UCLV
Universidad Central
"Marta Abreu" de Las Villas

MFC
Facultad de Matemática
Física y Computación

Departamento: Computación

TRABAJO DE DIPLOMA

Integración de técnicas de visualización para el algoritmo de búsqueda metaheurística Recocido Simulado.

Autor: Abel Alejandro Fleitas Perdomo

Tutor: MSc. Andy Morfa Hernández

Santa Clara, Junio, 2018

Dedicatoria.

A mi familia por permanecer en la vigilia de mis sueños profesionales.

A la Revolución Cubana por la oportunidad de llegar a la cima de nuestra formación profesional.

Agradecimiento.

Agradezco la colaboración brindada a los que de una forma u otra han contribuido al cierre exitoso de la realización de este trabajo investigativo.

A MSc. Andy Morfa Hernández por darme la oportunidad y ser mi tutor en esta investigación.

A todos los profesores de la carrera que me impartieron sus conocimiento para este momento.

A mis amigo Luis Alberto Martinez y Carlos Miguel Diaz por su apoyo brindado.

A Ing. Juan Daniel Pedraza Diaz por su experiencia brindada en este campo investigativo.

A mi padre Abel Fleitas Alvarez por su apoyo incondicional.

A mis tíos Silvia Perdomo Vergel y Andres López Olivera, por estar siempre presente en las dificultades y las alegrías.

A mis primos Manuel Orlando Garcia Perdomo y Johany López Perdomo por ser mis guías a seguir.

A mi abuela Yolanda Vergel Ribera, por ayudar en mi crianza.

En especial a mi madre Sandra Lourdes Vergel Perdomo por su aliento y apoyo constante a lo largo de todos estos años.

A todos muchas gracias.

Resumen.

Las técnicas de integración visual para apoyar algoritmos metaheurísticos es una variante muy poderosa para resolver problemas de optimización, debido a que los usuarios en su interacción a través de aplicaciones con estos algoritmos en tiempo de ejecución pueden a llegar a tomar buenas decisiones parciales o totales según el nivel de complejidad de la problemática aprovechando las ventajas que ofrece la percepción humana. La presente investigación propone un nuevo modelo de técnicas de integración visual para el software Atreeb desarrollado en la Universidad Central “Marta Abreu de Las Villas” el cuál implementa el algoritmo metaheurístico Recocido Simulado aplicado al problema del viajero vendedor. La eficiencia de estas nuevas técnicas se demuestra en el análisis del modelo propuesto.

Abstract.

The techniques of visual integration to support meta-heuristic algorithms is a very powerful variant to solve optimization problems, because users in their interaction through software with these algorithms at runtime can get to make good partial or total decisions according to the complexity level of the problem taking advantage of the advantages that offers human perception. The present research proposes a new model of visual integration techniques for the Atreeb software developed in the Central University "Marta Abreu de Las Villas" which implements the heuristic Meta algorithm Simulated annealing applied to the problem of the seller traveler. The efficiencies of these new techniques are demonstrated in the analysis of the proposed model.

Tabla de Contenido

Introducción.

La optimización combinatoria es una rama de la optimización. Su dominio se compone de problemas de optimización donde el conjunto de posibles soluciones es discreto o se puede reducir a un conjunto discreto. A la hora de tratar con problemas de optimización combinatoria, el objetivo consiste en encontrar la mejor solución posible existente o solución óptima, aquella que minimiza una función de coste dada. Si el problema no es complejo existen vías sencillas para resolver dichos problemas. A medida que la complejidad del espacio de búsqueda aumenta, el coste de ejecución de dichos algoritmos puede aumentar de forma exponencial, convirtiendo la resolución en prácticamente inviable.

Para enfrentarse a estos problemas existen varias técnicas entre ellas las más conocidas son los algoritmos heurísticos y los metaheurísticos, además de algo que posee un carácter invaluable como es la percepción humana que enfatizada en la visualización de la información que proveen los datos pueden discernir soluciones óptimas para estos problemas complejos. Para ello los teóricos identifican las propiedades “preatentivas” o “preconscientes”, que son aquellas que llaman la atención de nuestra percepción incluso antes de ser procesadas por nuestro cerebro. (Ribera, 2014).

En el proceso de percepción se distinguen tres niveles, en el primer nivel se determina la emergencia de las cualidades preatentivas como son la forma, el color, el movimiento y la posición espacial. En el segundo nivel se procesa estas características para favorecer la búsqueda de patrones y en el tercer nivel se determina la percepción de los objetos. (Ribera, 2014)

Planteamiento del problema.

Existirán nuevos mecanismos de integración visual adecuados para añadir al software Atreeb v1.0 que permitan interactuar con el algoritmo de búsqueda metaheurística Recocido Simulado en tiempo de ejecución y que mejoren la eficiencia de la búsqueda.

Objetivo General.

Valorar la aplicación de nuevas técnicas de integración visual al software Atreeb v.1.0, para mejorar la eficiencia del mismo.

Objetivos Específicos.

- Identificar cuál serían las nuevas interacciones visuales a añadir al software Atreeb v1.0.
- Crear un nuevo modelo de técnicas de visualización para el software Atreeb v1.0.
- Implementar computacionalmente las nuevas funcionalidades identificadas y añadirlas al software existente.
- Evaluar los beneficios del modelo creado teniendo en cuenta comparaciones experimentales.

Preguntas de Investigación.

1. ¿Cuáles serían las interacciones que pudieran adicionarse al software Atreeb v1.0 que permitan al usuario guiar la búsqueda con mayor eficiencia?
2. ¿Cómo podrían integrarse estas técnicas de visualización en el software Atreeb v1.0?
3. ¿Qué beneficios se podrían obtener al aplicar estas nuevas técnicas de visualización al software Atreeb v1.0?

Análisis Crítico.

Posterior a la revisión del trabajo de diploma del curso anterior y basado en las recomendaciones de este, surge la necesidad de darle continuación. Este trabajo tiene como principal impacto realizar ajustes al software anterior, finalizar ideas que quedaron inconclusas y corregir los errores encontrados, además de dotarlo con nuevas funcionalidades para que sea más eficiente en la búsqueda de las soluciones.

El software Atreeb v1.0 posee como principales desventajas que la visualización del grafo que modela la problemática del Viajero Vendedor es pequeña ante la percepción de los usuarios, las aristas que unen a los nodos son difícil de visualizar por su ancho e intensidad del color. Además, carece de información explicativa para el usuario que le ayude a realizar las

funcionalidades del software. También en el análisis de la aplicación se encontró un error en la representación manual ya que la validación del grafo realizado, en ocasiones no representa un ciclo de Hamilton, es decir el grafo no queda cerrado completamente y no cumple con la definición del problema del Viajero Vendedor.

Justificación.

La presente investigación, tiene como antecedentes las investigaciones que se han realizado y que se encuentran en progreso de desarrollo en el Laboratorio de Computación Gráfica del Centro de Estudios Informáticos de la Universidad Central de Las Villas, además, despierta el interés del Laboratorio de Inteligencia Artificial del mismo centro, y también cuenta con una tesis realizada por el estudiante de Ingeniería Informática Juan Daniel Pedraza Díaz en el curso 2016-2017 sobre este tema, en la tesis anterior se implementó una aplicación de escritorio utilizando el lenguaje de programación Java para resolver el Problema del Viajero Vendedor a través del algoritmo meta heurístico Recocido Simulado utilizando algunas técnicas de integración visual en tiempo de ejecución, este trabajo constituye una continuidad del anterior debido a la necesidad de enriquecer el software con nuevas técnicas de visualización que ayuden al algoritmo a mejorar su coste computacional, utilizando la percepción de los usuarios y herramientas que ayuden al algoritmo a encontrar soluciones iniciales basadas en un análisis heurístico que modele la problemática lo que implicaría que dicho algoritmo evaluaría soluciones atractivas en sus primeros pasos.

