Secretaría de Educación Pública
Tecnológico Nacional de México

Instituto Tecnológico de Veracruz

Una propuesta para resolver el problema del agente viajero basado en agrupamiento por cuadrantes y su implementación usando metaheurísticas

T E S I S

PARA OBTENER EL TÍTULO DE:

Ingeniero en Sistemas Computacionales

PRESENTA:

Guillermo Sebastián Medina Palacios

Asesores:

M.C. Rafael Rivera López

Dr. Marco Antonio Cruz Chávez

2017

Dedicatoria

Aquí iría las personas a quienes les dedicaría la obra, asesores, familiares y personas que hayan contribuido económicamente

Resumen

Este trabajo de tesis describe el desarrollo de una técnica para encontrar una solución al problema del agente viajero, que es una referencia clásica en el área de la optimización combinatoria y se utiliza como un caso de prueba estándar para evaluar la efectividad de diversos métodos de optimización.

La técnica propuesta en este documento utiliza una agrupación de soluciones parciales de un problema en cuadrantes del espacio euclidiano con el fin de obtener una solución cercana al óptimo a través de una regla determinística.

La solución inicial encontrada a través de este método es refinada en una etapa subsecuente utilizando un conjunto de metaheurísticas. Los resultados obtenidos indican que el uso de esta estrategia permite encontrar soluciones cercanas al óptimo para el problema del agente viajero.

En [JP85] se indica que el problema del agente viajero es posiblemente el más importante de los problemas de optimización combinatoria ya que fue uno de los primeros en ser clasificados como "duros" técnicamente hablando.

El texto está dividido en los siguientes capítulos:

- **Marco metodológico:** El marco metodológico es el inicio de cada investigación, es una breve introducción que plantea el documento: la descripción del problema, la hipótesis propuesta, los objetivos que pretende alcanzar, las justificaciones de realizarlo, así como los alcances y las limitaciones que tiene del trabajo. En este caso se explicará la problemática del TSP y que tipo de solución se usará para resolver dicho problema.
- **Marco teórico:** Para una mejor comprensión del tema, en este capítulo se describen todos los temas que fueron usados para el desarrollo de esta tesis, desde la definición del problema del agente viajero, así como de conceptos como metaheurística, la teoría de autómatas, etc. La intención de este capítulo es informar al lector y que pueda comprender la solución implementada en los siguientes capítulos. La información obtenida para el desarrollo de este capítulo es respaldada por fuentes de diversas instituciones.

- **Método de agrupamiento basado en cuadrantes:** Éste es el nombre que se le dio al algoritmo para resolver las instancias del problema del agente viajero y en este capítulo se explica la forma en que trabaja. Básicamente, este algoritmo usa técnicas de agrupamiento para dividir el problema en otros más pequeños con la intención de reducir la explosión combinatoria del problema.
- **Documentación del software usado en las pruebas:** En este capítulo se presenta una descripción del programa que se implementó en esta tesis. Se explica de que elementos se compone la interfaz, como funciona y el papel que desempeñan las metaheurísticas. Al final de este capítulo se hace una demostración de cómo se utilizó este programa para el desarrollo de los experimentos descritos en el siguiente capítulo.
- **Pruebas y análisis de resultados:** Por último se usará el programa descrito en el capítulo anterior para realizar experimentos sobre varias instancias del problema del agente viajero conocidos dentro del medio. Estos problemas fueron obtenidos en la página de la Universidad de Heidelberg ([RKH16]). Una vez terminado los experimentos se construyeron gráficas y se compararon los resultados obtenidos con los reportados en la literatura. Adicionalmente se aplica el método propuesto en este trabajo para resolver un grupo de problemas conocidos como "Arte con TSP". Una vez finalizado los capítulos, se presentan las conclusiones de este trabajo, así como las referencias bibliográficas.

Índice

Lista de figuras

Lista de tablas

Lista de códigos

CAPÍTULO 1 : Marco metodológico

1.1 Antecedentes

El problema del agente viajero (TSP por sus siglas en inglés) fue formulado en 1930 y según la teoría de la complejidad es considerado como un problema NP-Completo que, en otras palabras, no puede ser resuelto por métodos convencionales. El TSP está aplicado principalmente para el trazo de rutas vehiculares óptimas, al buscar la distancia más corta de un punto a otro.

Existen distintos enfoques y fuentes de información para el estudio y resolución del TSP. La fuente de información usada para este trabajo será la TSPLIB que es una agrupación de problemas de prueba para TSP para comparar el rendimiento de dichos enfoques. El conjunto de problemas que componen la TSPLIB fueron encontradas en la página de la Universidad de Heidelberg ([RKH16]).

Dentro de los enfoques conocidos se encuentra el uso de metaheurísticas, que han probado ser eficientes para resolver problemas computacionales complejos incluyendo el TSP.

1.2 Planteamiento del problema

El problema inicia bajo la siguiente premisa: "Un viajero quiere pasar por una cantidad específica de lugares recorriendo la menor distancia y en el menor tiempo y regresar al punto de origen pasando por cada ciudad una sola vez".

Este problema, aunque fácil de plantear es difícil resolver, ya que la cantidad de rutas posibles crecen de manera exponencial. En las tablas 1.1 y 1.2 se tiene un ejemplo de las combinaciones que se obtienen en función del número de ciudades pero partiendo del mismo punto de origen.

Tabla 1.1: Rutas posibles con 2 lugares.

1	2
A	B

En el ejemplo de la tabla 1.1 las combinaciones que se pueden realizar si el viaje fuera de 2 puntos sería solo 1; ahora realizando el mismo ejemplo pero calculando el número de rutas de 5 lugares se puede observar en la tabla 1.2 como el número de rutas aumenta.

Tabla 1.2: Ejemplo con 5 lugares.

1	2	3	4	5
A	B	C	D	E
A	B	C	E	D
A	B	D	C	E
A	B	D	E	C
A	B	E	B	C
A	B	E	C	B

1	2	3	4	5
A	C	B	D	E
A	C	B	E	D
A	C	D	B	E
A	C	D	E	B
A	C	E	B	D
A	C	E	D	B

1	2	3	4	5
A	C	B	D	E
A	C	B	E	D
A	C	D	B	E
A	C	D	E	B
A	C	E	B	D
A	C	E	D	B

1	2	3	4	5
A	E	B	C	D
A	E	B	D	C
A	E	C	B	D
A	E	C	D	B
A	E	D	B	C
A	E	D	C	B

Como se acaba de ver en la tabla 1.2 se pueden observar 24 combinaciones posibles a tomar; esta cantidad aumenta de manera exponencial de acuerdo al número de entradas (lugares) que recibe, lo que significa que entre más lugares tenga el problema más tiempo se ocupa para encontrar la ruta más corta.

La resolución del TSP a llamado la atención de la comunidad científica desde su formulación inicial. De acuerdo a [GP02] las aplicaciones del TSP van más allá del problema de planear una ruta ya que se utiliza en varias áreas de conocimiento que incluyen las matemáticas, las ciencias computacionales, la investigación de operaciones, la genética y otras disciplinas de la ingeniería.

De acuerdo a lo anterior existen propuestas para su resolución que abarcan el uso de diversos algoritmos extraídos tanto de la teoría de grafos como de la inteligencia artificial. Estos algoritmos utilizan heurísticas tanto determinísticas según [BT83] como aquellas basadas en la aleatoriedad según [GGRVG85].

TSP es considerado como un problema de optimización que busca dentro de un conjunto de soluciones candidatas (las posibles rutas de viaje) aquella que presente el mejor resultado (en caso del TSP costo menor) respecto a todas las demás. Existen propuestas que consisten en partir de una solución inicial aplicando reglas para mejorar dicha solución, ó aquellas que trabajan con un grupo se soluciones cuya información se vuelve a combinar para generar nuevas soluciones.

Un aspecto importante en los algoritmos para resolver problemas de optimización es definir la solución inicial, generalmente se utiliza una solución inicial completamente aleatoria o una simple secuencia ordenada de ciudades según [Per02].

Por lo anterior, es posible definir el siguiente planteamiento: ¿Es posible encontrar una solución inicial del problema del agente viajero utilizando una técnica de agrupamiento basada en cuadrantes que sea competitiva o esté cercana al óptimo?

1.3 Objetivos

El objetivo general es crear un algoritmo que servirá para obtener una solución inicial a instancias del TSP, que servirá como base para aplicar una serie de metaheurísticas que refinarán la solución con el fin de obtener soluciones cercanas al óptimo.

1.3.1 Objetivos específicos

- Desarrollar un algoritmo que permita obtener una solución cercana a la óptima.
- Hacer uso de los resultados obtenidos por medio del algoritmo anterior aplicándolo con metaheurísticas.
- Crear un programa que sea capaz de realizar los procedimientos anteriores, también que lea y guarde archivos en ".TSPLIB", este tipo de formato son los utilizados para resolver problemas de TSP.
- Comparar lo que se obtuvo por medio del programa con su respectivo benchmark y anotar los resultados tanto el costo de cada problema como de la imagen de la ruta creada por medio del programa.

1.4 Justificación

Aunque ya existen formas de resolver el problema del agente viajero, se aportará con una forma novedosa para encontrar una solución inicial al problema.

También se demostrará que se pueden obtener rutas con resultados cercanos al óptimo usando un algoritmo de agrupamiento combinado con metaheurísticas.

Por último el programa se creará con la finalidad para demostrar de manera gráfica el potencial de los procedimientos ejecutados y observar los cambios. No está de más mencionar que cualquier problema resuelto se puede guardar en un archivo para volverlos a cargar cuando se necesite.

1.5 Hipótesis

Basado en los resultados obtenidos se comprobará que el uso del método de agrupación basado en cuadrantes sirve para obtener resultados cercanos a sus respectivos benchmarks en un corto lapso de tiempo. El uso de metaheurísticas servirá para mejorar el resultado obtenido.

1.6 Alcances

- El uso del algoritmo basado en cuadrantes en conjunción con una serie de metaheurísticas proporcionará soluciones cercanas al óptimo para el TSP.
- El uso del programa, al poder leer archivos en formato .tsp permite que cualquier otro investigador del campo de TSP pueda utilizarlo para sus respectivas pruebas, un ejemplo están en los problemas encontrados en la página de la universidad de Waterloo según cite[MONALISA], esto se describirá más a detalle en el tema 5.6.

1.7 Limitaciones

- El algoritmo creado no supera a los mencionados benchmark.
- El uso del programa creado solo está limitado a problemas que se encuentren en archivos de la TSPLIB, además que solo resuelve problemas de tipo euclidiano en 2 dimensiones.

CAPÍTULO 2 : Marco teórico

2.1 Complejidad de los algoritmos

Según [Gim05] un algoritmo es un procedimiento o método de cálculo (con reglas bien determinadas) que conduce a la resolución de cualquier 'instancia' de un problema específico (en un número finito de pasos), la complejidad o costo de un algoritmo es una medida de los recursos (tiempo, memoria) que requiere su ejecución en función del tamaño de los datos de entrada (independiente de la arquitectura de la computadora).

Una vez desarrollado un algoritmo que funciona correctamente, es necesario definir criterios para medir su rendimiento o comportamiento. Estos criterios se centran principalmente en su simplicidad y en el uso eficiente de los recursos.

Según [GV98], respecto al uso eficiente de los recursos, éste suele medirse en función de dos parámetros: el espacio, es decir, la memoria que utiliza, y el tiempo, lo que tarda en ejecutarse. Ambos representan los costos que supone encontrar la solución al problema planteado mediante un algoritmo. Dichos parámetros van a servir además para comparar algoritmos entre sí, permitiendo determinar el más adecuado de entre varios que solucionan un mismo problema.

El tiempo de ejecución de un algoritmo va a depender de diversos factores como son los datos de entrada que se le suministre, la calidad del código generado por el compilador para crear el programa objeto, la naturaleza y rapidez de las instrucciones máquina del procesador concreto que ejecute el programa y la complejidad intrínseca del algoritmo. Hay dos estudios posibles sobre el tiempo:

- Uno que proporciona una medida teórica (a priori), que consiste en obtener una función que acote el tiempo de ejecución del algoritmo para unos valores de entrada dados.
- Otro que ofrece una medida real (a posteriori), consistente en medir el tiempo de ejecución del algoritmo para unos valores de entrada dados y una computadora específica.

Ambas medidas son importantes puesto que si bien la primera ofrece estimaciones del comportamiento de los algoritmos de forma independiente de la computadora en donde serán implementados y sin necesidad de ejecutarlos, la segunda representa las medidas reales del comportamiento del algoritmo. Estas medidas son funciones temporales de los datos de entrada.

Se entiende por tamaño de la entrada el número de componentes sobre los que se va a ejecutar el algoritmo. Por ejemplo, la dimensión del vector a ordenar o el tamaño de las matrices a multiplicar.

Teóricamente el tiempo de ejecución debe indicar el número de instrucciones ejecutadas por una computadora imaginaria, por tanto se deben buscar medidas simples y abstractas, independientes de la computadora a utilizar.

La unidad de tiempo a la que deben hacer referencia estas medidas de eficiencia no puede ser expresada en segundos o en otra unidad de tiempo concreta, pues no existe una computadora estándar al que puedan hacer referencia todas las medidas. Se entenderá como tiempo de ejecución a la cantidad de instrucciones en relación al número de entradas que reciba.

Como se puede observar en la figura 2.1 hay 3 fórmulas donde el eje horizontal sería la cantidad de parámetros a introducir y el eje vertical es el tiempo de ejecución.

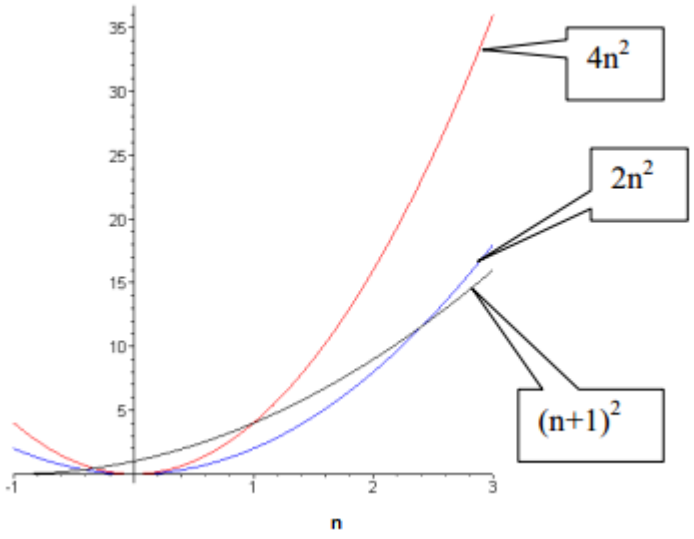

Figura 2.1: Complejidad de los algoritmos.

2.1.1 Orden de los algoritmos

Por lo general la mayoría de los problemas tienen un parámetro de entrada que es el número de datos que hay que tratar, esto es, N. La cantidad de recursos del algoritmo es tratada como una función de N. Según [Ziv07] de esa manera puede establecerse un tiempo de ejecución del algoritmo que suele ser proporcional a una de las siguientes funciones:

- 1 **(Tiempo de ejecución constante):** Significa que la mayoría de las instrucciones se ejecutan una cantidad constante de tiempo.
- $logN$ **(Tiempo de ejecución logarítmico):** Se puede considerar como una gran constante. La base del logaritmo (la más común es 2) cambia la constante, pero no demasiado. El programa es más lento cuanto más crezca *N*, pero es inapreciable, pues *logN* no se duplica hasta que *N* llegue a N^2.
- N **(Tiempo de ejecución lineal):** Estos algoritmos puede resumirse como un bucle que se repite un número *N* fijo de veces. Un caso en el que *N* valga 40, tardará el doble que otro en que *N* valga 20, pero si *N* vale 30 tendría un tiempo intermedio de los *N* anteriores.

- $NlogN$ **(Tiempo de ejecución en** $NlogN$ **):** Es común encontrarlo en algoritmos como Quick Sort y otros del estilo divide y vencerás. Si N se duplica, el tiempo de ejecución es ligeramente mayor del doble.
- N^2 **(Tiempo de ejecución cuadrático):** Suele ser habitual cuando se tratan pares de elementos de datos, como por ejemplo un bucle anidado doble. Si *N* se duplica, el tiempo de ejecución aumenta cuatro veces. El peor caso de entrada del algoritmo Quick Sort se ejecuta en este tiempo.
- N^3 **(Tiempo de ejecución cúbico):** Como ejemplo se puede dar el de un bucle anidado triple. Si *N* se duplica, el tiempo de ejecución se multiplica por ocho.
- 2^N **(Tiempo de ejecución exponencial):** No suelen ser muy útiles en la práctica por el elevado tiempo de ejecución. El problema de la mochila resuelto por un algoritmo de búsqueda exhaustiva es un ejemplo. Si N se duplica, el tiempo de ejecución crece de manera exponencial (de ahí el nombre).
- $n!$ **(Tiempo de ejecución factorial):** La mayoría de ellos son intratables debido a que su exponente es igual al número de entradas que recibe. Si se aplica búsqueda exhaustiva a TSP, el tiempo de ejecución sería factorial.

En la figura 2.2 se presenta una gráfica con el orden de crecimiento de cada uno de los conceptos en relación a su número de entradas y el tiempo que se tarda en resolverla.

2.1.2 Clasificación de los problemas

Los problemas que tratan de resolver con algoritmos se pueden clasificar de diferentes perspectivas. De acuerdo con [GJ79] se tienen estas posibles clasificaciones:

1. **Por su naturaleza:** Se toma en cuenta la dificultad para resolver los algoritmos.
   - **No computables:** Problemas que no admiten solución algorítmica, los problemas no computables que se basan en decisiones se llaman indecidibles.
   - **Tratables:**Son los problemas para los cuales existen algoritmos de complejidad

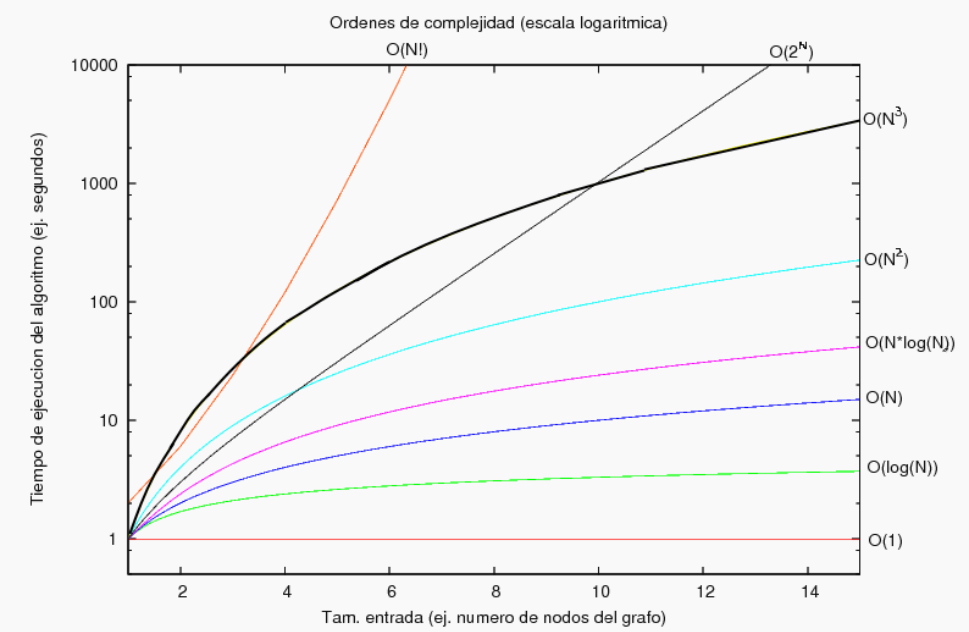

Figura 2.2: Ordenes de complejidad.

polinomial para resolverlos.

- **No tratables:** Son problemas que admiten solución y para los cuales de manera comprobada no pueden ser resueltos por algoritmos de complejidad polinomial.

2. **Por el tipo de respuesta:** Es el determinar o responder SI o NO a una pregunta dada o solicitada.
   - **Problemas de decisión:** Es el determinar o responder SI o NO a una pregunta dada o solicitada.
   - **Problemas de búsqueda:** Tienen como objetivo encontrar, en caso que exista, una estructura que verifique las restricciones del problema, dicha estructura es denominada de solución viable.
   - **Problemas de optimización:** Comparte el mismo objetivo que los problemas de búsqueda pero este tipo en particular tiene que encontrar una estructura que optimice un criterio pre-definido es decir, debe encontrar la mejor solución.
3. **Por su Tratabilidad:** Los problemas que admiten solución son clasificados de acuerdo

a la complejidad que presentan los algoritmos para resolverlos.

- **La clase P:** Está constituida por todos los problemas que pueden ser resueltos por algoritmos de complejidad polinomial en una máquina convencional (máquina de Turing deterministica).
- **La clase NP:** Es el conjunto de problemas que pueden ser resueltos en tiempo polinómico por una máquina de Turing no determinista, también contiene todos los problemas pertenecientes a las clases P, estos problemas tienen algoritmos ineficientes
- **La clase NP-Completo:** Se dice que un problema es NP-completo si todos los problemas de la clase NP pueden reducirse a él, son los problemas más difíciles de la clase de problemas NP. Debido a la reducción, si se encuentra un algoritmo en tiempo polinomial para un problema NP-completo, se sabría que todos los problemas de la clase NP requieren un tiempo polinómico para resolverse.

2.1.3 Ejemplos de problemas NP-Completos

Existen varios problemas NP-Completos, entre los más representativos se encuentran el problema de satisfacibilidad booleana y el problema del agente viajero

El problema de satisfacibilidad booleana: De acuerdo a [Wil94] el problema de satisfacibilidad booleana (SAT) fue el primer problema identificado como perteneciente a la clase de complejidad NP-completo en 1971. Según [Wil94] el objetivo de SAT se enfoca a encontrar una asignación de valores a un conjunto de variables lógicas que son utilizadas para construir una fórmula lógica, de forma que dicha asignación produzca que ésta sea verdadera (una asignación satisfacible).

Normalmente la fórmula lógica se debe presentar en su forma normal conjuntiva (FNC) donde las variables lógicas se agrupan en claúsulas, que son una conjunción de literales. Como ejemplo considerando el siguiente conjunto de variables x_1, x_2, x_3, x_4 se tiene una fórmula

lógica en su forma normal conjuntiva (FNC).

$$(x_1 \vee \neg x_3 \vee x_4) \wedge (\neg x_2 \vee x_3 \vee \neg x_4)$$

Como se muestra en la tabla 2.1 se verifican todas las combinaciones posibles de entradas en la que 10 de las 16 posibles generan una asignación verdadera lo que lo convierte en satisfacible. Por el contrario, si no existiera ninguna asignación que generara un resultado verdadero, entonces la fórmula sería insatisfacible.

Tabla 2.1: Ejemplo de satisfacibilidad booleana.

x_1	x_2	x_3	x_4	$(x_1 \vee \neg x_3 \vee x_4) \wedge (\neg x_2 \vee x_3 \vee \neg x_4)$
V	V	V	V	V
F	V	V	V	V
V	F	V	V	V
V	V	F	V	F
V	V	V	F	V
F	F	V	V	V
F	V	F	V	F
F	V	V	F	F
V	F	F	V	V
V	F	V	F	V
V	V	F	F	F
F	F	F	V	V
F	F	V	F	F
F	V	F	F	V
V	F	F	F	V
F	F	F	F	V

Problema de la mochila: Según [Kar72] el problema de la mochila es un problema que modela la situación de llenar una mochila limitada a cierta capacidad de peso y con cierta cantidad de objetos con diferentes pesos y valores cada uno. Los objetos que se pongan en la mochila deben de maximizar el valor total, sin superar el peso máximo permitido.

Como se puede ver en la figura 2.3 se muestra el ejemplo del problema de la mochila: dada una mochila con una capacidad de 15 kg para llenar con cajas de distinto peso y valor, se debe de elegir la mayor cantidad de cajas que pueden caber en la mochila sin exceder el peso permitido pero a la vez no sobre tanto espacio.

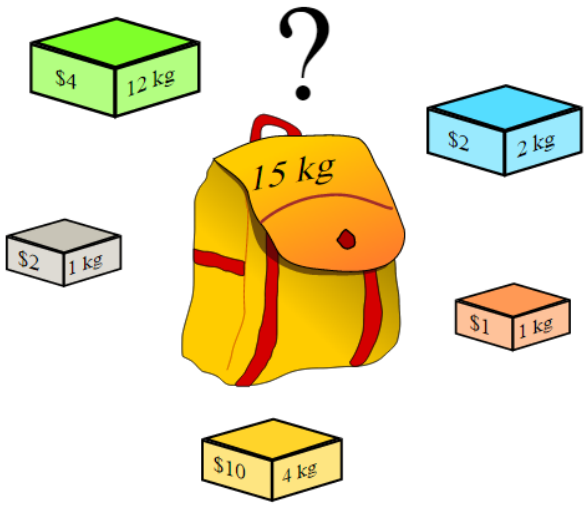

Figura 2.3: Ejemplo del problema de la mochila.

Los datos del problema se pueden expresar en términos matemáticos.:

- Los objetos están numerados por el índice i variando de 1 a N objetos (ítem).
- De cada tipo de ítem se tienen q_i ítems disponibles, donde q_i es un entero positivo que cumple $1 \leq q_i \leq \infty$.
- Los números w_i y v_i representan el peso y el valor del número i.
- La capacidad de la mochila se denomina en esta fórmula W.
- La solución al problema vendrá dado por la secuencia de variables x_n donde el valor de x_i indica cuantas copias se meterán en la mochila del tipo de ítem i

Con todo esto se puede formular matemáticamente de la siguiente forma:

$$\text{maximizar} \sum_{i=1}^{N} v_i x_i$$
$$\text{sujeto a} \sum_{i=1}^{n} w_i x_i \leq W$$
$$1 \leq q_i \leq \infty.$$

2.2 Problema del agente viajero (TSP)

En TSP se tiene un número de nodos (ciudades, localidades, tiendas, empresas, etc.) que deben ser visitados por una entidad (persona, agente viajero, automóvil, avión, autobús, etc.), sin visitar 2 veces el mismo nodo.

Si se consideran 3 nodos (a, b y c) por visitar, entonces se tendrá una función de combinaciones sin repetición que se calcula con una función factorial ($n!$) es decir, se tendrán 6 posibles soluciones: abc, acb, bac, bca, cab, cba; para el caso de 4 nodos serian 24 combinaciones, para 10 nodos habría 3628800 combinaciones y así sucesivamente. En la figura 2.4 se muestra otro ejemplo de los diferentes recorridos que puede dar un problema TSP, uno muestra un recorrido con un ineficiente resultado y otro con el resultado más eficiente.

De acuerdo a [DFJ54] la primera solución reportada para resolver una instancia del TSP fue en 1954, cuando Dantzig, Fulkerson y Johnson resolvieron una instancia de 49 ciudades.

2.2.1 Características de TSP

De acuerdo con [Len97], TSP se encuentra clasificado como un problema de optimización combinatoria, es decir, en el problema en el que intervienen cierto número de variables donde cada variable puede tener N diferentes valores y cuyo número de combinaciones es de carácter factorial.

Para obtener una solución a una instancia del TSP se emplean diferentes métodos entre

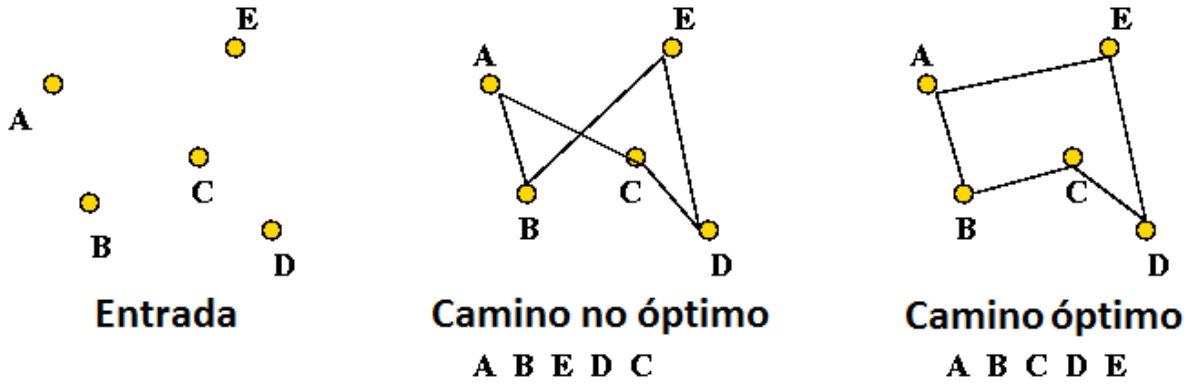

Figura 2.4: Ejemplo de una instancia de TSP con 2 posibles soluciones.

los cuales los principales se denominan heurísticas cuyo objetivo es generar soluciones de buena calidad en tiempos de cómputo mucho más pequeños (soluciones óptimas tiempo – respuesta). Las variables que han sido empleadas por la mayoría de los investigadores que dan solución a TSP son:

- **Tiempo de recorrido entre ciudades:** Horas, minutos, días, semanas, etc.
- **Distancia de recorrido entre ciudades:** Metros, kilómetros, millas, milímetros, etc.
- **Costo de traslado:** Dinero, desgaste de las piezas, gasto de energía, etc.

Las variables que se pueden adoptar dependen de cada problema, por ejemplo:

- **Circuitos electrónicos:** Cantidad de soldadura utilizada, menor espacio entre los puntos de soldadura de los circuitos, evitar el cruce entre las líneas de soldadura, tiempo de fabricación, distribución de los circuitos, entre otras.
- **Control de semáforos:** Número de semáforos (nodos), tiempo de traslado entre semáforos, cantidad de autos que pasan por un punto, entre otras variables.
- **Previsión del tránsito terrestre:** Puntos en una ciudad, cantidad de vehículos, tiempo de traslado, tipos de vehículos, horas pico, correlación entre variables, regresión lineal, etc.

- **Entrega de productos:** Peso de las entregas, número de entregas, nodos (domicilios) a visitar, recorridos, tiempos de traslado, tipo de vehículo, etc.
- **Estaciones de trabajo:** Secuencia de actividades, lugar de las herramientas (nodos), Tipo de herramientas, tiempo de uso, etc.
- **Edificación:** Puntos de edificación (construcciones), distancia entre las construcciones y los insumos, vehículos (grúas, camiones de volteo, etc.), cantidad de combustible que emplean, etc.

2.2.2 Algoritmo base

El TSP se puede modelar como un grafo cuyas aristas son los posibles caminos que puede seguir la entidad para visitar todos los nodos [ÖKL09], y cuyo algoritmo se puede representar como en el pseudocódigo 2.1.

Código 2.1 Algoritmo base del TSP.

```
* Definir el número de nodos, su posición y el costo por cada arista (i, j)
    donde i = ciudad 1 y j = ciudad 2.
* Elegir el nodo inicial i.
* Si el nodo más cercano no se ha visitado, visitar nodo j.
* Actualizar lista de nodos visitados.
* CostoTotal = CostoTotal + costo(i, j).
* Nodo i = nodo j.
* Repetir punto 3 hasta haber visitado todos los nodos.
```

2.2.3 Aplicaciones

El TSP se puede emplear en cualquier situación que requiere seleccionar nodos en cierto orden que reduzca los costos:

- **Reparto de productos:** Mejorar una ruta de entrega para seguir la más corta.
- **Transporte:** Mejorar el recorrido de caminos buscando la menor longitud.
- **Robótica:** Resolver problemas de fabricación para minimizar el número de desplazamientos al realizar una serie de perforaciones en un circuito impreso.

- **Turismo y agencias de viajes:** Aun cuando los agentes de viajes no tienen un conocimiento explícito del TSP, las compañías dedicadas a este giro utilizan un software que hace todo el trabajo. Estos paquetes son capaces de resolver instancias pequeñas del TSP.
- **Horarios de transportes laborales y/o escolares:** Estandarizar los horarios de los transportes es claramente una de sus aplicaciones, tanto que existen empresas que se especializan en ayudar a las escuelas a programarlos para optimizarlos en base a una solución del TSP.
- **Inspecciones a sitios remotos:** Ordenar los lugares que deberá visitar un inspector en el menor tiempo.
- **Secuencias:** Se refiere al orden en el cual n trabajos tienen que ser procesados de tal forma que se minimice el costo total de producción.

2.2.4 Técnicas de solución

El TSP puede resolverse de diferentes maneras:

- **Enumeración de todas las soluciones factibles:** Es decir, listar todas las posibles soluciones al problema, calcular sus costos asociados e identificar, por comparación, cuál es la solución con el costo más conveniente.
- **Métodos exactos:** Intentan descartar familias enteras de posibles soluciones, tratando así de acelerar la búsqueda y llegar a una óptima. Los que más se usan para resolver el TSP son Ramificación y Acotamiento, y Ramificación y Corte.
- **Heurísticas:** Son métodos obtienen buenas soluciones en tiempos de cómputo muy cortos, aunque sin garantizar la solución única.

En la actualidad investigadores han propuesto diferentes estrategias para dar solución a TSP, de las cuales se pueden mencionar algunas técnicas empleadas:

- **Algoritmos genéticos:** La solución consiste en encontrar un individuo cuya combinación de genes (cada gen es una variable), den solución al problema de visitar todas las

ciudades una vez. Otra solución es que cada gen es una ciudad y cuyo orden dependerá del orden en que serán visitadas.

- **Redes neuronales:** Una red neuronal simula las conexiones entre los nodos (lugares por visitar), y cada recorrido por las diferentes neuronas genera al final un camino que involucra el tour por todas las ciudades visitadas una sola vez.
- **Colonia de hormigas (ACO):** Las hormigas encuentran el camino más corto entre 2 puntos; para TSP, se considera el punto de inicio como el punto final, de esta manera, las hormigas deben recorrer todas las ciudades en un circuito, sin pasar 2 veces por la misma ciudad.
- **Búsqueda Tabú:** Consiste en buscar el vecino próximo cuyos costos de traslado del nodo actual al siguiente sea el de menor costo en cuanto al uso de recursos.
- **Combinación de propuestas:** Las técnicas de Inteligencia Artificial se pueden combinar para crear metaheurísticas conformando diferentes soluciones tales como: algoritmos genéticos con redes neuronales.

2.3 Metaheurísticas

El siguiente tema abordara sobre las metaheurísticas, sus características y principales ejemplos.

2.3.1 Concepto de heurística

Según [ZE81] un heurístico es un "procedimiento simple, a menudo basado en el sentido común, que se supone que ofrecerá una buena solución (aunque no necesariamente la óptima) a problemas difíciles, de un modo fácil y rápido". Las heurísticas se utilizan, por ejemplo, cuando no existe un método exacto de resolución, cuando existe un método exacto que consume mucho tiempo para ofrecer la solución óptima, cuando existen limitaciones de tiempo o como paso intermedio para obtener una solución inicial para la aplicación de otra técnica. Según [Sil80] se propone la siguiente clasificación de métodos de resolución mediante heurísticas:

- **Métodos constructivos:** Se caracterizan por construir una solución definiendo difer-

entes partes de ella en sucesivos pasos.

- **Métodos de descomposición:** Dividen el problema en varios más pequeños y la solución se obtiene a partir de la solución de cada uno de estos.
- **Métodos de reducción:** Tratan de identificar alguna característica de la solución que permita simplificar el tratamiento del problema.
- **Métodos de manipulación del modelo:** Obtienen una solución del problema original a partir de otra de otro problema simplificado (con menos restricciones, linealizando el problema, etc.)
- **Métodos de búsqueda por entornos:** En las que se parte de una solución inicial a la que se realizan modificaciones en sucesivas iteraciones para obtener una solución final. En cada iteración existe un conjunto de soluciones vecinas candidatas a ser nueva solución en el proceso. En este grupo se encuadran las técnicas metaheurísticas.

2.3.2 Características principales de las metaheurísticas

Las técnicas metaheurísticas son procedimientos de búsqueda que tampoco garantizan la obtención del óptimo del problema considerado y que también se basan en la aplicación de reglas relativamente sencillas.

A diferencia de las heurísticas, las técnicas metaheurísticas tratan de escapar de óptimos locales orientando la búsqueda en cada momento dependiendo de la evolución del proceso de búsqueda. [SH99] dice que las metaheurísticas tienen como características:

- **Ser ciegas:** No saben si llegan a la solución óptima. Por lo tanto, se les debe indicar cuándo deben detenerse.
- **Ser algoritmos de aproximación:** Por tanto no garantiza la obtención de la solución óptima.
- **Aceptan ocasionalmente soluciones ineficientes:** Sin embargo pueden ser de utilidad para acceder a nuevas regiones y escapar de óptimos locales.
- **Ser relativamente sencillos:** Todo lo que se necesita es una representación adecuada

del espacio de soluciones, una solución inicial (o un conjunto de ellas) y un mecanismo para explorar el campo de soluciones.

- **Ser generales:** Prácticamente se pueden aplicar en la resolución de cualquier problema de optimización de carácter combinatorio. Sin embargo, la definición de la técnica será más o menos eficiente en la medida en que las operaciones tengan relación con el problema considerado.
- **Ser impredecibles:** La regla de selección depende del instante del proceso y de la historia hasta ese momento. Si en dos iteraciones determinadas la solución es la misma, la nueva solución de la siguiente iteración no tiene por qué ser necesariamente la misma.

Aunque las soluciones que ofrecen los técnicas metaheurísticas no son las óptimas y, en general, ni siquiera es posible conocer la proximidad de las soluciones al óptimo, permiten estudiar problemas de gran complejidad de una manera sencilla y obtener soluciones suficientemente buenas en tiempos razonables.

De acuerdo a [BR03] las metaheurísticas se pueden clasificar según:

- Si están inspiradas en procesos naturales
- Si son dinámicas o estáticas
- Si manejan memoria
- Si están basados en poblaciones y manejan varias soluciones por proceso
- Si utilizan técnicas de agrupamiento

2.3.3 Algoritmo voraz (Greedy)

Los algoritmos voraces (del inglés greedy) reciben esta denominación por las siguientes razones según [CLR01]:

- Siempre escoge el mejor candidato para formar parte de la solución: Aquel que tenga mejor valor de la función objetivo, lo que constituye el cumplimiento de cierto criterio goloso de selección.

