n40w40.005	453	453	453	453	453	453	453	453
n40w60.001	494	470	494,75	494	494	494	494	494
n40w60.002	470	470	470	470	470	470	470	470
n40w60.003	408	408	408	408	408	408	408	408
n40w60.004	382	382	386,2	382	385,2	385	**384,8**	382
n40w60.005	328	328	328	328	328	328	328	328
n40w80.001	395	395	397	395	**395**	395	395,6	395
n40w80.002	431	454	431	431	431	431	431	431
n40w80.003	412	410	417,6	412	**416,4**	412	416,6	412
n40w80.004	417	417	420,25	417	**417**	417	418,25	417
n40w80.005	344	344	344	344	344	344	344	344
n40w100.001	429	453	430,6	430	**429,8***	429	430,4	429
n40w100.002	358	370	364	361	**361,6***	358	360,8	358
n40w100.003	364	361	364	364	364	364	364	364
n40w100.004	357	357	370,8	357	358,8	357	361	357
n40w100.005	377	393	377	377	377	377	378,6	377
Probabilidad concurrencia			0,824		**0,848**		0,816	
Δ(%)			-0,416		**-0,143**		-0,266	

Tabla.4.3.Comparativa de resultados obtenidos por las diferentes heurísticas para n =40 (TSPTW)

- Tiempo y desviación

				M. Penalización			
	M. exacto	**CAS**		**SA**		**LAHC**	
Archivo	Tiempo	Tiemp. medio	σ	Tiemp. medio	σ	Tiemp. medio	σ
n40w20.001	0,460	-	-	190,02	0,000	-	-
n40w20.002	1,615	36,210	0,000	189,98	0,000	92,333	0,000
n40w20.003	1,342	45,760	0,000	192,36	0,000	104,59	0,000
n40w20.004	1,029	45,695	0,000	191,52	0,000	98,920	0,000
n40w20.005	1,521	33,760	0,000	174,45	0,000	92,910	0,000
n40w40.001	0,884	40,102	1,600	166,36	0,000	82,982	4,000
n40w40.002	7,552	42,488	0,000	168,14	0,000	93,515	0,000
n40w40.003	1,373	45,883	0,000	163,14	0,000	91,763	0,000
n40w40.004	7,264	45,726	0,000	173,11	0,000	91,830	8,786
n40w40.005	22,10	45,505	0,000	140,11	0,000	97,713	0,000
n40w60.001	49,79	42,405	1,162	171,26	0,000	101,00	0,000
n40w60.002	34,39	45,853	0,000	172,75	0,000	95,658	0,000
n40w60.003	64,34	45,698	0,000	172,20	0,000	96,418	0,000
n40w60.004	150,9	45,736	2,315	172,43	0,400	106,80	1,470
n40w60.005	14,47	46,250	0,000	171,98	0,000	94,367	0,000
n40w80.001	175,2	45,680	1,095	172,47	0,000	91,732	1,200
n40w80.002	300,0	46,220	0,000	169,91	0,000	97,010	0,000
n40w80.003	300,0	45,826	2,871	167,75	3,878	101,65	3,980
n40w80.004	150,5	45,815	1,830	172,37	0,000	93,260	1,936
n40w80.005	300,0	45,706	0,000	171,89	0,000	99,278	0,000
n40w100.001	300,0	45,710	1,200	173,11	0,400	99,106	2,800
n40w100.002	300,0	45,772	2,966	140,11	1,960	100,35	2,400
n40w100.003	300,0	45,638	0,000	171,62	0,000	91,770	0,000
n40w100.004	300,0	45,848	16,522	171,97	2,227	96,330	2,280
n40w100.005	300,0	45,778	0,000	175,45	0,000	105,26	1,960
Tiemp. medio	123,385	**44,378***		171,865		95,524	
Desviación(σ)	-	1,315		**0,335**		1,284	