Estructura del documento:

El documento se encuentra estructurado de la siguiente forma:

Presenta una introducción donde se presenta el contexto, la problemática y los objetivos: generales y específicos.

Presenta cuatro capítulos:

Capítulo 1: contiene la fundamentación teórica, dentro de la cual se van a encontrar el objeto de estudio, la fundamentación de los objetivos y los elementos teóricos de la investigación.

Capítulo 2: contiene el modelo de negocio y requisitos, dentro de ellos se encontrarán el modelo de negocio, los actores a automatizar, los requisitos funcionales y no funcionales, el diagrama de paquetes, el diagrama de casos de uso del sistema, así como una descripción de los casos de uso más significativos.

Capítulo 3: posee una descripción de la propuesta de solución, dentro de ella se encontrarán: la arquitectura del sistema, el diagrama de clases del diseño y diagrama de secuencia de los casos de uso más significativos, el tratamiento de los errores del sistema, y una descripción del modelo propuesto.

Capítulo 4: contiene las Pruebas y Análisis de factibilidad, dentro de ella se van a mostrar la planificación basada en uno de los métodos de estimación, en este caso se utilizó la estimación basada en casos de uso, y los Casos de Pruebas, donde se verán las secciones y escenarios a probar.

Posteriormente se muestran las conclusiones, las recomendaciones propuestas, y la bibliografía utilizada.

Capítulo 1: Fundamentación Teórica.

Introducción.

Este capítulo ofrece una visión teórica del algoritmo de búsqueda meta heurística Recocido Simulado, aborda la temática de las técnicas de visualización empleadas para representar el Problema del Viajero Vendedor modelado con grafos. Para ello, se ofrece una variedad de conceptos importantes de diferentes autores e investigadores.

Problemas y Complejidad.

Resolver un problema de optimización es encontrar la mejor solución posible a un problema formulado en lenguaje matemático, donde el criterio que evalúa la calidad de una solución es cuantitativo, generalmente asociado a un costo y denominado función objetivo. En un problema combinatorio de optimización se desea encontrar un orden específico sobre un conjunto de elementos discretos.(Mario César Vélez, 2007) .

Ante un problema de optimización, la primera pregunta que se debe responder es si es fácil o difícil de resolver. Aunque parece una pregunta simple, sólo a partir de la década de los setenta los investigadores abordaron este tema, introduciendo un nuevo campo de investigación: la complejidad computacional, la cual, entre otros usos, determina si un problema es fácil o no de acuerdo con los algoritmos conocidos para resolverlo.(Mario César Vélez, 2007).

Para la teoría de la complejidad computacional, la capacidad de un algoritmo para resolver un problema la determina el número de operaciones aritméticas necesarias para su ejecución. Un problema es fácil si existe un algoritmo que lo resuelve en tiempo polinomial; es decir, si el número de operaciones necesarias para que el algoritmo resuelva el problema es una función polinomial del tamaño del problema. Si esta función no es polinomial, se dice que el algoritmo es no polinomial y el problema se considera difícil.(Mario César Vélez, 2007).

Mientras los problemas se clasifican en fáciles o difíciles, los algoritmos se clasifican en exactos o completos y de aproximación o heurísticos. Los algoritmos exactos son aquellos en los que existe la garantía de que

encontrarán la solución óptima; mientras que para los algoritmos de aproximación sólo se puede afirmar que encontrarán una solución aceptable, no necesariamente óptima.(Mario César Vélez, 2007).

Técnicas Heurísticas y metaheurísticas.

Las heurísticas son algoritmos que encuentran soluciones de buena calidad para los problemas combinatorios complejos. Los algoritmos heurísticos son fáciles de implementar y encuentran buenas soluciones con esfuerzos computacionales relativamente pequeños en un tiempo razonable, pero no garantizan encontrar la solución óptima global de un problema.(Jesús del Carmen Peralta Abarca, 2015)

Las metaheurísticas son métodos aproximados diseñados para resolver problemas de optimización combinatoria, en los que los heurísticos clásicos no son efectivos. Las metaheurísticas proporcionan un marco general para crear nuevos algoritmos híbridos, combinando diferentes conceptos derivados de la inteligencia artificial, la evolución biológica y los mecanismos estadísticos. La mayor ventaja de los metaheurísticas frente a otros métodos está en su gran flexibilidad, lo que permite usarlos para abordar una amplia gama de problemas.(Mario César Vélez, 2007).

Algoritmo de búsqueda metaheurística Recocido Simulado

El método de Recocido Simulado (RS) fue propuesto por Kirkpatrick, Gelatt y Vecchi en 1983, e inicialmente se creó para minimizar funciones de costo. Su nombre está inspirado en el proceso de recocido de sólidos, el cual utiliza un procedimiento que va disminuyendo la temperatura, con lo cual se modifica la estructura del material. El enfriamiento debe hacerse de manera lenta para obtener configuraciones moleculares resistentes. Cada etapa del enfriamiento tiene asociada una energía y una configuración del material determinadas.(Jesús del Carmen Peralta Abarca, 2015)

EL RS es una generalización de los métodos Monte Carlo para localizar estadísticamente al óptimo global de una función multi variable. Este algoritmo realiza una búsqueda parcial estocástica dentro de la región permitida para un conjunto de variables de optimización. En problemas de minimización, las perturbaciones que ocasionan incrementos en el valor de

la función objetivo son aceptadas con una probabilidad controlada empleando el criterio de la Metropolis. Estas perturbaciones se realizan en varias ocasiones y permite que el algoritmo escape de mínimos locales.(Adrián Bonilla Petricolet, 2005).

$$\text{Prob}(\text{aceptar } x') = \begin{cases} 1 & , f(x') < f(x) \\ exp\left(-\frac{f(x')-f(x)}{c}\right) & , f(x') \geq f(x) \end{cases}$$

Figura 1: Criterio de la Metropolis.

Generalmente el RS puede localizar al óptimo global de la función objetivo o una aproximación a este en tiempos de cómputos razonables. Diversos algoritmos han sido desarrollados para el RS, que difieren principalmente en los mecanismos para perturbar a las variables de optimización y en el procedimiento para modificar los parámetros del RS durante la secuencia de optimización.(Adrián Bonilla Petricolet, 2005) .

Actualmente es una de las metaheurísticas más aplicadas en optimización combinatoria e inclusive se ha combinado con otras estrategias heurísticas y metaheurísticas por su gran eficiencia.

Aplicaciones del algoritmo Recocido Simulado.

La aplicación del RS es muy variada dentro de los campos de la ingeniería: en logística, producción, transporte, mecánica, electrónica, entre otros.(Jesús del Carmen Peralta Abarca, 2015).

Algoritmo de búsqueda local Hill Climbing.

También conocido como el método de ascenso de colinas es típicamente un algoritmo de búsqueda local, usa técnicas de mejoramiento iterativo, su inicio es a partir de un punto actual en el espacio de búsqueda. Si el nuevo punto es mejor, se transforma en el punto actual, si no, otro punto vecino es seleccionado y evaluado, el método termina cuando no hay mejorías, o cuando se alcanza un número predefinido de iteraciones.