- Esta elección es única e inmodificable dado que no analiza más allá los efectos de haber seleccionado un elemento como parte de la solución. No deshacen una selección ya realizada: una vez incorporado un elemento a la solución permanece hasta el final y cada vez que un candidato es rechazado, lo es permanentemente.

Se sabe sin embargo la calidad de los algoritmos voraces está en relación con las características de las instancias que pretenden resolver: puede arrojar muy buenos resultados para determinadas instancias del problema pero para otras no. Otro inconveniente que presentan es que se estancan en óptimos locales de las funciones que pretenden optimizar y quizá no analizan vecindades más allá del criterio goloso por lo que pueden estar dejando de considerar al óptimo global.

A continuación el pseudocodigo 2.2 que muestra cómo funciona:

Código 2.2 Pseudocódigo base para el algoritmo greedy.

```
/* Algoritmo de Greedy*/
Inicio
  S=Solucion Inicial
  Mientras (Condición establecida){
   S  = Se genera una solución nueva en base a la inicial
    if (S  > S){
        S = S
    }
  }
  retorno S
Fin;
```

A diferencia de otros algoritmos éste no almacena soluciones adicionales y entre sus virtudes se encuentra en la simplicidad que tiene y por tanto, menor manejo de recursos, sin embargo su defecto se encuentra en el hecho de que no toma decisiones alejadas de la solución inicial, si ésta no cumple con su objetivo se deshace el cambio anterior y continua con el procedimiento hasta que cumpla con la condición establecida.

2.3.4 Búsqueda local

Según [DP05], la idea básica de los algoritmos de búsqueda local es iniciar con una solución inicial ya sea generada aleatoriamente o hallada con algún otro algoritmo; una vez obtenida

se realiza una transformación aleatoria de sus valores para mejorar la solución. Dicha transformación se repite hasta que ya no se pueda mejorar más la solución o bien, que cumpla con ciertas condiciones.

En la figura 2.5 se puede ver un ejemplo de una gráfica formada por las ciudades (v, x, y, z), analizando la ruta de la izquierda se puede crear una nueva ruta reemplazando dos caminos que comprenden los puntos (x, y) y (v, z) haciendo los caminos (x, v) y (y, z) produciendo un nuevo tour que es el que se puede apreciar a la derecha de la misma imagen.

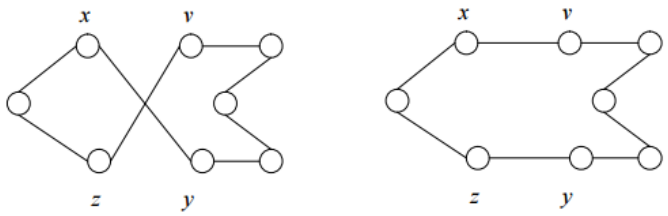

Figura 2.5: Ejemplo de búsqueda local aplicado al TSP.

2.3.5 Recocido simulado

Recocido Simulado (SA por sus siglas en inglés) de acuerdo a [RRTT53] es un método de optimización inspirado en el proceso de templado de metales. El algoritmo de recocido simulado es un método iterativo que inicia con una perturbación de cierto estado s. Mediante un proceso particular genera un estado vecino s' al estado actual. Si la energía, o evaluación, del estado s' es menor que la del estado s, se cambia el estado s por s'. Si la evaluación de s' es mayor que la de s entonces se puede elegir s' en lugar de s con una cierta probabilidad que depende de las diferencias de las evaluaciones $\Delta f = f(s) - f(s')$ y de la temperatura actual del sistema T. La posibilidad de elegir un estado peor al actual es lo que le permite a SA salir de óptimos locales para poder llegar a los óptimos globales. La probabilidad de aceptar elegir un peor estado normalmente se calcula por la distribución de Boltzmann:

$$P(\Delta f, T) = e^{\Delta f/T}$$

La cualidad de SA es que la temperatura va disminuyendo gradualmente conforme avanza la simulación. El código 2.3 representa una versión general del algoritmo de recocido simulado:

Código 2.3 Pseudocódigo base para el algoritmo de recocido simulado.

```
s := GeneraUnaSolucionInicial();
T := T0; g := 0;
mientras (CondicionesParoNoActivas(g, T)) hacer
    s' := Toma Un VecinoAleatorioDe(s);
    si ( f(s') < f(s))
        s := s';
    No
    //se usara la fórmula de temperatura
        si (Random(0, 1.0) < exp((f(s) - f(s'))/T))
            s := s';
        fin si;
    fin si;
g := g + 1; T := Actualiza(g, T);
fin mientras
```

2.3.6 Algoritmo genético

Los algoritmos genéticos (AG) son métodos adaptativos que están basados en la teoría evolutiva de Darwin y la teoría genética de Mendel. En la naturaleza los individuos de una población compiten entre sí en la búsqueda de recursos tales como comida, agua y refugio. Incluso los miembros de una misma especie compiten a menudo en la búsqueda de un compañero. Aquellos individuos que tienen más éxito en sobrevivir y en atraer compañeros tienen mayor probabilidad de generar un gran número de descendientes.

Como se puede ver en la figura 2.6 se tiene un grupo de individuos representado por círculos de colores, en algún momento 2 de estos círculos se juntarán y darán origen a un nuevo círculo, aunque posean algunas de las mismas características que sus padres es un ser único, este nuevo individuo se integrará al grupo y continuará su ciclo.

A continuación se presentará un resumen basado en la idea general del algoritmo genético según [Hol75], [Ree93], [Mic92], [Dav91], [Gol89]. Las soluciones del problema se representan como un conjunto de parámetros llamados genes, los cuales agrupados forman un grupo de valores llamado cromosoma; el alfabeto que se usa para tratar con estos problemas puede ser cualquiera, pero para la siguiente explicación se usará el 1 y el 0. El conjunto de parámetros

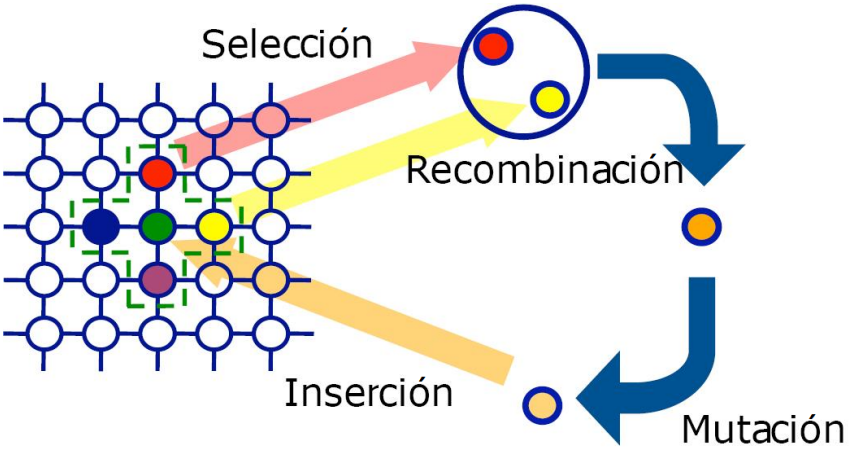

Figura 2.6: Ejemplo de cómo funciona el algoritmo genético.

representando un cromosoma en específico se denomina fenotipo, el fenotipo contiene la información requerida para construir una solución el cual se refiere como genotipo.

La función de adaptación debe ser diseñada para cada problema de manera específica. Dado un cromosoma particular, la función de adaptación le asigna un valor numérico que se supone refleja el nivel de adaptación al problema del individuo representado por el cromosoma. Durante la fase reproductiva se seleccionan los individuos de la población para cruzarse y producir descendientes que constituirán, una vez mutados, la siguiente generación de individuos. La selección de padres se efectúa al azar usando un procedimiento que favorezca a los individuos mejor adaptados, ya que a cada individuo se le asigna una probabilidad de ser seleccionado que es proporcional a su función de adaptación. Es decir, los individuos bien adaptados se escogerán probablemente varias veces por generación, mientras que los pobremente adaptados al problema no se escogerán más que de vez en cuando. Una vez seleccionados los 2 padres, sus cromosomas se combinan utilizando habitualmente los operadores de cruce y mutación. Las formas básicas de dichos operadores se describen a continuación:

- El operador de cruce toma a los 2 padres seleccionados y divide sus genes en una posición escogida al azar para producir dos pares de cromosomas cada uno. Después se intercambian los cromosomas finales para crear una solución nueva (Figura 2.7). Ambos

descendientes heredan genes de cada uno de los padres. Este operador se conoce como operador de cruce basado en un punto.

- El operador de mutación se aplica a cada hijo de manera individual y consiste en la alteración aleatoria (normalmente con probabilidad pequeña) de cada gen componente del cromosoma.

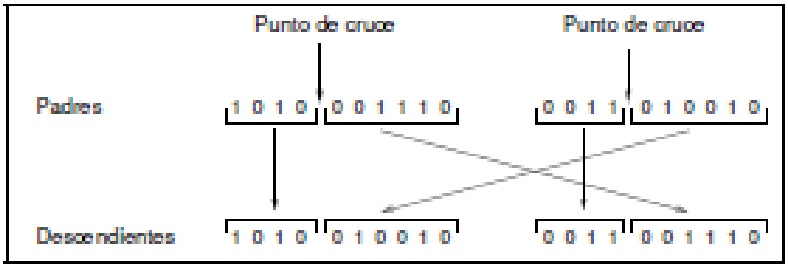

Figura 2.7: Operador de cruce basado en un punto.

La figura 2.8 muestra la mutación del quinto gen del cromosoma. Aunque el operador de cruce es más importante que el operador de mutación por proporcionar una exploración rápida del espacio de búsqueda, el operador de mutación asegura que ningún punto del espacio de búsqueda tenga probabilidad cero de ser examinado y es importante para asegurar la convergencia de un AG.

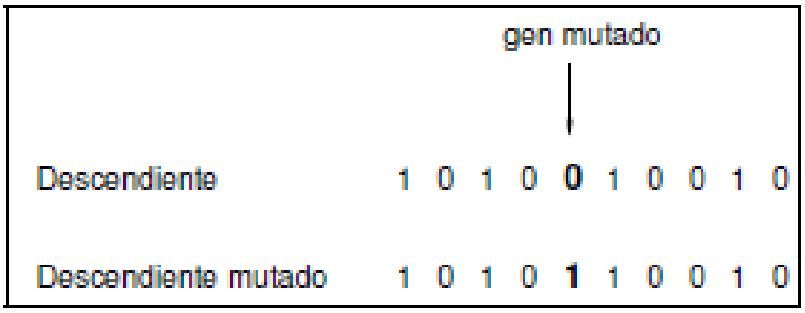

Figura 2.8: Operador de mutación.

El código 2.4 representa una versión general del algoritmo genético:

Código 2.4 Pseudocódigo base para el algoritmo algoritmo genético.

```
/* Algoritmo Genético Simple */
Generar una población inicial().
Computar la función de evaluación de cada individuo().
MIENTRAS(NOT Terminado){
    /* Producir nueva generación */
    Ciclo (Tama o población/2){
     /*Ciclo Reproductivo */
        1.-Seleccionar dos individuos de la anterior generación para el cruce (
            probabilidad de selección proporcional a la función de evaluación
            del individuo);
        2.-Cruzar con cierta probabilidad los dos individuos obteniendo dos
            descendientes;
        3.-Mutar los dos descendientes con cierta probabilidad. Computar la
            función de evaluación de los dos descendientes mutados;
        4.-Insertar los dos descendientes mutados en la nueva generación;
    }
}
SI( la población ha convergido) {
    Terminado := TRUE
}
Fin;
```

2.4 Autómata finitos

Según [HMU08a] un autómata finito (máquina de estados finitos), es un modelo computacional que realiza cómputos en forma automática sobre una entrada para producir una salida.

Este modelo está conformado por un alfabeto, un conjunto de estados y un conjunto de transiciones entre dichos estados. Su funcionamiento se basa en una función de transición, que recibe a partir de un estado inicial una cadena de símbolos pertenecientes al alfabeto (la entrada), y que va leyendo dicha cadena a medida que el autómata se desplaza de un estado a otro, para finalmente detenerse en un estado final o de aceptación, que representa la salida.

La finalidad de los autómatas finitos es la de reconocer lenguajes regulares. Formalmente, un autómata finito es una 5-tupla $< Q, \Sigma, q0, \delta, F >$ donde:

- Q es un conjunto finito de estados.
- Σ es un alfabeto finito de símbolos.
- $q0$ es el estado inicial en Q.
- δ es la relación de transiciones de la forma $< qi, x, qj >$ con qi y qj como estados de Q y x, símbolo de Σ ó puede ser también la cadena vacía.
- F es el conjunto de estados finales o de aceptación y (evidentemente) subconjunto de

Q.

2.4.1 Autómatas finitos deterministas

Según [HMU08b] un autómata finito determinista (AFD) es un autómata finito que además es un sistema determinista; es decir, para cada estado $q \in Q$ en que se encuentre el autómata, y con cualquier símbolo $\alpha \in \Sigma$ del alfabeto leído, existe siempre a lo más una transición posible $\delta(q, a)$. En la figura 2.9 se puede apreciar un ejemplo de un AFD que consta de los elementos mencionados en la lista anterior:

- Q: (q0,q1,q2).
- Σ: (a,b).
- $q0$: (q0).
- δ: es la relación de transiciones que se muestran en la tabla 2.2.
- F: (q2).

Tabla 2.2: Ejemplo de transiciones de un AFD.

	a	b
q0	q1	q0
q1	q1	q2
*q2	q1	q0

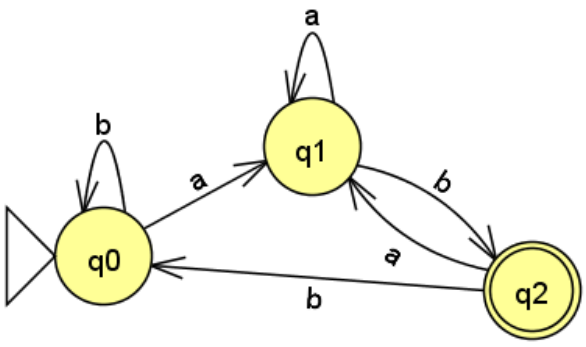

Figura 2.9: Ejemplo de autómata finito determinista.

El autómata en cuestión acepta solamente las cadenas de símbolos que terminen con ab, una vez que ya no existen más caracteres y éste cae en el estado $q2$ (aquél que está en doble círculo) significa que es una cadena válida para el autómata.

En un AFD no pueden darse ninguno de estos dos casos:

- Que existan dos transiciones del tipo $\delta(q, a) = q1$ y $\delta(q, a) = q2$, siendo $q1 \neq q2$;
- Que existan transiciones del tipo $\delta(q, \epsilon)$, salvo que q sea un estado final, sin transiciones hacia otros estados.

2.4.2 Autómatas finitos no deterministas

Según [HMU08c] un autómata finito no determinista (AFND) es aquél que, a diferencia de los autómatas finitos deterministas, posee al menos un estado $q \in Q$, tal que para un símbolo $\alpha \in \Sigma$ del alfabeto, existe más de una transición $\delta(q, a)$ posible. Haciendo la analogía con los AFDs, en un AFND puede darse cualquiera de estos dos casos:

- Que existan transiciones del tipo $\delta(q, a) = q1$ y $\delta(q, a) = q2$, siendo $q1 \neq q2$;
- Que existan transiciones del tipo $\delta(q, \epsilon)$, siendo q un estado no-final, o bien un estado final pero con transiciones hacia otros estados.
- Cuando se cumple el segundo caso, se dice que el autómata es un autómata finito no determinista con transiciones vacías o transiciones ϵ (abreviado AFND-ϵ). Estas transiciones permiten al autómata cambiar de estado sin procesar ningún símbolo de entrada.
- También hacen transiciones ϵ que son aquellas donde se mueve a otro estado sin hacer nada, siempre y cuando el estado actual lo indique.

Haciendo un ejemplo con el autómata de la figura 2.9 se puede observar que se puede hacer un AFND con la misma cadena de aceptación:

Como se puede observar en la figura 2.10 tiene una menor cantidad de transiciones, como se puede ver en la tabla 2.3 el estado $q0$ produce 2 estados en la transición a, una que lleva al estado $q1$ y otra que lleva nuevamente al estado $q0$, es este momento en el que se

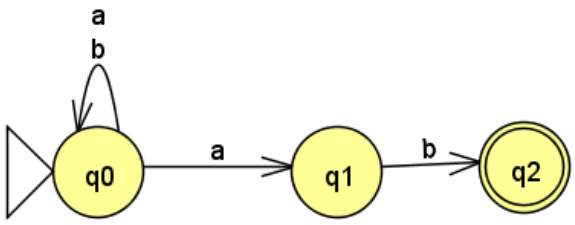

Figura 2.10: Ejemplo de autómata finito no determinista.

hacen 2 procesos por separado, cada uno siguiendo un distinto estado así sucesivamente hasta que termine la cadena.

Tabla 2.3: Ejemplo de transiciones de un AFND.

	a	b
q0	q0,q1	q0
q1		q2
*q2		

Como se menciona anteriormente, las transiciones ϵ son, de acuerdo a [HMU08d], transiciones que permiten ir a otro estado sin necesidad de un símbolo. La figura 2.4 muestra un autómata que acepta la expresión regular $(0 * 1 * 2*)$:

- Q: (q0,q1,q2).
- Σ: (0,1,2).
- $q0$: (q0).
- δ: es la relación de transiciones que se muestran en la tabla 2.4.
- F: (q2).

Tabla 2.4: Ejemplo de AFND con transición ϵ.

	0	1	2	ϵ
q0	q0			q1
q1		q1		q2
*q2			q2	

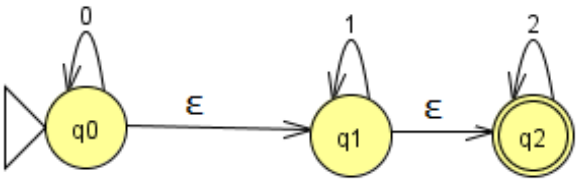

Figura 2.11: Ejemplo de AFND con transición ϵ.

Como se puede observar en este ejemplo, la transición ϵ permite saltar de un estado a otro sin usar un símbolo; en caso de que el siguiente estado no posea ninguna transición con el símbolo actual puede tomar (en caso de que tenga) la transición ϵ actuando como un sustituto. También se puede dar el caso de que exista una transición ϵ y una transición con el símbolo actual. En este caso se generan 2 procesos en paralelo para satisfacer ambas transiciones. En el siguiente ejemplo se puede explicar lo mencionado anteriormente:

- Q: (q0,q1,q2,q3).
- Σ: (a,b).
- $q0$: (q0).
- δ: es la relación de transiciones que se muestran en la tabla 2.5.
- F: (q2).

Tabla 2.5: Segundo ejemplo de un AFND con transición ϵ.

	a	b	ϵ
q0		q1,q3	q0
q1		q2	
*q2			
q3			q2

En este ejemplo se puede observar que este autómata acepta solamente las cadenas b y bb. En el estado inicial la transición b lleva a dos estados distintos, en ese momento se tomarán en cuenta los 2 estados $q1$ y $q3$ donde el estado $q1$ podrá llegar al estado de aceptación $q2$ si encuentra otra cadena b mientras que el estado $q3$ podrá llegar a su estado de aceptación si

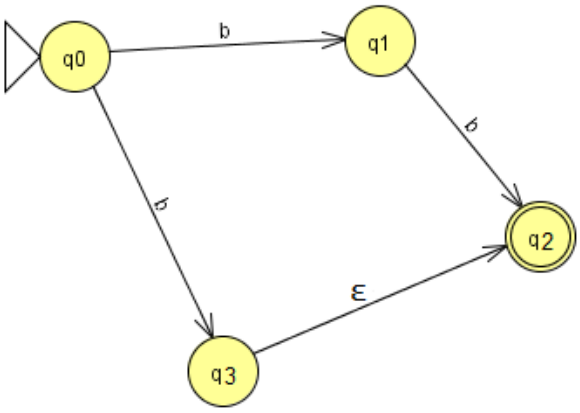

Figura 2.12: Segundo ejemplo de AFND con transición ϵ.

no existe otra cadena (que se interpreta con la transición ϵ).

2.5 Técnicas de agrupamiento (Clustering)

Según [Pas07] y [PS07] el problema del agrupamiento (también conocido como clustering) puede definirse como sigue: dados n puntos en un espacio n-dimensional, particionar los mismos en k grupos tales que los puntos dentro de un grupo son más similares que cada uno a los de los otros grupos; dicha similaridad se mide atendiendo a alguna función de distancia o alguna otra función. En la figura 2.13 aparece un ejemplo del agrupamiento.

Los métodos de agrupamiento se pueden dividir en paramétricos y no paramétricos. Entre los métodos de agrupamiento paramétricos se encuentran las mixturas finitas, éstas son una poderosa herramienta para modelar densidades de probabilidad de conjuntos de datos univariados y multivariados, modelan observaciones las cuales se asume que han sido producidas por un conjunto de fuentes aleatorias alternativas e infieren los parámetros de estas fuentes para identificar qué fuente produjo cada observación, lo que lleva a un agrupamiento del conjunto de observaciones. Los métodos de agrupamiento no paramétricos pueden dividirse en

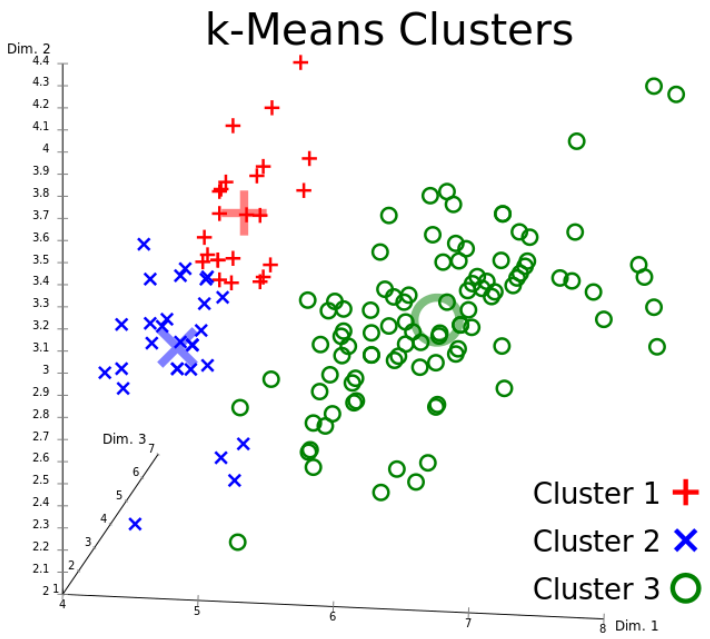

Figura 2.13: Ejemplo de agrupamiento.

tres grupos fundamentales: jerárquicos, particionales y basados en densidad.

2.5.1 Algoritmos jerárquicos

Según [Pas07] los algoritmos jerárquicos son aquellos en los que se va particionando el conjunto de datos por niveles, de modo que en cada nivel generalmente se unen o dividen dos grupos del nivel anterior de acuerdo al tipo de algoritmo: aglomerativo o divisivo. Según [DHS01] las estrategias jerárquicas más conocidas son:

- **Single Link (SL):** En cada paso se unen los dos grupos cuyos elementos más cercanos tienen la mínima distancia.
- **Average Link (AL):** En cada paso se unen los dos grupos que tienen la mínima distancia promedio entre sus puntos.
- **Complete Link (CL):** En cada paso se unen los dos grupos tal que su unión tiene el diámetro mínimo o los dos grupos con la menor distancia máxima entre sus elementos.

- **Chameleon (AC):** Según [KHK99] este método consta de dos fases. En la primera fase se construye un grafo con los k vécinos más cercanos y usa un algoritmo de partición para agrupar los puntos en subgrupos. En la segunda fase usa un algoritmo jerárquico aglomerativo para encontrar los clusters genuinos combinando repetidamente estos subgrupos. Esta fase determina el par de subgrupos más similares tomando en cuenta su conectividad y cercanía. A diferencia del algoritmo SL, este algoritmo permite unir varios pares de subgrupos en la misma iteración.

2.5.2 Algoritmos particionales

Los algoritmos particionales son los que realizan una división inicial de los datos en grupos y luego mueven los objetos de un grupo a otro según se optimice alguna función objetivo. Algunos de estos son:

- **Algoritmo K-Means:** Según [DHS01], la idea principal de este algoritmo es definir k centroides (uno para cada grupo) y luego tomar cada punto y situarlo en la clase de su centroide más cercano. El próximo paso es recalcular el centroide de cada grupo y volver a distribuir todos los puntos según el centroide más cercano. El proceso se repite hasta que ya no hay cambio en los grupos dentro del proceso iterativo.
- **Algoritmo CURE:** De acuerdo a [GRS98] este algoritmo es un enfoque híbrido entre los dos enfoques (jerárquico y particional), que trata de emplear las ventajas de ambos y de eliminar las limitaciones. En lugar de usar un solo punto como representante de un grupo se emplea n puntos representativos del grupo. La similaridad entre dos grupos se mide por la similaridad del par de puntos representativos más cercanos, uno de cada grupo. Para tomar los puntos representativos selecciona los n puntos más dispersos y los atrae hacia el centro del mismo por un factor de contracción α; en cada paso se unen los dos grupos más cercanos y una vez unidos se vuelve a calcular para éste su centro y los c puntos representativos.

2.5.3 Algoritmos basados en densidad

Los algoritmos basados en densidad enfocan el problema de la división de una base de datos en grupos teniendo en cuenta la distribución de densidad de los puntos, de modo tal que los grupos que se forman tienen una alta densidad de puntos en su interior mientras que entre ellos aparecen zonas de baja densidad. Entre los algoritmos que emplean esta técnica se pueden mencionar:

- **Algoritmo DBSCAN:** Como se indica en [EKX96] el algoritmo comienza seleccionando un punto p arbitrario y si p es un punto central, se comienza a construir un grupo y se ubican en su grupo todos los objetos denso-alcanzables desde p. Si p no es un punto central se visita otro objeto del conjunto de datos.

  El proceso continúa hasta que todos los objetos han sido procesados. Los puntos que quedan fuera de los grupos formados se llaman puntos ruido y los puntos que no son ni ruido ni centrales se llaman puntos borde.

  De esta forma DBSCAN construye grupos en los que sus puntos son o puntos centrales o puntos borde, un grupo puede tener más de un punto central.
- **Algoritmo OPTICS:** Según [ABKS99] la motivación para la realización de este algoritmo se basa en la necesidad de introducir parámetros de entrada en casi todos los algoritmos de agrupamiento existentes que en la mayoría de los casos son difíciles de determinar, además en conjuntos de datos reales no existe una manera de determinar estos parámetros globales, el algoritmo OPTICS trata de resolver este problema basándose en el esquema del algoritmo DBSCAN creando un ordenamiento de la base de datos para representar la estructura del agrupamiento basada en densidad, además puede hacer una representación gráfica del agrupamiento incluso para conjuntos de datos grandes.

2.6 Comentarios finales

Como se acaba de ver la resolución de problemas es un tema bastante extenso, aunque teóricamente se puede resolver mediante el método de fuerza bruta que consiste en probar

con todas las combinaciones existentes y tomar la correcta, es el más tardado y en algunos casos imposibles de resolver, por eso se busca otros métodos capaces de encontrar una buena solución, no necesariamente tiene que ser la mejor, pero si lo más cercano o bien, que pueda resolver una parte especifica del problema.

Uno de los algoritmos mencionados fue el de agrupamiento (Clustering) que permite dividir todo el conjunto de parámetros en diferentes grupos y de esa manera simplificar el proceso sin tener que realizar un proceso con soluciones de más y que a simple vista son ineficientes.

Además de los clásicos algoritmos que son capaces de dar la misma solución cada vez que se usa la respectiva fórmula, existen también las metaheurísticas que permite obtener secuencias aleatorias que proporciona soluciones que en los algoritmos convencionales no se permitiría ver, aunque son soluciones generadas al azar están controlados mediante ciertas normas que hacen que no se salgan del camino pero puedan seguir buscando diferentes soluciones.

Como comentario final, se pueden combinar los 2 tipos de algoritmos, primero usando algoritmos determinístas para crear una solución inicial, y luego aplicar metaheurísticas para ir explorando zonas alrededor de la solución inicial y poder encontrar una mejor solución.

CAPÍTULO 3 : Método de agrupamiento basado en cuadrantes

En este capítulo se describe el método propuesto para agrupar y reordenar los nodos de un problema de TSP, que consiste en dividir el espacio en cuadrantes dentro de un proceso iterativo.

3.1 Introducción

Este método de agrupamiento jerárquico recibe un grupo de puntos y lo divide en 4 subgrupos (cuadrantes), cada uno de estos subgrupos en caso de no cumplir con la condición requerida se volverá a dividir en otros 4 subgrupos, y así sucesivamente. Este proceso de división puede repetirse hasta que cada grupo debe tener una cantidad igual o menor al límite establecido previamente.

3.2 Aplicación de agrupamiento

En la figura 3.1 se puede apreciar un diagrama de flujo cuyos pasos serán explicados con detalle en los siguientes párrafos.

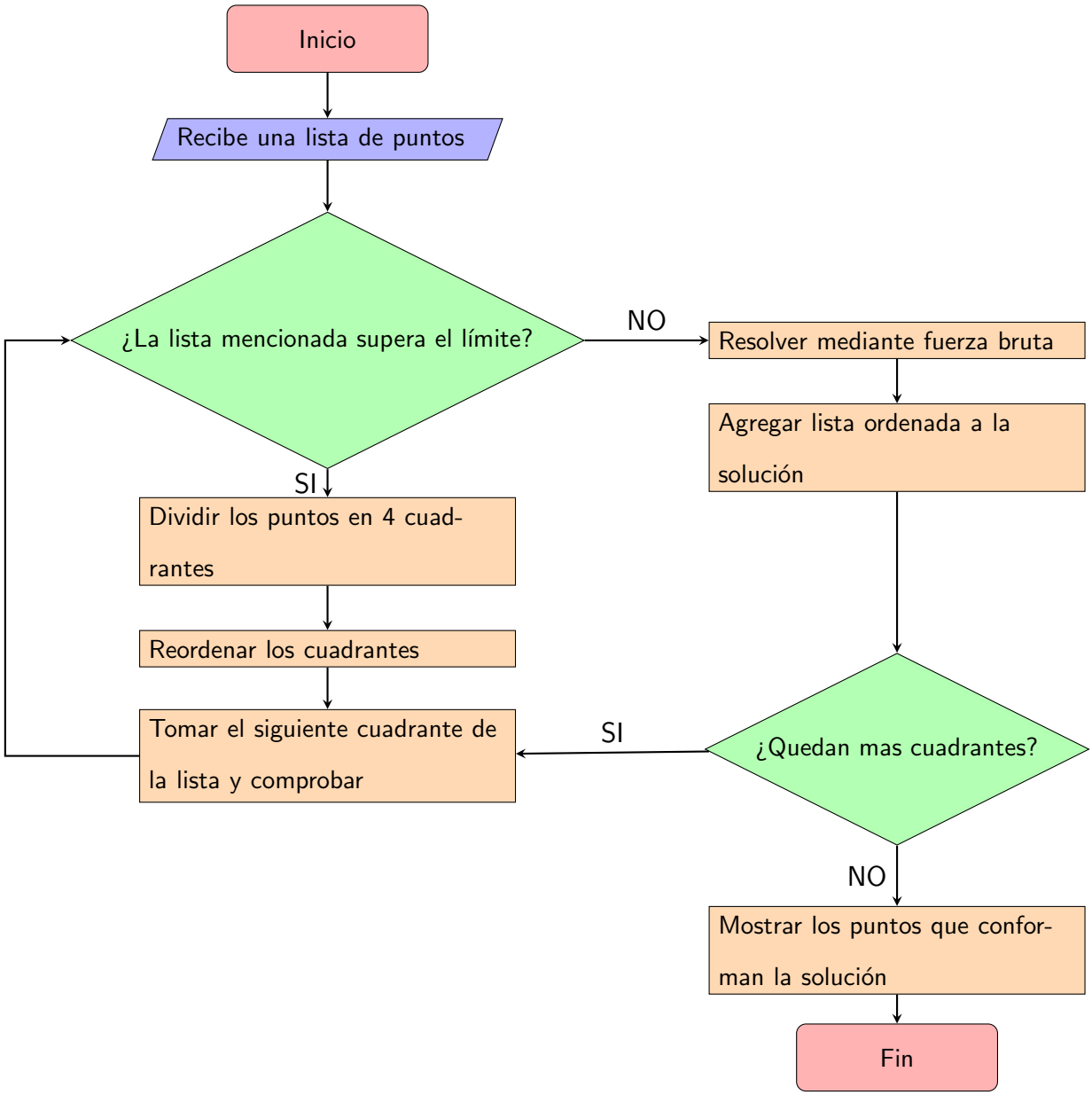

Figura 3.1: Diagrama de flujo del método de cuadrantes.

Como indica la figura 3.1 después de recibir una lista de puntos y que su tamaño supere el límite establecido se dividirá en 4 cuadrantes de un plano cartesiano; una vez distribuidos se comprobará cada cuadrante para determinar si supera el límite. Cada cuadrante se dividirá en 4 de manera recursiva.

Si en algún momento del proceso la lista de puntos a comprobar no supera el límite se procederá a resolver mediante un proceso de fuerza bruta buscando la secuencia de puntos que produzca la ruta más corta; ya con estos puntos reordenados se agregarán a una nueva lista que representará la solución del problema, estos al leerse en ese orden mostrará la ruta a recorrer.

Ahora ya con este diagrama explicado se necesita determinar la forma en que el algoritmo determina hacia que cuadrante irá cada punto, para ello se tendrá que crear un punto adicional que servirá de centro. Para calcular el centro es necesario tomar como base la coordenada del punto más lejano tanto a la izquierda como a la derecha para el eje X y el más lejano de arriba hacia abajo para hacer el eje Y; una vez que se encuentre un subgrupo que satisfaga el criterio se colocará el cursor (el supuesto agente viajero) e iniciará a recorrer el cuadrante. La figura 3.2 muestra un ejemplo de la definición del centro.

En la figura 3.2 se observa que el punto 5 es el que se encuentra más alejado a la izquierda y el 4 más alejado a la derecha, así como los puntos 2 y 4 de abajo hacia arriba respectivamente, por tanto el centro es el punto medio entre el ancho y alto de ambos.

El siguiente proceso consiste en buscar la mejor combinación que exista para llegar desde el punto donde está iniciando el recorrido al siguiente cuadrante. Para lograr este objetivo se usarán unos puntos creados para la ocasión que son intersecciones de un cuadrante a otro, estos servirán para indicar la salida y entrada de los cuadrantes.

Como se explicó anteriormente estos cuadrantes creados a partir de estos agrupamientos se unirán de cuadrante a cuadrante hasta que se recorran todos; al final del problema todos los puntos se encontrarán conectados. Todos los cuadrantes ya resueltos se unirán, de preferencia con el cuadrante que tenga adyacente y cuando los 4 subcuadrantes del cuadrante se hayan

unido, se buscará otro cuadrante adjunto y así sucesivamente, el proceso terminará hasta resolver todos los cuadrantes.

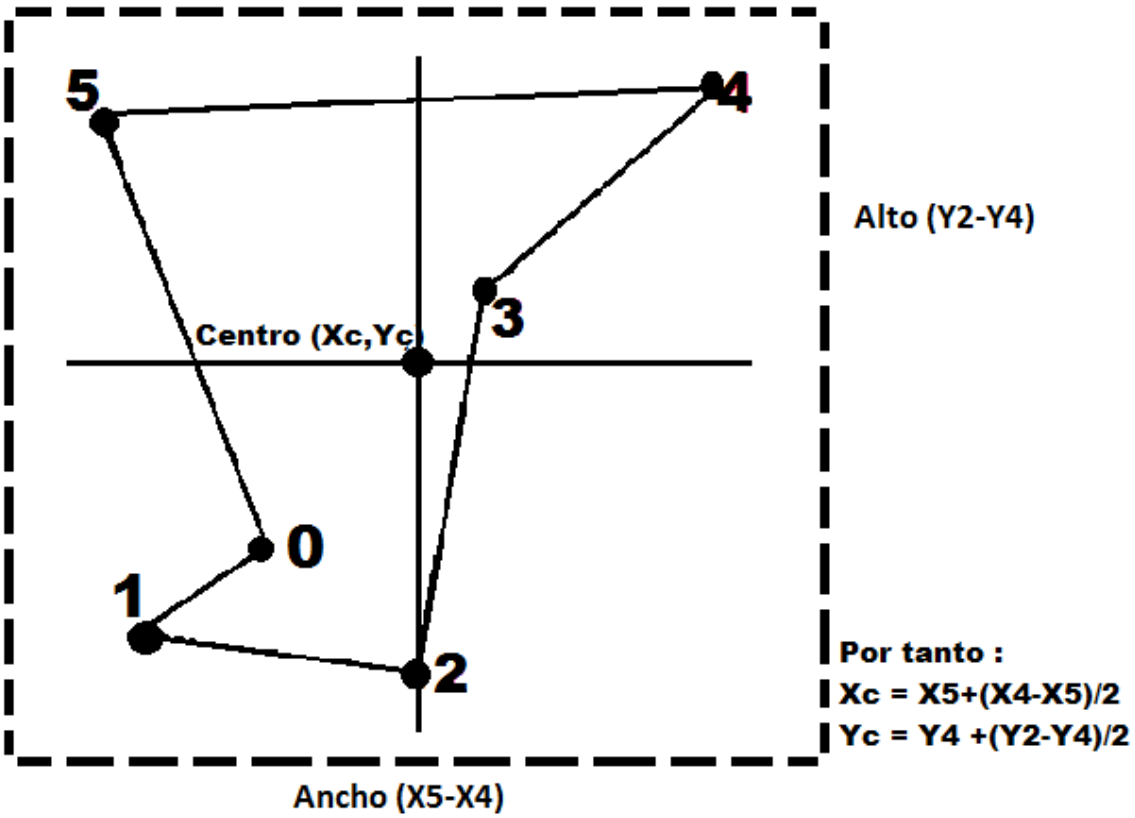

Figura 3.2: Ejemplo para calcular el centro del plano de búsqueda.