Tabla 4.4.Comparativa de tiempo y desviación obtenidos por las diferentes heurísticas para n =40 (TSPTW)

Para el caso de 40 nodos, parece que el método de *Simulated Annealing* presenta un comportamiento mejor al resto; al menos en lo que a proximidad al óptimo y robustez se refiere. Si bien es verdad, que necesita aproximadamente el doble de tiempo que el resto de heurísticas estudiadas, aunque en un problema con una dificultad de este calibre, el tiempo que tarda en dar con la solución no sería descabellado (menos de 3 minutos). Entre los dos restantes, si el factor tiempo fuera el más importante elegiríamos el CAS; sin embargo ofrece una menor robustez y soluciones peores que LAHC.

En el caso de los grupos de 60 y 80 nodos se han omitido las tablas porque la probabilidad de convergencia al óptimo era baja y muy aleatoria, lo que podría distorsionar los resultados. Ya dijimos que este problema tendría una dificultad mayor que el TSP, y los resultados lo han demostrado. Mientras que con el TSP llegamos sin dificultad a obtener soluciones para un tamaño de n = 100, en el caso del TSPTW a partir de 40 nodos este hallazgo se complica.

Al igual que hicimos con el problema del TSP se quiere determinar cuál es el algoritmo que mejor rendimiento presenta para la resolución de este problema particular, para ello y como decíamos al comienzo de este apartado; un algoritmo tenía que tener tres cualidades para ser considerado un "buen" algoritmo: esfuerzo computacional realista, solución cerca del óptimo y robustez. Por ello, se resumen los datos recogidos durante este estudio en los siguientes tres gráficos que dan buena medida de estas tres cualidades. En ellos, se representa para cada modelo: el tiempo (Figura 4.8), la desviación típica (Figura 4.9) y el porcentaje de mejora o empeoramiento (Figura 4.10) en función del tamaño del problema.

- **Esfuerzo computacional realista**: cómo podemos ver en la Figura 4.8, todas las heurísticas, salvo *Simulated Annealing*, requieren un menor tiempo para encontrar la solución que el método exacto. Dentro de estas la que ofrecería un mejor comportamiento frente al factor tiempo sería la heurística del *Compressed-Annealing Simulated* (CAS).

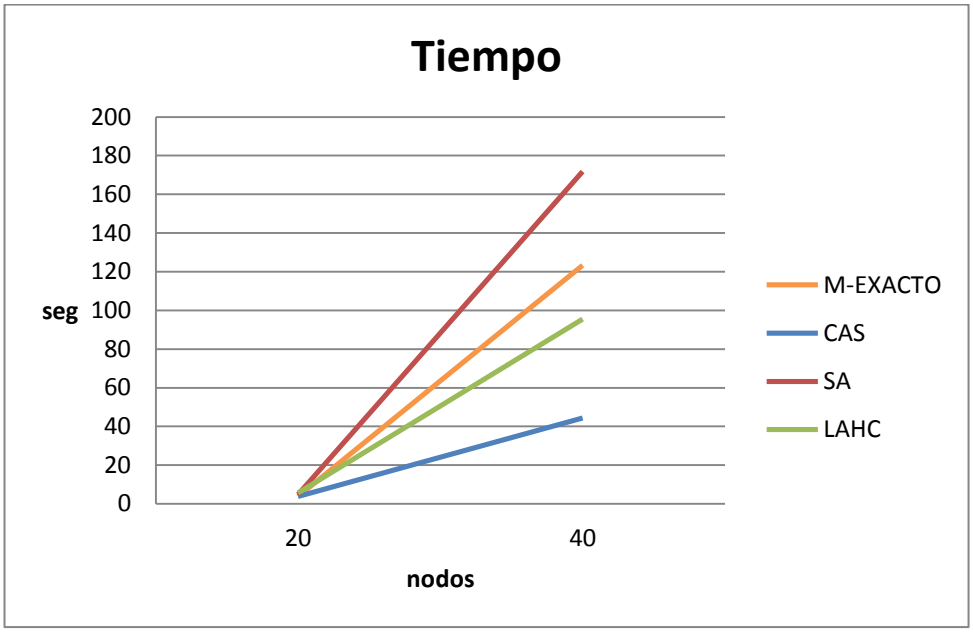

Figura 4.8. Tiempos obtenidos para los distintos métodos (TSPTW).