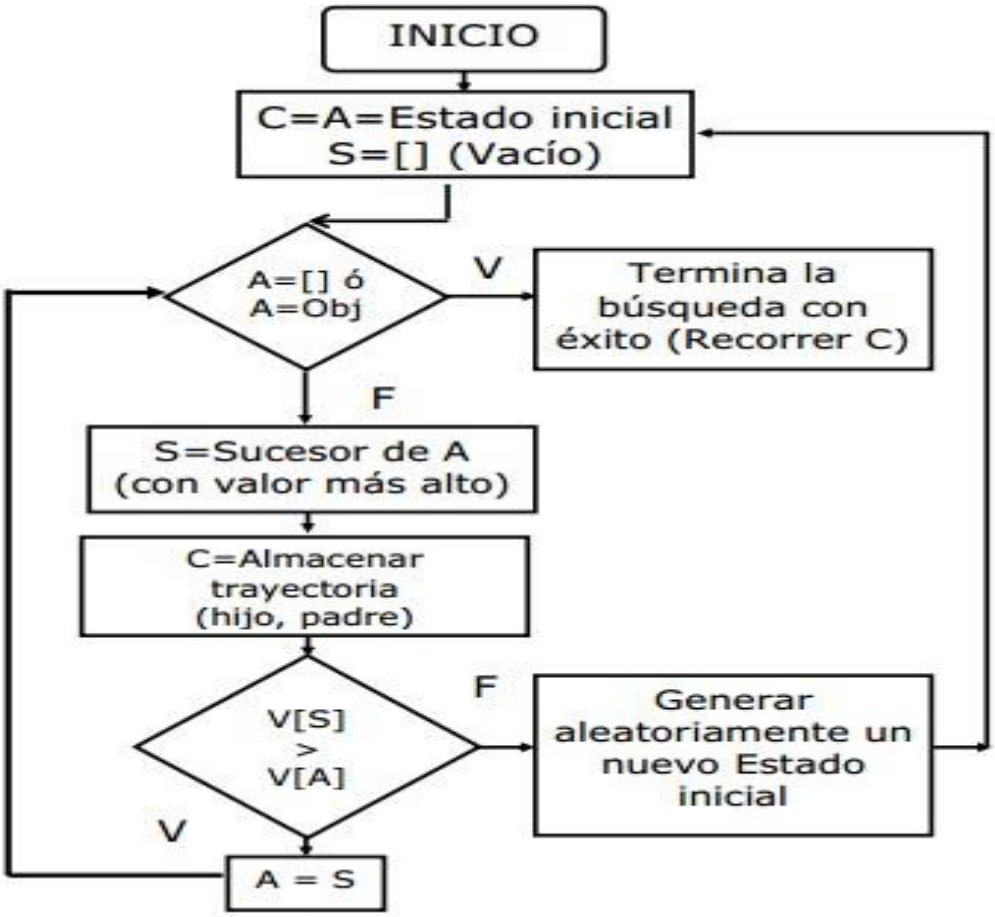


Figura 2: Seudocódigo del algoritmo Hill Climbing.

Problema del Viajero Vendedor.

Definición: Dado un conjunto finito de ciudades, y costos de viaje entre todos los pares de ciudades, encontrar la forma más barata de visitar todas las ciudades exactamente una vez, y volver al punto de partida.

El Problema del Viajero Vendedor, que denotaremos como TSP (*traveling salesman problem en inglés*) fue definido en el siglo XIX por el matemático irlandés W. R. Hamilton y por el matemático británico Thomas Kirkman. El TSP es uno de los problemas más intensamente estudiados tanto en ciencias de la computación, investigación de operaciones y en matemáticas, y pese a eso aún no existen métodos computacionalmente efectivos para resolverlo en el caso general.(Vargas, 2012).

La complejidad del cálculo del problema del agente viajero ha despertado múltiples iniciativas por mejorar la eficiencia en el cálculo de rutas. El método más básico es el conocido con el nombre de fuerza bruta, que consiste en el

cálculo de todos los posibles recorridos, lo cual se hace extremadamente ineficiente y casi que se imposibilita en redes de gran tamaño. También existen heurísticas que se han desarrollado por la complejidad en el cálculo de soluciones óptimas en redes robustas, es por ello que existen métodos como el vecino más cercano, la inserción más barata y el doble sentido entre muchos otros.(López, 2016).

Aplicaciones del TSP.

La mayor parte de los avances en la historia del TSP han sido motivados por aplicaciones del mismo en casos cotidianos. Hoy en día, hay multitud de campos en los que este problema juega un papel importante a la hora de tomar decisiones de eficiencia. A continuación, se muestran algunos ejemplos de ello.(García, 2015)

- **Logística:** La elección de un recorrido eficiente a la hora de transportar mercancías, pasajeros, vehículos en torno a una serie de ciudades puede suponer un ahorro significativo de recursos.(García, 2015).
- **Industria:** se aplica en la optimización de secuencias de tareas, en la producción de circuitos electrónicos, conexiones entre aeropuertos, conexiones entre cadenas de ADN o incluso en la programación de robots en los almacenes de Amazon que se encargan de distribuir los paquetes que ordenan los clientes.(García, 2015).

Visualización de Grafos.

El dibujado de grafos direcciona el problema de visualizar información estructural o relacional construyendo representaciones visuales geométricas de grafos o redes que son los modelos subyacentes en una gran cantidad de datos abstractos.(Roxana Curino, 2009).

La generación automática del dibujo de un grafo tiene importancia en aplicaciones claves tales como la Ingeniería de Software, la Visualización de Información, Data Mining, el diseño de Bases de Datos e Interfaces Visuales, la Representación del conocimiento y las Telecomunicaciones entre otros dominios.(Roxana Curino, 2009).

El problema de dibujado de un grafo puede plantearse simplemente del siguiente modo:

Dado un conjunto de nodos y un conjunto de arcos que representan las relaciones entre los nodos, obtener la posición de los nodos y la curva que debe ser dibujada para cada arco.(Roxana Curino, 2009).

La utilidad de una representación dada depende de diversos factores y hay varios criterios considerados estándar que se tienen en cuenta en la visualización de grafos. El tamaño del mismo juega un rol preponderante cuando se lo quiere visualizar: no pueden usarse los mismos criterios de dibujado cuando el grafo es chico que cuando tiene miles de nodos. Usualmente, los vértices son representados por símbolos como cajas o puntos y las aristas por curvas abiertas conectando los símbolos que representan a los vértices asociados. Sin embargo, estos estándares varían, sin duda, según la aplicación. Por otro lado, si la cantidad de elementos es grande, es decir, mayor a miles de nodos, podrá mostrarse todo el grafo para dar una idea de su estructura general, pero se hace difícil si no imposible ver el detalle de una parte del mismo. El tamaño del grafo plantea problemas adicionales de navegabilidad e interactividad.(Roxana Curino, 2009).

Representación del TSP.

Sea G = (N, A) un grafo completo, donde N = 1,…, n es el conjunto de nodos o vértices y A es el conjunto de arcos. Los nodos i = 2,…, n se corresponden con los clientes a visitar mientras que el nodo 1 es considerado la ciudad de origen y destino. A cada arco (i, j) se le asocia un valor no negativo dij, que representa la distancia del vértice i al j. El uso de los arcos (i, i) no está permitido, por lo que se impone que dii= 1 $\forall$ i $\in N$. (Moreno, 2015).

En general podremos concluir que el TSP se define como un grafo completo con costes asociados a los arcos y consiste en encontrar el camino Hamiltoniano de coste mínimo. (Moreno, 2015).

Historia de la Visualización de Datos.

Los primeros ejemplos que se conocen de visualización de datos corresponden a diagramas geométricos, a tablas de las posiciones de las estrellas y otros cuerpos celestes; y al armado de mapas para ayudar en la navegación y la exploración.

Junto con los comienzos de la teoría estadística y la colección sistemática de datos empíricos, fueron introducidos gráficos abstractos y gráficos funcionales. Las innovaciones tecnológicas generaron la materia prima que facilitó la reproducción de imágenes de datos. En una primera instancia, muchas de esas formas graficas aparecían en publicaciones de circulación limitada, de manera que no eran conocidas popularmente. (Pontis, 2007).