El siguiente paso es definir el criterio que se aplica para saber en qué orden de subcuadrantes debe viajar para recorrer toda la sección.

3.3 Aplicación de autómatas para el orden de los cuadrantes

Para determinar el orden de los cuadrantes por recorrer se utilizará un autómata que recibirá ciertos parámetros y devolverá una de varias soluciones predeterminadas. Los 3 parámetros del autómata son:

- El cuadrante origen: El cuadrante donde se encuentra el cursor.
- El cuadrante destino: El cuadrante a donde se dirigirá el cursor.
- El origen del subcuadrante: El subcuadrante del cuadrante origen de donde inicia la ruta.

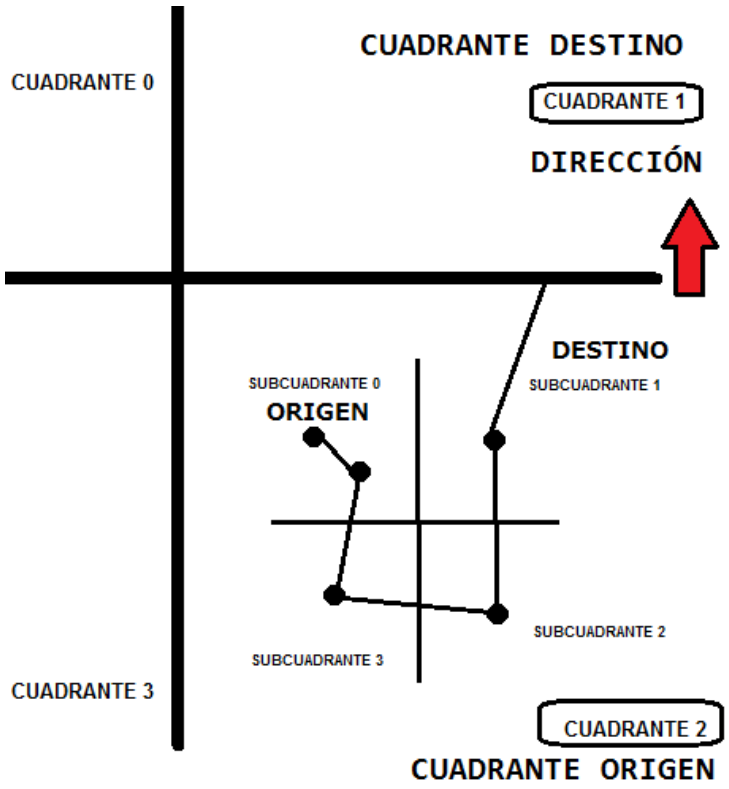

Figura 3.3: Fragmento de ejemplo del método de cuadrantes.

En la figura 3.3 se puede observar el fragmento de un ejemplo donde se presentan los cuadrantes 1 y 2; dentro del cuadrante 2 se puede observar que también está dividido en otros subcuadrantes y el punto de origen se encuentra en el subcuadrante 0; de esta forma se obtienen los parámetros necesarios donde:

- El cuadrante origen es 2.
- El cuadrante destino es 1.
- El origen del subcuadrante es 0.

Lo primero a notar es que el cursor tendrá que viajar hacia arriba recorriendo los 4 cuadrantes sin pasar por el anterior ya que el cuadrante destino está sobre el cuadrante origen. Para ello se hará uso de una matriz de direcciones presentada en la figura 3.4.

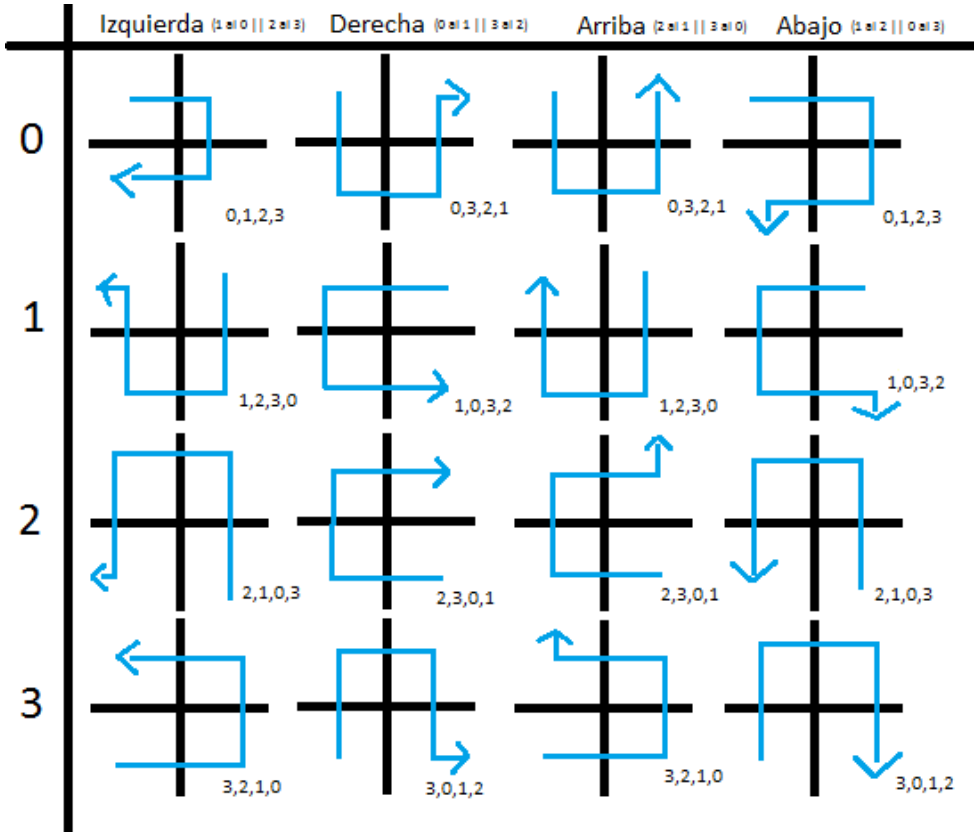

Figura 3.4: Matriz de direcciones.

Como se puede ver en la figura 3.4 ésta es una matriz de 4 filas y 4 columnas, las filas representan el origen del subcuadrante en donde se va a empezar (en este caso 0) y las columnas representan la dirección a la que tiene que ir (en este caso hacia arriba y del cuadrante 2 al 1), al tener estas 2 coordenadas se determina como solución la secuencia (0,3,2,1) que determina el orden de subcuadrantes por los que tiene que viajar para pasar por todos los puntos. Cada subcuadrante puede seguir dividiéndose, para tal caso el subcuadrante del origen es siempre el que se está resolviendo al momento, el cuadrante origen y cuadrante

destino forman parte del cuadrante anterior.

La primera vez que se vaya a dividir el conjunto de puntos, el parámetro de destino será el mismo punto de origen por tanto; ya teniendo los otros 2 parámetros se puede tomar la solución de cualquiera de las 4 columnas, dando 4 posibles soluciones dependiendo del giro principal. En la figura 3.5 se podrá ver un ejemplo completo aplicando todas las reglas mencionadas.

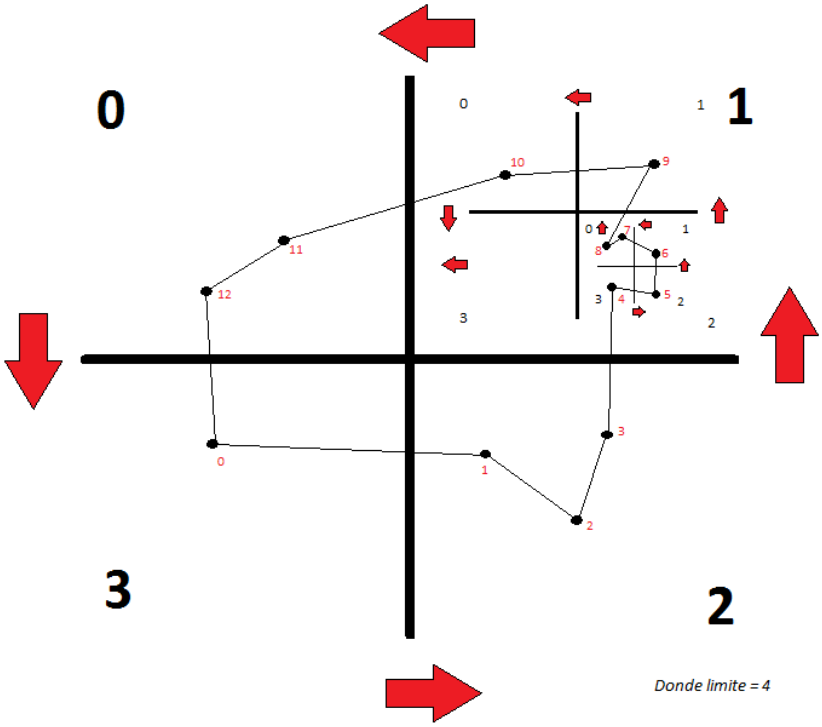

Figura 3.5: Ejemplo completo de un problema TSP aplicando las reglas mencionadas.

3.3.1 Descripción del autómata

En la figura 3.6 se puede ver el autómata que recibe los 3 parámetros (el cuadrante inicial, el cuadrante donde se tiene que dirigir, y por último el origen del subcuadrante). En la tabla 3.1

se podrá ver su declaración formal.

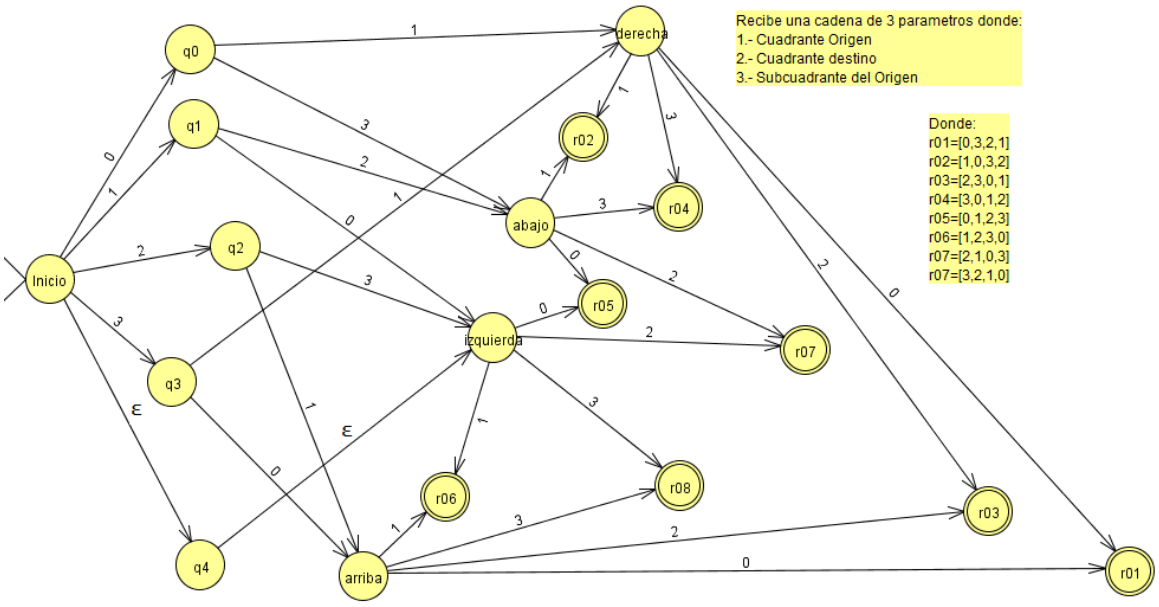

Figura 3.6: Autómata del método de cuadrantes.

Como se puede ver en la tabla 3.2, la transición de eventos que realiza el autómata es del tipo no determinista ya que posee estados vacíos y transiciones ϵ.

En el ejemplo a la figura 3.5 el cuadrante 1 está dividido en otros subcuadrantes, el origen del subcuadrante sería el punto el 4 que se encuentra en el cuadrante 2, ya que a la dirección por la que va tiene que ir al cuadrante principal 0. Por tanto, los parámetros que recibiría el autómata en ese mismo orden sería:

- El cuadrante origen: El cuadrante 1.
- El cuadrante destino:El cuadrante 0 que se encuentra a su izquierda.
- El origen del subcuadrante: El subcuadrante 2.

Como se observa en la figura 3.6, primero se recorrería al estado $q2$, luego pasaría por el estado $izquierda$ y por último al estado $r7$, dando como resultado los parámetros (2,1,0,3) que

Tabla 3.1: Elementos del autómata del método de cuadrantes.

Declaración formal	
Σ:	0, 1, 2, 3
Q:	Inicio, q0, q1, q2, q3,q4 izquierda, derecha, arriba, abajo, r01, r02, r03, r04, r05, r06, r07, r08
$q0$:	Inicio
F:	r01, r02, r03, r04, r05, r06, r07, r08
δ:	Es la relación de transiciones que se muestran en la tabla 3.2.

Tabla 3.2: Transiciones del autómata del método de cuadrantes.

	0	1	2	3	ϵ
Inicio	q0	q1	q2	q3	q4
q0		derecha		abajo	
q1	izquierda		abajo		
q2		arriba		izquierda	
q3	arriba		derecha		
q4					izquierda
izquierda	r05	r06	r07	r08	
derecha	r01	r02	r03	r04	
arriba	r01	r06	r03	r08	
abajo	r05	r02	r07	r04	
*r01					
*r02					
*r03					
*r04					
*r05					
*r06					
*r07					
*r08					

vendrían siendo los cuadrantes por recorrer en la iteración inicial. Sin embargo el subcuadrante 1 excede con el límite de puntos establecido teniendo que resolver primero este inconveniente repitiendo el proceso de subdivisión esta vez poniendo el origen, destino y subcuadrante del origen como 2, 1 y 3 respectivamente, lo que daría el estado $r8$ con los parámetros (3,2,1,0).

De ahí en adelante se resuelven los demás cuadrantes usando el método de fuerza bruta, que gracias a las divisiones que se fueron haciendo agilizan el proceso, terminando mucho más rápido que haber intentado hacer el método de fuerza bruta con todos los puntos.

3.4 Comentarios finales

Este método hace uso del sentido común y de la distribución uniforme para cumplir con el objetivo deseado, no se busca obtener todavía un resultado definitivo, sino una buena solución que servirá de base para la aplicación de las metaheurísticas correspondientes.

El uso de cuadrantes permite seccionar por zonas de manera finita y no salir de cada una de ellas hasta pasar por todos los lugares, reduciendo el tiempo de cálculo y evitando realizar combinaciones innecesarias.

Sin embargo y a pesar de obtener una buena solución, el resultado obtenido a través de este algoritmo no es la mejor ruta que se haya creado hasta la fecha, sino es una respuesta cercana que se puede refinar aplicando metaheurísticas, tema que se verá en el capítulo 5.

Lo que destaca de este método es la de obtener buenos resultados y la preferencia de recorrer zonas cercanas con tiempos de cálculo breves en lugar de intentar hacer movimientos audaces o combinaciones reiterativas que pueden ser solo una pérdida de tiempo, aunque eso también implica una desventaja ya que jamás podría obtener resultados mejores a costa de mayor trabajo.

Como se muestra en la figura 3.7, es un retrato de la Mona Lisa hecho a base de un problema de TSPLIB obtenido de [UW16], usando el algoritmo de cuadrantes se pudo unir los puntos que componían el problema mostrando la figura presentada.

Figura 3.7: Ejemplo de ejercicio de la MonaLisa hecha en TSP.

CAPÍTULO 4 : Documentación del software usado en las pruebas

4.1 Introducción

Para poder aplicar el algoritmo propuesto con las instancias obtenidas de la TSPLIB, se desarrolló un programa utilizando el lenguaje de programación JAVA que permita almacenar datos en diferentes formatos (XML, JSON,TSPLIB).

Otro propósito de este programa es que sea de código abierto con el fin de que otras personas puedan continuar con la investigación o utilizarlo como referencia para otros proyectos. Una vez que se explique la composición del software y la forma en la que se utiliza en el siguiente capítulo se procederá al desarrollo de los experimentos.

4.2 Componentes

En la figura 4.1 se muestra la ventana principal del programa que está compuesto por:

A.- Una barra de herramientas

B.- 3 pestañas cuyo contenido se explicará a continuación

C.- La pantalla principal que muestra un problema de TSP

Como se observa en la figura 4.1 la ventana muestra la posición de los puntos del problema y las rutas que éste recorrió de principio a fin; también se muestran los cuadrantes que se trazaron para realizar el cálculo así como el orden de puntos en que se fue recorriendo.

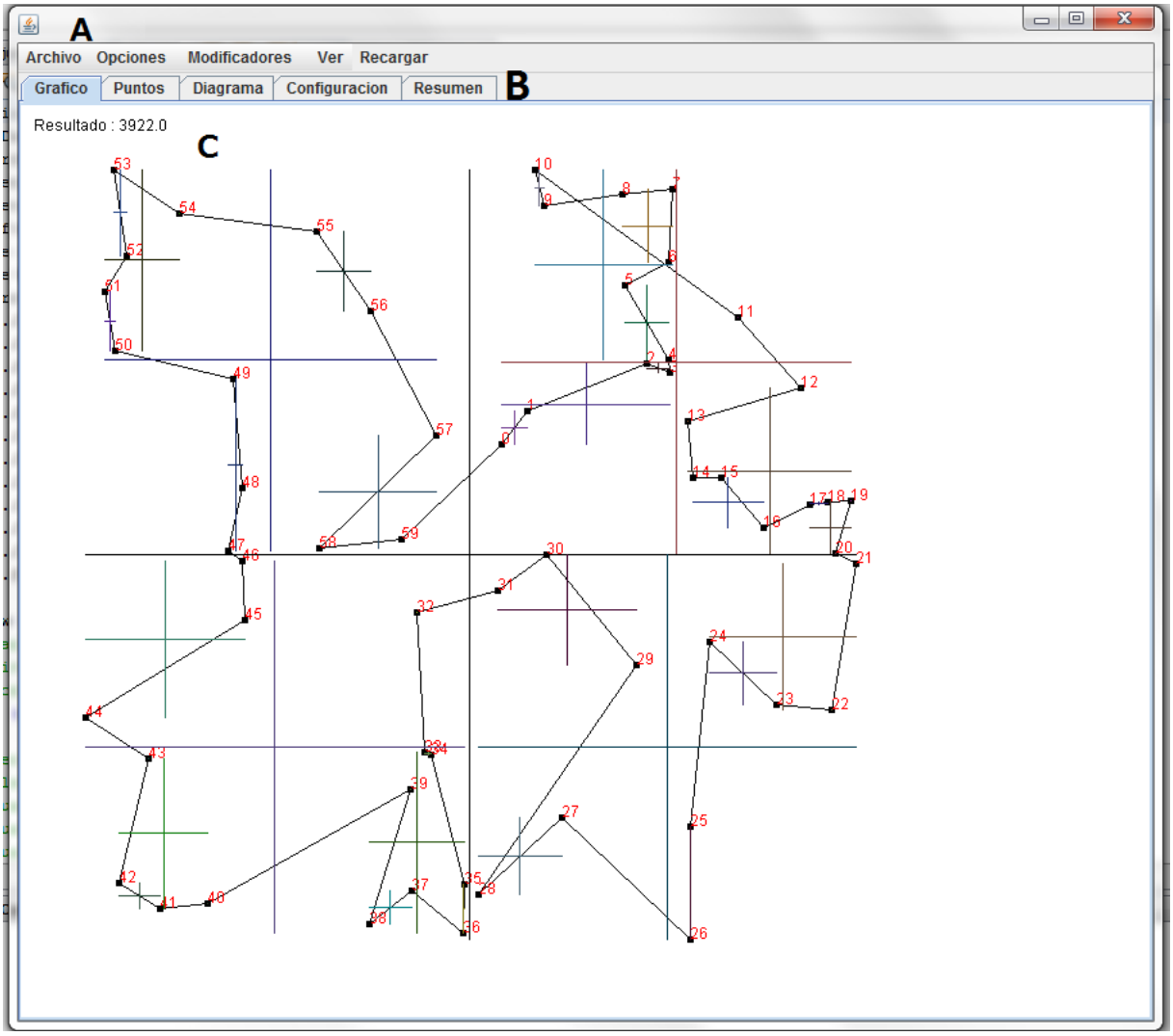

Figura 4.1: Descripción de la ventana principal del software.

En la figura 4.2 se muestran las coordenadas de los puntos del problema junto con el resultado mostrado en la figura 4.1.

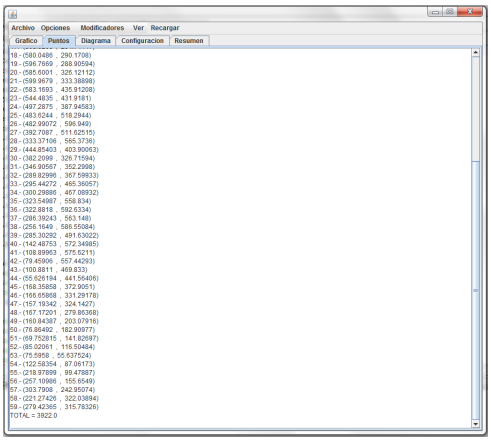

Figura 4.2: Coordenadas de los puntos del problema.

En la figura 4.3 se muestra la gráfica de los resultados obtenidos en el problema utilizando diferentes metaheurísticas. En este ejemplo se aplicó únicamente el algoritmo de recocido simulado.

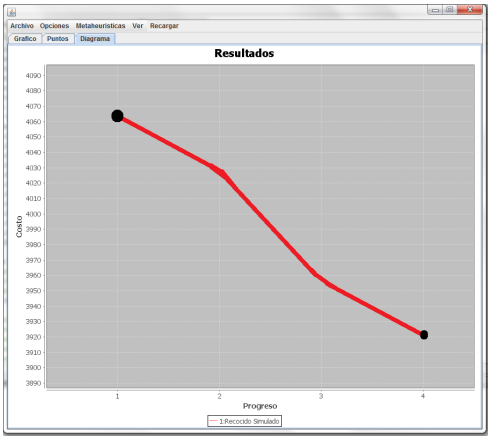

Figura 4.3: Gráfica de los resultados del problema.

En la figura 4.4 se muestra la ventana de configuraciones que se puede realizar para cada metaheurística, como la cantidad de ciclos que se puede hacer, el tamaño de los genes,

factor de temperatura para el proceso de recocido simulado, etc. El uso de estas variables se explicarán en el siguiente capítulo.

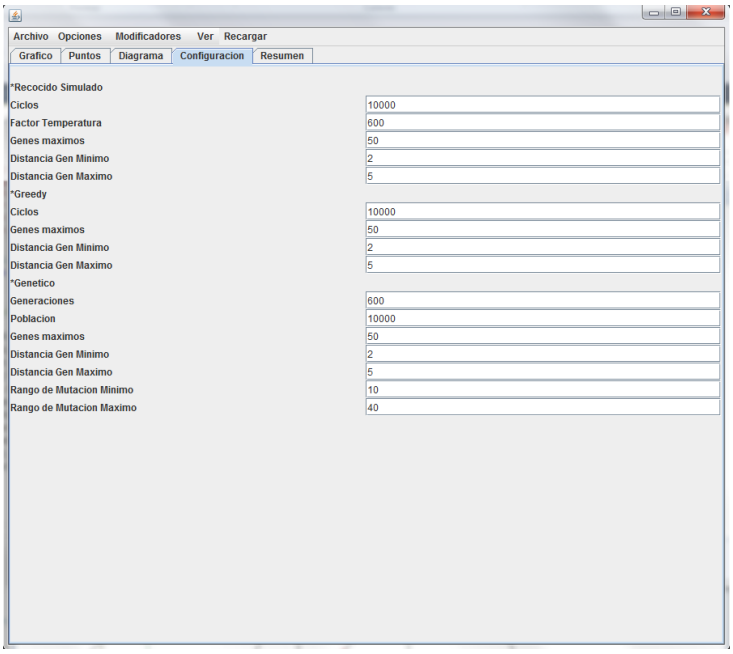

Figura 4.4: Ventana de configuración.

En la figura 4.5 se muestra una pantalla que despliega la cantidad de tiempo que tarda el algoritmo cada vez que se ejecuta un problema.

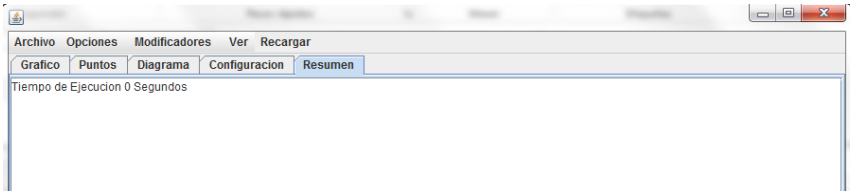

Figura 4.5: Ventana de resultados.

En la figura 4.6 se puede observar el menú de Archivo de la barra de herramientas, que tiene las siguientes opciones:

A.- **Guardar:** Permite guardar las coordenadas para poder usarse en experimentos futuros en un archivo .TSP.

B.- **Abrir:** Abre un documento .TSPLIB o .TXT y carga sus coordenadas en el programa, una vez que se carguen los datos el problema se resolverá automáticamente.

C.- **Abrir sin resolver:**Lo mismo que el botón anterior, solo que no resuelve el problema de forma automática.

D.- **Abrir Prueba:** Permite abrir un archivo XML ó JSON y cargar un problema con sus respectivos resultados. Estos problemas se consiguen a través de los siguientes botones.

E.- **Guardar (XML):** Permite guardar un problema con todos los resultados obtenidos en un archivo XML.

F.- **Guardar (JSON):** Lo mismo que el anterior, pero lo guarda en formato JSON.

G.- **Realizar Experimento:** Este será la función que se usará para realizar las actividades del siguiente capítulo, permite realizar de manera consecutiva una serie de ejecuciones del mismo listado de puntos que se encuentra cargados, guardando los resultados de forma física con el archivo .JSON de cada resultado y su gráfica correspondiente. Por ejemplo si se seleccionan 30 ejecuciones devolverá 30 archivos .json y 30 gráficas de cada uno.

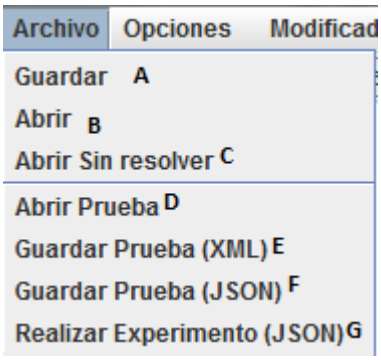

Figura 4.6: Barra de herramientas, Archivo.

En la figura 4.7 se muestra el menú de Opciones, la cual contiene funciones de diversos propósitos:

A.- **Aleatorio:** Permite generar una serie de coordenadas al azar y que resolverá de manera automática, el propósito original de esta función fue la de probar el algoritmo de búsqueda por cuadrantes conforme se fue desarrollando.

B.- **Borrar resumen:** Permite limpiar la pantalla de la ventana de resumen de la figura 4.5.

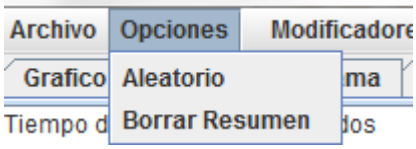

Figura 4.7: Barra de herramientas, Opciones.

En la figura 4.8 se puede observar el menú de modificadores que se encarga de aplicar las perturbaciones de la solución base; éstas se harán al final de la ejecución del método de cuadrantes (si se marcó como seleccionado el método de cuadrantes) con el fin de optimizar la solución base, los algoritmos que se pueden seleccionar son:

A.- Cuadrantes

B.- Búsqueda Exhaustiva

C.- Recocido Simulado

D.- Búsqueda Greedy

E.- Algoritmo Genético

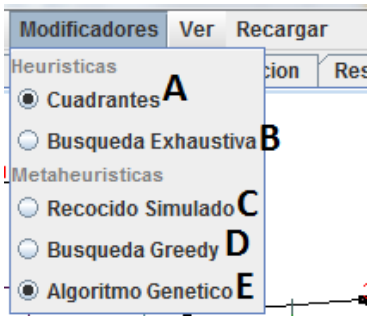

Figura 4.8: Barra de herramientas, Modificadores.

En la figura 4.9 se puede observar el menú de Ver, que permite dibujar el diagrama de la figura 4.1 de acuerdo a los elementos que aparecen en el listado:

A.- **Ruta:** Las líneas que muestra el recorrido de punto a punto.

B.- **Puntos:**Los nodos que indican la ubicación exacta de cada punto.

C.- **Lineas:** Las divisiones de colores que los cuadrantes de acuerdo al método de cuadrantes.

D.- **Números:** El orden de los puntos que recorrió la ruta de principio a fin.

E.- **Final:** La última ruta que se recorre del último punto hacia el punto de inicio.

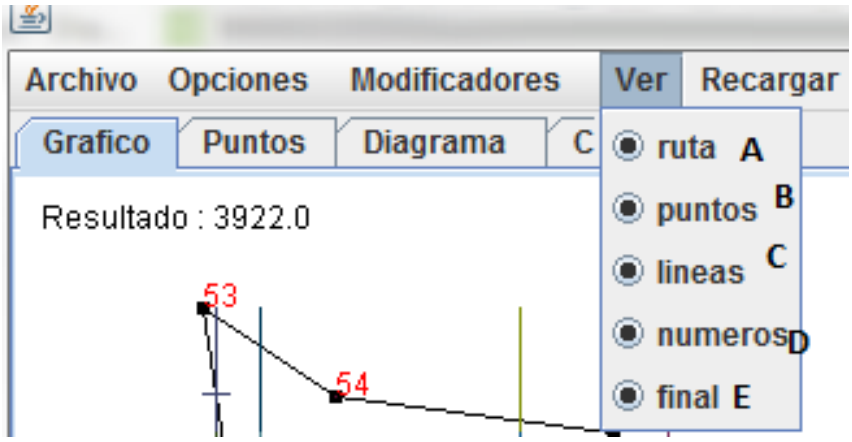

Figura 4.9: Barra de herramientas, Ver.

En la figura 4.10 se puede observar la opción de Recargar que permite volver a generar el problema; la utilidad de esta función es la de reutilizar el mismo problema usando los diferentes métodos de la figura 4.8 o bien, esperar obtener resultados diferentes.

Figura 4.10: Barra de herramientas, Recargar.

4.3 Uso del software

Ya conociendo los componentes que conforman el programa se procederá a explicar la manera en que se usará para los experimentos.

Al iniciar el programa se mostrará una ventana parecida a la de la figura 4.1 donde se

muestra un problema formado por un conjunto de puntos al azar ya resuelto, sin embargo para esta ocasión se usará un archivo .TSP como prueba. Primero se usará la opción de Abrir sin resolver como se muestra en la figura 4.11.

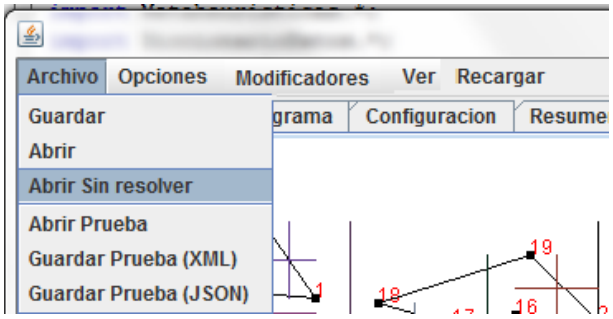

Figura 4.11: Abrir sin resolver.

Para este ejemplo se usará el archivo 'ch150.tsp' como se muestra en la figura 4.12, la mejor ruta conocida para este problema es de 6528 unidades.

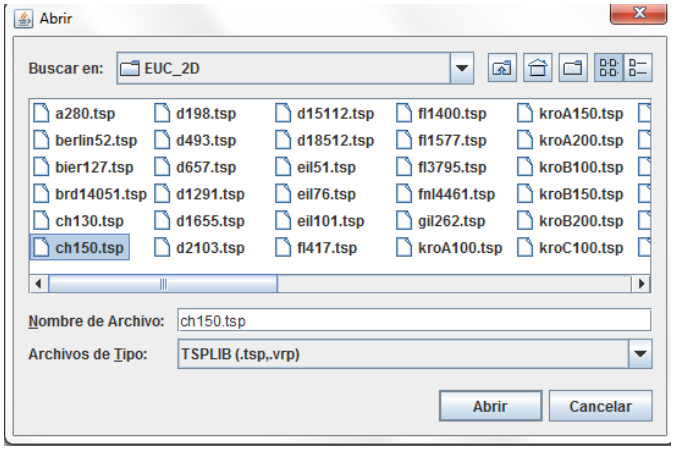

Figura 4.12: Archivo .TSP.

En la figura 4.13 se puede apreciar que los puntos se encuentran desordenados, por tanto genera una ruta de lo más ineficiente con un costo de 52890 unidades ya que uno los puntos en el mismo orden que los va leyendo.

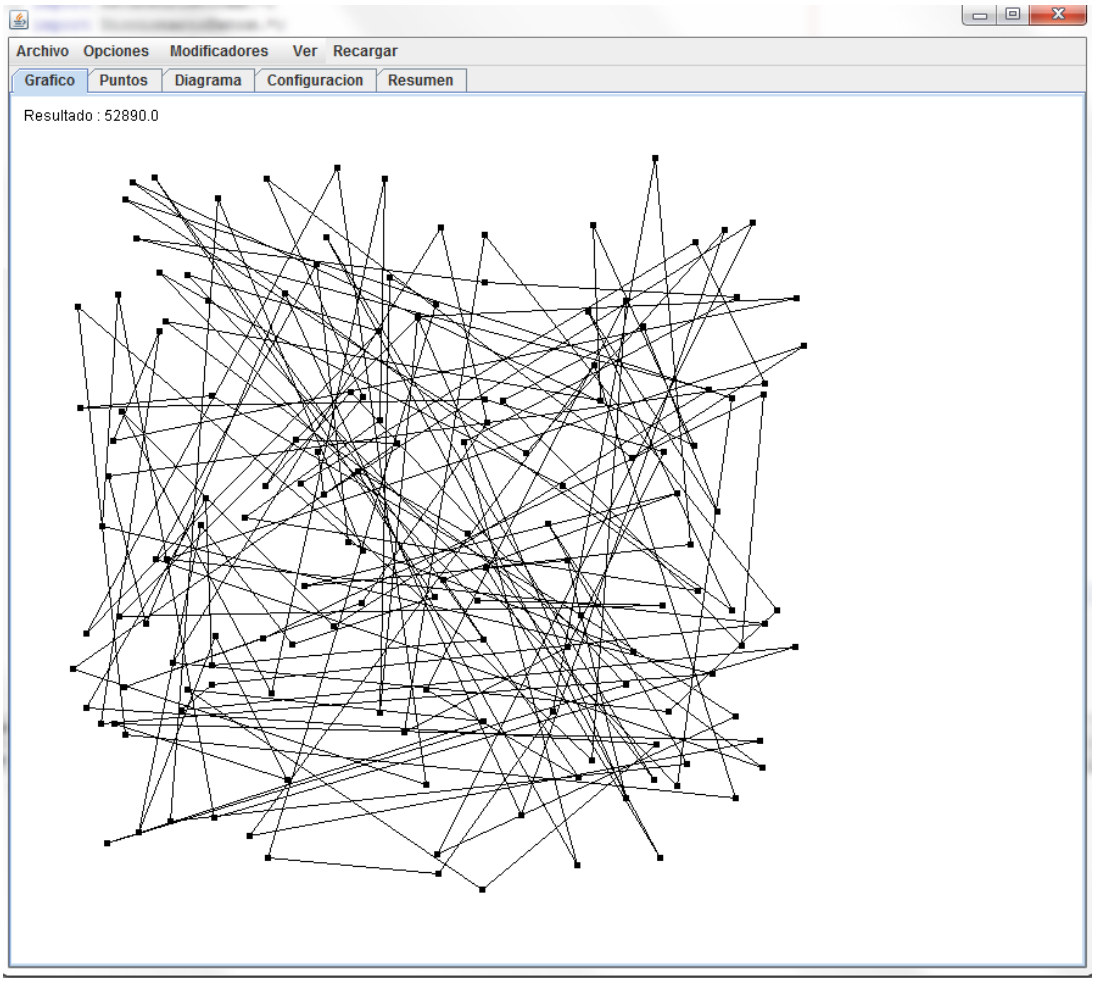

Figura 4.13: Archivo sin resolver.

Sin embargo si se aplica la opción de recargar se aplicará las fórmulas que se tienen seleccionadas, en este caso solo se aplicará el método de cuadrantes, tal como se muestra en la figura 4.14. Los puntos siguen siendo los mismos pero el trazado de rutas está ordenado de acuerdo dicho método trayendo un costo de 8579 unidades, siendo menos de una sexta parte del costo de la figura 4.13.