- **Robustez**: Para medir la robustez nos vamos a basar en la desviación típica. Se persigue obtener una desviación típica próxima a 0. La heurística *Simulated Annealing* es, de nuevo, la más alejada del ideal que se persigue. Sin embargo, para n = 40, es la heurística más robusta. Esto puede ser, entre otras cosas, debido a lo comentado en un principio; sólo se han considerado los resultados de las soluciones factibles por lo que necesitaremos una gráfica más que muestre esta probabilidad de acercarse a soluciones factibles para esclarecer estos resultados. En cuanto a los resultados de las otras heurísticas en lo que a robustez, como vemos en la Figura 4.9, son muy parecidos; de hecho los valores de la desviación típica para los distintos tamaños se encuentran "solapados".

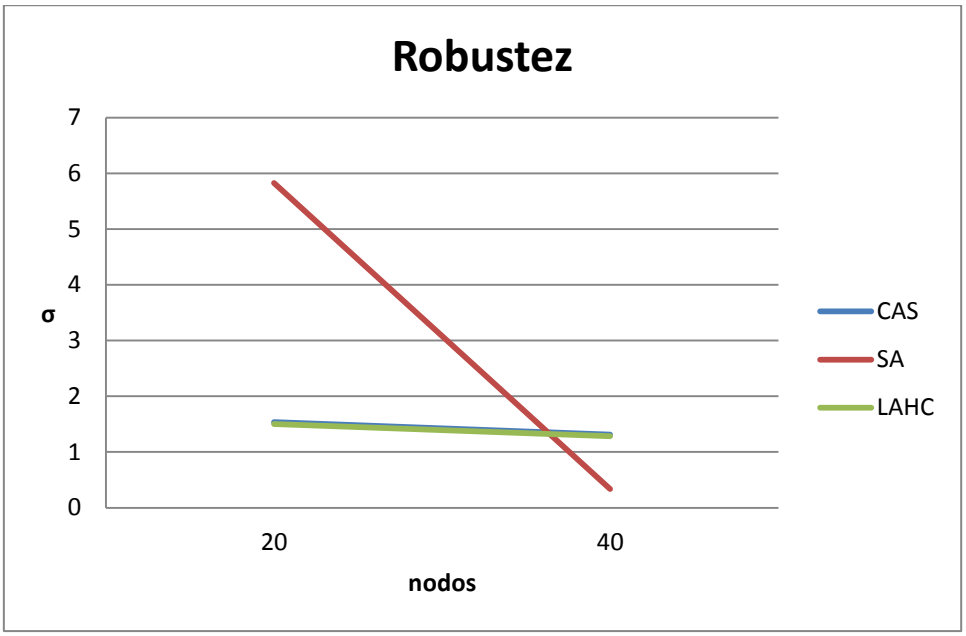

Figura 4.9. Desviación típica para los distintos métodos (TSPTW)

- **Solución cercana al óptimo**: Para estos tamaños de problema relativamente pequeños es difícil que las heurísticas consigan mejorar las soluciones que obtiene el método exacto. Sin embargo, vemos que aunque el *Simulated Annealing* no mostraba buenas aptitudes en cuanto a tiempo y robustez, si lo hace a la hora de ofrecer buenas soluciones.