Las primeras décadas del siglo XX fueron una época de cambios en la vida social, política, cultural y económica que alteraron de manera radical los diferentes aspectos de la condición humana. El desencadenamiento de la primera guerra mundial movilizó a la civilización occidental generando un cambio en la concepción de la vida. En medio de este contexto, las formas graficas de comunicación experimentaron una serie de revoluciones creativas que cuestionaron sus valores, su aproximación a la organización del espacio y su función en la sociedad. Por estos motivos, se denominó a esta etapa como "oscuros años modernos" de la representación visual de datos. (Pontis, 2007).

Durante el siglo XX, la gestación de nuevas ideas y el desarrollo de las tecnologías permitieron explorar las posibilidades representativas de las 2D del plano y generar nuevos métodos que permitieran representar información de 3D, favoreciendo el desarrollo grafico multi-variable. Grandes corporaciones estadounidenses cumplieron una función importante en el desarrollo de productos y servicios, gracias a los cuales fue posible que el diseño de información comenzara a salir de la inactividad. La invención del primer ordenador, hizo posible realizar pruebas y experimentar distintas técnicas gráficas para la elaboración de diagramas y mapas. También con la información se crearon nuevas incertidumbres, motivando a retomar los estudios relacionados con el método gráfico. (Pontis, 2007).

La manera de visualizar en la historia ha sido de gran variedad según el tipo información que se desea transmitir. Una cuestión sumamente importante es el enfoque y la selección del soporte visual que se desea dar a la información, si bien puede suponerse una objetividad y un limpio deseo de informar, de parte del investigador, una mala decisión en cuanto a la visualización de la información puede, por el contrario, desinformar, alarmar o confundir al usuario. La visualización es una herramienta tan poderosa y no solo sirve para mostrar,

sino que puede encausar los criterios del espectador, la forma en que un dato se presenta visualmente, puede facilitar el flujo de información, pero a su vez también la puede manipular y causar especifico un efecto en el otro. (Shanahan, 2017)

Visualización ¿Disciplina o Ciencia?

Con el aporte de distintas áreas de la investigación como son ciencia de la información, las ciencias cognitivas y la ciencia de la computación, se han elaborado conceptos, métodos y técnicas en función de las distintas transformaciones computacionales, cognitivas y mecánicas necesarias para convertir y representar espacialmente a la información. La existencia de una ciencia visual se podría definir como aquella que estudia el efecto de producir imágenes en la mente, y presenta una naturaleza interdisciplinar, entre sus ramas de investigación estarían la semiótica de la visualización, la estética de la visualización, la filosofía de la visualización, y la comunicación de la visualización entre otras posibles. Lo cierto es que, aun cuando no existe un consenso, en general se acepta a la visualización como una disciplina científica que presenta una gran comunidad académica dedicada a su estudio e investigación. (Ponjuán, 2009).

Categorías de la visualización.

La visualización científica se centra en la visualización de datos científicos no abstractos; sin embargo, no se puede decir que no sea informativa, ni que la visualización de información no sea científica. La distinción de nombres tal vez se relacione con la propia evolución histórica de ambos campos, que delimitó de alguna forma alcances y propósitos de la investigación de cada uno por separado. La visualización científica ha tenido mayor interés en la representación de fenómenos físicos y ha estado, por tanto, más cercana al campo de los gráficos y de los datos, con interpretaciones espacio-temporales de los problemas investigados. La visualización de información, por su parte, se ha interesado más por los temas cognitivos al interactuar con grandes volúmenes de información, y ha estado más cerca de los estudios de la interacción, de la recuperación de la información y de los estudios cognitivos. (Ponjuán, 2009).

Con respecto a la representación, la visualización del conocimiento y la visualización de la información tienen como propósito común visualizar estructuras, las cuales comprenden elementos de conocimiento o de información. Con este enfoque de organizar la información y los conocimientos de manera que puedan accederse fácil y exhaustivamente, se combinan estructuras de conocimiento extraídas automáticamente de los textos con estructuras de conocimientos creadas por expertos humanos para proveer varias funciones interactivas que facilitan la interacción visual coherente del usuario con los conceptos y los documentos. Existe entonces un interés común en ambos campos por facilitar la accesibilidad y obtener el sentido de los conocimientos y elementos de información representados a partir del desarrollo de artefactos visuales.(Ponjuán, 2009).

Fundamentación del lenguaje de desarrollo y librerías.

El lenguaje de programación seleccionado para implementar las nuevas funcionalidades identificadas es Java, para mantener una coherencia de lenguaje con la aplicación Atreeb V1.0 desarrollada anteriormente.

Características del lenguaje.

Java es un lenguaje de desarrollo de propósito general, y como tal es válido para realizar todo tipo de aplicaciones profesionales, es orientado a objetos. Aprovecha características de la mayoría de los lenguajes modernos evitando sus inconvenientes, tiene una gran funcionalidad gracias a sus variadas librerías, el manejo de la memoria lo gestiona el propio lenguaje y no el programador. Genera aplicaciones con pocos errores e Incorpora Multi-Threading para permitir la ejecución de tareas concurrentes dentro de un mismo programa.

Librería Graph Stream.

Graph Stream es una librería Java, que se enfoca en los aspectos de la dinámica de gráficos. La meta de la biblioteca es proporcionar una manera para representar los gráficos y trabajar en ellos.(Team, 2010).

Conclusiones parciales.

En el presente capítulo se abordaron las definiciones teóricas que presenta este trabajo, se enfatizó en el algoritmo metaheurístico Recocido Simulado, en

la representación visual o modelación de un problema como lo es el TSP y se hizo mención sobre la ayuda que ofrece la percepción humana para solucionar problemas de optimización utilizando técnicas de visualización.

Capítulo 2: Modelo del negocio y requisitos.

2.1 Introducción.

En este capítulo se describen las características del sistema mediante el uso de diagramas que representan el modelado del negocio para su posterior implementación y entendimiento. Además, se abordan los requisitos funcionales y no funcionales que presenta el software para su correcto funcionamiento y utilización.

Un modelo de negocio explica cómo trabaja una organización, indicando quienes serán los clientes que iniciarán el caso de uso, cómo generamos utilidades, y cómo podemos entregar a los clientes un valor, en fin, describe la forma en que una organización crea, captura y entrega la información.

2.2 Actores del sistema a automatizar.

Todos los usuarios que interactúen con el software serán actores del sistema, esta aplicación desktop no presenta roles, es decir no existen usuarios con privilegios adicionales sobre otros usuarios. El usuario tiene como único requisito tener conocimientos sobre el problema del viajero vendedor y conocer los algoritmos Hill Climbing y Recocido Simulado para el correcto entendimiento y manejo del software.

2.3 Definición de los requisitos funcionales.

Los requisitos funcionales describen las funciones que llevan a cabo el software, cómo debe reaccionar ante ciertas entradas y cómo debe comportarse en situaciones particulares. Estos dependen del tipo de software, las expectativas de los usuarios y del tipo de sistema donde el software se usará. Además, estos se dividen en 2 categorías requisitos de usuario, que son descripciones de alto nivel de lo que el sistema debe hacer y en requisitos del sistema que describen los servicios con más detalles.(Ruiz) En la Tabla 1 se muestra un listado de los requisitos funcionales del sistema.