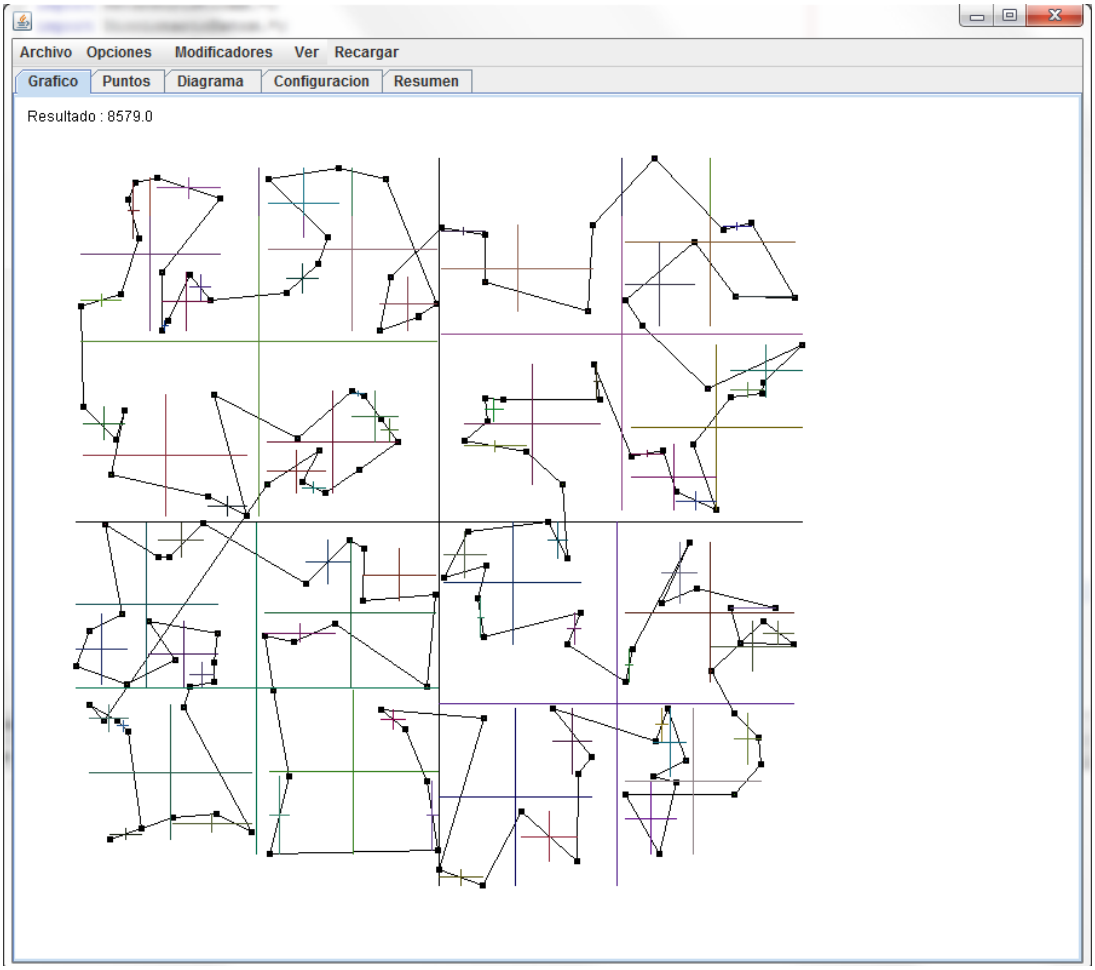

Figura 4.14: Problema resuelto con el método de cuadrantes.

En la figura 4.15 se volvió a recargar el mismo problema pero esta vez usando el método de búsqueda exhaustiva la cual trae un valor de 8252 unidades, un valor ligeramente menor al anterior con 327 unidades de diferencia.

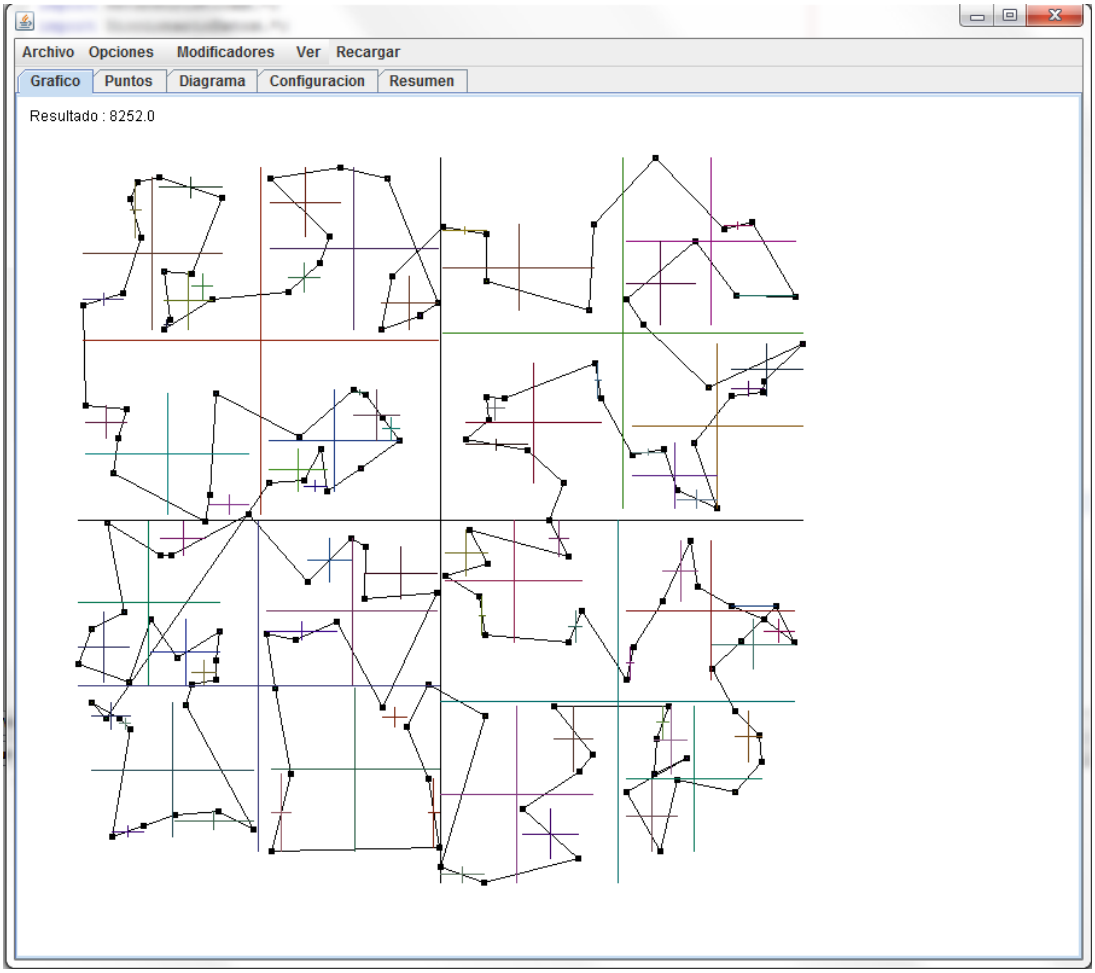

Figura 4.15: Problema resuelto con el método de cuadrantes y la búsqueda exhaustiva en el orden correspondiente.

En la figura 4.16 se volvió a recargar el mismo problema pero esta vez marcando también el método de recocido simulado con un resultado de 8179 unidades, una mejora de la figura 4.15 con 73 unidades de diferencia.

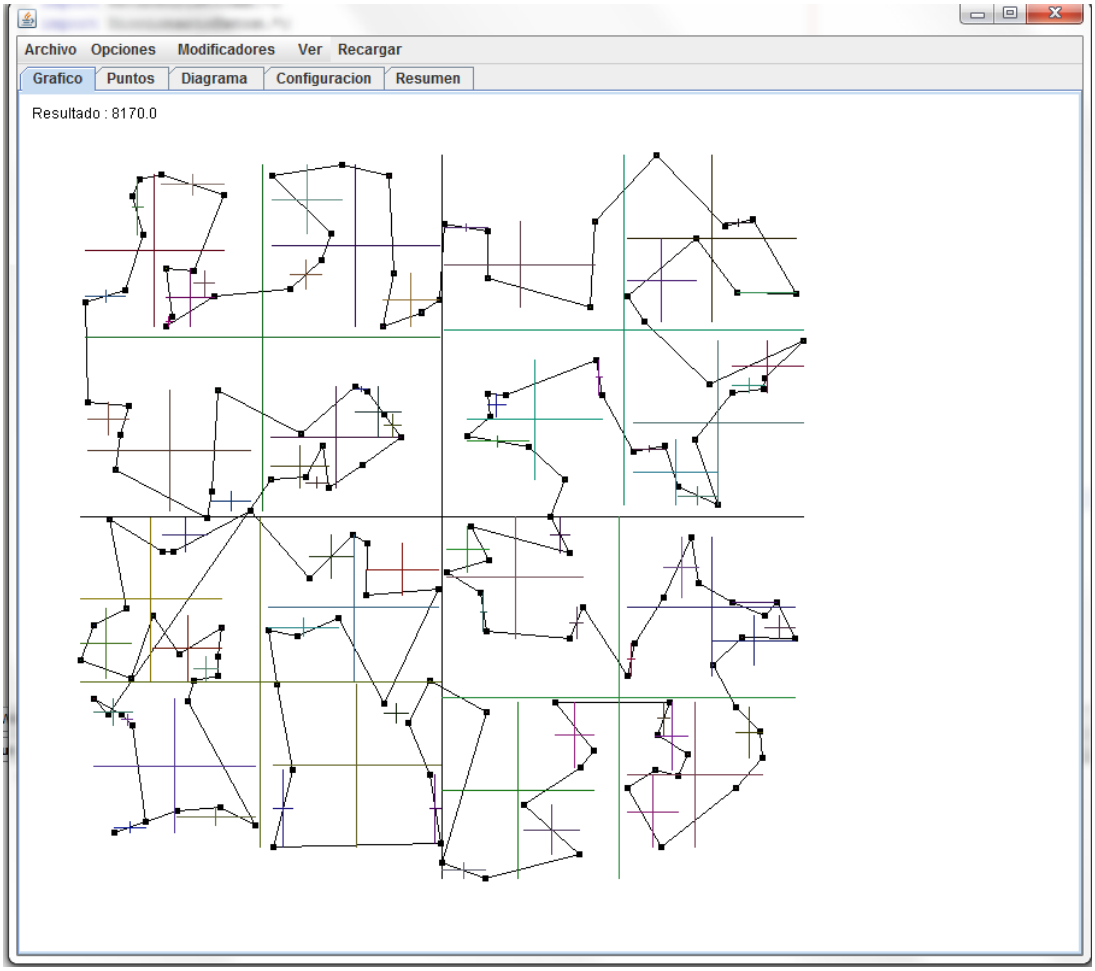

Figura 4.16: Problema resuelto con el método de cuadrantes, recocido simulado y búsqueda exhaustiva en el orden correspondiente.

En la figura 4.17 se puede mostrar la gráfica de la forma en que fue mejorando la solución obtenida por el programa. El método de Búsqueda exhaustiva siempre se ejecutará al final del proceso.

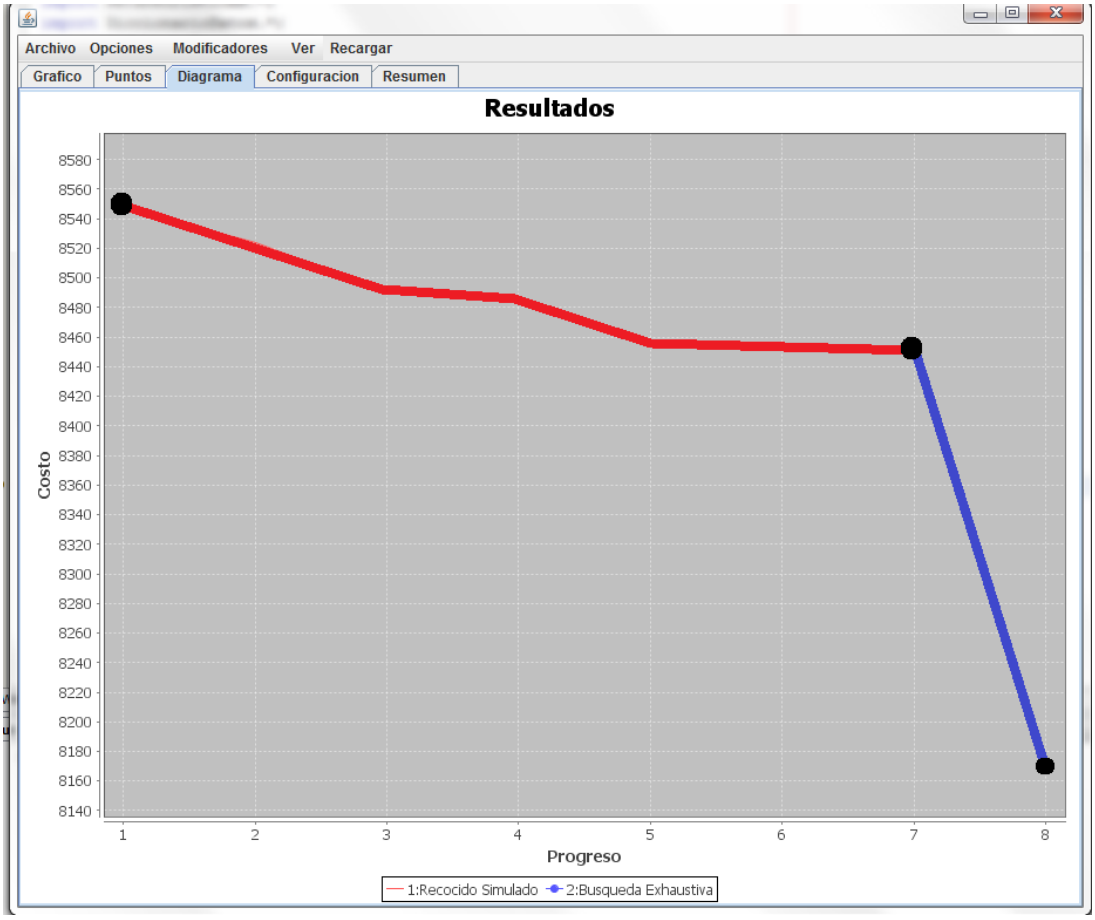

Figura 4.17: Gráfica que muestra los resultados.

Por último, aplicando el algoritmo genético en la figura 4.18 con el fin de obtener el mejor resultado posible, deja un costo de 7342 unidades con una notable diferencia de 837 unidades.

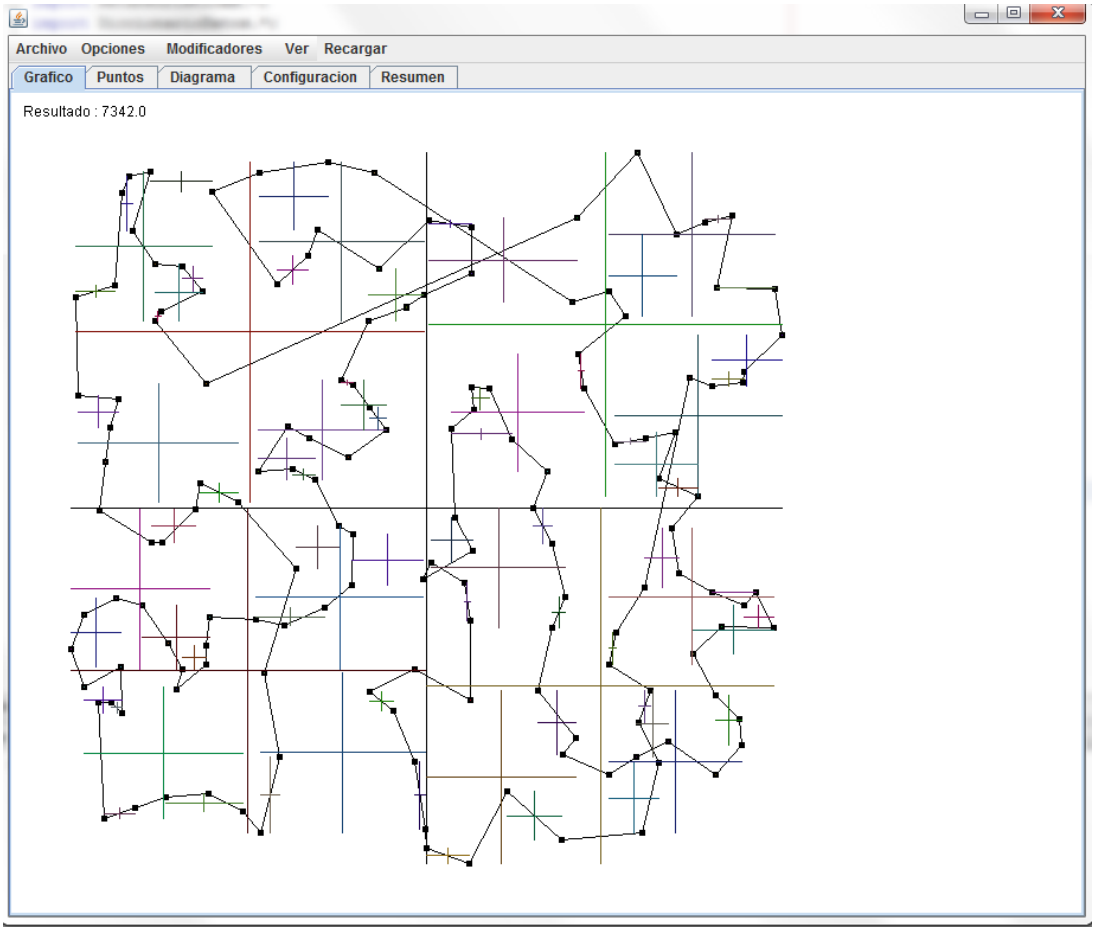

Figura 4.18: Problema resuelto usando todas las metaheurísticas.

En la figura 4.19, se puede observar la gráfica con el resultado de la figura 4.18 el cual muestra un resultado favorable en la línea que representa el algoritmo genético, debido a su simplicidad el método de búsqueda exhaustiva se aplica al final de las demás metahuerísticas.

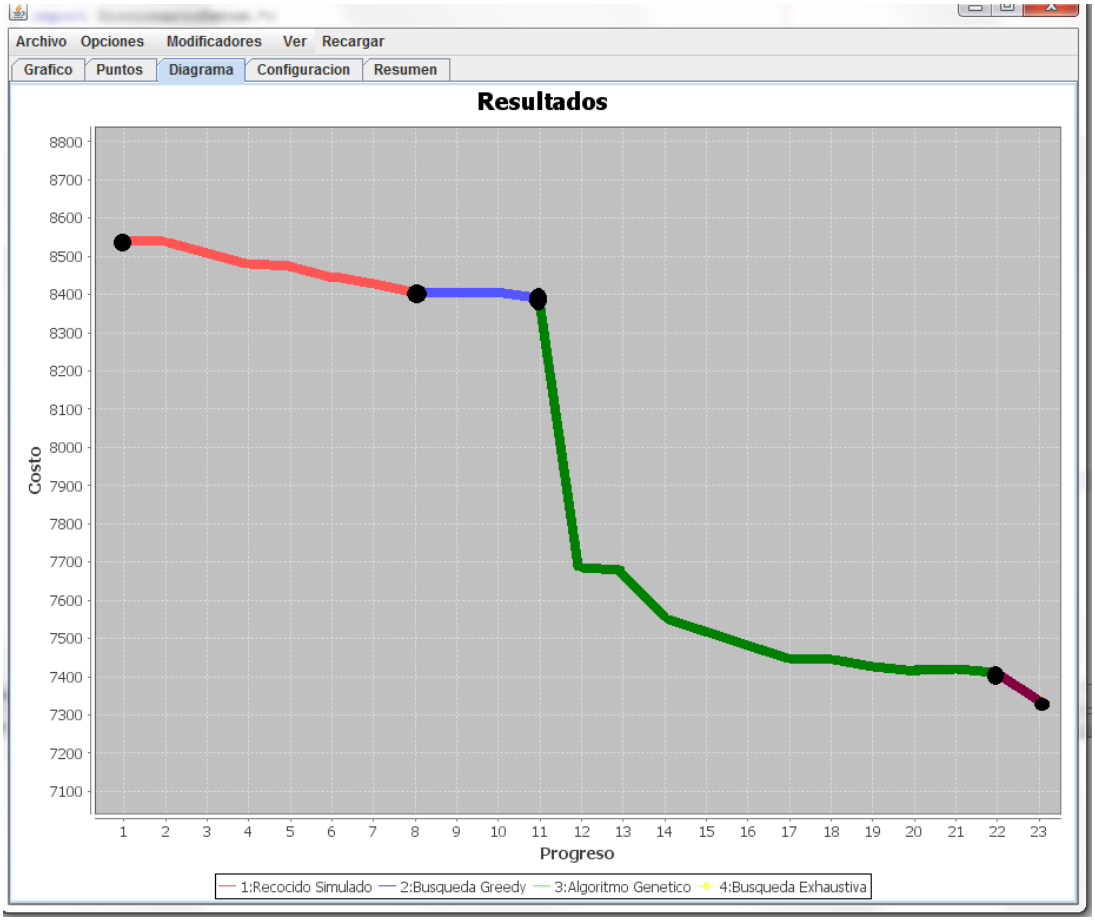

Figura 4.19: Gráfica que muestra el progreso después de aplicar todas las metahuerísticas.

Para finalizar si se desea mostrar el problema resuelto se puede guardar por medio de un archivo .XML o un .JSON descritos en la figura 4.6. En la figura 4.20 se eligió en formato .JSON.

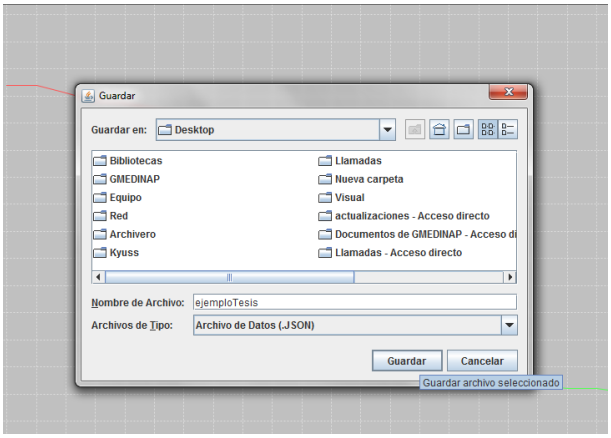

Figura 4.20: Problema solucionado guardado en un JSON.

4.4 Comentarios finales

La ayuda de este programa fue esencial para poder realizar los experimentos, ya que gracias a esta herramienta fue posible leer, interpretar y almacenar los resultados correspondientes.

Una de las ventajas de este programa es que sigue el mismo formato de las TSPLIB, por tanto puede utilizarse para otros proyectos futuros, sin embargo solo está limitado a resolver problemas que utilizan distancia euclidiana (EUC2D).

CAPÍTULO 5 : Pruebas y análisis de resultados

En este capítulo se aplicará lo expuesto en los capítulos 3 y 4 para realizar los experimentos correspondientes, y se describirán los resultados obtenidos.

5.1 Introducción

El último capítulo trata sobre el uso de los algoritmos y metaheurísticas aplicados sobre problemas reales de TSP, primero se explicará el proceso que se realizó y al final se mostrarán los resultados obtenidos.

Cada uno de los problemas de TSP se probará tanto con el método de cuadrantes como sin este método; en ambos casos se probará con algo conocido como "mutación" que alterará la solución base reordenando algunos de sus puntos con el fin de obtener resultados aleatorios pero coherentes. Como se muestra en la tabla 5.1 estos serán los problemas a utilizar obtenidos en la página de la Universidad de Heidelberg [RKH16] y cuyas descripciones se obtuvieron de la Universidad de Waterloo [ZIB99].

Tabla 5.1: Descripción de los problemas de TSP usados para las pruebas.

Problema	Ciudades	Descripción	Autor
a280.tsp	280	Problema de perforación	Ludwig
brd14051.tsp	14051	Republica federal de Alemania en 1989	Bachem y Wottawa
ch150.tsp	150	Problema de 150 ciudades	Churritz
d1655.tsp	1655	Problema de perforación	Reinelt
d493.tsp	493	Problema de perforación	Reinelt
eil101.tsp	101	Problema de 101 ciudades	Christofides y Eilon
fl417.tsp	417	Problema de perforación	Reinelt
lin318.tsp	318	Problema de 318 ciudades	Lin y Kernighan
p654.tsp	654	Problema de perforación	Reinelt
pcb3038.tsp	3038	Problema de perforación	Reinelt y Juenger
rd400.tsp	400	Problema de aleatorio de 400 ciudades	Reinelt
rl5934.tsp	5934	Problema de 5934 ciudades	Reinelt
u159.tsp	159	Problema de perforación	Reinelt
u724.tsp	724	Problema de perforación	Reinelt
vm1084.tsp	1084	Problema de 1084 ciudades	Reinelt

5.2 Procedimiento

5.2.1 Mutación de la solución base

Una mutación consiste en obtener un resultado distinto en base a uno fijo, para ello se usará el concepto de Especie, una especie es un conjunto de variables numéricas compuestas por 2 arreglos del mismo tamaño, estos arreglos se llaman respectivamente cromosomas y genes cuya función se explicará a continuación.

Como se explicó con el método de cuadrantes, la constante división en partes más pequeñas permite realizar cálculos más rápidos; al momento de trazar una ruta se espera que el siguiente destino sea el más cercano al lugar de donde se partió y el uso de cuadrantes permite crear zonas aisladas donde algunos de los puntos se encuentran más cerca entre sí.

Sin embargo, ¿qué pasa si hay un punto más cercano a un grupo, pero debido a la división hecha por el algoritmo se ubica en un cuadrante diferente? En este caso se perdería un importante ahorro de distancia.

Para solucionar este problema se hará uso de las metaheurísticas creando especies de valores aleatorios permitiendo alterar la solución base y así obtener una solución diferente.

Como ejemplo se mostrará el problema eil51.tsp; en la figura 5.1 se muestra la solución obtenida por el método de cuadrantes con el resultado de 507 unidades de distancia, sin embargo se puede mejorar como muestra el recuadro dentro de la figura mencionada.

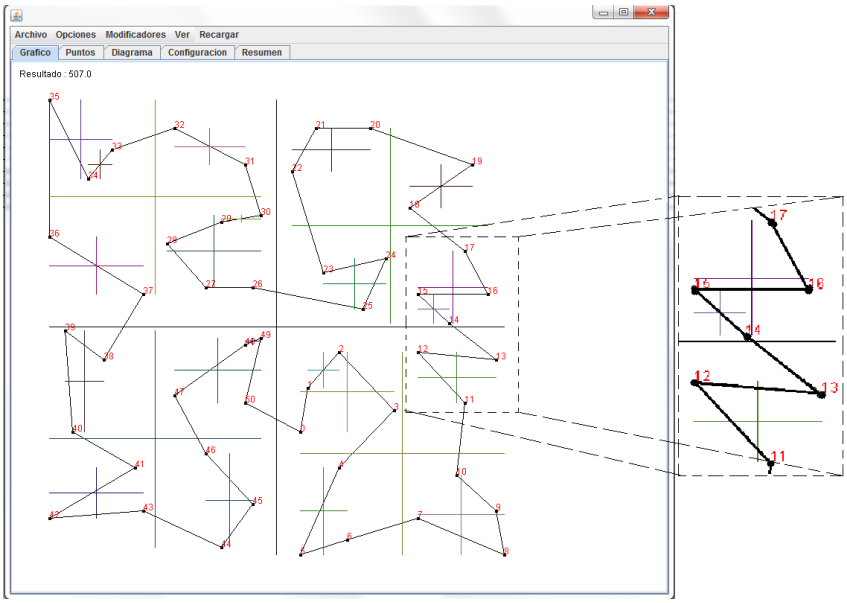

Figura 5.1: Problema eil51.tsp resuelto por el método de cuadrantes.

Como se ha mencionado anteriormente, una especie está formada por 2 arreglos, cada arreglo tiene uno de estos valores: el gen que se encargará de seleccionar uno de los puntos y el cromosoma que indicará hacia donde se desplazará el intercambio; el objetivo de la especie es reordenar los puntos seleccionados, y debido a que ya se resolvió el problema con el método de cuadrantes, no es necesario hacer intercambio con todos los elementos de la lista sino solo con los más cercanos.

En la figura 5.2 se podrá ver que se generó una especie formada por 2 arreglos, una señala al punto 11 y otro al 12,en ambos indica como tipo de cromosoma el número 2. El proceso se dividió en 3 fases:

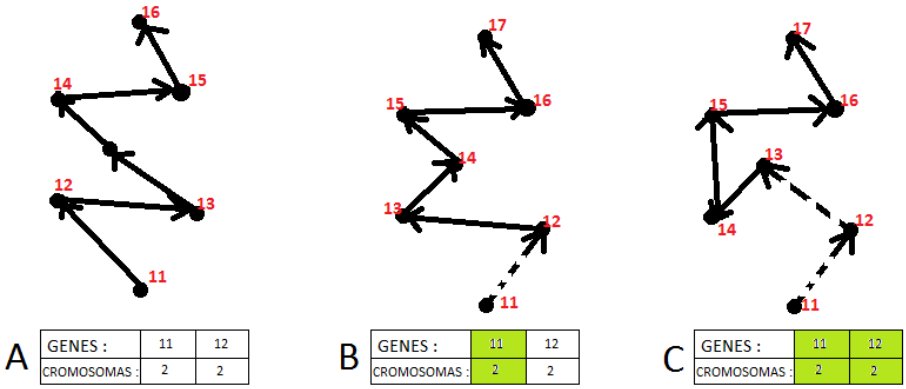

Figura 5.2: Problema eil51.tsp resuelto por el método de cuadrantes con una especie aplicada.

- **A:** Éste es el tramo inicial que donde se puede observar que aún no se ha modificado el segmento de la ruta.
- **B:** Aquí se hace el reemplazo, el gen marca el punto 11 y el cromosoma 2, eso quiere decir que el punto 11 tendrá como siguiente objetivo aquél que este 2 lugares delante de él (13), se intercambiarán las posiciones 12 con el 13, generando una nueva ruta, a diferencia de la fase A, el punto 12 y 13 se encuentran en lugares distintos.
- **C:** Por último se repite lo mostrado en la fase B, esta vez con el nuevo punto 12 y el 14, de nuevo intercambiarán posiciones los puntos 13 y 14 volviendo a generar una nueva ruta.

En la figura 5.3 se muestra la ruta completa con el cambio realizado que tendrá de resultado 502 unidades de distancia, ahorrándose 5 unidades. Este ejemplo es solo hipotético, y para que pueda funcionar se tendrán que hacer varias generaciones de especies, algunos haciendo movimientos para llegar a dicha solución, además de que el reordenamiento puede generar rutas más cortas pero a su vez otras más largas. El objetivo de este procedimiento es ir refinando las especies generadas con el fin de obtener una que mejore la solución obtenida.

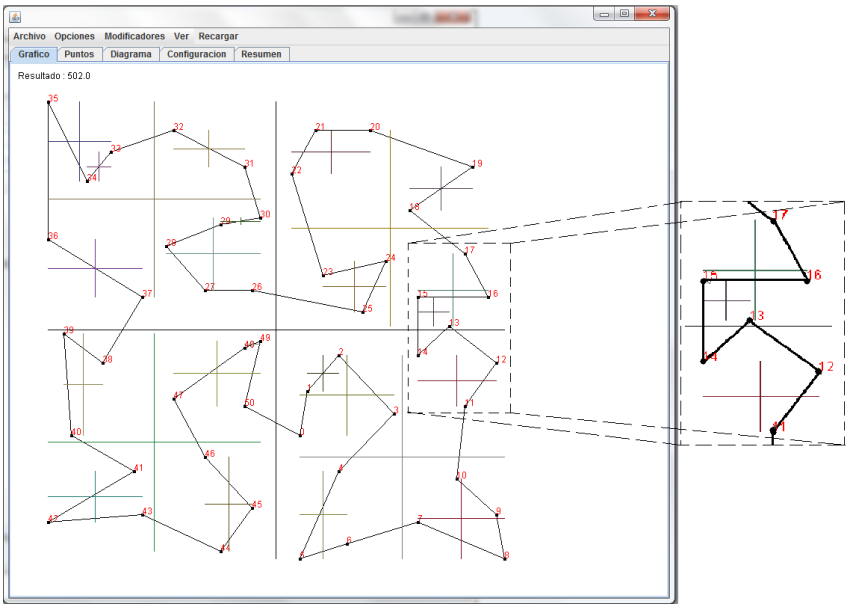

Figura 5.3: Problema eil51.tsp resuelto por el método de cuadrantes explicando cómo funciona el uso de las especies.

5.3 Metaheurísticas aplicadas en las pruebas

A continuación se mostrarán las metaheurísticas aplicadas en los experimentos, primero se explicarán los pasos realizados que se muestran en el diagrama de la figura 5.4, después se mostrará la funcionalidad de cada metaheurística programada para este experimento junto con la configuración de parámetros usados en el mismo.

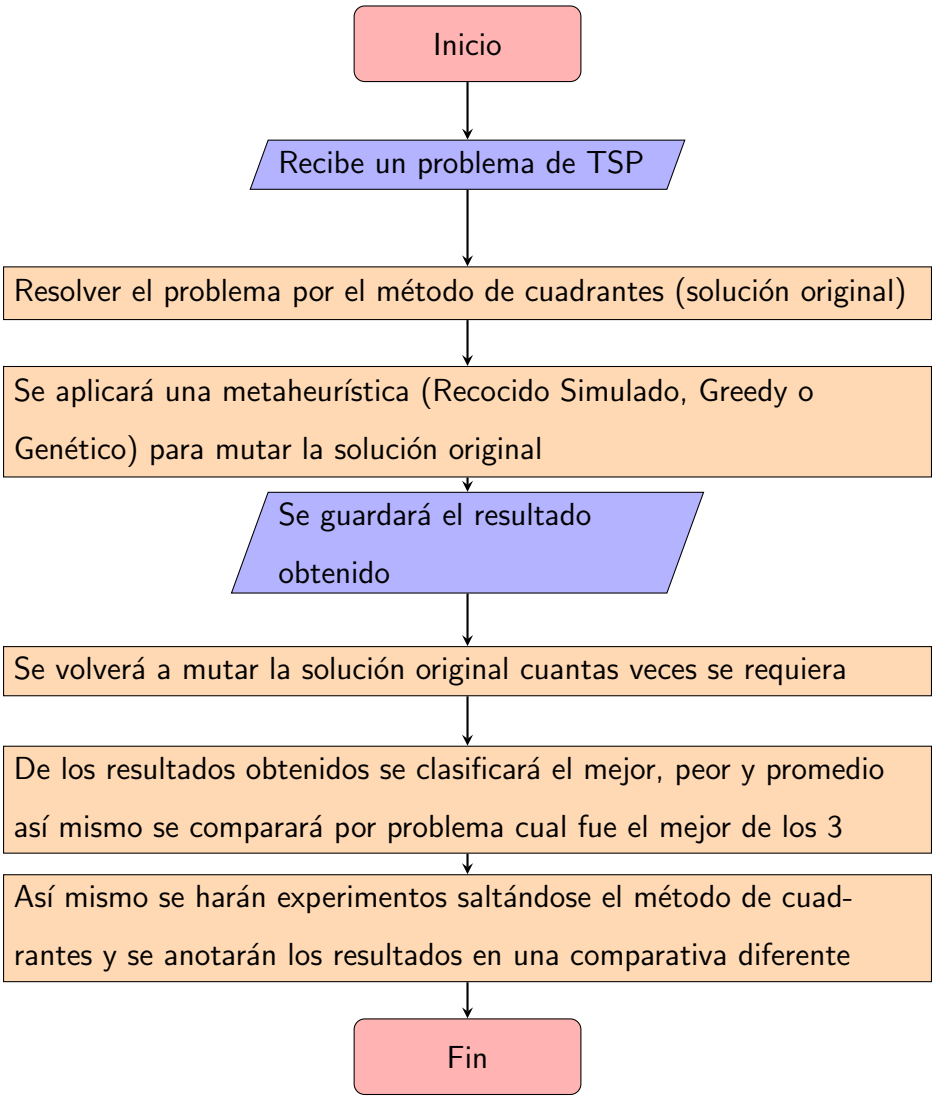

Figura 5.4: Diagrama de flujo del experimento de método de cuadrantes con metaheurísticas.

5.3.1 Algoritmo de Recocido Simulado

Aquí se aplicará lo aprendido en el código 2.3 donde usando un valor de aleatoriedad combinado con la diferencia de resultados obtenidos se irá viendo si el resultado (mejor o peor que el anterior) será el objetivo actual. En la tabla 5.2 se describen las variables usadas y en el código 5.1 se muestra la implementación del algoritmo de Recocido simulado.

Tabla 5.2: Configuración de las variables de recocido simulado durante las pruebas.

Nombre	Descripción	Valor
Ciclos	Cantidad de veces en que se repetirá el proceso	10000
Genes máximos	La cantidad de genes que tendrá la especie durante el proceso de búsqueda local	100
Distancia gen mínimo	El número mínimo que tendrá en el intercambio de posiciones	2
Distancia gen máximo	El número máximo que tendrá en el intercambio de posiciones	5

Código 5.1 Algoritmo de recocido simulado aplicado en las pruebas

```
Inicio

Ciclos
Genes máximos
Distancia gen mínimo
Distancia gen máximo
Factor de temperatura

Recibir una lista de puntos para trabajar convirtiéndose en la lista de
    puntos actual.
MIENTRAS(NO se hayan cumplido la cantidad de Ciclos){
 1.- Crear una nueva especie con una longitud igual a la cantidad de genes m
    áximos.
 2.- Los números que conformara la especie oscilara entre el número mínimo y
     máximo declarados anteriormente.
 3.- Una vez declarada alterara la lista de puntos actual para obtener una
    lista nueva.
 4.- Una vez alterada la lista de puntos actual con la nueva, en caso de
    obtener un mejor resultado
 SI(el resultado es mejor){
  1.-esta lista es sustituida por la nueva.
 }SINO{
  1.-Aplicar formula de temperatura
  SI(APLICA){
      1.-La especie, aunque ineficiente se convierte en la lista de puntos
         actual.
      2.-Se ajusta la temperatura.
  }
 }
}

Fin
```

5.3.2 Algoritmo Greedy

A diferencia del algoritmo de recocido simulado, el algoritmo Greedy (o voraz) es más directo, intenta sacar al azar una solución y verifica que éste sea el mejor, si lo es sustituye al original, sino lo es lo descarta y continua el proceso hasta que se cumpla el criterio de paro, que en este caso son el número de veces (o ciclos) que se repetirá el proceso. En la tabla 5.3 se describen las variables usadas y en el código 5.2 muestra la implementación del algoritmo Greedy.

Tabla 5.3: Configuración de las variables de método greedy durante las pruebas.