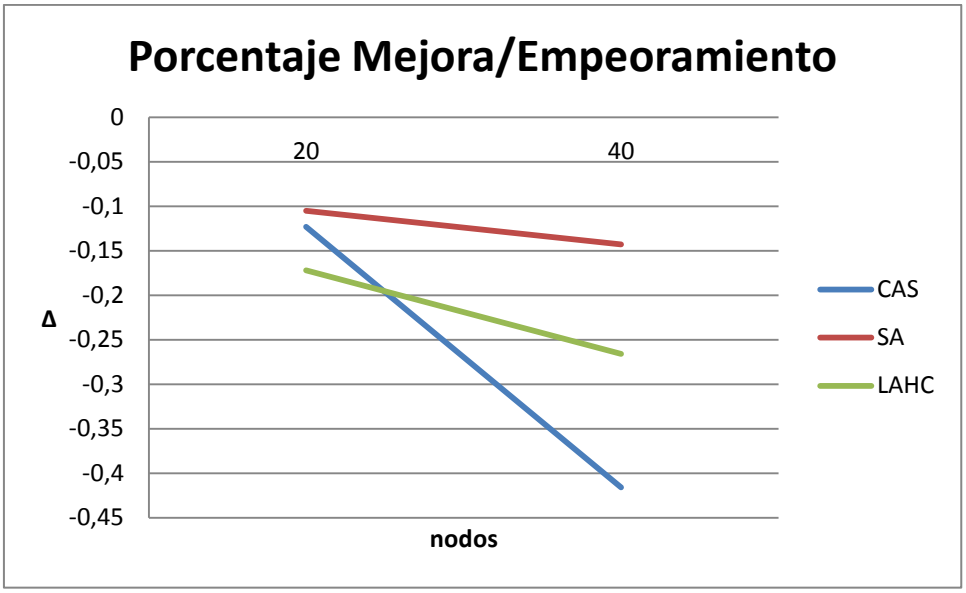

Figura 4.10. Porcentaje mejora/empeoramiento para los distintos métodos (TSPTW)

Los promedios finales de estos datos son:

	CAS	SA	LAHC
Tiempo	**24,072**	88,347	50,616
σ	1,4271	3,0795	**1,3927**
Δ	-0,2695	-0,1243	**-0,2191**

Figura 4.11. Promedios finales (TSPTW)

Por tanto, si lo que buscábamos es determinar el mejor algoritmo de estos tres, antes de ello, tendremos que decidir cuál es nuestro criterio de elección principal: tiempo, robustez o mejores soluciones. En este caso, y puesto que hemos realizado también el análisis de probabilidad de convergir hacía una solución óptima, utilizaremos esta probabilidad como criterio de desempate cuantitativo.