RF1	Dibujar arista
Descripción	El sistema tiene que permitir dibujar aristas entre los nodos, siempre validando que todos los nodos queden unidos por un único camino, esta acción se

	puede realizar para determinar una solución inicial para ejecutar el algoritmo o dibujar un camino parcial lo suficientemente bueno a la visión del usuario.
RF2	**Eliminar arista**
Descripción	El sistema tiene permitir eliminar aristas con el objetivo de cambiar un camino parcial a la visión del usuario que su costo es superior a uno existente.
RF3	**Exportar imagen**
Descripción	El sistema tiene que permitir como opción al usuario guardar los resultados obtenidos de la optimización de las distancias, para ello se brinda la elección de salvar los lienzos solución actual, mejor solución y peor solución como una imagen jpg.
RF4	**Añadir nodos a lista de exclusión**
Descripción	El sistema permite añadir nodos a una lista de exclusión, este requerimiento tiene como objetivo que el nodo añadido no sea válido para el intercambio en las iteraciones del algoritmo Recocido Simulado manteniendo un camino que a la visión del usuario es suficientemente bueno, lo que facilita que el algoritmo haga menos permutaciones y el proceso de optimizar la distancias sea un poco más rápido que si tuviese que explorar todo el espacio de búsqueda.
RF5	**Conocer distancia entre dos nodos**
Descripción	El sistema permite al usuario conocer el costo de la distancia de ir de un nodo a otro, lo que le sugiere una perspectiva real para dibujar una representación de una solución inicial.
RF6	**Hacer zoom sobre los lienzos**
Descripción	El sistema permite hacer zoom sobre todos los lienzos de representación esta acción se realiza con

	la combinación del teclado Ctrl+Scroll del mouse, facilita al usuario la visión cuando la cantidad de nodos es muy abundante.
RF7	**Hallar solución Inicial por Hill Climbing**
Descripción	El sistema permite encontrar una solución inicial aplicando el algoritmo Hill Climbing Simple para determinar una solución inicial lo suficientemente buena.
RF8	**Llevar el tiempo de ejecución de la corrida**
Descripción	El sistema, al iniciar la representación visual de la solución inicial comienza a contar el tiempo que demora dicha representación al terminar el sistema detiene el contador y muestra el tiempo en el panel de resultados numéricos. Al iniciar la ejecución del algoritmo Recocido Simulado el sistema continúa contabilizando el tiempo, iniciando en el valor que se detuvo hasta que el algoritmo termine todas las iteraciones pertinentes o sea detenido por el usuario.

Tabla 1 : Requisitos funcionales

2.4 Definición de los requisitos no funcionales.

Los requisitos no funcionales son aquellos requerimientos que no se refieren directamente a las funciones específicas que entrega el sistema, sino a las propiedades emergentes de esté como la fiabilidad, la respuesta en tiempo y la capacidad de almacenamiento. Es decir, los requisitos no funcionales definen las restricciones del sistema. En la Tabla 2 se muestra un listado de los requisitos no funcionales que presenta el sistema.

RNF1	**Software**
Plataforma	➢ El software es una aplicación de escritorio programada en el lenguaje de alto nivel Java, por tanto, es una aplicación multiplataforma y requiere que el

	computador donde se va a ejecutar tenga previamente instalado la máquina virtual de Java (JVM) correspondiente al sistema operativo.
RNF2	**Usabilidad**
Exploración	➢ La aplicación está compuesta por un menú principal en la parte superior de la ventana, tres lienzos de representación, un panel de control a la izquierda y un panel inferior para mostrar los resultados. ➢ Los botones tienen la representación de la acción que realiza mediante un icono descriptivo, además de un toltip que muestra la acción en formato de texto.
RNF3	**Apariencia o interfaz**
Vistas	➢ El sistema ofrece una interfaz amigable e intuitiva con apariencia a una ventana estándar del sistema operativo Windows. ➢ El software provee una serie de mensajes para notificar al usuario en cada situación. ➢ Las utilizaciones de los colores se manifiestan en tonos de grises para el menú principal, blanco para los lienzos, fondo negro para los botones con iconos en blanco y los textos se muestran en color blanco con fondo negro.
RNF4	**Rendimiento**
Tiempo de respuesta	➢ Para cualquier petición del usuario al sistema no debe tardarse más de 5 segundos en la respuesta.
RNF5	**Escalabilidad**
Descripción	➢ El sistema mantiene una arquitectura basada en paquetes que le permitirá

	agregar nuevas funcionalidades al sistema sin afectar las ya existentes.
RNF6	**Documentación**
Descripción	➢ El software presenta un manual de ayuda para cualquier incógnita que surja en su utilización.

Tabla 2 : Requisitos no funcionales.

2.5 Paquetes y sus relaciones.

Un paquete es una agrupación de elementos, bien sea casos uso, clases o componentes. Los paquetes pueden contener a su vez otros paquetes anidados que en la última instancia contendrá alguno de los elementos anteriores. Un paquete se representa con un símbolo en forma de 'carpeta'. (Alejandra Ortiz).

La asignación entre paquetes debe seguir un cierto principio racional, tal como funcionalidad común, implementación estrechamente relacionada y un punto de vista común (James Rumbaugh, 1998). En la Tabla 3 se muestra un listado de los paquetes que presenta el sistema con su descripción.

Paquete	Descripción
Vistas	Almacena todas las interfaces graficas que tiene la aplicación.
Iconos	Se encuentran todos los iconos utilizados por las vistas.
HillClimbing	Almacena las clases que tienen que ver directamente con el algoritmo Hill Climbing.
Útiles	Almacena las clases adicionales usadas por la aplicación, además contiene la clase controladora que controla las peticiones del usuario a las vistas y le gestiona una respuesta
Clase Recocido Simulado	Almacena las clases que tienen que

	ver directamente con el algoritmo Recocido Simulado.
Servicios	Contiene las clases que manejan cada servicio del software, procesan las tareas pesada y se complementa con el controlador para gestionar las peticiones a las vistas por el usuario

Tabla 3 : Paquetes y sus descripciones.

2.6 Diagrama de Paquetes.

Un diagrama de paquetes muestra como un sistema está dividido en agrupaciones lógicas mostrando las dependencias entre esas agrupaciones, Dado que normalmente un paquete está pensado como un directorio, los diagramas de paquetes suministran una descomposición de la jerarquía lógica de un sistema(Alejandra Ortiz).

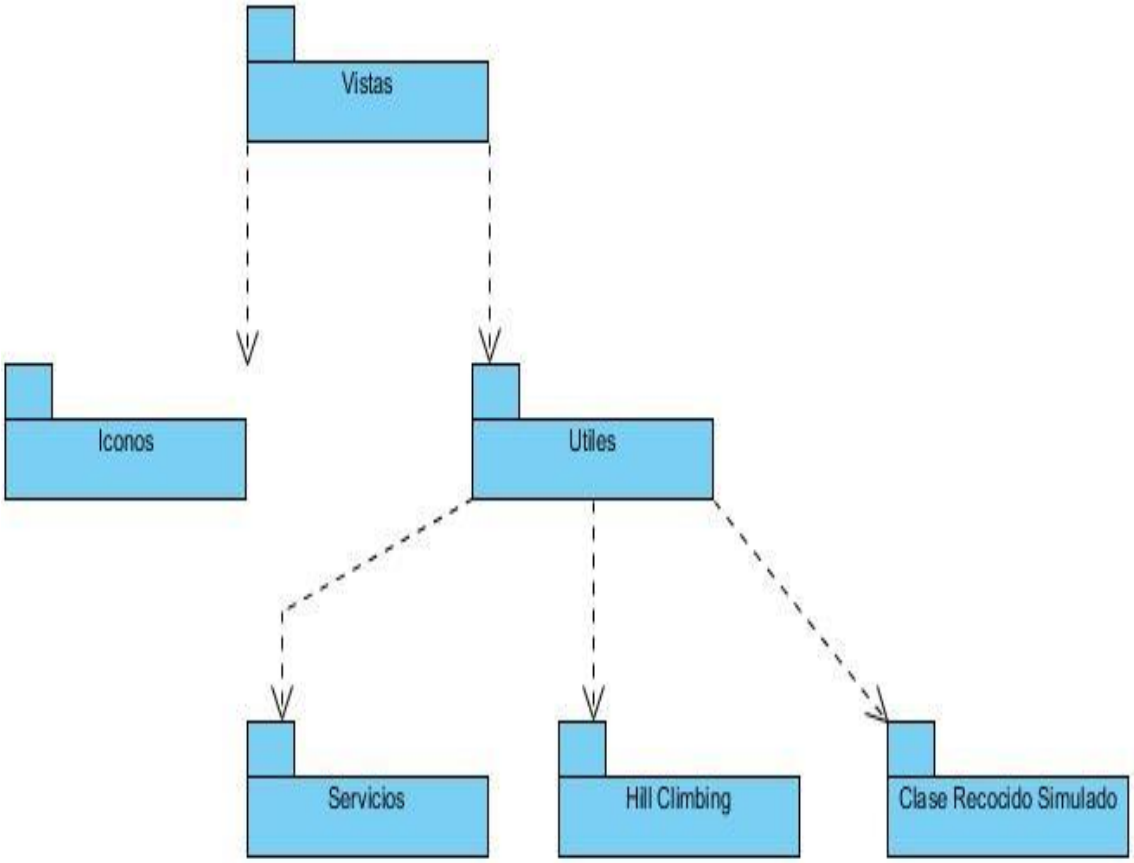


Figura 3: Diagrama de paquetes.