Nombre	Descripción	Valor
Ciclos	Cantidad de veces en que se repetirá el proceso.	10000
Genes máximos	La cantidad de genes que tendrá la especie durante el proceso de búsqueda local.	100
Distancia gen mínimo	El número mínimo que tendrá el intercambio de posiciones.	2
Distancia gen máximo	El número máximo que tendrá el intercambio de posiciones.	5

Código 5.2 Algoritmo greedy aplicado en las pruebas

```
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
    Inicio

    Ciclos
    Genes máximos
    Distancia gen mínimo
    Distancia gen máximo

    Recibir una lista de puntos para trabajar convirtiéndose en la lista de
        puntos actual.
    MIENTRAS(NO se hayan cumplido la cantidad de Ciclos){
     1.- Crear una nueva especie con una longitud igual a la cantidad de genes m
        áximos
     2.- Los números que conformará la especie oscilará entre el número mínimo y
         máximo declarados anteriormente
     3.- Una vez declarada alterara la lista de puntos actual para obtener una
        lista nueva.
     4.- Una vez alterada la lista de puntos actual con la nueva, en caso de
        obtener un mejor resultado esta lista es sustituida por la nueva
    }

    Fin
```

5.3.3 Algoritmo Genético

Por último el algoritmo genético en vez de modificar una solución candidata, utiliza una población de ellas donde soluciones con mejor resultado tendrá la prioridad para reproducirse, y durante varias generaciones los descendientes irán modificando el resultado original hasta que al final devuelva uno mejor. Entre los aspectos a destacar son el hecho de dividir los genes en dominantes y recesivos: los primeros jamás se modificarán y representan aquellos cambios que son capaces de mejorar el resultado y luego están los recesivos que estarán sujetos a una mutación devolviendo otro conjuntos de valores numéricos que pueden o no mejorar la solución. En la tabla 5.4 se describen las variables usadas y en el código 5.3 muestra la implementación del algoritmo genético.

Tabla 5.4: Configuración de las variables de algoritmo genético durante las pruebas.

Nombre	Descripción	Valor
Generaciones	Cantidad de veces que se repetirá el proceso, en este caso la cantidad de nuevas generaciones de hijos.	1000
Población	Cantidad máxima que tendrá cada generación, en caso de ser impar se eliminará el único que quede sin pareja durante el proceso de cruza.	100
Genes máximos	La cantidad de genes que tendrá la especie durante el proceso de búsqueda local.	100
Distancia gen mínimo	El número mínimo que tendrá el intercambio de posiciones.	2
Distancia gen máximo	El número máximo que tendrá el intercambio de posiciones.	5
Rango de mutación mínimo	Porcentaje aleatorio mínimo de genes recesivos que serán modificados durante la mutación, esta cantidad se calcula junto con el valor de genes máximos.	10
Rango de mutación máximo	Porcentaje aleatorio máximo de genes recesivos que serán modificados durante la mutación, esta cantidad se calcula junto con el valor de genes mínimos.	40

Código 5.3 Algoritmo genético aplicado en las pruebas

```
Inicio

Generaciones
Población
Genes máximos
Distancia gen mínimo
Distancia gen máximo
Rango de mutación mínimo
Rango de mutación máximo

1.-Recibir una lista de puntos para trabajar convirtiéndose en la lista de
    puntos actual.

MIENTRAS(NO se hayan creado la cantidad de Generaciones establecidas)
{
 1.-Se crea una nueva población conformada de Especies.

 2.-Cada Especie tiene una cantidad de genes igual al valor de genes máximos
     y sus valores oscilan entre los números máximos y mínimos de distancia
    .

 3.-Todas las especies se cruzan con la lista de puntos actual, se ordenan
    de acuerdo a su desempe o.

 4.-Si mejor especie supera a la solución actual, es reemplazada, de lo
    contrario la solución actual permanecerá dentro de la población
    sustituyendo a la solución más débil.

 5.-Una vez que se haya terminado la evaluación se procederá el siguiente
    paso.

 MIENTRAS (Existan Especies sin pareja)
 {
  1.- La mejor especie selecciona al azar cualquier miembro de la población
      convirtiéndose en padre y madre de la siguiente especie.

  2.- Este hijo sera el resultado de la combinación de los genes en
      posiciones pares del padre y los genes de las posiciones impares de la
       madre.

  3.- Los genes que ayuden a mejorar el rendimiento de la solución se
      llamaran dominantes mientras que los que afecten se llamaran recesivos
      .

  4.- Los genes recesivos se someterán a un proceso de mutación, donde sus
      valores serán alterados de acuerdo a las distancias mínimas y máximas,
       la cantidad de genes mutados será determinado por los rangos mínimos
      y máximos de mutación.

  5.- El hijo sera agregado a la nueva población y los padres serán
      descartados.

  6.- El proceso se repite esta vez con la siguiente mejor Especie.

 }
}

Fin
```

5.4 Experimentos

A continuación se presenta una breve explicación de los elementos que compondrán los experimentos. Se llevaron a cabo 15 experimentos con las instancias de la TSPLIB. Se hicieron 100 corridas de cada problema con el fin de obtener diferentes resultados. Todos los experimentos mantienen la misma configuración y se prefirió no ajustarla con el fin de asegurar el mismo rendimiento en ambos problemas.

Los datos arrojados quedaron registrados en las siguientes 3 presentaciones:

A.- Una tabla comparativa del mismo experimento (figura 5.5) que muestra los resultados agrupados por tipo de metaheurística y si se aplicó o no el método de cuadrantes. Las celdas coloreadas ayuda a representar entre los 3 el mejor resultado, el mejor porcentaje y de los peores resultados el más pequeño.

B.- Una imagen comparativa del resultado original (sin aplicar metaheurísticas) y otro que muestra el mejor resultado usando la metaheurística ganadora (en ambos están aplicando el método de cuadrantes). Como se puede observar en la figura 5.6, los círculos marcados muestran los puntos cuyo orden es distinto en comparación a la imagen de la izquierda.

C.- Una gráfica comparativa mostrando los resultados de las 100 corridas (figura 5.7) que están ordenados de manera descendente para observar el progreso comparado con las demás metaheurísticas. Esta gráfica muestra los resultados obtenidos aplicando o no el método de cuadrantes.

NOMBRE DEL EXPERIMENTO

lin318.tsp				
Resultado Original : 55340		Promedio	Mejor	Peor
Con cuadrantes	Recocido	54438.22	53752	55085
	Greedy	54367.97	53773	55073
	Genético	53997.96	53479	54344
Sin cuadrantes	Recocido	110739.67	102042	118920
	Greedy	110691.43	104454	117441
	Genético	120441.11	114461	125074

Resultado obtenido únicamente por el método de cuadrantes.

La tabla está dividida en 2 categorías: si se aplicó o no el método de cuadrantes y cada clasificación se divide en el tipo de metaheurística aplicado sobre el resultado original.

Se realizaron 100 pruebas por cada configuración posible, dando un total de 600 corridas.

Los resultados obtenidos se ordenan del mejor a peor y se calcula un promedio.
El resultado más destacado es coloreado de gris.

Figura 5.5: Explicación del experimento (punto A), tabla comparativa.

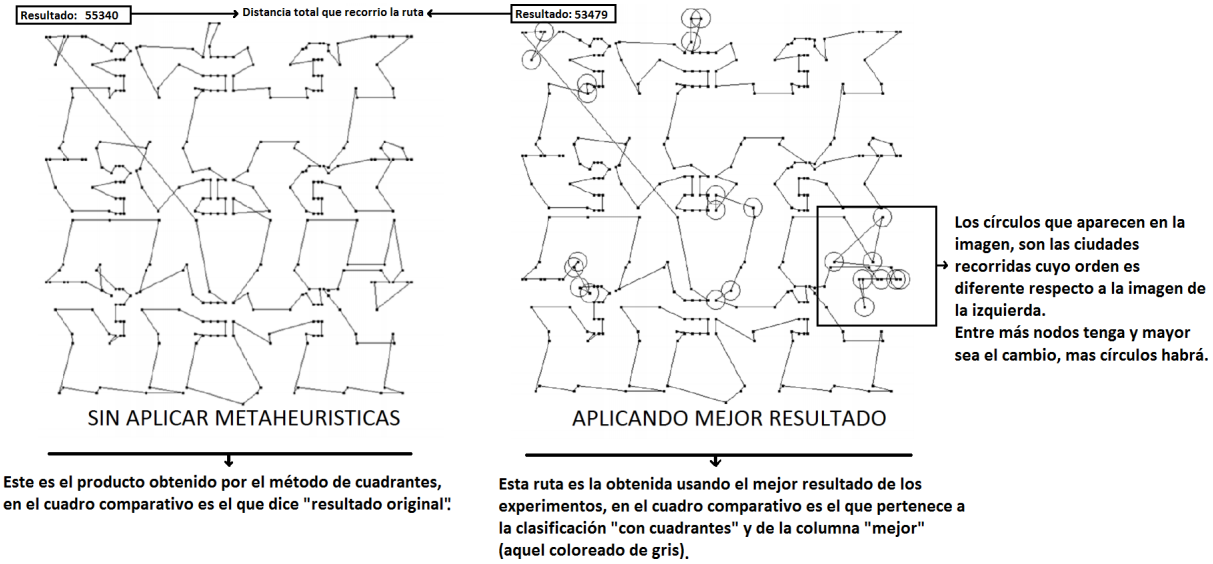

Figura 5.6: Explicación del experimento (punto B), comparación de rutas.

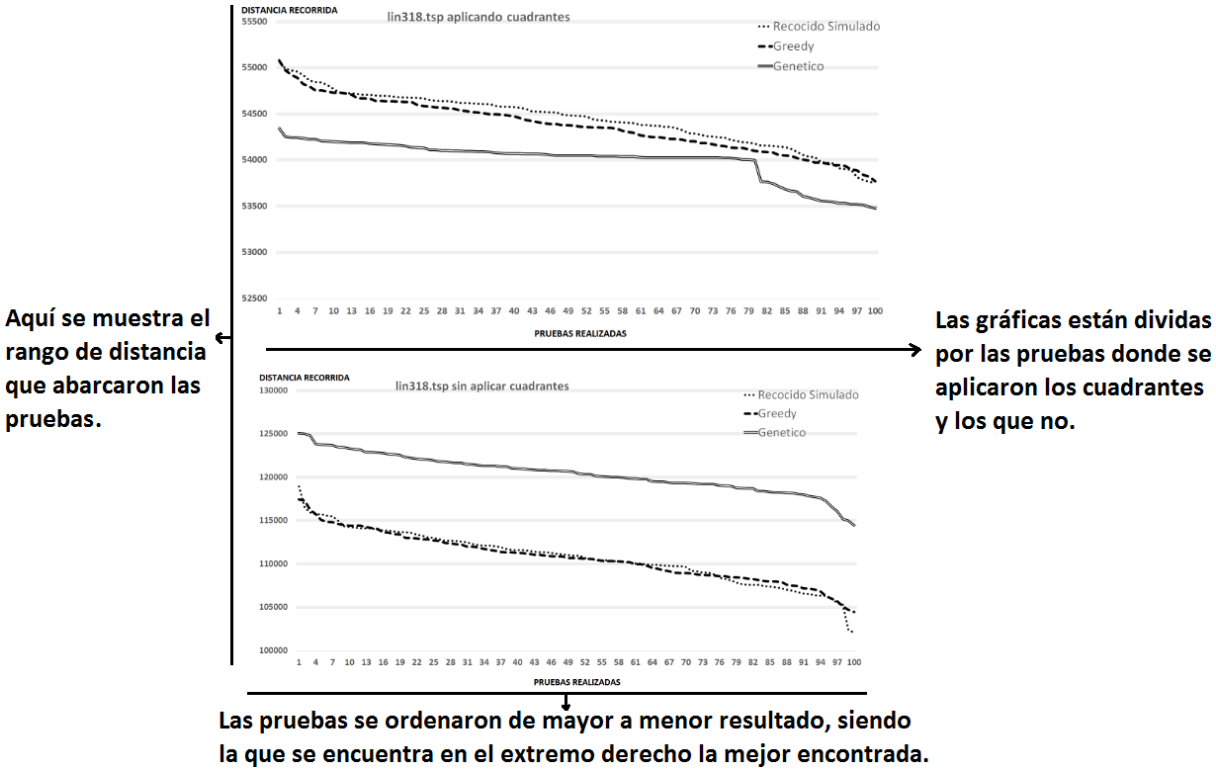

Figura 5.7: Explicación del experimento (punto C), gráfica que compara los resultados obtenidos aplicando o no el método de cuadrantes.

5.4.1 a280.TSP

Tabla 5.5: Experimento con el problema a280.TSP.

a280.TSP				
Resultado Original : 3418		Promedio	Mejor	Peor
Con cuadrantes	Recocido	3364.17	3329	3394
	Greedy	3363.79	3335	3396
	Genético	3354.06	3318	3418
Sin cuadrantes	Recocido	5078.51	4711	5345
	Greedy	5101.88	4804	5410
	Genético	4876.34	4747	5066

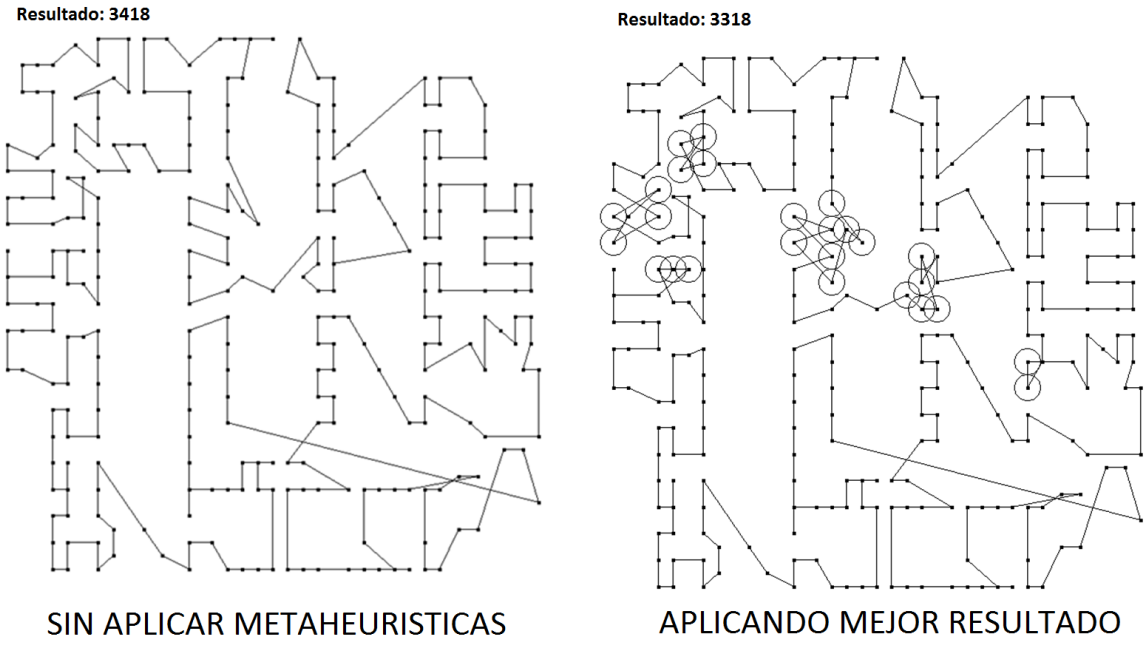

Figura 5.8: Comparativa a280.tsp.

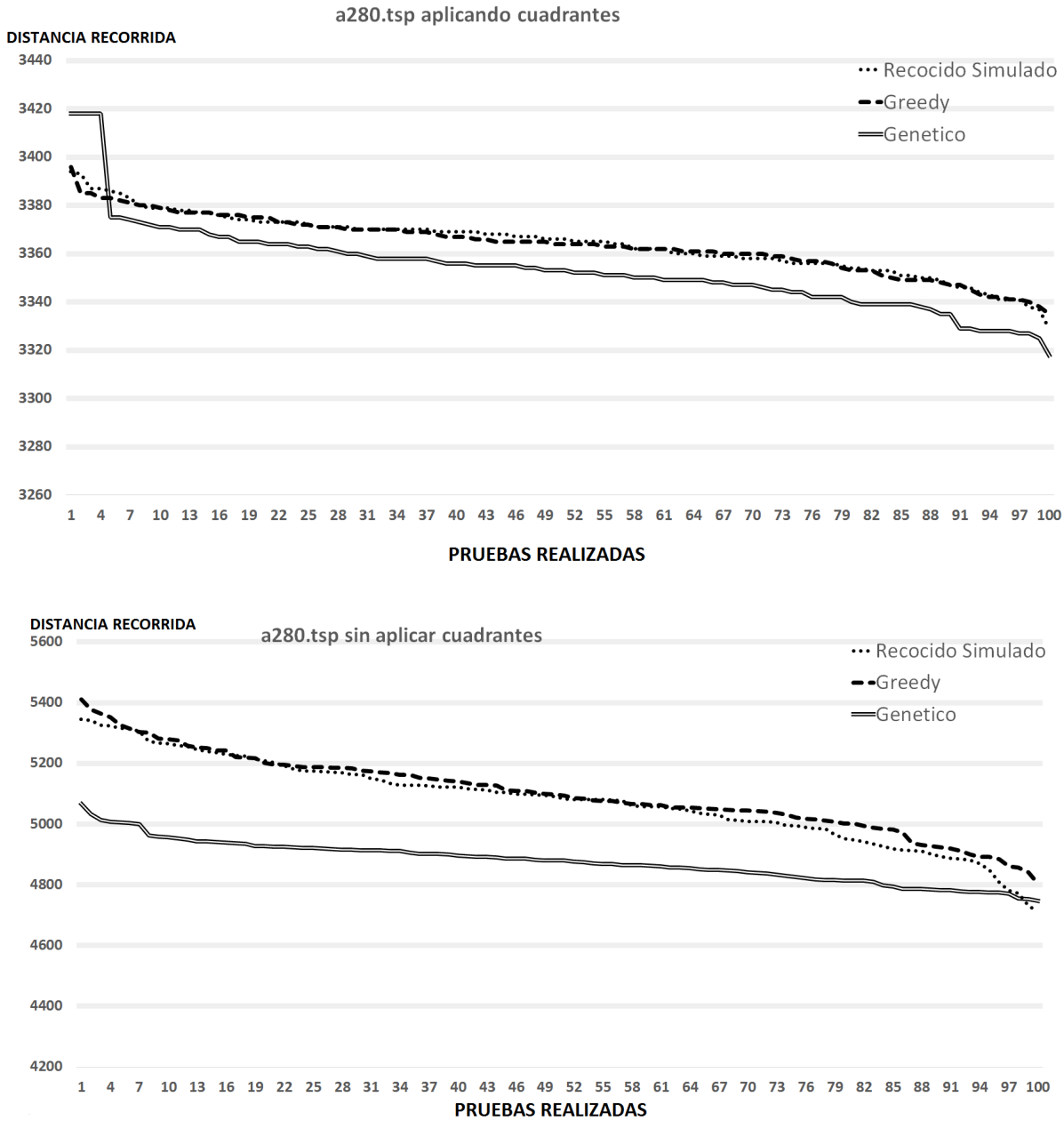

Figura 5.9: Gráficos a280.tsp con cuadrantes y sin cuadrantes.

Como se puede observar en la figura 5.10, los círculos marcan las zonas mutadas de la solución obtenida mediante el método de cuadrantes. En la figura 5.8 algunos puntos que se muestran en la imagen de la derecha están encerrados en círculos, estos puntos están marcados así porque fueron colocados fuera de la posición original que se le asignó por el método de cuadrantes.

Regresando a la figura 5.10 se puede ver 3 imágenes:

- **A:** Es el problema resuelto por el método de cuadrantes.
- **B:** El mismo problema, solo que fue modificado usando metaheurísticas, aquí se puede apreciar que varios puntos fueron encerrados en círculos ya comentando su funcionalidad
- **C:** Es la imagen B sin los puntos marcados, para que se puede apreciar con mejor detalle las diferencias que tiene con la imagen A.

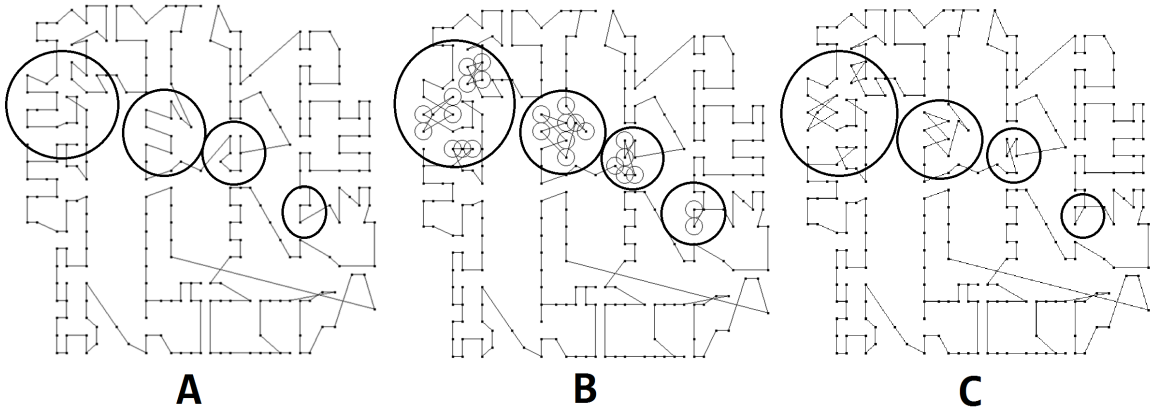

Figura 5.10: Explicación de los cambios realizados en el a280.tsp.

Ya conociendo para que sirve esta imagen se procederá con los demás experimentos.

5.4.2 brd14051.TSP

Tabla 5.6: Experimento con el problema brd14051.tsp.

brd14051.tsp				
Resultado Original :623324		Promedio	Mejor	Peor
Con cuadrantes	Recocido	619876.89	618878	620546
	Greedy	619899.03	619060	620743
	Genético	621842.30	621273	622688
Sin cuadrantes	Recocido	13091307.46	12612710	13458979
	Greedy	13071094.56	12811921	13411800
	Genético	23478563.40	23433764	23520252

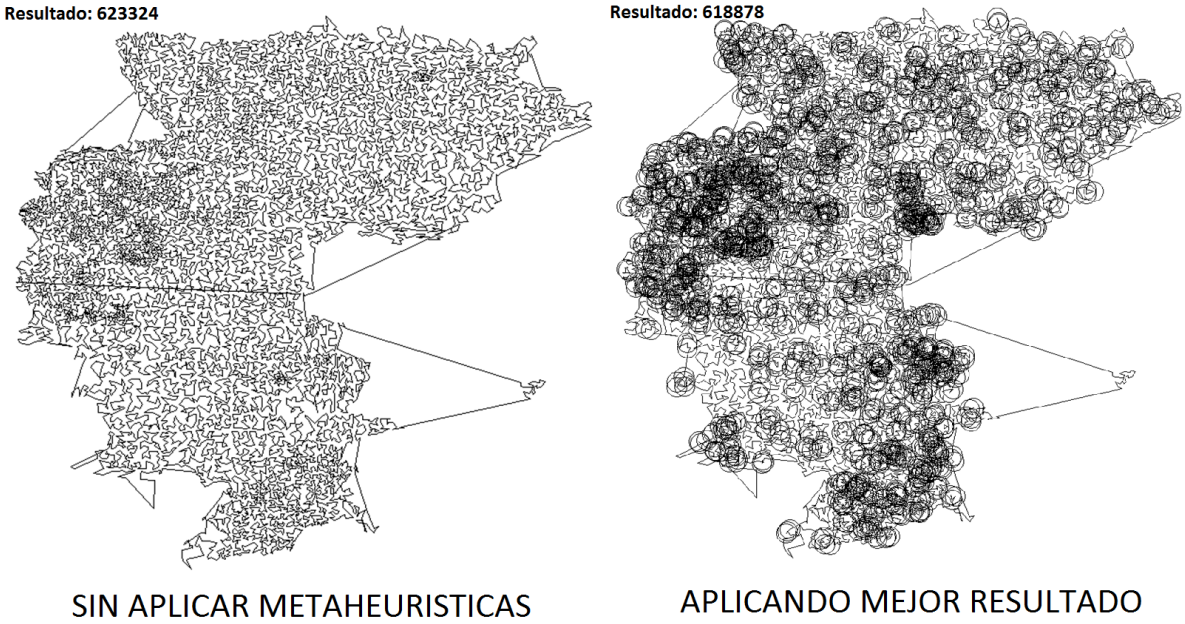

Figura 5.11: Comparativa brd14051.tsp.

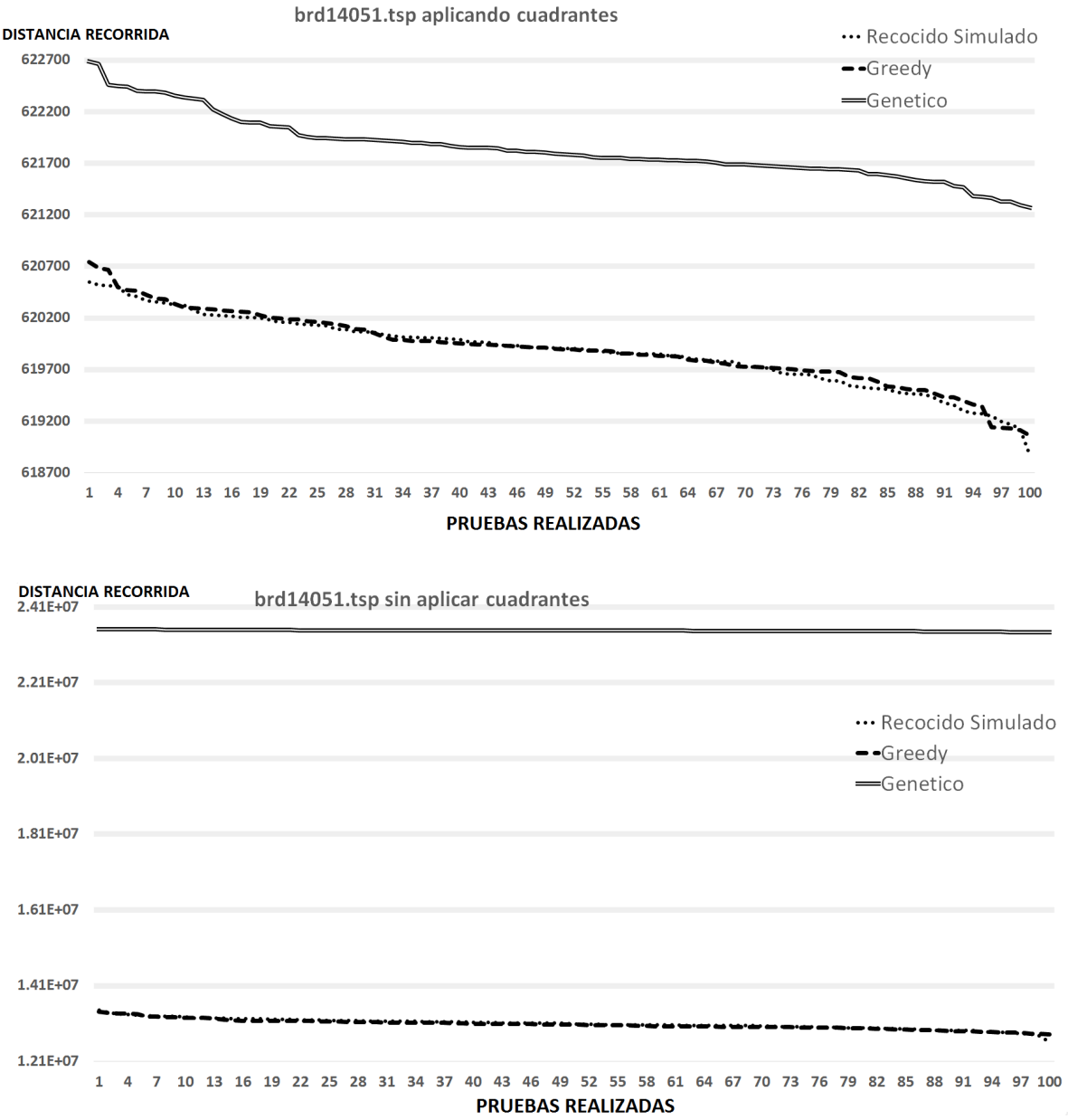

Figura 5.12: Gráficos brd14051.tsp con cuadrantes y sin cuadrantes.

5.4.3 ch150.TSP

Tabla 5.7: Experimento con el problema ch150.tsp.

ch150.tsp				
Resultado Original : 8579		Promedio	Mejor	Peor
Con cuadrantes	Recocido	8445.98	8272	8535
	Greedy	8446.02	8315	8540
	Genético	8358.81	8231	8427
Sin cuadrantes	Recocido	26500.85	23162	28674
	Greedy	26614.55	22417	29187
	Genético	34548.11	32091	37844

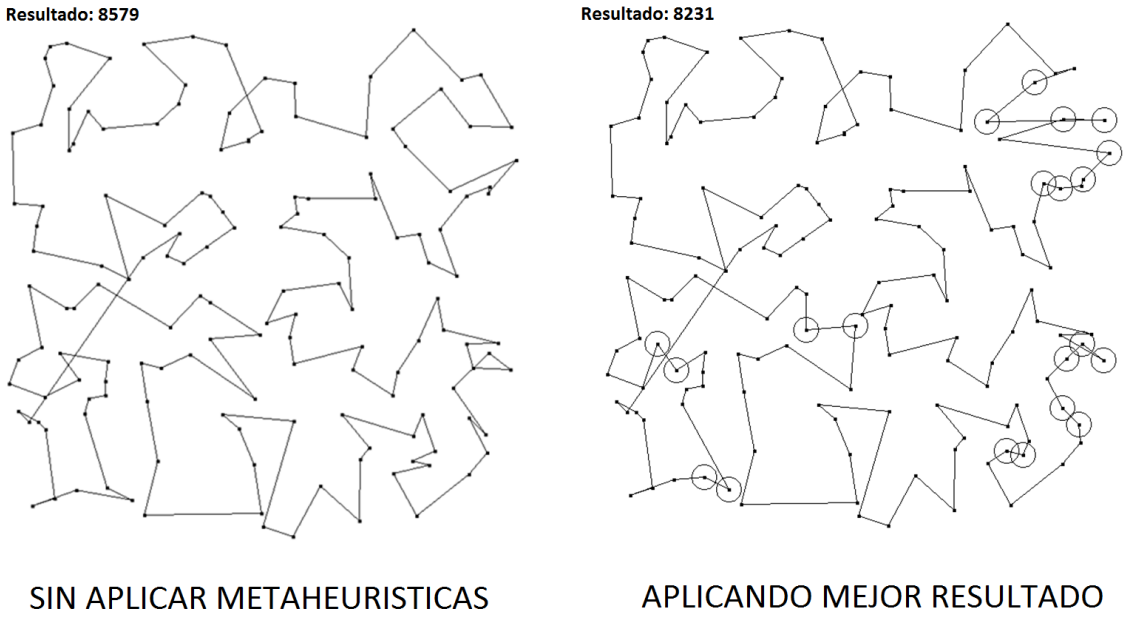

Figura 5.13: Comparativa ch150.tsp.

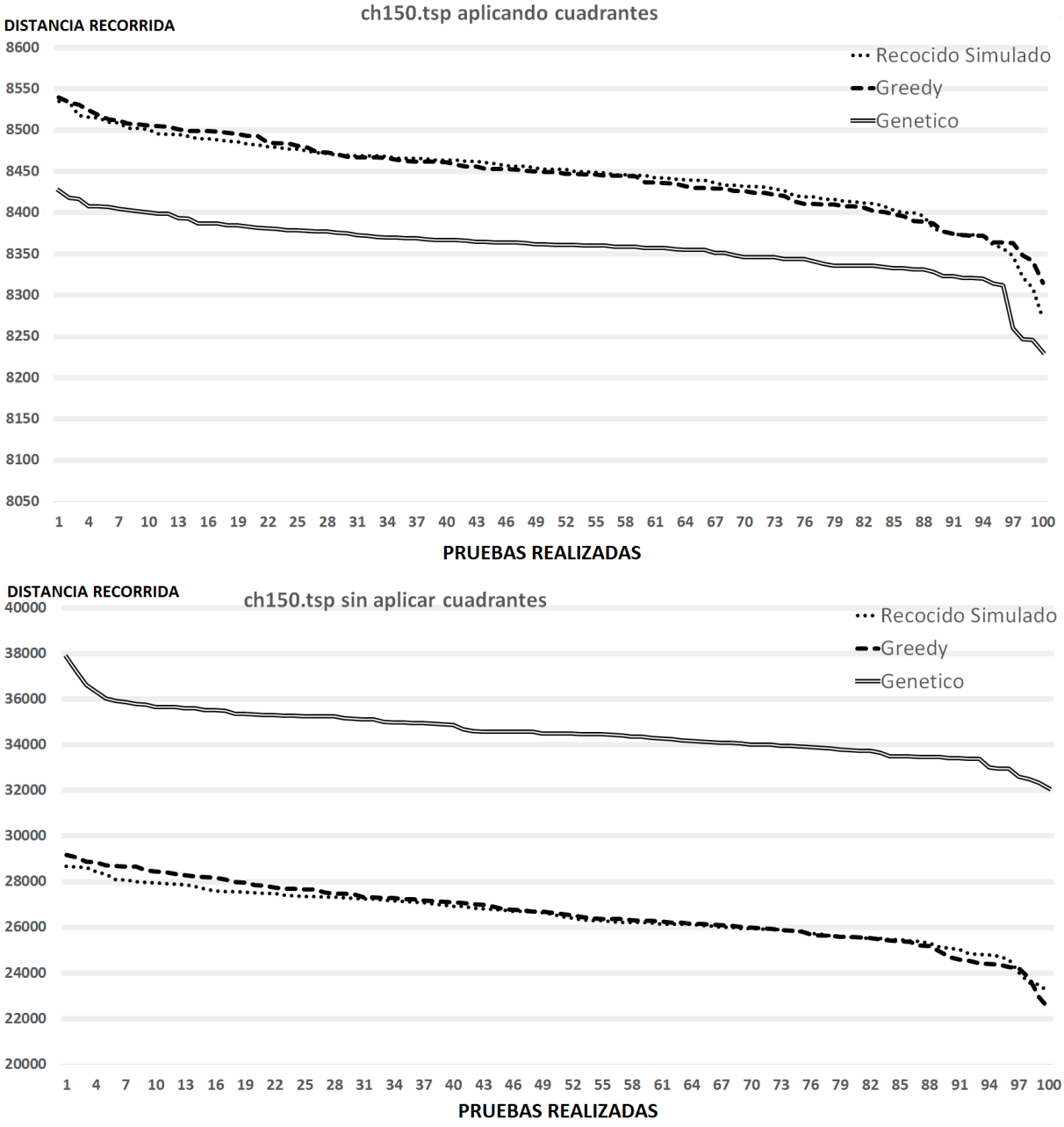

Figura 5.14: Gráficos ch150.tsp con cuadrantes y sin cuadrantes.

5.4.4 d1655.TSP

Tabla 5.8: Experimento con el problema d1655.tsp.

d1655.tsp				
Resultado Original :83605		Promedio	Mejor	Peor
Con cuadrantes	Recocido	82500.78	82098	82881
	Greedy	82489.63	82014	82839
	Genético	82244	81996	82447
Sin cuadrantes	Recocido	235605.87	222473	243750
	Greedy	236306.2	226740	246172
	Genético	264556.11	262348	266415

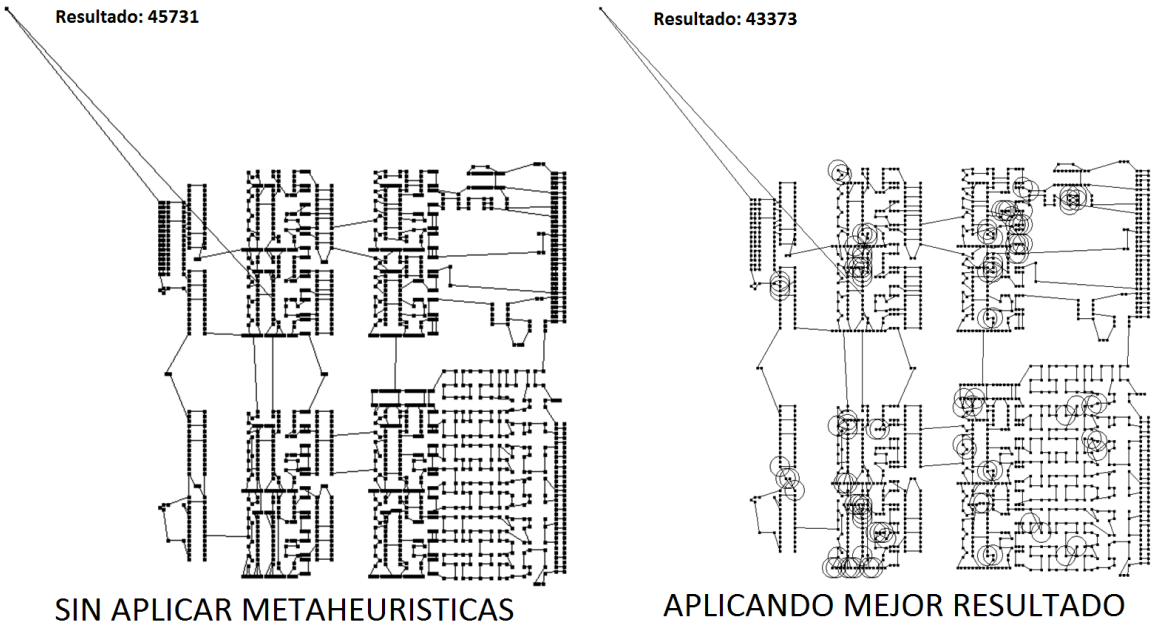

Figura 5.15: Comparativa d1655.tsp.