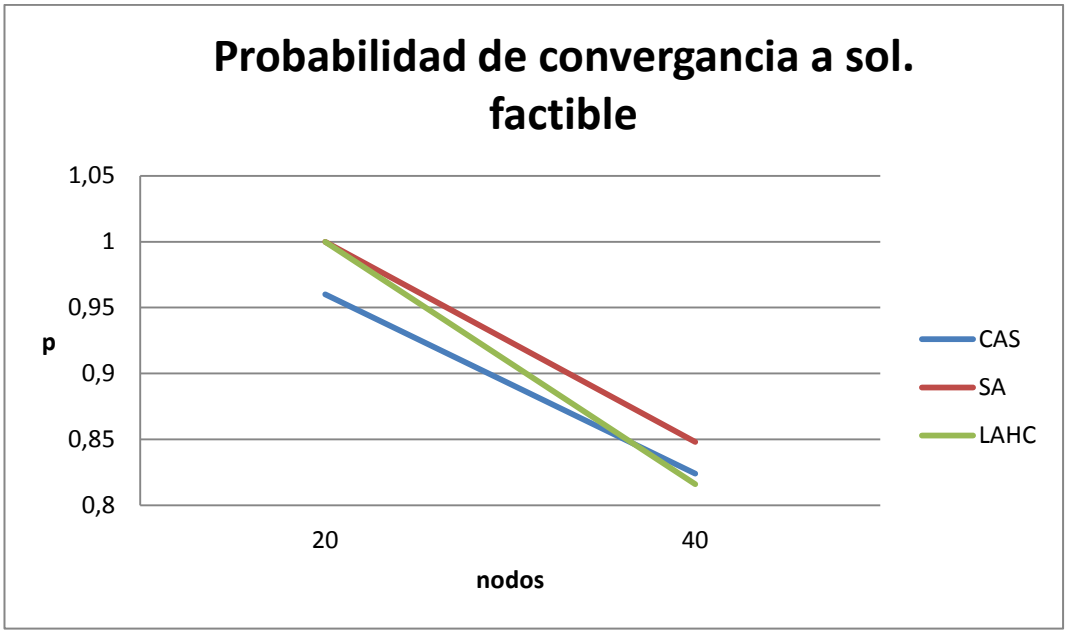

Figura 4.12. Probabilidad de convergencia a solución factible para los distintos métodos (TSPTW)

Como vemos, el método más fiable es el *Simulated Annealing*; seguido del *Late Acceptance Hill-Climbing.*

Así, podríamos hacer una elección meramente cuantitativa a raíz de los resultados obtenidos en los cuatro parámetros (ver Figura 4.13), de forma que elegiríamos como mejor método la heurística *Simulated Annealing* combinada con el factor de penalización ρ. No obstante, se recalca de nuevo la importancia de determinar el criterio de elección principal a la hora de optar por un método u otro: tiempo, robustez o mejores soluciones.

	seg	Puesto	σ	Puesto	Δ	Puesto	p	Puesto
SAC	24,07	1º	1,427	2º	-0,295	3º	0,892	3º
SA	88,34	3º	3,0796	3º	-0,124	1º	0,924	1º
LAHC	50,62	2º	1,393	1º	-0,219	2º	0,908	2º

Figura 4.13. Ranking de las heurísticas para los distintos parámetros analizados

4.5. Conclusiones

En este tema se han presentado 3 métodos para la resolución del TSPTW: dos de ellos aplicados por primera vez a este problema como son el *Simulated Annealing* y *Late Acceptance Hill-Climbing*, y otro, *Compressed-Annealing Simulated*, del que se esperaba fuera la panacea, a juzgar por los resultados que mostraban los investigadores que propusieron dicho método [31]. Nuestro principal objetivo era comprobar si esta novedosa heurística ofrecía tal rendimiento, así como probar suerte con las heurísticas que mejor habían rendido para el TSP añadiendo el factor de penalización.

Tras los análisis efectuados en el apartado anterior se puede afirmar que, aunque los resultados obtenidos no fueron los esperados; puesto que se pretendía llegar a los 80 nodos y la dificultad de resolución de este problema nos dejó "a mitad de camino", la aproximación que consideramos realizar del algoritmo SA y LAHC con el factor de penalización presentó un rendimiento mejor de lo esperado, logrando superar al laureado y reconocido *Compressed-Annealing Simulated* en tres de los cuatro parámetros analizados, como veíamos en la Figura 4.13.

Además y como comentábamos también en el análisis del capítulo 2, no sólo son importantes los resultados obtenidos, sino el número de parámetros de entrada que ha de ser "tuneados", debido a que a mayor número de parámetros mayor es el tiempo requerido para conseguir ajustarlos. Por ello, la principal ventaja que podemos corroborar de la aproximaciones que hemos realizado con el factor de penalización frente a CAS es el reducido número de parámetros de entrada y la facilidad para "tunearlos". Mientras que SAC tiene 9 parámetros de entrada, SA tiene 7 y LFA tan sólo 4; de los cuales sólo 1 depende del tamaño del problema.

Así, aunque las conclusiones no son tan claras como en el caso del TSP, y todos los métodos obtienen un rendimiento similar, nosotros nos decantaríamos por la utilización de SA y LAHC con el factor de penalización; no sólo por los resultados que reporta sino por su relativa facilidad de "tuneado".