2.7 Diagrama de casos de uso del sistema.

Un caso de uso define una secuencia de acciones que da lugar a un resultado de valor observable. Los casos de uso proporcionan una estructura para expresar requisitos funcionales en el contexto de procesos empresariales y de sistema. Los casos de uso pueden representarse como un elemento gráfico en un diagrama y como una especificación de caso de uso en un documento textual.

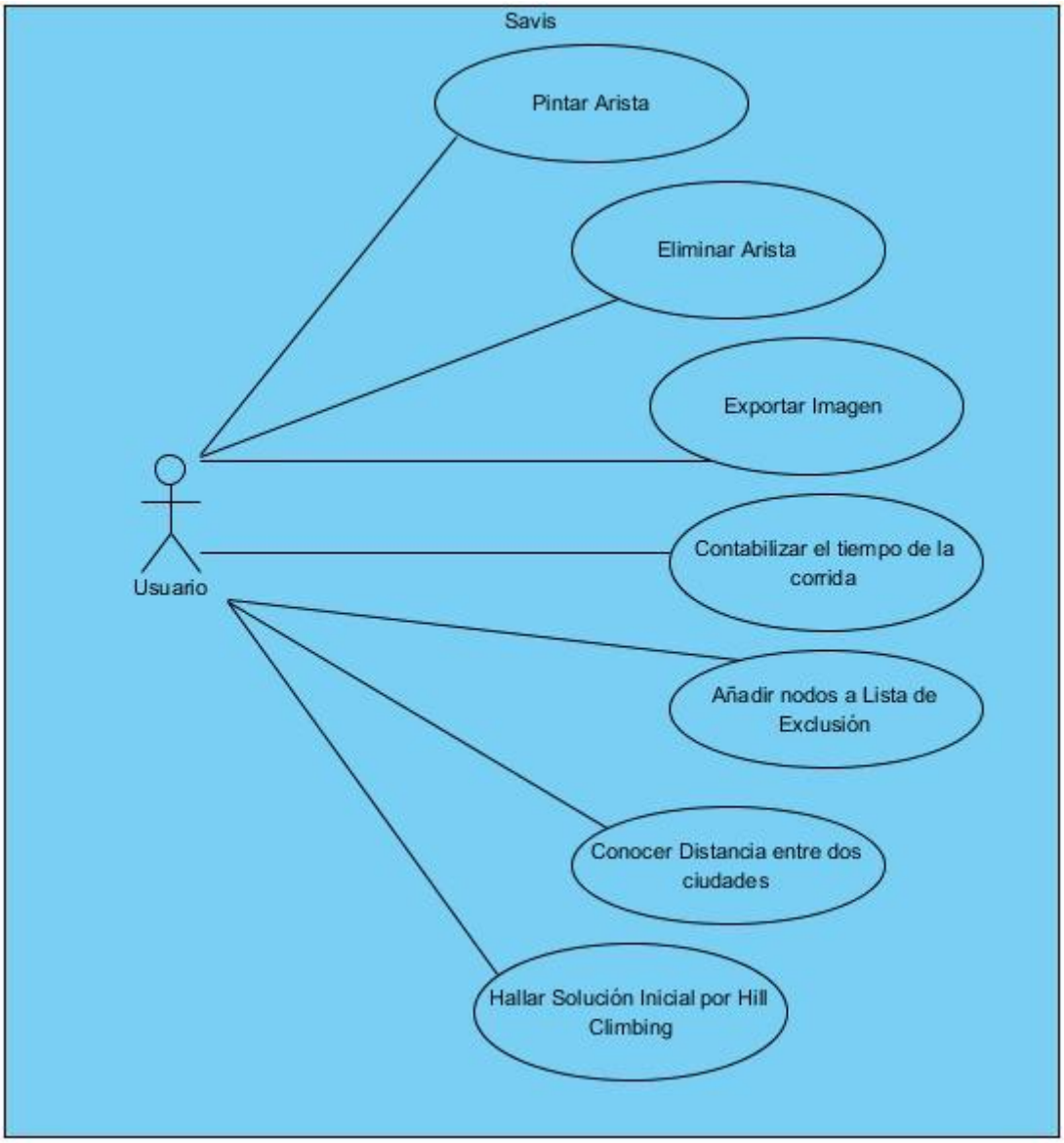


Figura 4: Diagrama de casos de uso del sistema.

2.8 Determinación de los casos de uso del sistema más significativos.

Para determinar los casos de uso más significativos hay que tener en cuenta su clasificación y estos se clasifican en:

- **Primario:** los casos primarios de uso representan los procesos comunes más importantes**.**
- **Secundarios:** los casos secundarios de uso representan procesos menores o raros.
- **Auxiliares**: Casos de uso que no son determinantes para la arquitectura del sistema.
- **Opcional:** los casos opcionales de uso representan procesos que pueden no abordarse.

Clasificaciones	Casos de uso del Sistema
Primarios	• Hallar Solución Inicial por Hill Climbing. • Contabilizar el tiempo de la corrida.
Secundarios	• Pintar Arista. • Eliminar Arista. • Añadir nodos a la Lista de Exclusión.
Auxiliares	• Exportar Imagen.
Opcionales	• Conocer distancias entre dos ciudades.

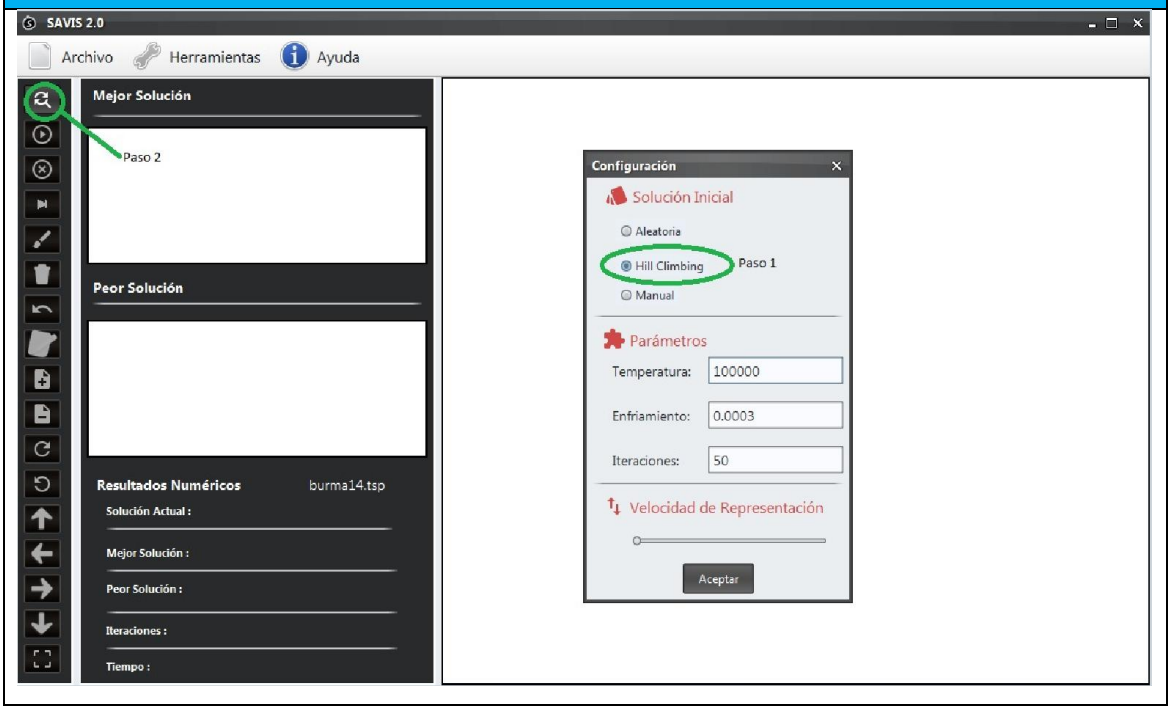
Tabla 4: Clasificación de los Casos de Uso.