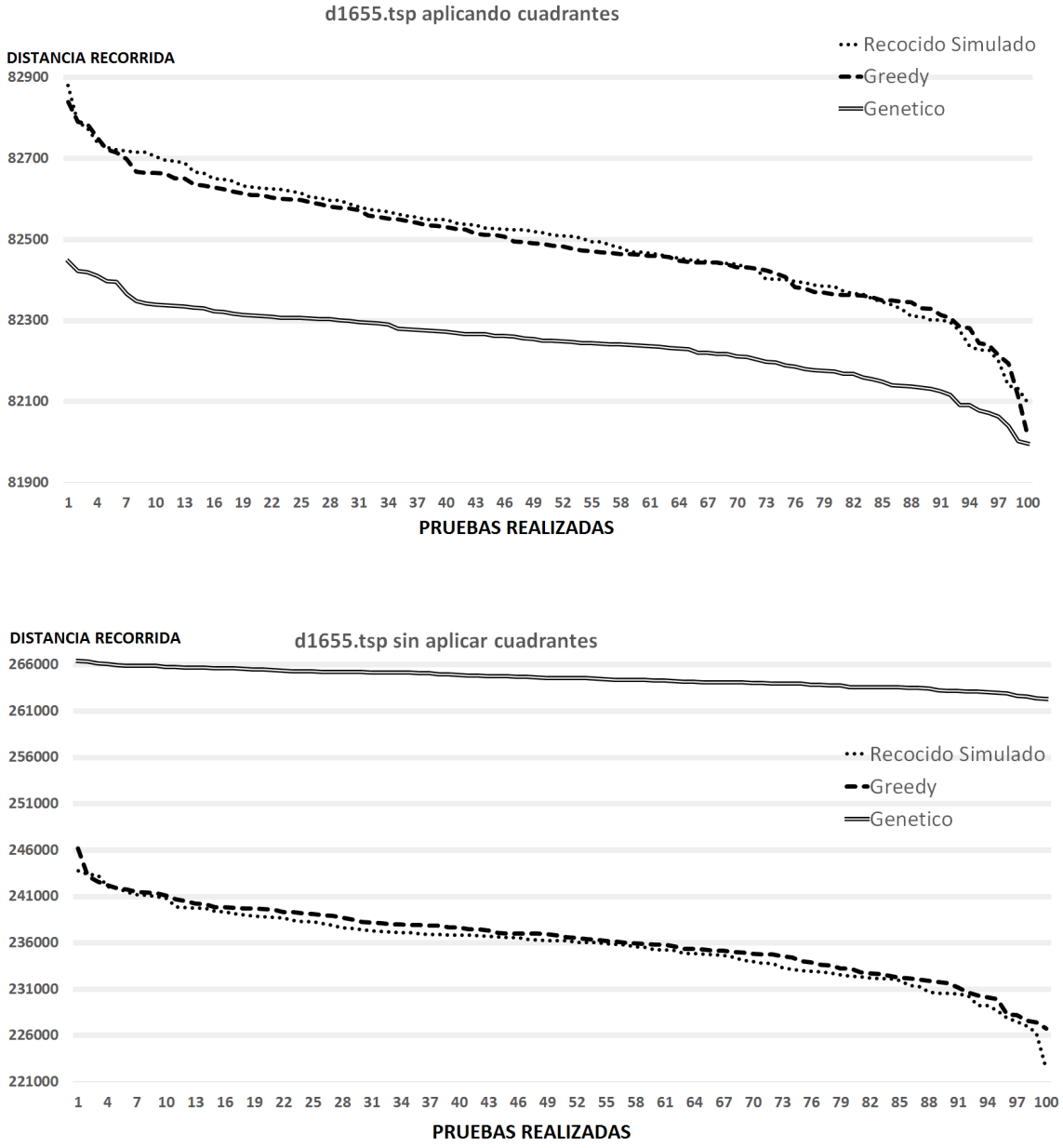

Figura 5.16: Gráficos d1655.tsp con cuadrantes y sin cuadrantes.

5.4.5 d493.TSP

Tabla 5.9: Experimento con el problema d493.tsp.

d493.tsp				
Resultado Original : 45731		Promedio	Mejor	Peor
Con cuadrantes	Recocido	44582.69	44124	45012
	Greedy	44604.21	44054	45275
	Genético	43987.43	43373	44918
Sin cuadrantes	Recocido	96827.13	91468	102751
	Greedy	96002.89	89956	100955
	Genético	121326.2	115991	124989

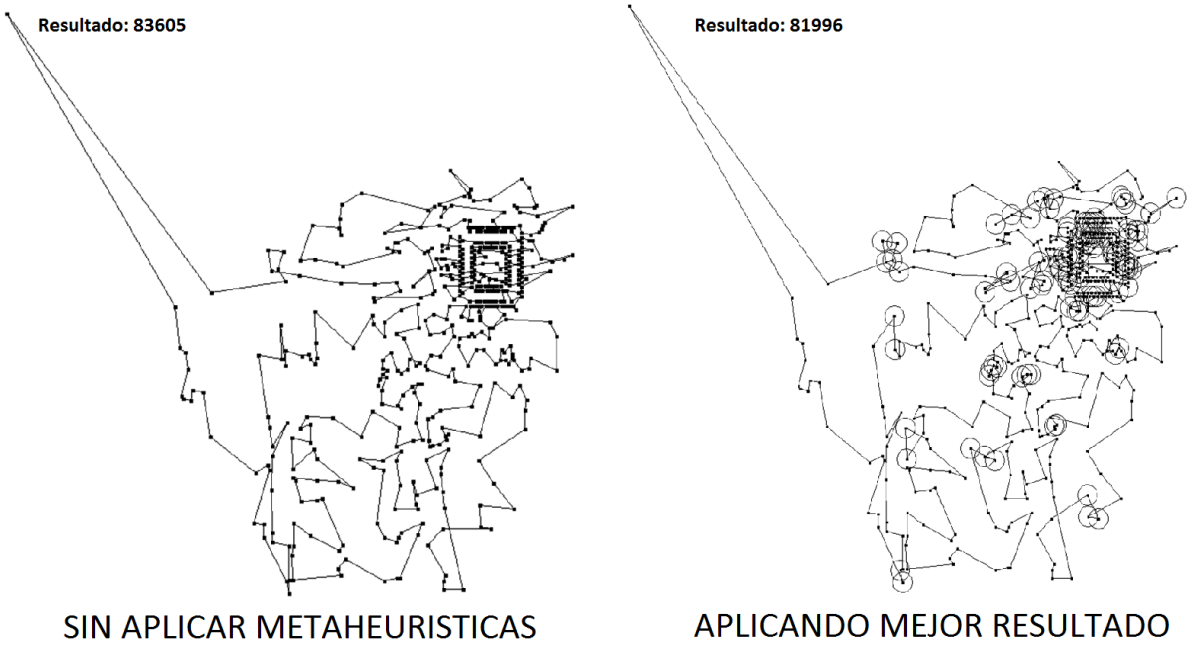

Figura 5.17: Comparativa d493.tsp.

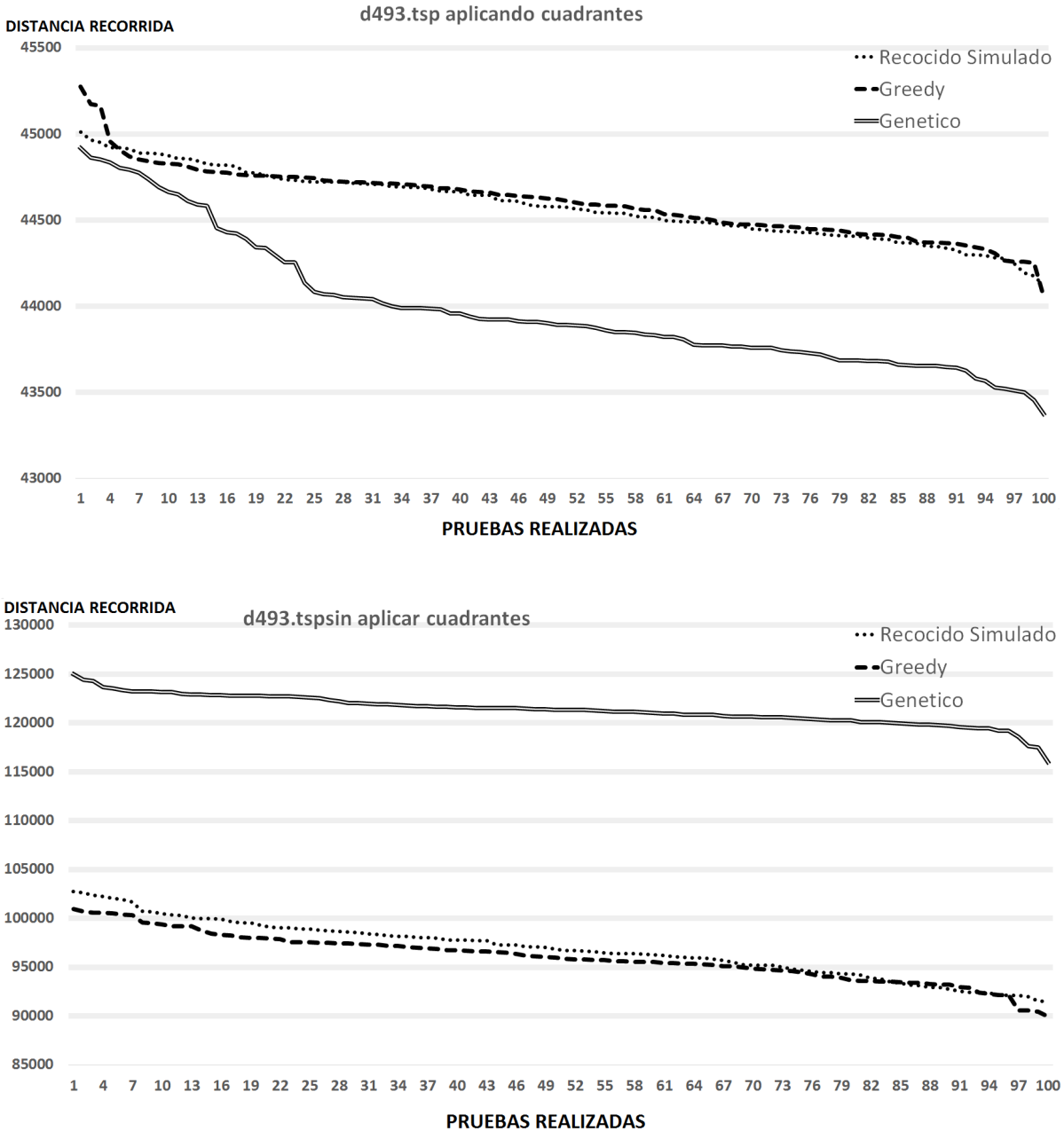

Figura 5.18: Gráficos d493.tsp con cuadrantes y sin cuadrantes.

5.4.6 eil101.TSP

Tabla 5.10: Experimento con el problema eil101.tsp.

eil101.tsp				
Resultado Original : 828		Promedio	Mejor	Peor
Con cuadrantes	Recocido	804.91	788	822
	Greedy	806.93	788	825
	Genético	787.33	781	802
Sin cuadrantes	Recocido	1747.41	1585	1917
	Greedy	1744.97	1630	1855
	Genético	1705.49	1602	1740

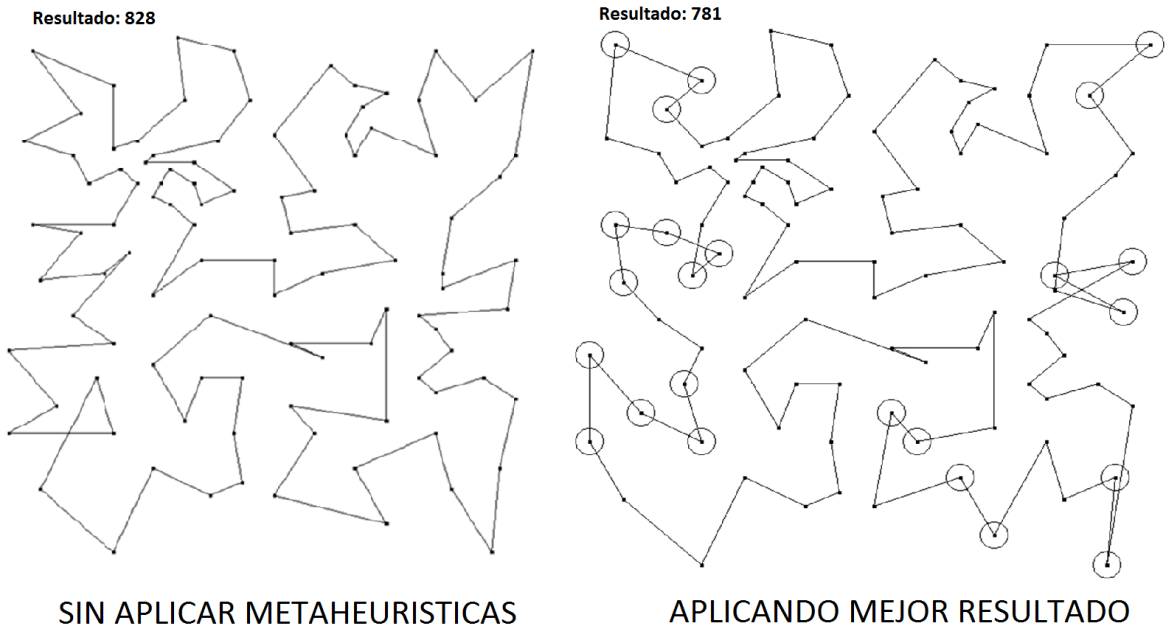

Figura 5.19: Comparativa eil101.tsp.

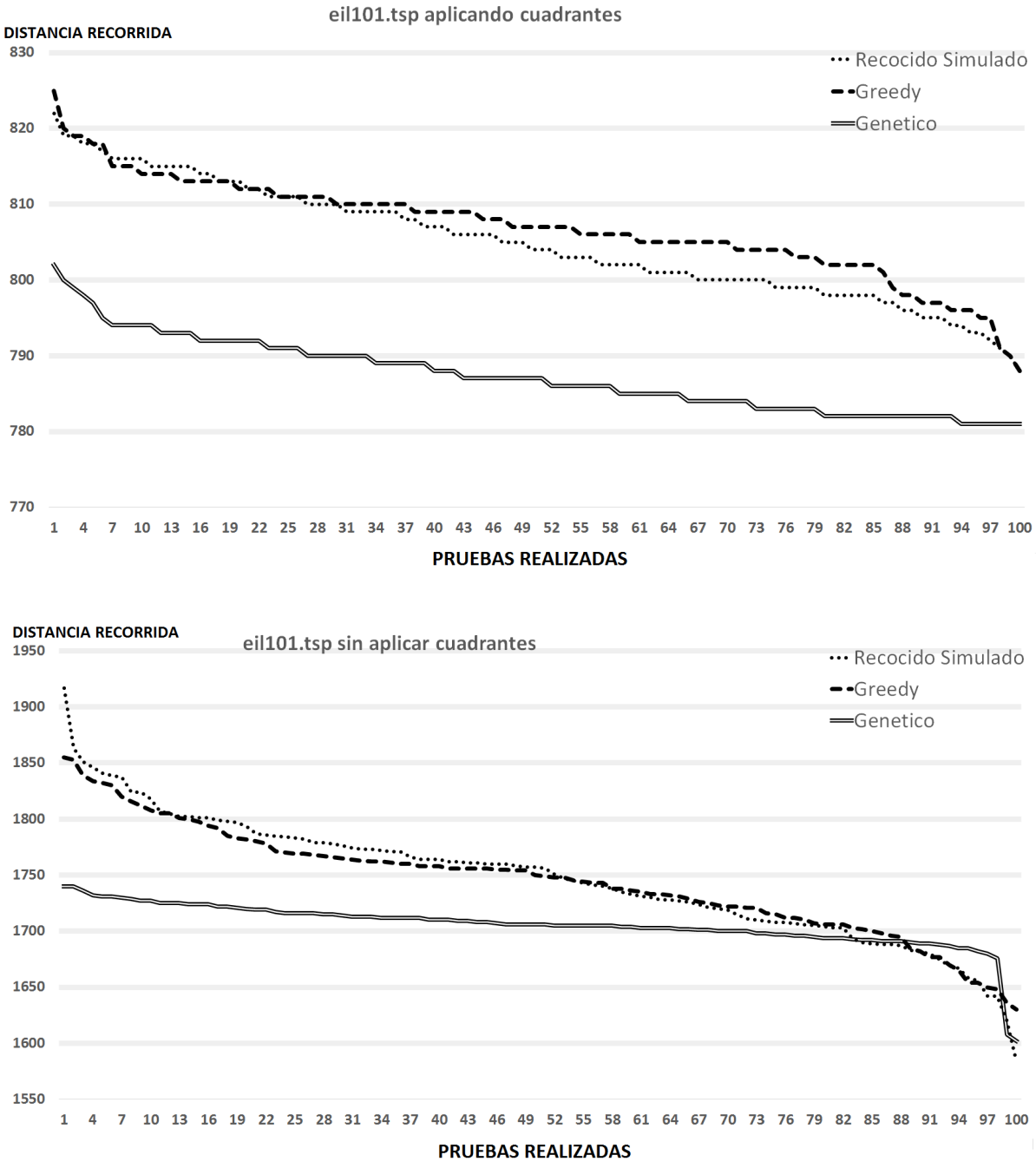

Figura 5.20: Gráficos eil101.tsp con cuadrantes y sin cuadrantes.

5.4.7 fl417.TSP

Tabla 5.11: Experimento con el problema fl417.tsp.

fl417.tsp				
Resultado Original : 17419		Promedio	Mejor	Peor
Con cuadrantes	Recocido	16835.95	16672	17070
	Greedy	16824.81	16610	17028
	Genético	16599.01	16450	16709
Sin cuadrantes	Recocido	48845.13	46858	50870
	Greedy	48689.67	46460	50570
	Genético	52319.03	50896	53761

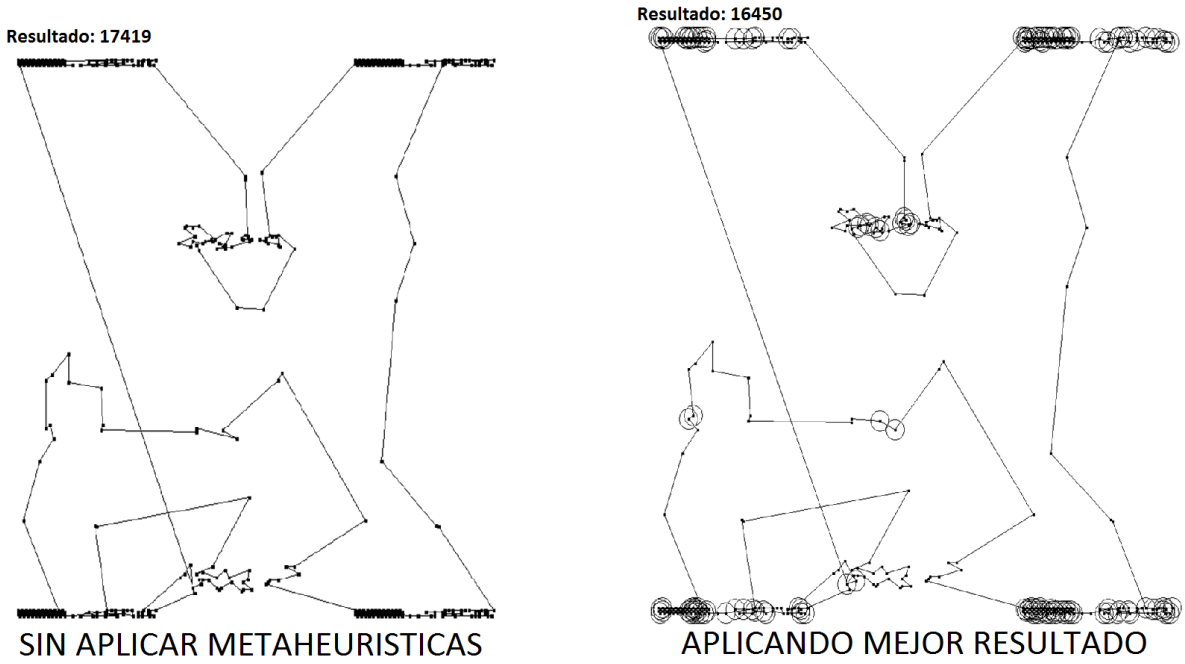

Figura 5.21: Comparativa fl417.tsp.

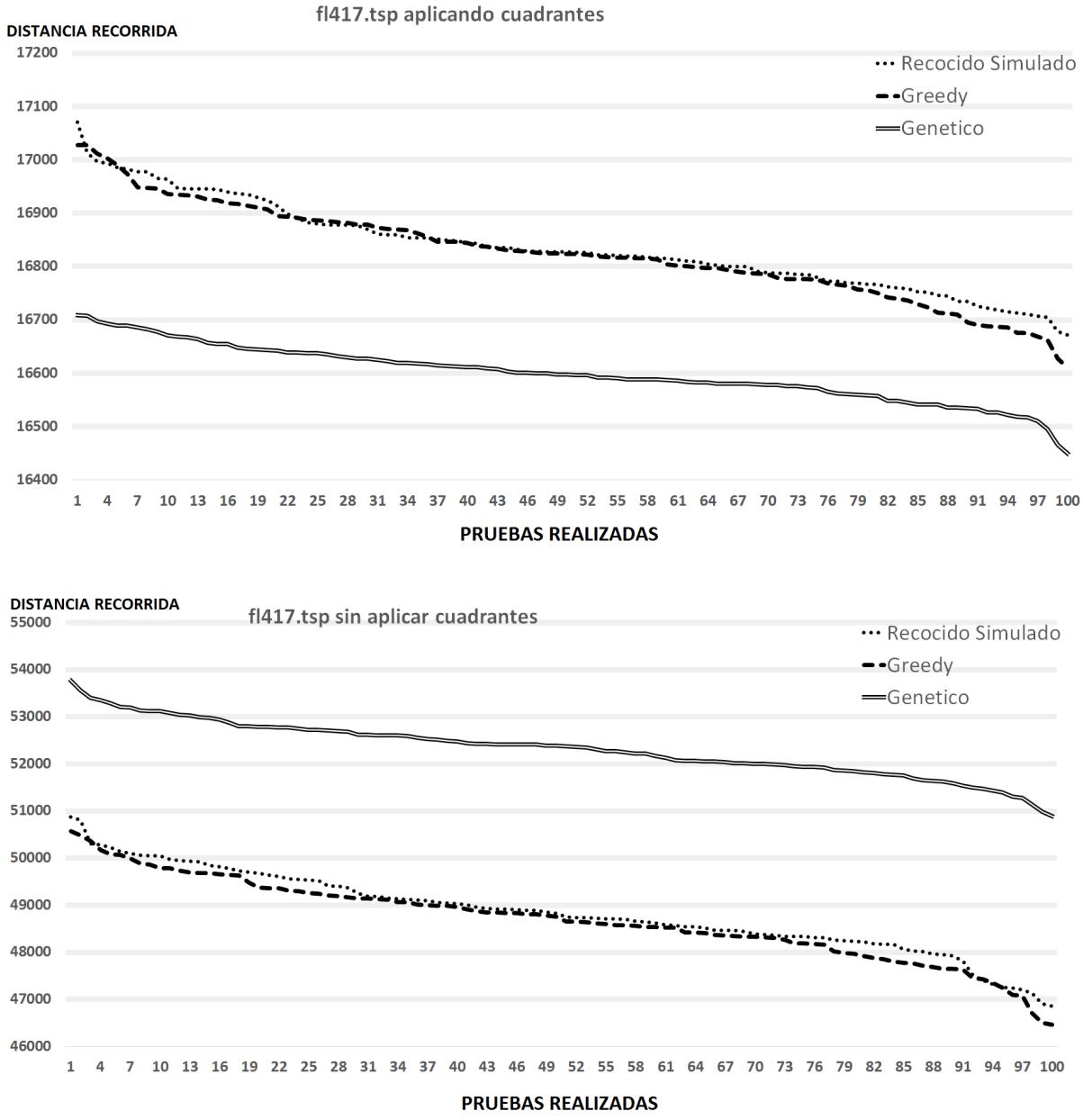

Figura 5.22: Gráficos fl417.tsp con cuadrantes y sin cuadrantes.

5.4.8 lin318.TSP

Tabla 5.12: Experimento con el problema lin318.tsp.

lin318.tsp				
Resultado Original : 55340		Promedio	Mejor	Peor
Con cuadrantes	Recocido	54438.22	53752	55085
	Greedy	54367.97	53773	55073
	Genético	53997.96	53479	54344
Sin cuadrantes	Recocido	110739.67	102042	118920
	Greedy	110691.43	104454	117441
	Genético	120441.11	114461	125074

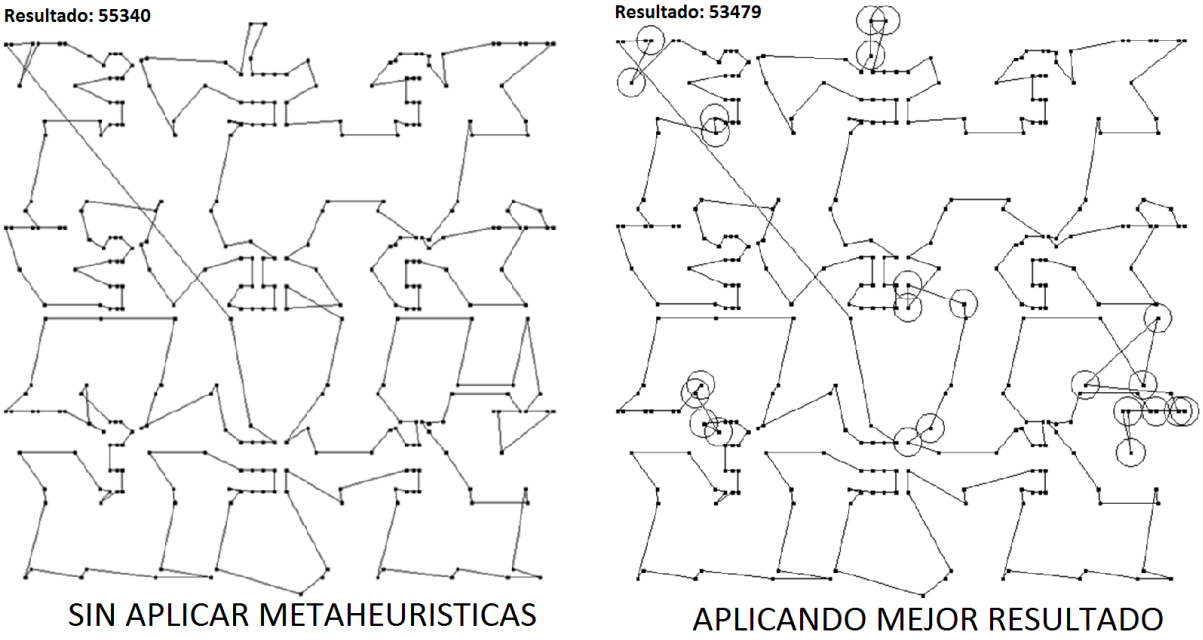

Figura 5.23: Comparativa lin318.tsp.

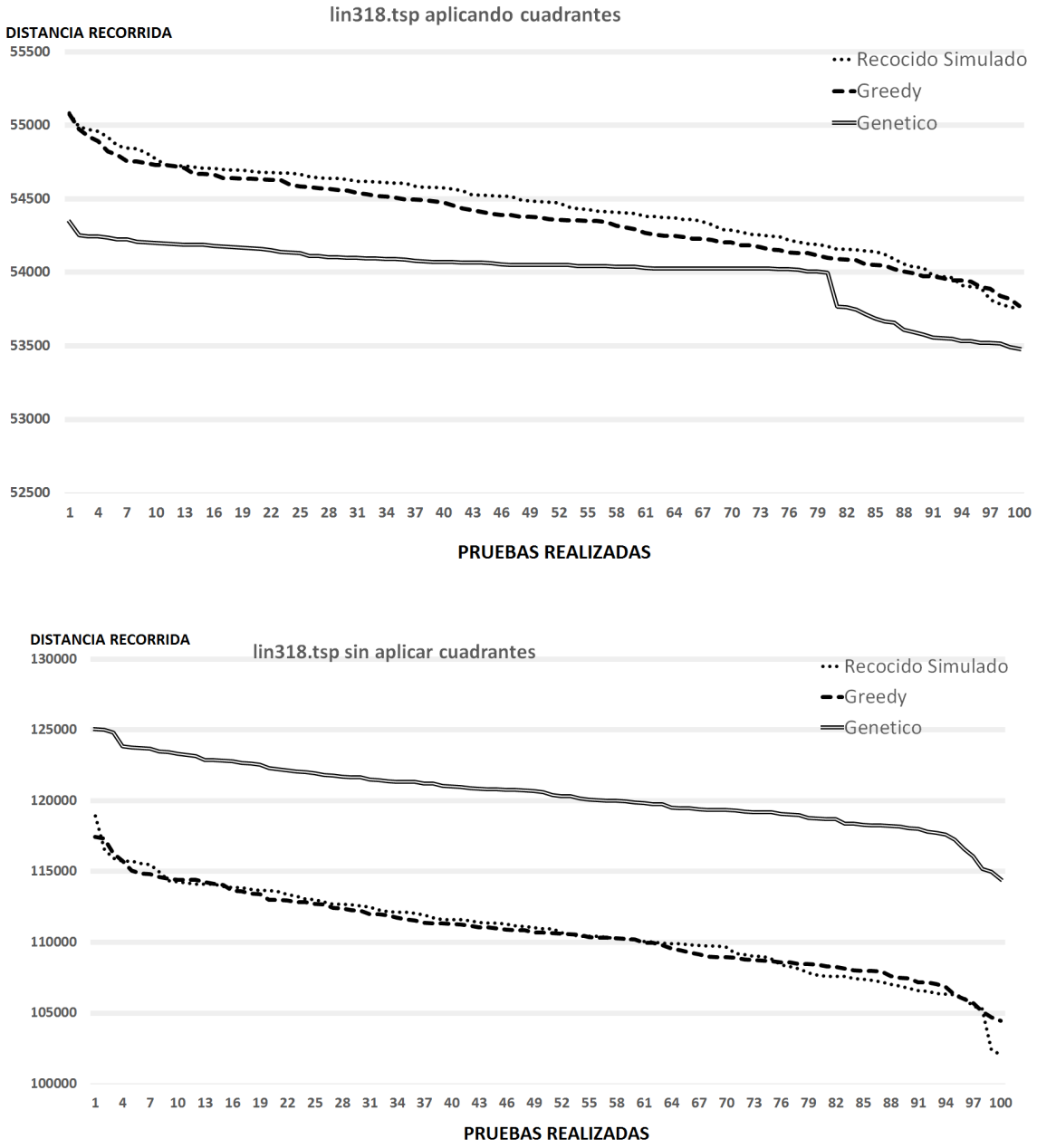

Figura 5.24: Gráficos lin318.tsp con cuadrantes y sin cuadrantes.

5.4.9 p654.TSP

Tabla 5.13: Experimento con el problema p654.tsp.

p654.tsp				
Resultado Original : 47464		Promedio	Mejor	Peor
Con cuadrantes	Recocido	46190.13	45753	46767
	Greedy	46146.53	45588	46541
	Genético	45916.31	45645	46292
Sin cuadrantes	Recocido	121597.61	118683	127226
	Greedy	121537.77	118676	126211
	Genético	132643.30	131454	135144

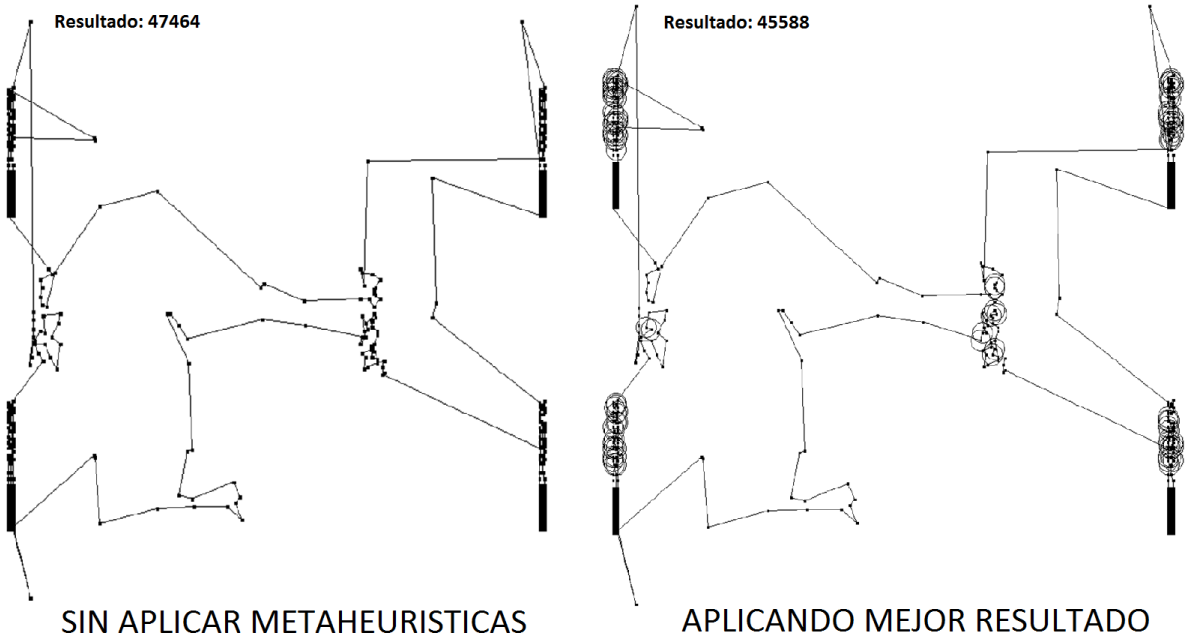

Figura 5.25: Comparativa p654.tsp.

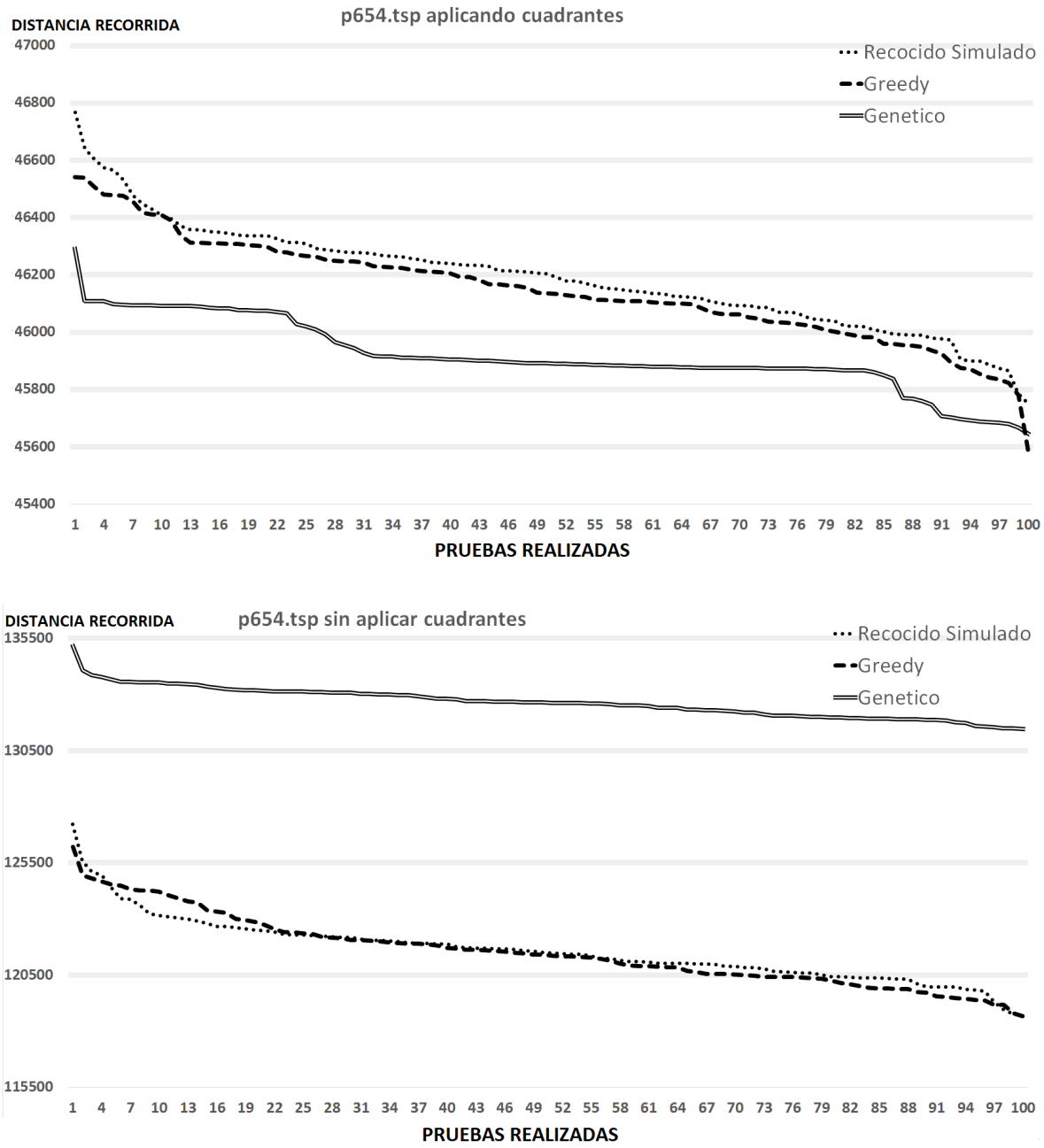

Figura 5.26: Gráficos p654.tsp con cuadrantes y sin cuadrantes.