Capítulo 5

Conclusiones

Cuando se habla de la resolución de problemas de optimización combinatoria pocas veces se plantea la posibilidad de usar nuevos métodos, ya que tenemos la creencia de que aquellas heurísticas que llevan años siendo aplicadas son las que reportan mejores soluciones. Tal es el caso de GRASP, Algoritmos Genéticos o Colonia de Hormigas. Sin embargo, la principal conclusión que sacamos de este proyecto es que hay muchas heurísticas poco conocidas que presentan resultados similares o incluso mejores que éstas últimas.

Nuestro principal descubrimiento ha sido el algoritmo *Late Acceptance Hill-Climbing*; el cual a pesar de su sencillez de programación y de funcionamiento; ha resultado ser un método que presenta resultados tan buenos o incluso mejores que el conocido Recocido Simulado.

Otro de los grandes protagonistas de éste proyecto ha sido la aproximación del algoritmo *Late Acceptance Hill-Climbing* y *Simulated Annealing* para la resolución del TSPTW gracias al parámetro de penalización ρ. El rendimiento que se ha obtenido con ello ha sido incluso superior a la heurística reconocida como la que mejor se adapta y mejor rendimiento presenta para la resolución de esta variante del TSP, la conocida como *Compressed-Annealing Simulated.*

Aunque el proceso de investigación seguido para la elaboración del proyecto ha sido en ocasiones lento e incluso frustrante, los resultados son muy satisfactorios; se abre una puerta nueva a la investigación en este campo. Concretamente plantearíamos las siguientes líneas de investigación futura:

- En cuanto al algoritmo *Late Acceptance Hill-Climbing*, debido a los excelentes resultados que ha obtenido para el TSP, se podría estudiar su aplicación para diferentes problemas de optimización combinatoria dado que es un algoritmo bastante inexplorado (Google Académico muestra 856.000 publicaciones con la palabra clave "Simulated Annealing" y sólo 61 publicaciones sobre “Late Acceptance Hill-Climbing”).

- En cuanto a la resolución del TSPTW se ha visto que aún queda mucho campo de mejora. Además de probar nuevos métodos, sobre todo se propone continuar la búsqueda de algoritmos que reporten buenas soluciones para el caso del TSP y después usar el parámetro de penalización como vimos en el capítulo anterior, hasta dar con un algoritmo que consiga resolver problemas de un tamaño relativamente grande.
  Además, no existen publicaciones, al menos que sepamos, sobre la resolución del TSPTW con el algoritmo *Simulated Annealing* ni tampoco con *Late Acceptance Hill-Climbing*. Por lo que se podría plantear el perfilado de estos dos métodos que se han propuesto en el presente proyecto.

- Por último, a la hora de decidir en qué heurísticas de búsqueda local íbamos a basar este proyecto tuvimos que prescindir de muchas de ellas que posiblemente hubiesen dado buenos resultados. Por ello, se propone continuar con esta línea de investigación de nuevas heurísticas de búsqueda local como son: Demon Algorithms [43], Old Bachelor Accepting [24], Step Counting Hill-Climbing [9] o Nosing Methods [10], entre otras.

Así, animando al lector a continuar con estas líneas de investigación propuestas, ponemos punto y final a este proyecto.

BIBLIOGRAFÍA

[1] ANAS ARRAM, M., ZAKREE, M. *Comparative Study of Meta-Heuristic Approaches for Solving Traveling Salesman Problems*. Asian Journal of Applied Sciences 7 (7), 662-670, Knowledgia Review, Malaysia, 2014

[2] ANTOSIEWICZ, M., KOLOCH G., KAMINSKI, B. *Choice of best possible metaheuristic algorithm for the travelling salesman problem with limited computational time: quality, uncertainty and speed*. Journal of Theoretical and Applied Computer Science, vol. 7 (1), 46-55, 2013.