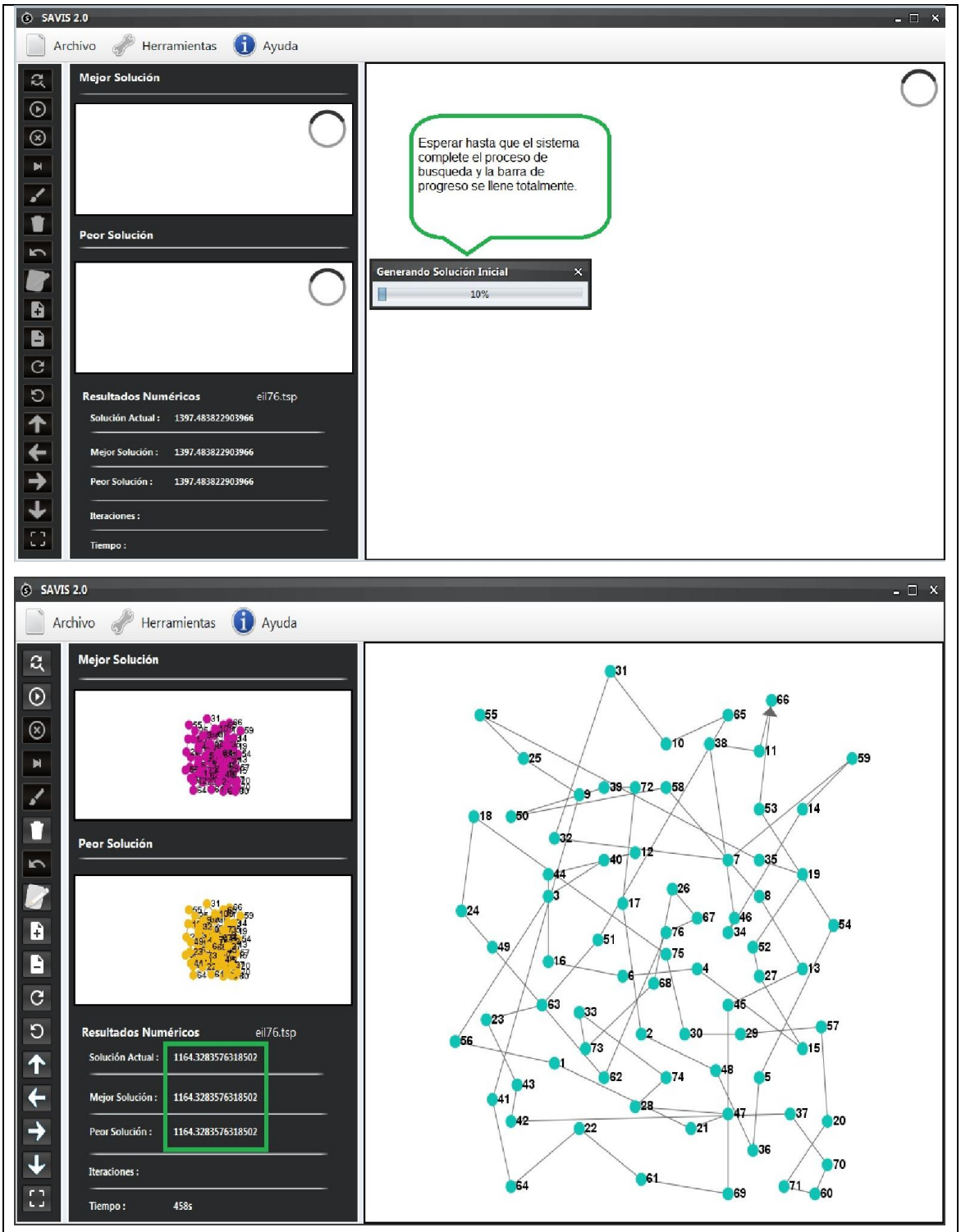
2.9 Descripción de los casos de uso del sistema más significativos.

A continuación, se muestra una descripción de los casos de uso más significativos para su mejor comprensión y entendimiento.

Caso de uso del Sistema	Hallar Solución Inicial por Hill Climbing
Actores	Usuario
Propósito	Encontrar una solución inicial eficiente para cuando el software empiece a ejecutar el algoritmo Recocido Simulado parta de una solución que optimice el costo computacional.
Resumen	El usuario inicia el software, selecciona la

	opción de solución inicial por Hill Climbing en la configuración (predefina por defecto) y ejecuta la acción de representación, posterior el software comienza el proceso de encontrar la mejor solución a través del algoritmo en un total de 100 iteraciones, luego de terminar la ejecución se representa en el panel principal.
Responsabilidades	Encontrar una solución inicial eficiente.
Requisitos especiales	-
Precondiciones	Que el software haya cargado un fichero tsp.
Descripción	