5.4.10 pcb3038.TSP

Tabla 5.14: Experimento con el problema pcb3038.tsp.

pcb3038.tsp				
Resultado Original :179474		Promedio	Mejor	Peor
Con cuadrantes	Recocido	176904.37	176488	177468
	Greedy	176903.86	176282	177403
	Genético	176989.64	176391	177957
Sin cuadrantes	Recocido	389970.63	384530	396772
	Greedy	390708.99	382955	398492
	Genético	425459.53	422981	427722

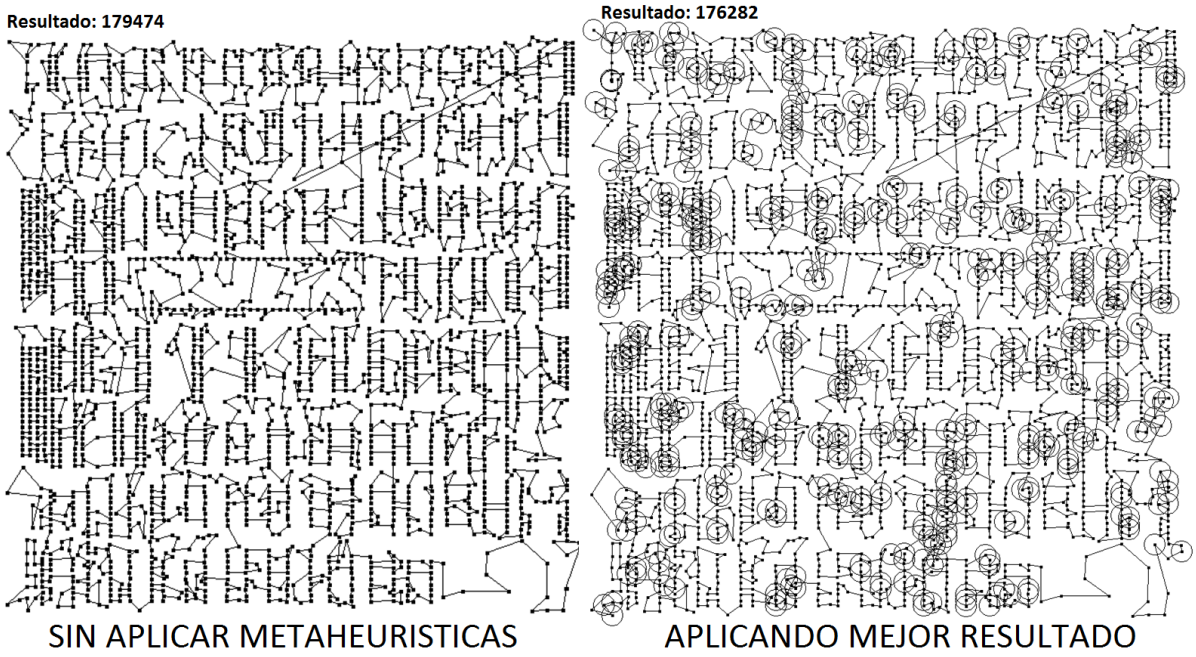

Figura 5.27: Comparativa pcb3038.tsp.

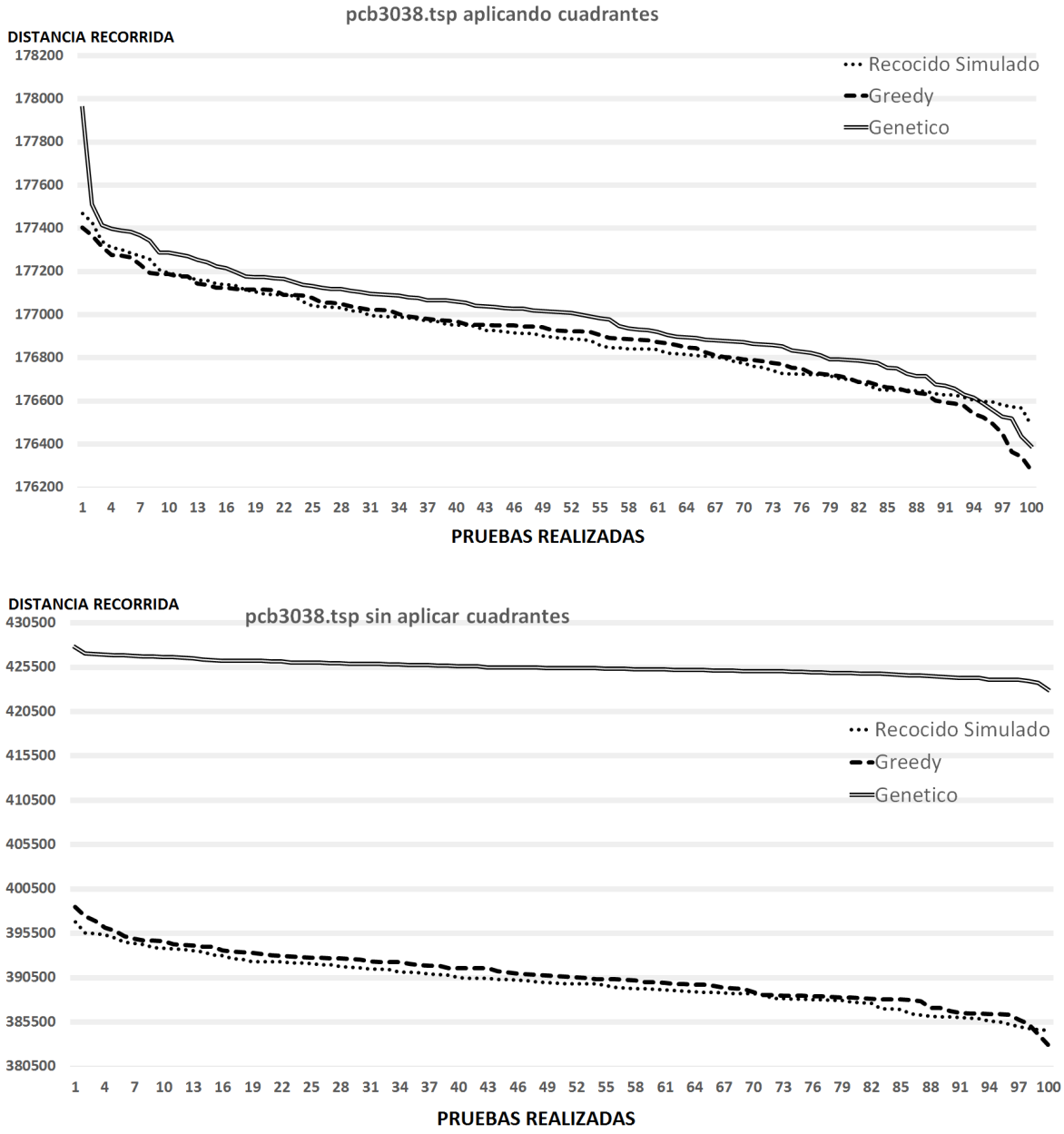

Figura 5.28: Gráficos pcb3038.tsp con cuadrantes y sin cuadrantes.

5.4.11 rd400.TSP

Tabla 5.15: Experimento con el problema rd400.tsp.

rd400.tsp				
Resultado Original : 19094		Promedio	Mejor	Peor
Con cuadrantes	Recocido	18773.66	18562	18922
	Greedy	18766.69	18559	18919
	Genético	18633.28	18510	18790
Sin cuadrantes	Recocido	109863.35	101860	118554
	Greedy	109948.16	102698	117363
	Genético	178000.38	174218	181970

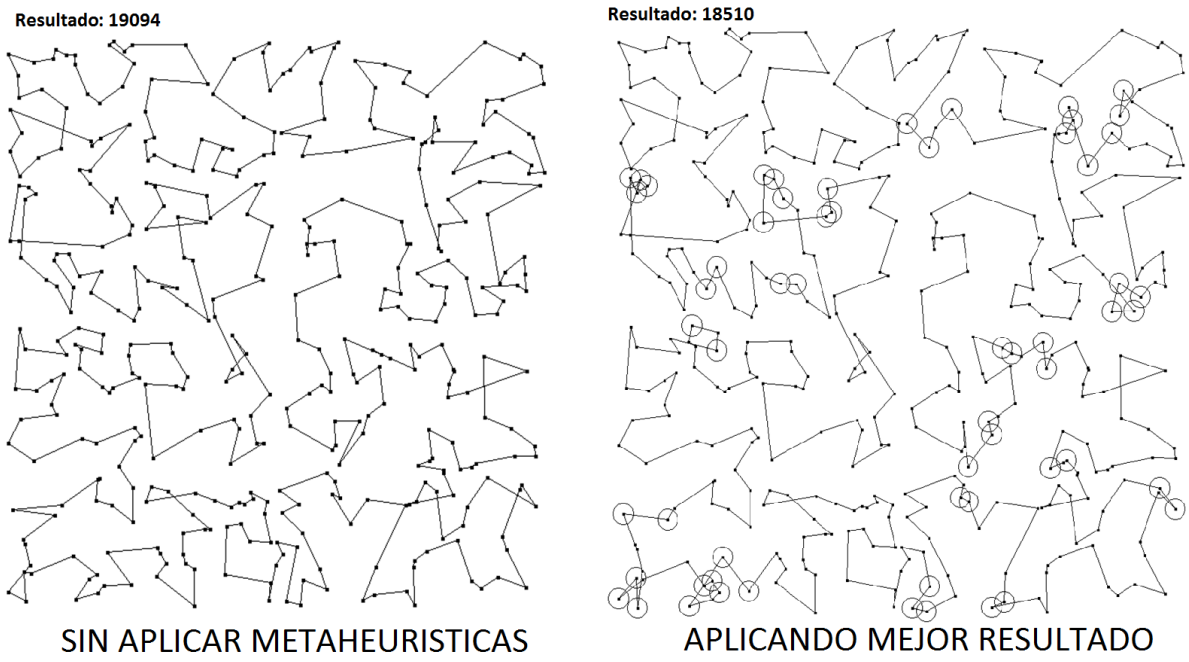

Figura 5.29: Comparativa rd400.tsp.

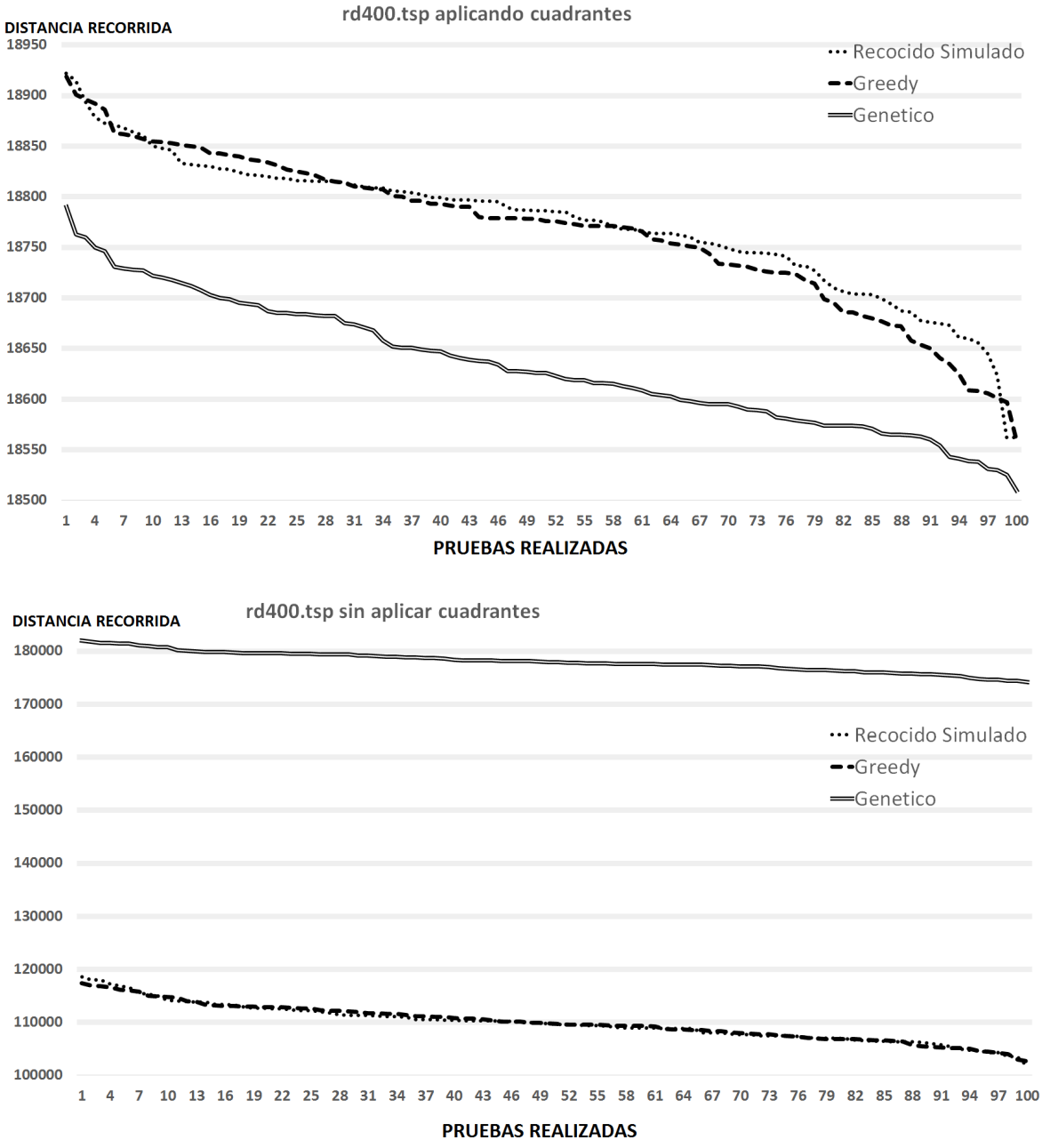

Figura 5.30: Gráficos rd400.tsp con cuadrantes y sin cuadrantes.

5.4.12 rl5934.TSP

Tabla 5.16: Experimento con el problema rl5934.tsp.

rl5934.tsp				
Resultado Original :785540		Promedio	Mejor	Peor
Con cuadrantes	Recocido	774838.4	772530	777479
	Greedy	774761.73	772083	778205
	Genético	777549.96	776338	779039
Sin cuadrantes	Recocido	9349554.00	9184381	9492444
	Greedy	9348025.21	9240374	9497166
	Genético	10249756.12	10213188	10282877

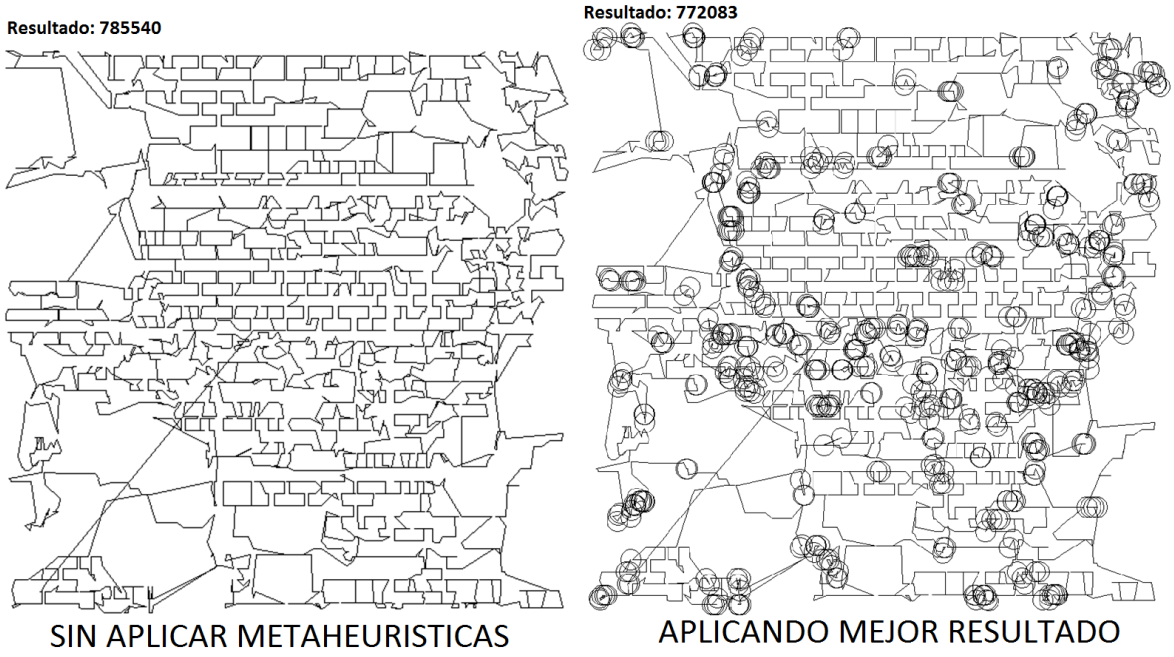

Figura 5.31: Comparativa rl5934.tsp.

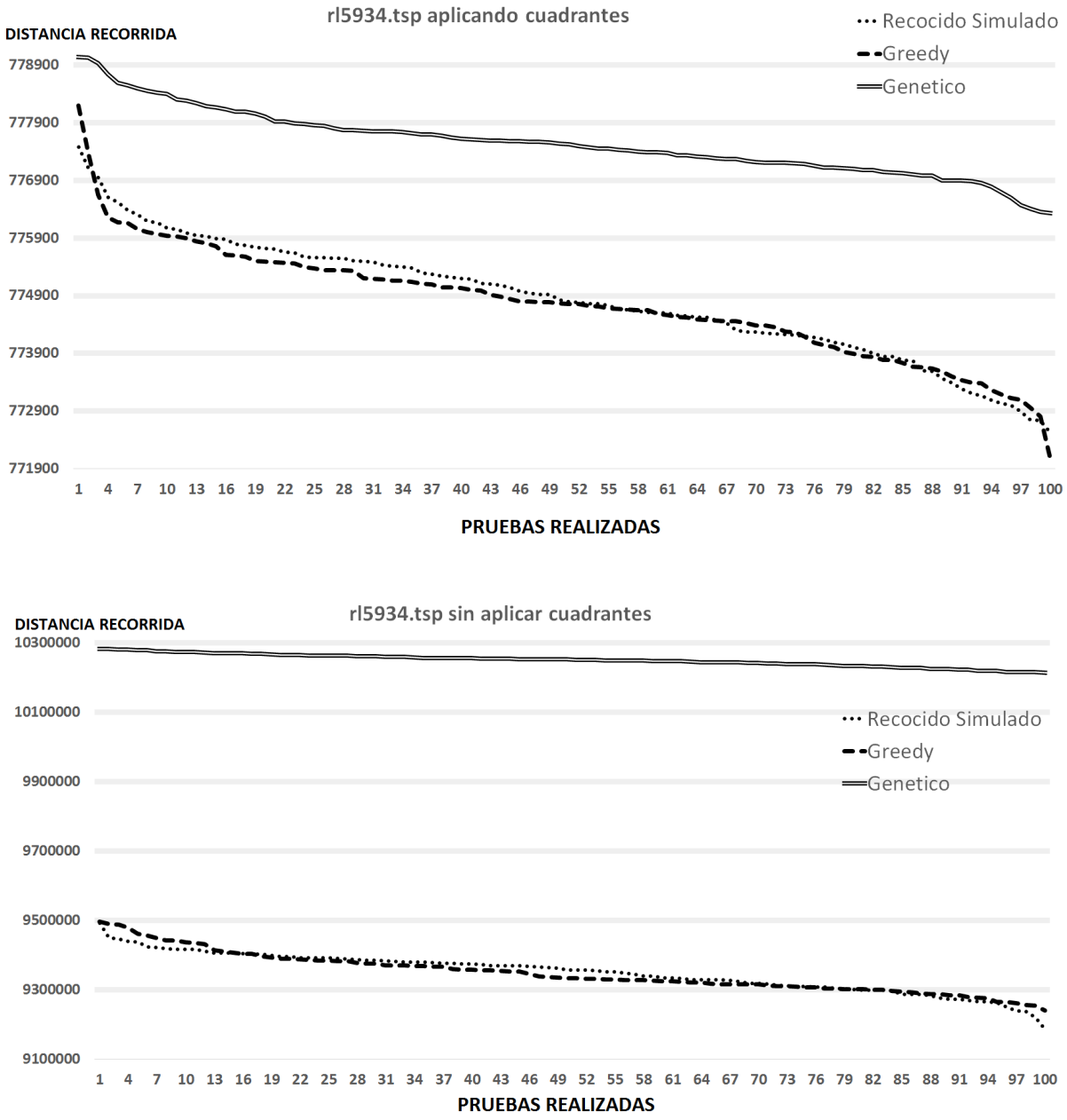

Figura 5.32: Gráficos rl5934.tsp con cuadrantes y sin cuadrantes.

5.4.13 u159.TSP

Tabla 5.17: Experimento con el problema u159.tsp.

u159.tsp				
Resultado Original : 56495		Promedio	Mejor	Peor
Con cuadrantes	Recocido	55506.95	54737	56021
	Greedy	55525.77	54992	56004
	Genético	54834.24	54418	55238
Sin cuadrantes	Recocido	68629.83	65324	74869
	Greedy	68829.22	65058	73971
	Genético	62582.97	59906	65035

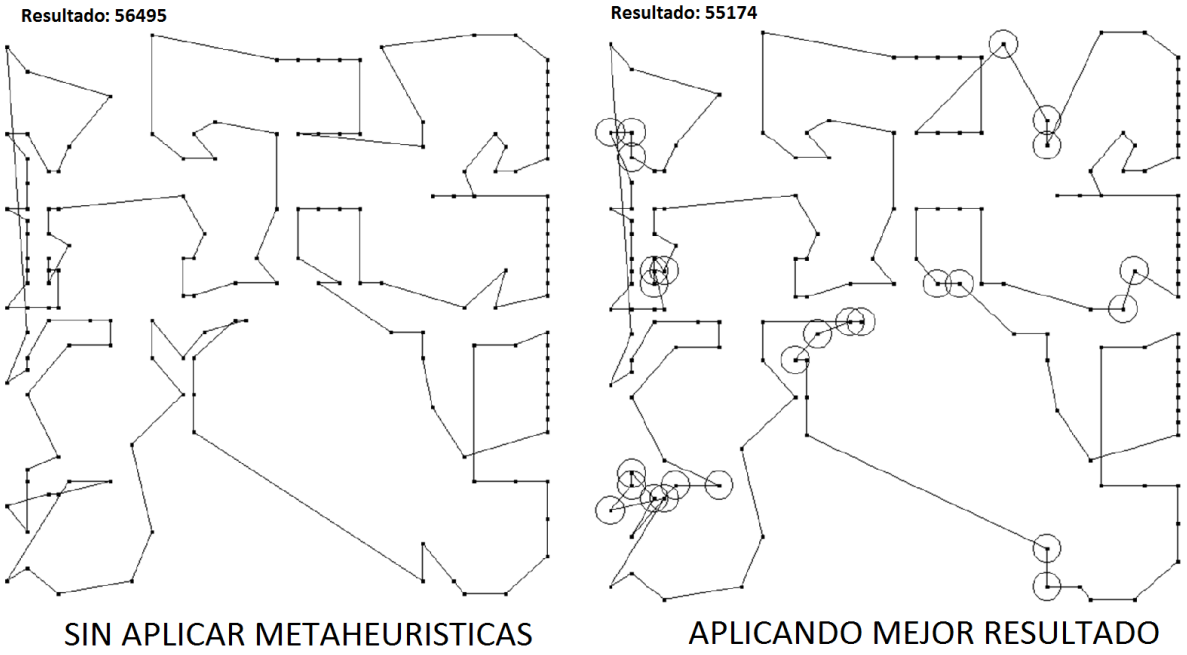

Figura 5.33: Comparativa u159.tsp.

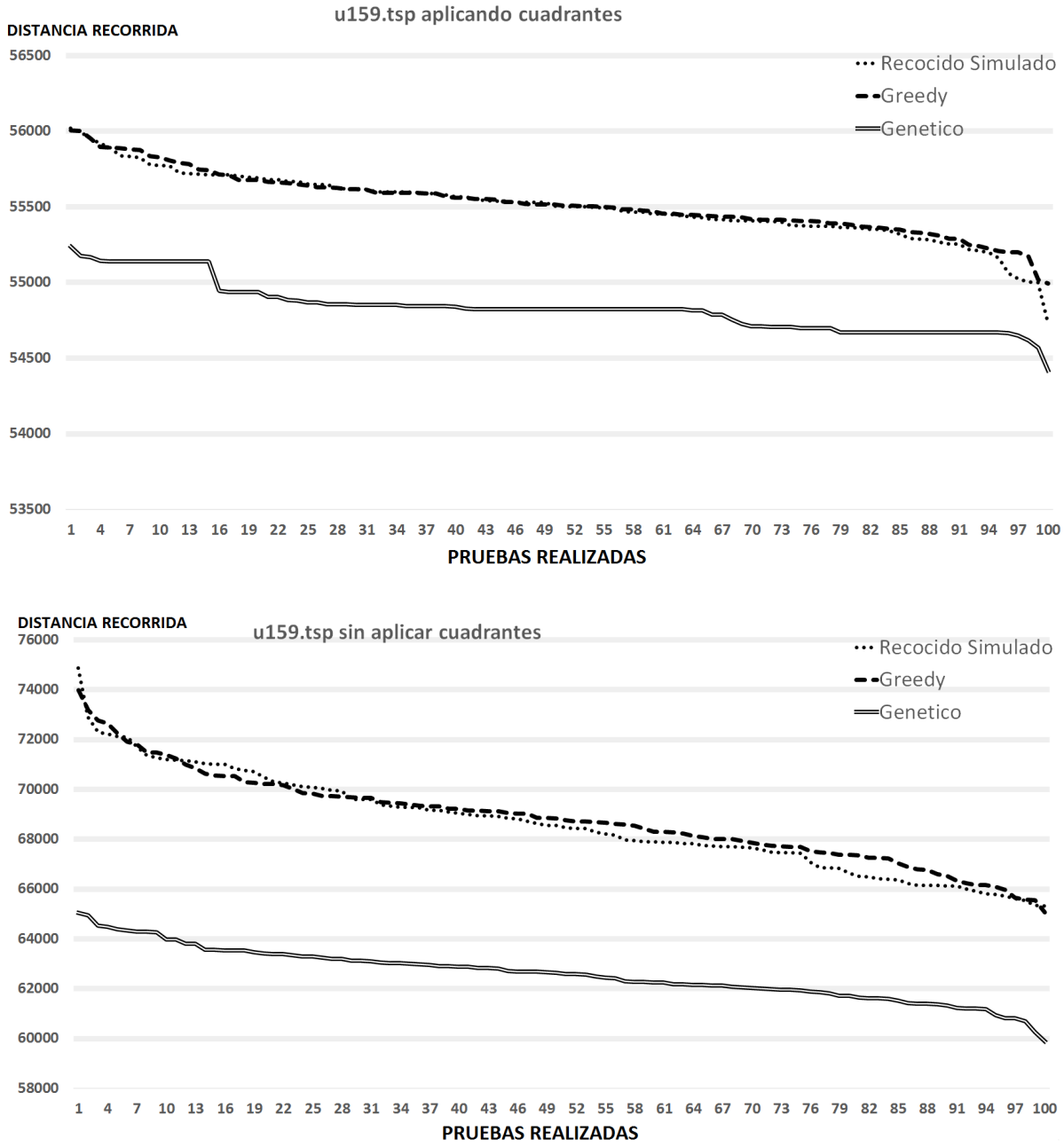

Figura 5.34: Gráficos u159.tsp con cuadrantes y sin cuadrantes.

5.4.14 u724.TSP

Tabla 5.18: Experimento con el problema u724.tsp.

u724.tsp				
Resultado Original : 57641		Promedio	Mejor	Peor
Con cuadrantes	Recocido	55976.89	55514	56479
	Greedy	56010.58	55474	56725
	Genético	55004.45	54685	55283
Sin cuadrantes	Recocido	140950.10	134129	149630
	Greedy	140872.24	132686	149355
	Genético	168973.59	164770	171946

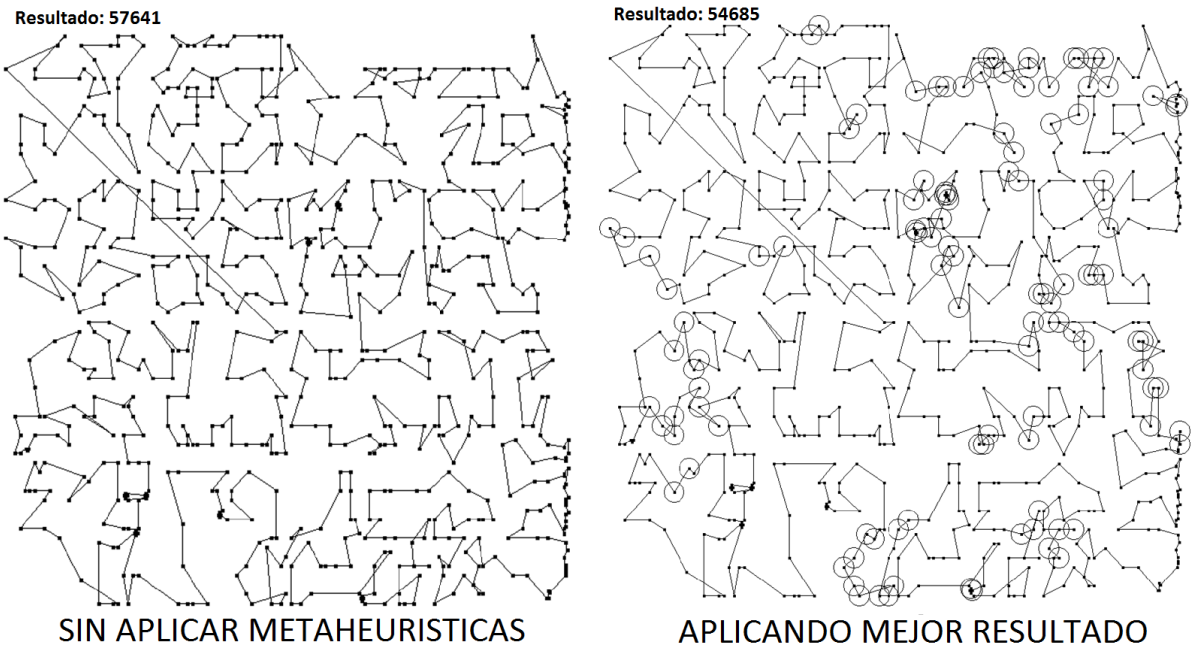

Figura 5.35: Comparativa u724.tsp.

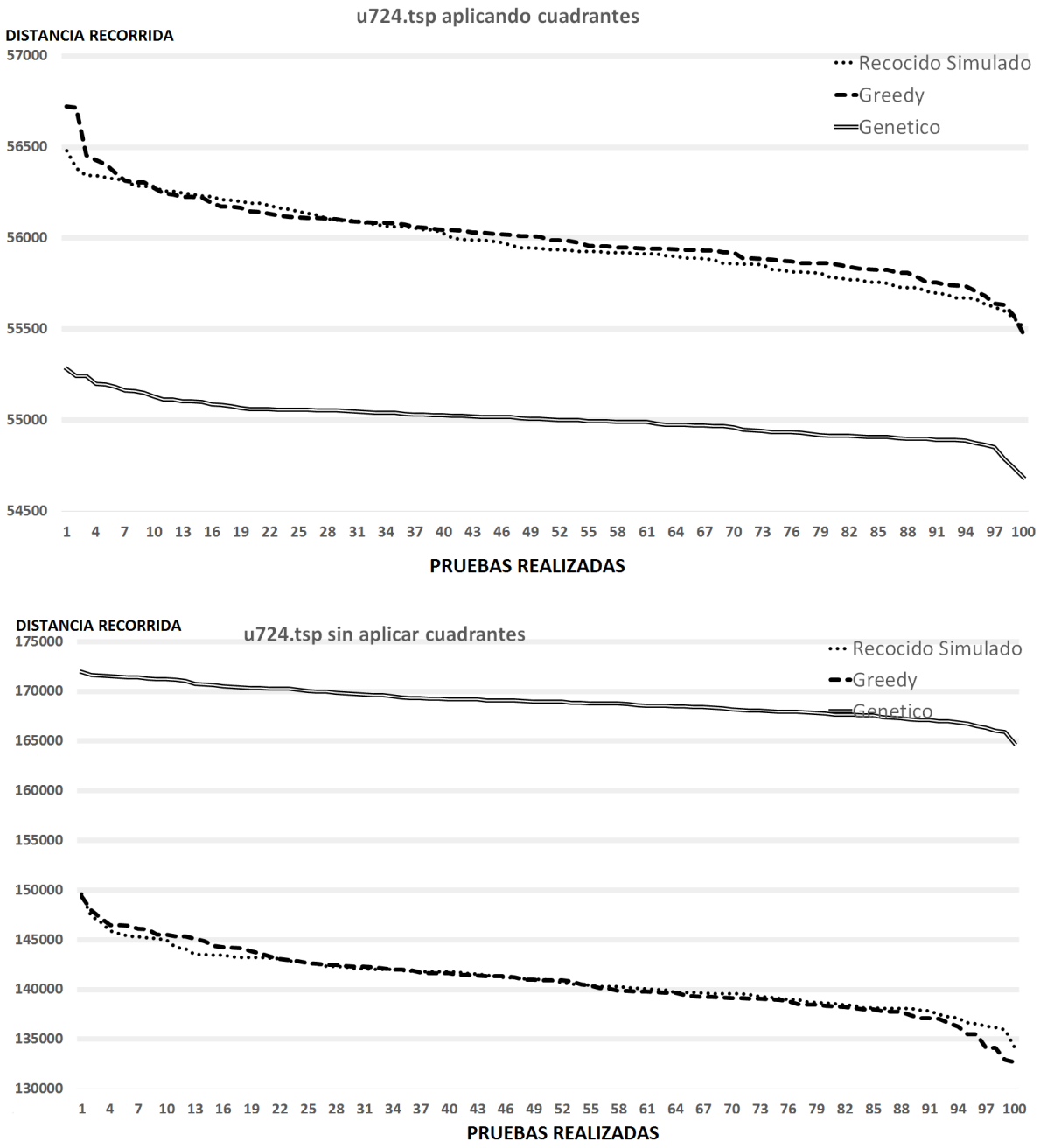

Figura 5.36: Gráficos u724.tsp con cuadrantes y sin cuadrantes.

5.4.15 vm1084.TSP

Tabla 5.19: Experimento con el problema vm1084.tsp.

vm1084.tsp				
Resultado Original :344972		Promedio	Mejor	Peor
Con cuadrantes	Recocido	334821.39	330999	338557
	Greedy	335133.04	331589	339023
	Genético	331384.21	329200	333749
Sin cuadrantes	Recocido	3224207.18	3064309	3380496
	Greedy	3214476.72	3063644	3363287
	Genético	4772212.30	4671155	4850260

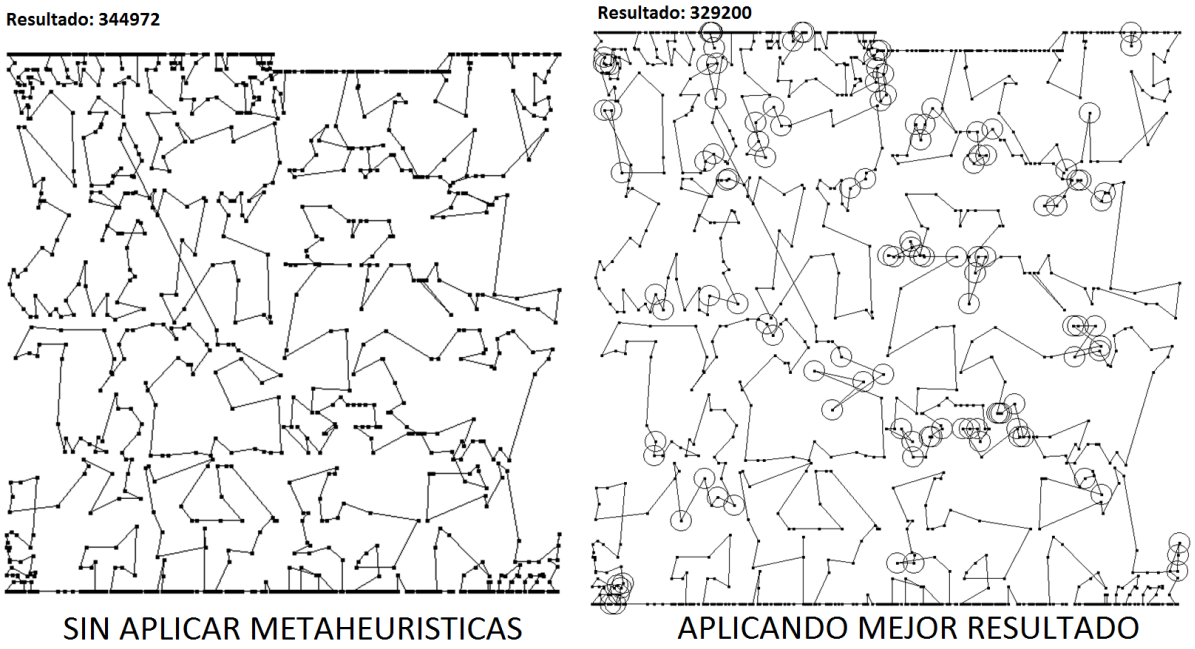

Figura 5.37: Comparativa vm1084.tsp.