[3] ASCHEUER, N., FISCHETTI, M., GRÖTSCHEL, M. *Solving the Asymmetric Travelling Salesman Problem with time windows by branch-and-cut*. Math. Program., Ser. A 90: 475-506, 2001.

[4] B. CARLTON, W., WESLEY BARNES, J. *Solving the traveling-salesman problem with time windows using tabu search*. IIE Transactions, Vol 28, 617-629, 1996.

[5] BALL, M.O., MAGNATI, T.L., MONMA C.L., NEMHAUSER, G.L. *Network routing*. Elsevier, 1995.

[6] BAYRAM, HÜSAMETTING, SAHIN, R. *A New Simulated Annealing Approach for Travelling Salesman Problem.* Mathematical and Computational Applications, Vol 18 (3), 331-22, 2013

[7] BURKE, E.K., KENDALL, G. *Search methodologies*. Introductory Tutorials in Optimization and Decision Support Techniques. Springer, 2005.

[8] BURKE, E.K, BYKOV, Y. *The Late Acceptance Hill-Climbing Heuristic.* Department of Computing Science and Mathematics, University of Stirling, Technical Report CSM-192, 2012.

[9] BYKOV, Y., PETROVIC, S. *An initial study of a novel Step Counting Hill Climbing Heuristic applied to timetabling problems*. 6th Multidisciplinary International Scheduling Conference: Theory & Applications (MISTA), Gent, Belgium, 2013.

[10] CHARON, I., HUDRY O. *The noising methods: A generalization of some metaheuristics*. European Journal of Operational Research 135, 86-101, 2001.

[11] Clay Mathematic Institute (2000). Millenium problems. http://www.claymath.org/millennium-problems/rules-millennium-prizes

[12] COOK, W. J. *In pursuit of the Traveling Salesman*. Princeton University Press, 2012.

[13] COOK, W.J, CUNNINGHAM W.H, PULLEYBLANK W.R, SCHRIJVER, A. *Combinatorial Optimization*. John Wiley & Sons, 1998.

[14] DÍAZ, A., GLOVER, F. M. GHAZIRI, H. GONZÁLEZ, J.L., LAGUNA, M., T.TSENG, F. *Optimización Heurística y Redes Neuronales en Dirección de Operaciones e Ingeniería*. Editorial Paranifo, 1995.

[15] DUECK, G., SCHEUER, T. *Threshold Accepting: A General Purpose Optimization Algorithm Appearing Superior to Simulated Annealing*. Journal of Computational Pyhisics 90, 161-175, 1990.

[16] DUMAS, Y., DESROSIERS, J., GELINAS, E., SOLOMON, M. *An optimal algorithm for the traveling salesman problem with time windows*. Operations Research 43 (2), 367-371,1995.

[17] FERREIRA DA SILVA, R., URRUTIA, S. *A General VNS heuristic for the traveling salesman problem with time windows*. Discrete Optimization 7, 203-211, 2010.

[18] FICO. Xpress-Mosel (User Guide), 2013.

[19] FILIPPO, F., LODI, A., MILANO, M. *A Hybrid Exact Algorithm for the TSPTW*. INFORMS Journal on Computing, Vol 14 (4), 403-417, 2002.

[20] GENDREAU, M., HERTZ, A., LAPORTE, G., STAN, M. *A generalized insertion heuristic for the traveling salesman problem with time windows.* Operations Research, Vol 43 (3), 1998.

[21] GENDREAU, M., POTVIN, J-Y. *Handbook of metaheuristics*, Springer, 2010.

[22] GUTIN G., P.PUNNEN A. *The Traveling Salesman Problem and Its Variations*. Kluwer Academic Publishers, 2002.

[23] JANIAK, A., LICHTENSTEIN, M. *Advanced algorithms in combinatorial optimization*. Wroclaw University of Technology, Wroclaw, 2011.