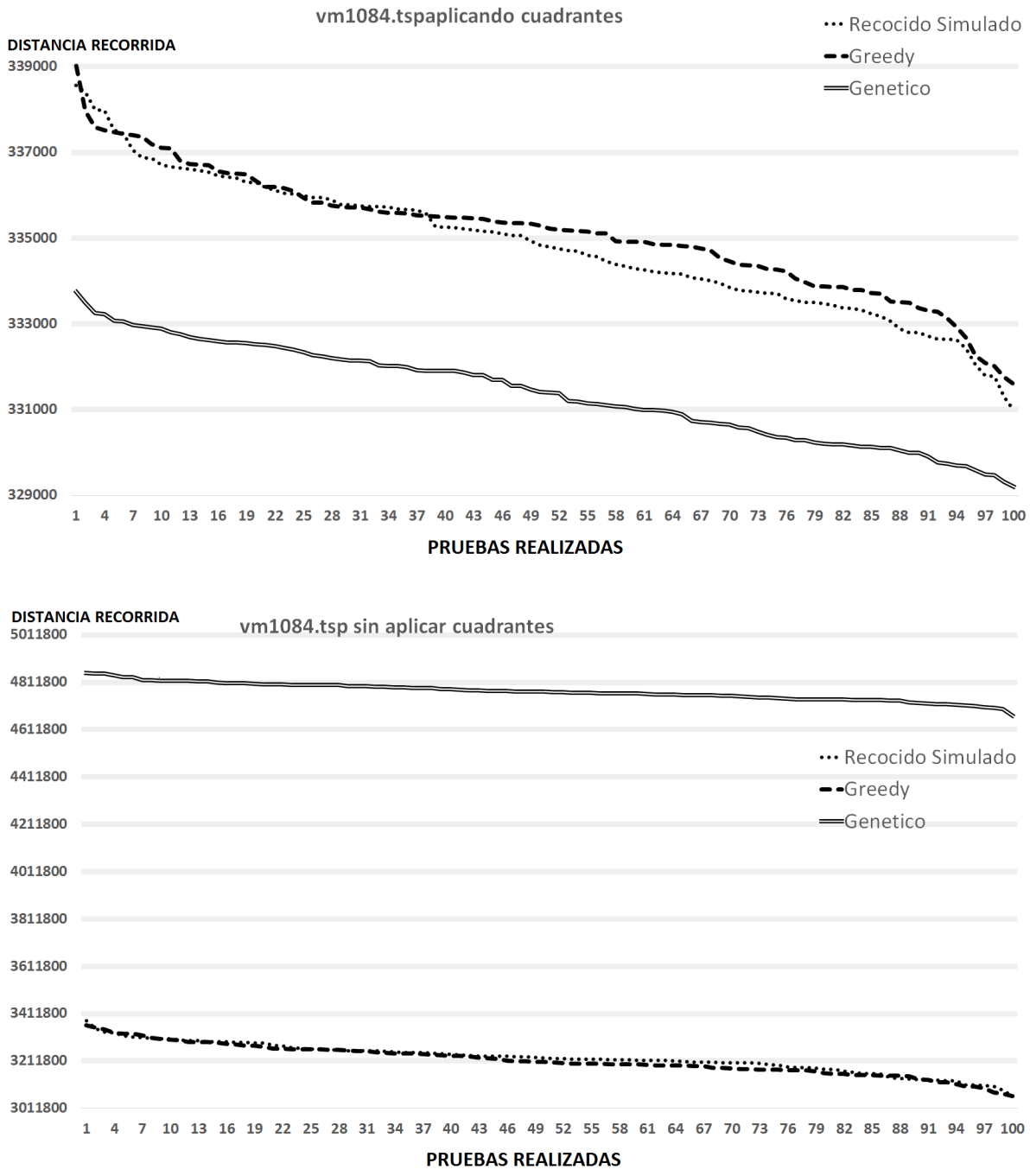

Figura 5.38: Gráficos vm1084.tsp con cuadrantes y sin cuadrantes.

5.5 Resumen de experimentos

Después de presentar los experimentos se recolectaron y vaciaron los resultados en las siguientes tablas cuyas descripciones se detallarán más adelante:

- Experimentos aplicando el método de cuadrantes (tabla 5.20).
- Experimentos sin aplicar el método de cuadrantes (tabla 5.21).
- Análisis global de los mejores resultados (tabla 5.22).

5.5.1 Experimentos aplicando el método de cuadrantes

La tabla 5.20 muestra los resultados obtenidos de los experimentos aplicando el método de cuadrantes y aplicando una de las 3 metaheurísticas en cada uno de ellos. De las 100 corridas que se realizó sobre cada problema se tomó la mejor solución, la peor y se hizo un promedio de todos los resultados obtenidos.

Tabla 5.20: Experimentos aplicando el método de cuadrantes.

		Experimentos aplicando método de cuadrantes								
		Recocido Simulado			Greedy			Genético		
Nombre	Ciudades	Peor	Mejor	Promedio	Peor	Mejor	Promedio	Peor	Mejor	Promedio
a280.tsp	280	3394	3329	3361.5	3396	3335	3365.5	3418	3318	3368
brd14051.tsp	14051	620546	618878	619712	620743	619060	619901.5	622688	621273	621980.5
ch150.tsp	150	8535	8272	8403.5	8540	8315	8427.5	8427	8231	8329
d1655.tsp	1655	82881	82098	82489.5	82839	82014	82426.5	82447	81996	82221.5
d493.tsp	493	45012	44124	44568	45275	44054	44664.5	44918	43373	44145.5
eil101.tsp	101	822	788	805	825	788	806.5	802	781	791.5
fl417.tsp	417	17070	16672	16871	17028	16610	16819	16709	16450	16579.5
lin318.tsp	318	55085	53752	54418.5	55073	53773	54423	54344	53479	53911.5
p654.tsp	654	46767	45753	46260	46541	45588	46064.5	46292	45645	45968.5
pcb3038.tsp	3038	177468	176488	176978	177403	176282	176842.5	177957	176391	177174
rd400.tsp	400	18922	18562	18742	18919	18559	18739	18790	18510	18650
rl5934.tsp	5934	777479	772530	775004.5	778205	772083	775144	779039	776338	777688.5
u159.tsp	159	55996	55996	55996	56003	56003	56003	55174	55174	55174
u724.tsp	724	56479	55514	55996.5	56725	55474	56099.5	55283	54685	54984
vm1084.tsp	1084	338557	330999	334778	339023	331589	335306	333749	329200	331474.5

5.5.2 Experimentos sin aplicar el método de cuadrantes

En la tabla 5.21 se usó las mismas columnas, esta vez muestra los datos de los experimentos sin haber aplicado el método de cuadrantes antes, como se puede notar las cantidades presentadas son mucho más altas que las anteriores. También de las 100 corridas que se realizó sobre cada problema se tomó la mejor solución, la peor y se hizo un promedio de todos los resultados.

Tabla 5.21: Experimentos sin aplicar método de cuadrantes.

		Experimentos sin aplicar método de cuadrantes								
		Recocido Simulado			Greedy			Genético		
Nombre	Ciudades	Peor	Mejor	Promedio	Peor	Mejor	Promedio	Peor	Mejor	Promedio
a280.tsp	280	5345	4711	5028	5410	4804	5107	5066	4747	4907
brd14051.tsp	14051	1.35E+07	1.26E+07	1.30E+07	1.34E+07	1.28E+07	1.31E+07	2.35E+07	2.34E+07	2.35E+07
ch150.tsp	150	28674	23162	25918	29187	22417	25802	37844	32091	34967.5
d1655.tsp	1655	243750	222473	233111.5	246172	226740	236456	266415	262348	264381.5
d493.tsp	493	102751	91468	97109.5	100955	89956	95455.5	124989	115991	120490
eil101.tsp	101	1917	1585	1751	1855	1630	1742.5	1740	1602	1671
fl417.tsp	417	50870	46858	48864	50570	46460	48515	53761	50896	52328.5
lin318.tsp	318	118920	102042	110481	117441	104454	110947.5	125074	114461	119767.5
p654.tsp	654	127226	118683	122954.5	126211	118676	122443.5	135144	131454	133299
pcb3038.tsp	3038	396772	384530	390651	398492	382955	390723.5	427722	422981	425351.5
rd400.tsp	400	118554	101860	110207	117363	102698	110030.5	181970	174218	178094
rl5934.tsp	5934	9492444	9184381	9338412.5	9497166	9240374	9368770	1.03E+07	1.02E+07	1.02E+07
u159.tsp	159	72895	72895	72895	73170	73170	73170	64946	64946	64946
u724.tsp	724	149630	134129	141879.5	149355	132686	141020.5	171946	164770	168358
vm1084.tsp	1084	3380496	3064309	3222402.5	3363287	3063644	3213465.5	4850260	4671155	4760707.5

5.5.3 Análisis global de los mejores resultados

Por último en la tabla 5.22 se muestra la metaheurística del mejor resultado de las 3 (con cuadrantes y sin cuadrante) por cada problema expuesto, además de compararse con los mejores resultados (benchmark) obtenidos. Esta tabla está formada por 4 grupos que se explicará de izquierda a derecha:

- **Con cuadrantes:** Aquí se muestra por cada problema de TSP, el mejor resultado obtenido de entre las 3 metaheurísticas empleadas en la tabla 5.20. Está dividido en las siguientes columnas:
  - **Metaheurística:** El nombre de la metaheurística que produjo el mejor resultado.
  - **Resultado:** La puntuación obtenida a través de dicha metaheurística.

- **Con cuadrantes:** Aquí se muestra por cada problema de TSP, el mejor resultado obtenido de entre las 3 metaheurísticas empleadas en la tabla 5.21. Está dividido en las siguientes columnas.
  - **Metaheurística:** El nombre de la metaheurística que trajo el mejor resultado.
  - **Resultado:** La puntuación obtenida a través de dicha metaheurística.
- **Otros Datos:** Aquí se muestran información adicional:
  - **Solo M.C:** Muestra el resultado que se obtuvo solo con el método de cuadrantes.
  - **Benchmark:** Muestra el resultado de los benchmarks.
- **Comparaciones con los benchmarks:** Por último aquí se hace una comparación del resultado de los benchmark que se puede ver en la columna de Otros datos, con los resultados de las columnas anteriores. El valor de la columna es la diferencia de ambos valores mostrados en forma de porcentaje.
  - **Con cuadrantes:** Muestra la diferencia con el mejor resultado aplicando el método de cuadrantes.
  - **Sin cuadrantes:** Muestra la diferencia con el mejor resultado sin aplicar el método de cuadrantes.
  - **Solo M.C:** Muestra la diferencia con el resultado que se obtuvo solo con el método de cuadrantes.

Tabla 5.22: Análisis global de los mejores resultados.

		Análisis global de los mejores resultados								
		Con Cuadrantes		Sin Cuadrantes		Otros datos		Comparaciones entre el Benchmark (%)		
Nombre	Ciudades	Metaheurística	Resultado	Metaheurística	Resultado	Solo M.C.	Benchmark	Con Cuadrantes	Sin Cuadrantes	Solo M.C.
a280.tsp	280	Genético	3318	Recocido Simulado	4711	3418	2579	28.65	82.66	32.53
brd14051.tsp	14051	Recocido Simulado	618878	Recocido Simulado	12612710	623324	469385	31.84	2587.07	32.79
ch150.tsp	150	Genético	8231	Greedy	22417	8579	6528	26.08	243.39	31.41
d1655.tsp	1655	Genético	81996	Recocido Simulado	222473	83605	62128	31.97	258.08	34.56
d493.tsp	493	Genético	43373	Greedy	89956	45731	35002	23.91	157.00	30.65
eil101.tsp	101	Genético	781	Recocido Simulado	1585	828	629	24.16	151.98	31.63
fl417.tsp	417	Genético	16450	Greedy	46460	17419	11861	38.68	291.70	46.85
lin318.tsp	318	Genético	53479	Recocido Simulado	102042	55340	42029	27.24	142.78	31.67
p654.tsp	654	Greedy	45588	Greedy	118676	47464	34643	31.59	242.56	37.00
pcb3038.tsp	3038	Greedy	176282	Greedy	382955	179474	137694	28.02	178.12	30.34
rd400.tsp	400	Genético	18510	Recocido Simulado	101860	19094	15281	21.13	566.57	24.95
rl5934.tsp	5934	Greedy	772083	Recocido Simulado	9184381	785540	556045	38.85	1551.73	41.27
u159.tsp	159	Genético	55174	Genético	64946	56495	42080	31.11	54.33	34.25
u724.tsp	724	Genético	54685	Greedy	132686	57641	41910	30.48	216.59	37.53
vm1084.tsp	1084	Genético	329200	Greedy	3063644	344972	239297	37.56	1180.26	44.16

5.6 Arte con TSP

El arte con TSP son una serie de ejercicios que ponen a prueba la capacidad de los algoritmos de TSP. Se selecciona una imagen, se eligen algunos puntos y se conectan entre ellos como si fuera un problema de TSP. Según [KB05] el matemático Robert Bosch y el profesor Craig S. Kaplan presentaron este tema con una serie de imágenes que se muestran en la figura 5.39 basadas en fotografías de Phil Greenspun.

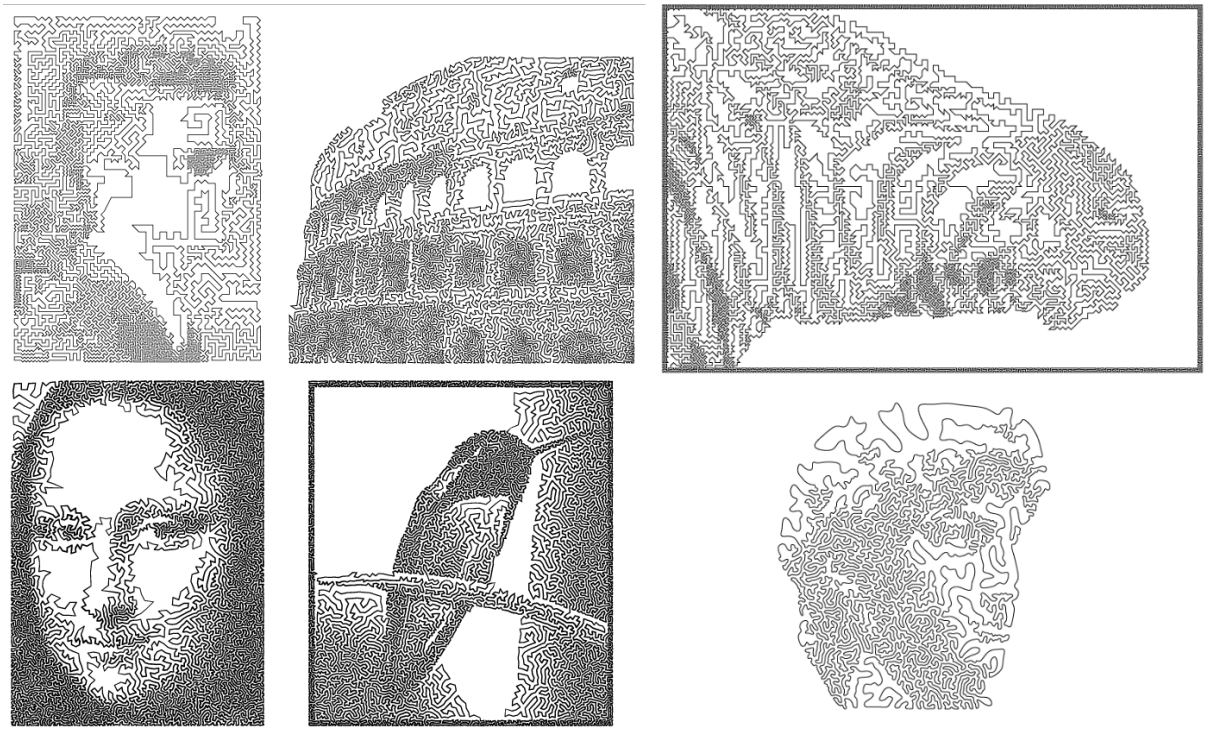

Figura 5.39: Ejemplos de arte con TSP.

Aunque no se abarcará este tema es importante señalar que se aplicó también un algoritmo que permite ver la densidad de una zona de puntos, eliminando y redistribuyendo zonas de puntos para obtener matices de blanco y negro, este método en particular fue desarrollado por Robert Bosch y Adriane Herman

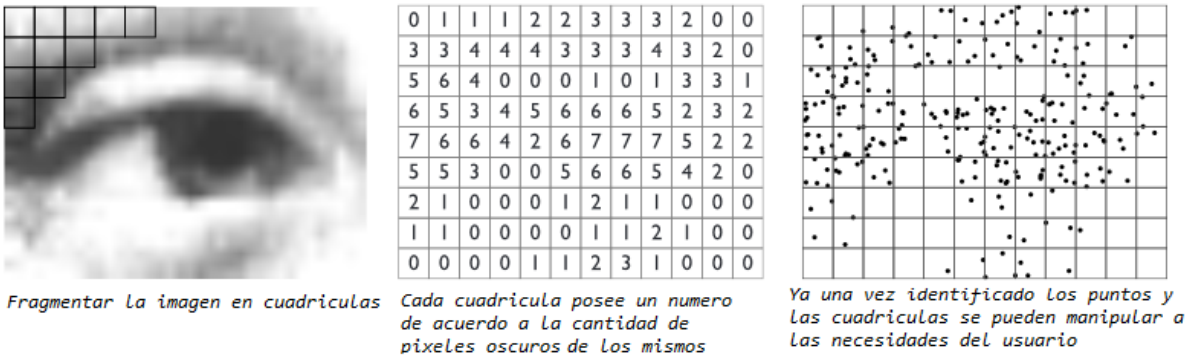

Figura 5.40: Breve explicación del proceso de semitonos utilizado por Bosch y Herman.

A continuación se presentarán algunos problemas de TSP hechos por Robert Bosch basándose en cuadros famosos, estos problemas fueron resueltos usando el método de cuadrantes sin aplicar metaheurísticas, el nombre de la pintura y el resultado se pueden ver en las mismas figuras.

También se encuentra la tabla 5.23 que recompila los resultados mostrados comparado con la persona que posee el record actual según [UW16] cuyo nombre es Yuichi Nagata [TTR16].

Tabla 5.23: Comparación con los resultados de Yuichi Nagata.

Cuadro original	Autor	Ciudades	Record actual	Cuadrantes	Diferencia (%)
Mona Lisa	Leonardo Da Vinci	100,000	5,757,191	6,471,018	12.39
Auto retrato	Vincent Van Gogh	120,000	6,543,610	7,401,998	13.11
El nacimiento de Venus	Sandro Botticelli	140,000	6,810,665	7,653,034	12.36
Juan de Pareja	Diego Velázquez	160,000	7,619,953	8,586,457	12.68
El desesperado	Gustave Courbet	180,000	7,888,733	8,951,892	13.47
La joven de la perla	Johannes Vermeer	200,000	8,171,677	9,357,118	14.50

Figura 5.41: "Mona Lisa" de Leonardo Da Vinci hecho en TSP.

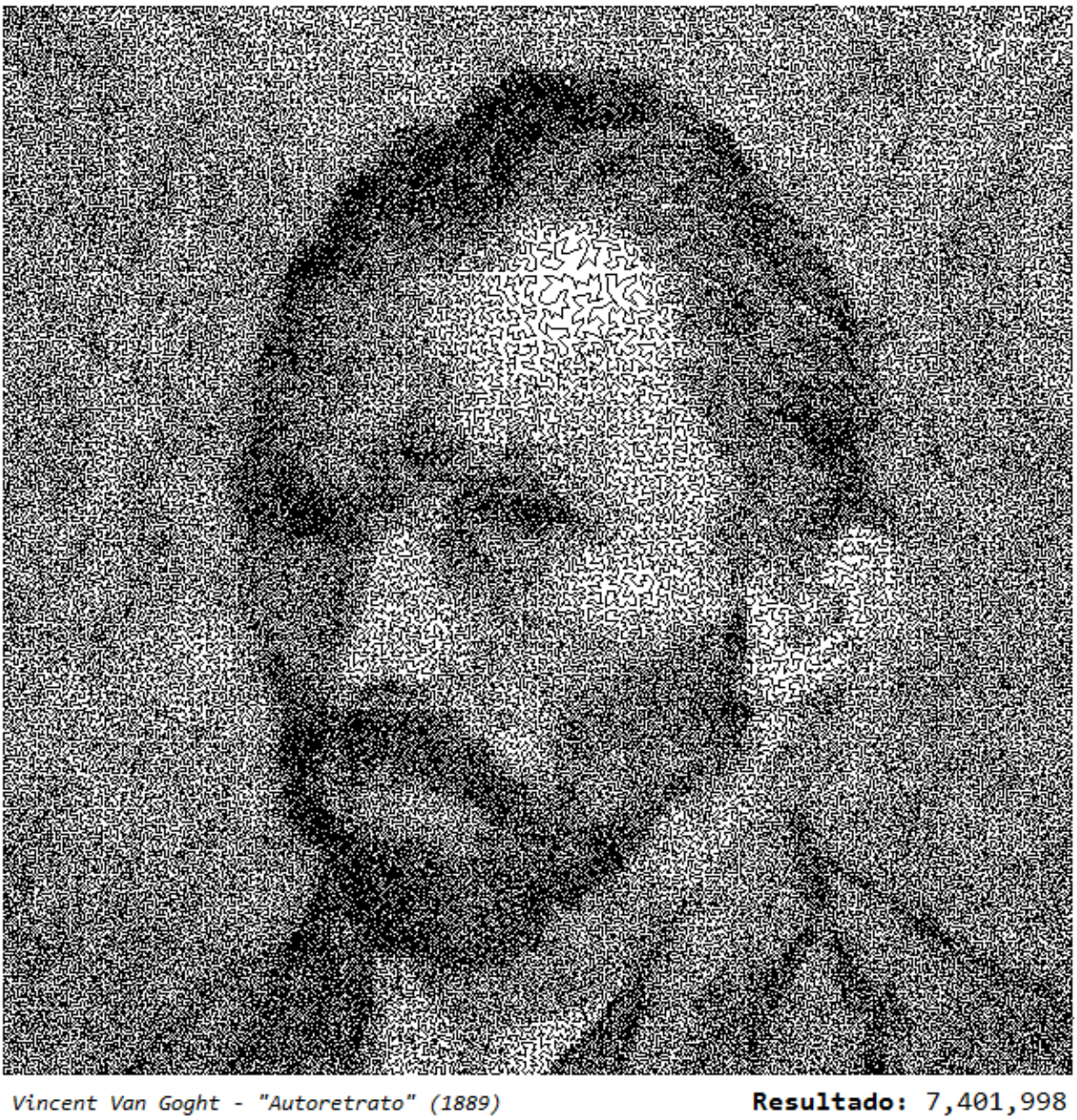

Figura 5.42: "Autoretrato" de Vincent Van Gogh hecho en TSP.

Figura 5.43: "El nacimiento de Venus" de Sandro Botticelli hecho en TSP.

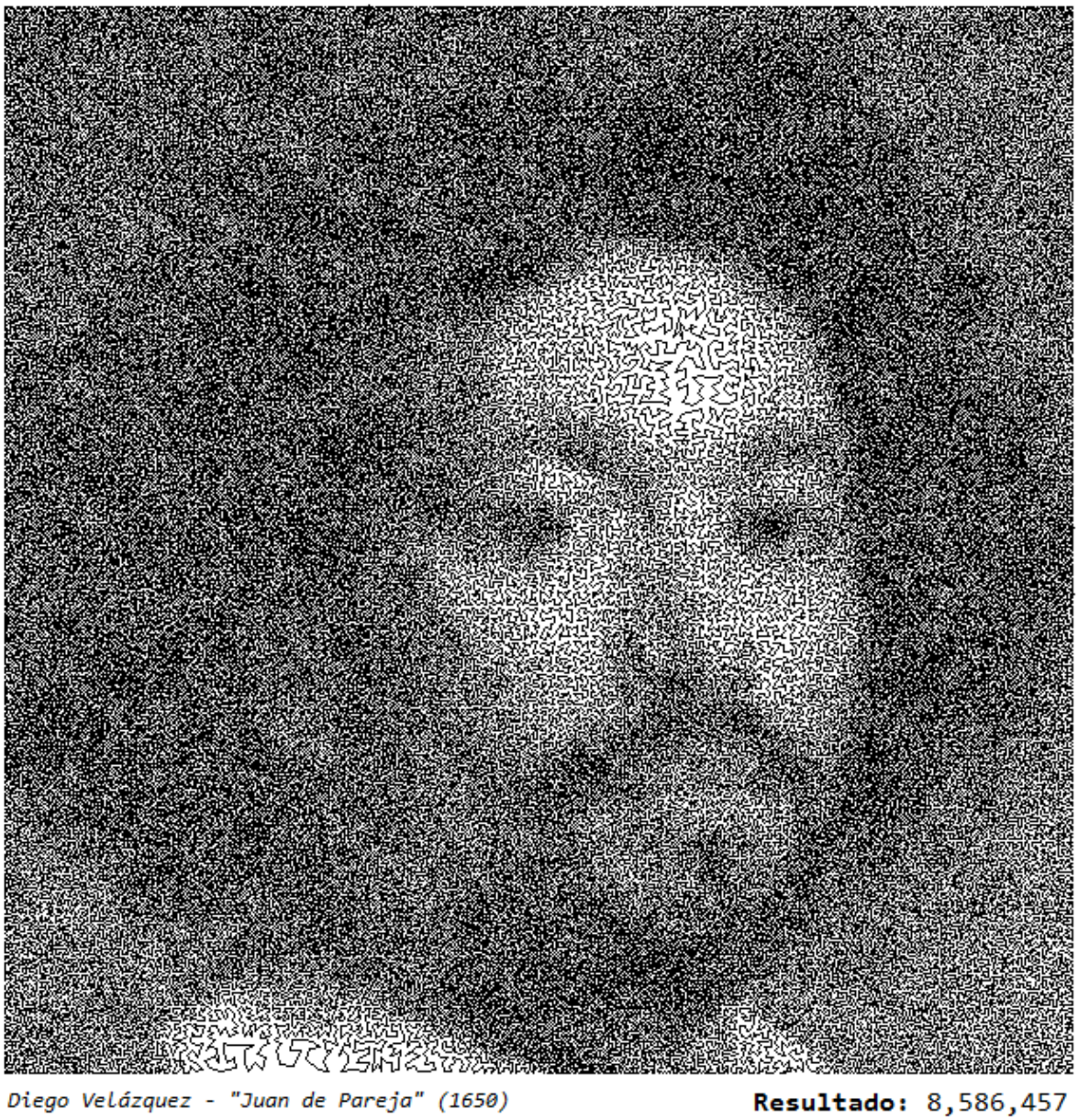

Figura 5.44: "Juan de Pareja" de Diego Velázquez hecho en TSP.

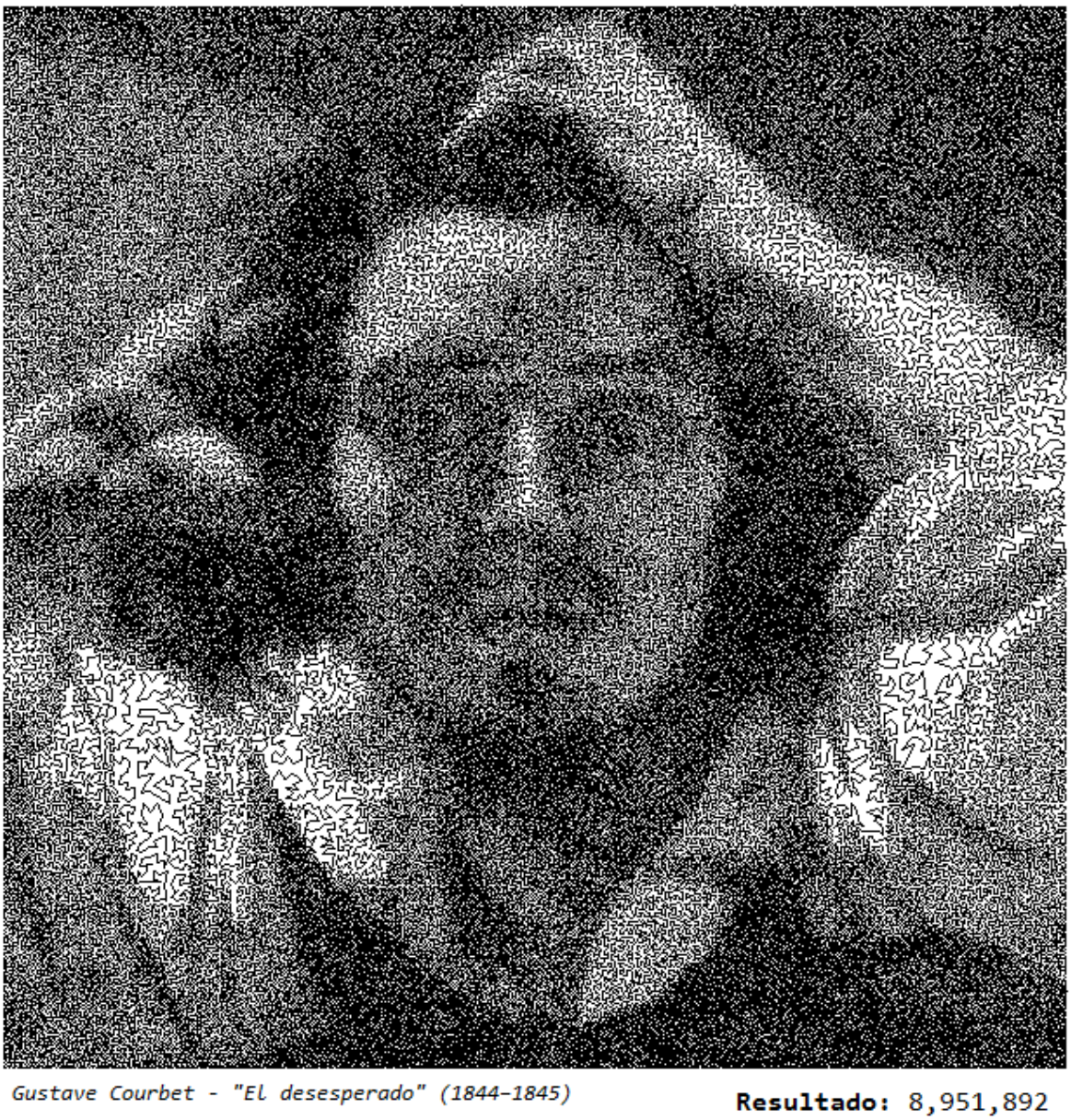

Figura 5.45: "El desesperado" de Gustave Courbet hecho en TSP.

Figura 5.46: "La joven de la perla" de Johannes Vermeer hecho en TSP.

5.7 Comentarios finales

Como últimos comentarios cabe señalar que una de las ventajas del método de cuadrantes es que, al tratarse de un método determinista, permite obtener un resultado de manera rápida y acertada sin depender de elementos aleatorios. Aunque la desventaja más notable es que siempre se obtendrá la misma respuesta y no siempre será la mejor posible, aquí es donde entraron las metaheurísticas que permitieron obtener mejores resultados sin tener que revisar manualmente cada desperfecto de la solución anterior.

La razón por la que se comentó acerca del arte con TSP fue para demostrar la enorme utilidad que tiene el método de cuadrantes en otro tipo de aplicación real aparte de los problemas cotidianos de TSP, además de que fueron los problemas que más puntos abarcaron, el de la Mona Lisa de 100 mil puntos y la de Vermeer con 200 mil.

Conclusión

La aplicación de un algoritmo que divide un problema de TSP en cuadrantes logró obtener soluciones lo suficientemente eficientes para el tiempo de ejecución, además de que el uso de metaheurísticas para modificar el resultado anterior permitió mejorar la solución inicial.

Después de realizar los experimentos quedo demostrado que el uso de algoritmos determínistas en conjunto con alguna metaheurística permite obtener resultados cercanos al óptimo, aunque no superan los resultados obtenidos en los benchmark, si son lo suficientemente eficientes para presentar un gran avance.

El conjunto de problemas usados en la prueba (TSPLIB y arte) y los resultados obtenidos demuestran que es factible aplicar esta técnica para resolver el problema del agente viajero, con lo cual la hipótesis queda demostrada.

Otro objetivo que se esperaba de este trabajo es que pueda ser utilizado a futuro en otros trabajos similares de optimización combinatoria como el problema de ruteo de vehículos y el problema de asignación de trabajos (scheduling).

Como ejemplo está el tema 5.6 (Arte de TSP) que muestra el uso los algoritmos del agente viajero para crear imágenes, incluyendo usar dicha metodología para otras funciones como el renderizado de imágenes al redistribuir los puntos de un determinado cuadrante,

aunque no se abarco este tema a profundidad es posible usar el método de aplicación de cuadrantes para poder realizar dicha función.

Referencias bibliográficas

[ABKS99] Ankerst, M., M. Bruenig, H. P. Kriegel, and J. Sander: *Optics: ordering points tu identify the clustering structure*. In Proceedings of ACM SIGMOD International Conference on Management of Data, pages 49–60, 1999.

[BR03] Blum, C. and A. Roli: *Metaheuristics in combinatorial optimization: Overview and conceptual comparison*. Technical report, Universite Libre de Bruxelles, Universita' degli Studi di Bologna, 2003.

[BT83] Balas, E. and P. Toth: *Tech report no. msrr-488*. Technical report, Universidad Carnegie-Mellon, 1983.

[CLR01] Cormen, T., Ch. Leiserson, and R. Rivest: *Introduction to Algorithms, MIT Press*. McGraw Hill, 2001.

[Dav91] Davis, L.: *Handbook of Genetic Algorithms*. Van Nostrand Reinhold, 1991.

[DFJ54] Dantzig, G., R. Fulkerson, and S. Johnson: *Solution of a large-scale traveling-salesman problem*. Journal of the Operations Research Society of America, 2(4):393–406, 1954.

[DHS01] Duda, R.O., P.E. Hart, and D. Stork: *Pattern Classification*. Wiley series in Probabilistic and Statistics, 2001.

[DP05] Díaz Pérez, A.: *Análisis y diseño de algoritmos*. Technical report, Instituto Politécnico Nacional, Sección de Computación Departamento de Ingeniería Eléctrica, 2005.

[EKX96] Ester, M., J. Kriegel, H.P. Sander, and X. Xu: *A density-based algorithm for discovering clusters in large spatial databases with noise*. In *In Proceedings of International Conference on Knoowledge Discovery and Data Mining (KDD96)*, number 2, pages 226–231, Portland, Oregon, 1996.

[GGRVG85] Grefenstette, J., R. Gopal, B. Rosmaita, and D. Van Gucht: *Proceedings of the first international conference on genetic algorithms and their applications*. In *Genetic algorithms for the traveling salesman problem*, pages 160–168, 5000 Forbes Ave, Pittsburgh, PA 15213, EE. UU, 1985. Universidad Carnegie-Mellon.

[Gim05] Gimbert, J.: *Curso de doctorado, problemas de optimización sobre grafos*. Technical report, Universitat de Lleida, Departament de Matemàtica, 2005.

[GJ79] Garey, M. and D. Johnson: *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W.H. Freeman y Cía, 1979.

[Gol89] Goldberg, D.E.: *Genetic Algorithms in Search, Optimization and Machine Learning*. Addison-Wesley, 1989.

[GP02] Gutin, G. and A.P. Punnen: *The traveling salesman problem and its variations*. Springer, 2002.

[GRS98] Guha, S., R. Rastogi, and K. Shim: *Cure: An efficient clustering algorithm for large databases*. In *ACM SIGMOD98*, number 24, pages 73–84, Seattle, Washington, 1998.

[GV98] Guerrequeta, G.R. and M.A Vallecillo: *Técnicas de Diseño de Algoritmos*. Universidad de Malaga, 1998.

[HMU08a] Hopcroft, J., R Motwani, and J. Ullman: *Introducción a la teoría de autómatas lenguajes y computación*. McGraw Hill, 2008.

[HMU08b] Hopcroft, J., R Motwani, and J. Ullman: *Introducción a la teoría de autómatas lenguajes y computación*. McGraw Hill, 2008.

[HMU08c] Hopcroft, J., R Motwani, and J. Ullman: *Introducción a la teoría de autómatas lenguajes y computación*. McGraw Hill, 2008.

[HMU08d] Hopcroft, J., R Motwani, and J. Ullman: *Introducción a la teoría de autómatas lenguajes y computación*. McGraw Hill, 2008.

[Hol75] Holland, J.: *Adaptation in natural and articial systems*. Technical report, University of Michigan Press,Ann Arbor, 1975.

[JP85] Johnson, D.S. and C.H. Papadimitriou: *Computational Complexity*. Wiley, 1985.

[Kar72] Karp, R.: *Reducibility among combinatorial problems*. Technical report, University of California at Berkeley, 1972.

[KB05] Kaplan, Craig S. and Robert Bosch: *Renaissance banff: Bridges 2005: Mathematical connections in art, music and science*. In *TSP Art*, pages 301–308, Alberta, Canada, 2005.

[KHK99] Karypis, G., E. H. Han., and V. Kumar: *Chameleon: A hierarchical clustering algorithm using dynamic modeling*, volume 32. IEEE Computer, 1999.

[Len97] Lenstra, J. K: *Local Search in Combinatorial Optimisation*. John Wiley and Sons Ltd, 1997.

[Mic92] Michalewicz, Z.: *Genetic Algorithms + Data Structures = Evolution Programs*. SpringerVerlag, 1992.

[ÖKL09] Öncan, T., A. Kuban, and G. Laporte: *A comparative analysis of several asymmetric traveling salesman problem formulations*. Computers and Operational Research, 36 (3):637–654, 2009.

[Pas07] Pascual, D.: *Algoritmos de agrupamiento*. Technical report, Departamento de Computación Universidad de Oriente,Cuba, 2007.

[Per02] Perttunen, J.: *The traveling salesman problem and its variations*. Springer, 2002.

[PS07] Pla, F. and S. Sánchez: *Algoritmos de agrupamiento*. Technical report, Departamento de Lenguajes y Sistemas Informáticos Universitat Jaume I, España, 2007.

[Ree93] Reeves, ZC.: *Modern Heuristic Techniques for Combinatorial Problems*. Blackwell Scientific Publications, 1993.

[RKH16] Ruprecht-Karls-Universität Heidelberg: *The TSPLIB*, 2016. http://comopt.ifi.uni-heidelberg.de/software/TSPLIB95/.

[RRTT53] Rosenbluth, N., A., M. Rosenbluth, A. Teller, and E. Teller: *Equation of state calculations by fast computing machines*. J. Chem. Phys., pages 1087–1092, 1953.

[SH99] Sadiq, S. M. and Y. Habib: *Iterative Computer Algorithms with Applications in Engineering: Solving Combinatorial Optimization Problems*. Wiley-IEEE Computer Society Press, 1999.

[Sil80] Silver, E.A.: *A tutorial on heuristic methods*. European Journal of Operational Reseach, 5:153–162, 1980.

[TTR16] Tokyo Tech Research Repository: *Yuichi Nagata*, 2016. http://t2r2.star.titech.ac.jp/cgi-bin/researcherinfo.cgi?lv=en&q_researcher_content_number=CTT100574614.

[UW16] University of waterloo: *TSP Art*, 2016. http://www.math.uwaterloo.ca/tsp/data/art/index.html.

[Wil94] Wilf, H.S.: *Algorithms and Complexity*. AK Peters, 1994.

[ZE81] Zanakins, S. H. and J. R. Evans: *Heuristic 'optimization': Why, when and how to use it*. Interfaces, 8(5):84 – 91, 1981.

[ZIB99] Zuse Institute Berlin: *The TSPLIB Symmetric Traveling Salesman Problem Instances*, 1999. http://elib.zib.de/pub/mp-testdata/tsp/tsplib/tsp/.

[Ziv07] Ziviani, N.: *Diseño de algoritmos con implementaciones en Pascal y C*. Thomson, 2007.