[24] KAHGN A.B., ALBERT TSAO, C,W. *Old Bachelor Acceptance: A New Class of Non-Monotone Threshold Accepting Methods.* Dept. of Computer Sciences, Los Angeles CA, 1995.

[25] LANGEVIN, A., DESROCHERS, M., DESROSIERS J., GÉLINAS, S., SOUMIS, F. *A Two-Commodity Flow Formulation for the Traveling Salesman and the Makespan Problems with Time Windows*. NETWORKS, Vol 23, 631-640, John Willey & Sons, 1993.

[26] LAWLER, E.L, LENSTRA, J.K., RINNOOY KAN, A.H.G. SHMOYS, D.B. *The traveling salesman problem*. John Wiley & Sons, 1984.

[27] LEE, KWANG Y., EL-SHARKAWI, M. *Modern Heuristic Optimization Techniques. Theory and Applications to power systems*. Wiley-Interscience, 2008.

[28] MARTÍNEZ RÍOS, F., FRAUSTO-SOLIS, J., *An Hybrid Simulated Annealing Threshold Accepting Algorithm for Satisfiability Problems using Dynamically Cooling Schemes*. Universidad Panamericana, Ciudad de México, 2008.

[29] MARTÍ CUNQUERO, R. *Algoritmos heurísticos en optimización combinatoria*. Departamento de Estadística e Investigación Operativa, Universidad de Valencia.

[30] OHLMANN J. W., BEAN, J.C., HENDERSON S.G. *Convergence in Probability of Compressed Annealing.* Mathematics of Operations Research, vol 29 (4), 837-860, 2004.

[31] OLHMANN, J. W., THOMAS, B.W. *A Compressed-Annealing Heuristic for the Traveling Salesman Problem with Time Windows*. INFORMS Journal on Computing, Vol.19, 80-90, 2007.

[32] ORMAN, A. J, WILLIAMS, H.P. *A survey of different integer programming formulations of the traveling salesman problem. Operational Research working papers*, LSEOR 04.67. Department of Operational Research, London School of Economics and Political Science, London, 2004.

[33] PATALIA T.P., KULKARNI, G.R., *Comparative Analysis of Threshold Acceptance Algorithm, Simulated Annealing Algorithm and Genetic Algorithm for Function Optimization*. Global Journal of Researches in Engineering, Vol 12 (1), 2012.

[34] PIRLOT, M. *General local search methods*. European Journal of Operational Research, 92, 493-511, 1996.

[35] REINELT, G. *The Traveling Salesman. Computational Solutions for TSP Applications*. Springer-Verlag, 1994.

[36] SÁEZ AGUADO, JESÚS. *Introducción a Xpress-Mosel*. Departamento de Estadística e Investigación Operativa, Universidad de Valladolid, 2012.

[37] SANJEEB, D., GÜNLÜK, O., LODI, A., TRAMONTANI, A. A *Time Bucket Formulation for the TSP with Time Windows*. INFORMS Journal on Computing, Vol 24 (1), 132-147, 2010.

[38] SCHNEIDER, J.J., KIRKPATRICK, S. *Stochastic Optimization*. Springer, 2006.

[39] SOLOMON M.M., *Algorithms for the vehicle routing and scheduling problems with time window constraints*. Operations Research, Vol 35 (2), 1987.

[40] http://www.yuribykov.com/LAHC/

[41] WANG, H-J., TING, C-J. *A Threshold Accepting Algorithm for the Uncapacitated Single Allocation p-Hub Median Problem*. Journal of the Eastern Asia Society for Transportation Studies, Vol.8, 2009.

[42] WOLFER CALVO, R., *A New Heuristic for the Traveling Salesman Problem with Time Windows*, Transportation Science, Vol 34 (1),113-124, 2000.

[43] WOOD, I., DOWNS, T. *Fast Optimization by Demon Algorithms, Department of Electrical and Computer Engineering, University of Queensland, Australia, 1998*.