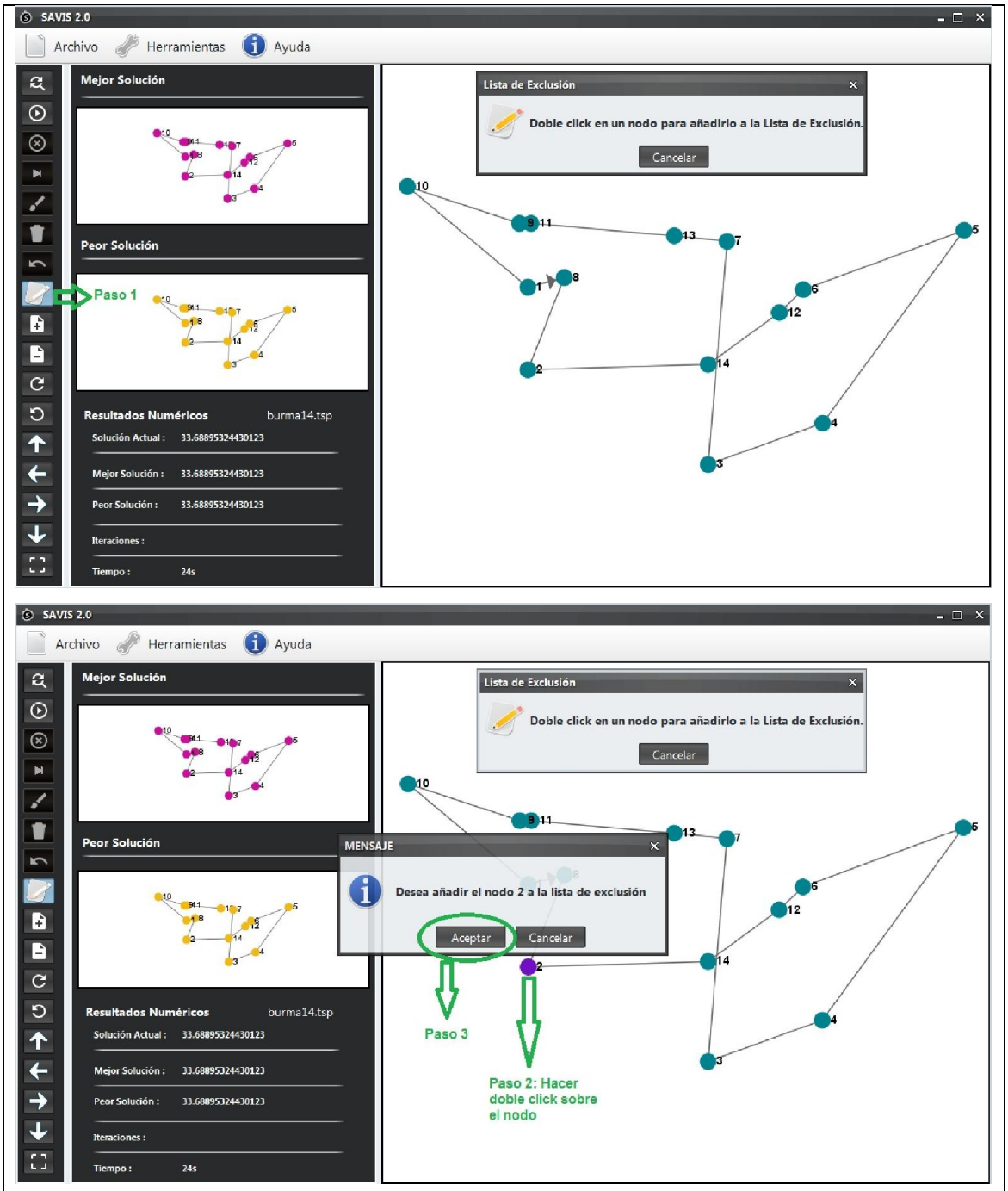


Flujo normal de los eventos	
Acción del actor	Respuesta del sistema
1. El usuario selecciona en la barra de menú la opción de configuración	2. El sistema muestra la vista correspondiente a la configuración
3. El usuario selecciona en	4. El sistema guarda la opción

la solución inicial Hill Climbing.	seleccionada.
5. El usuario selecciona en la barra de herramientas la opción de Representar.	6. El sistema muestra una ventana con una barra de progreso que empieza a llenarse mientras el algoritmo encuentra la mejor solución. 7. El sistema representa la mejor solución.

Tabla 5: Caso de uso Hallar solución inicial por Hill Climbing.

Caso de uso del Sistema	**Añadir nodos a la Lista de Exclusión**
Actores	Usuario
Propósito	Reducir el espacio de búsqueda del algoritmo Recocido Simulado añadiendo nodos a una lista de exclusión y así posibilitar que los nodos marcados no estén disponibles para el intercambio que realiza el algoritmo.
Resumen	El usuario selecciona la opción Lista de Exclusión el sistema responde mostrando una ventana informativa, el usuario selecciona en el lienzo principal los nodos haciendo doble click sobre ellos.
Responsabilidades	Reducir el espacio de búsqueda.
Requisitos especiales	-
Precondiciones	Haber realizado una representación visual.
Descripción	



Flujo normal de los eventos	
Acción del actor	Respuesta del sistema
1. El usuario selecciona en el panel de herramientas la opción Lista de Exclusión.	2. El sistema muestra una ventana informativa con la descripción del proceso seleccionado.
3. El usuario hace doble click sobre un nodo en el lienzo principal.	4. El sistema muestra una ventana de confirmación.

5. El usuario confirma en el botón aceptar de la ventana.	

Tabla 6: Añadir nodos a la Lista de Exclusión.

2.10 Conclusiones parciales.

En este capítulo se analizó el modelo de negocio y los requisitos que presenta software, se visualizaron los casos de uso del sistema y se hizo una breve descripción de los más significativos, además se presentó la estructura de los paquetes de clases que se utilizaron en la realización de este producto.

Capítulo 3: Descripción de la propuesta de solución.

3.1 Introducción.

En este capítulo se ofrece una visión a la propuesta de solución de este trabajo, se analizará la arquitectura del software utilizada, las clases del diseño referente a los casos de uso más significativos con sus respectivos diagramas de secuencia, además se mencionarán los patrones de arquitectura y diseño utilizados en la aplicación, el modelo de técnicas de visualización y un experimento comparativo para medir la eficiencia de dicho modelo.

3.2 Arquitectura del sistema.

La arquitectura de software constituye un amplio marco que describe la forma, estructura, componentes y cómo estos interactúan en nuestro sistema. Un conjunto de decisiones significativas de un sistema software, la selección de los elementos estructurales a partir de los cuales se compone el sistema, las interfaces entre ellos, su comportamiento, sus colaboraciones y su composición dan lugar a un diseño centrado en la arquitectura.

Para la realización de este software se utilizó la Arquitectura Monolítica que es aquella en la que el software se estructura en grupos funcionales muy acoplados, involucrando los aspectos referidos a la presentación, procesamiento y almacenamiento de la información.(Reveco, 2010).

Este tipo arquitectura es muy utilizada en aplicaciones para escritorio: sistemas operativos, Office, juegos monousuario, etc.(Reveco, 2010).

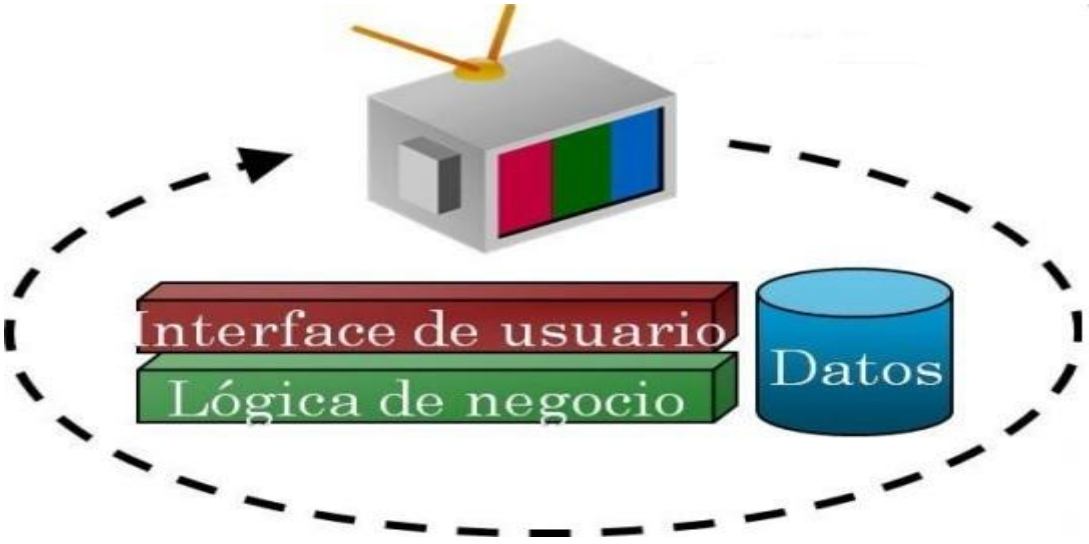


Figura 5: Arquitectura Monolítica

3.3 Diagrama de clases del diseño.

Un diagrama de clases del diseño muestra la especificación para las clases software de una aplicación. A diferencia del modelo conceptual, un diagrama de clases del diseño muestra definiciones de entidades software más que conceptos del mundo real. Un diagrama de clases del diseño brinda la siguiente información:

- Clases, asociaciones y atributos.
- Interfaces, con sus operaciones y constantes.
- Métodos.
- Navegabilidad.
- Dependencias.

Diagrama de clase del diseño para el caso de uso Hallar solución inicial por Hill Climbing.

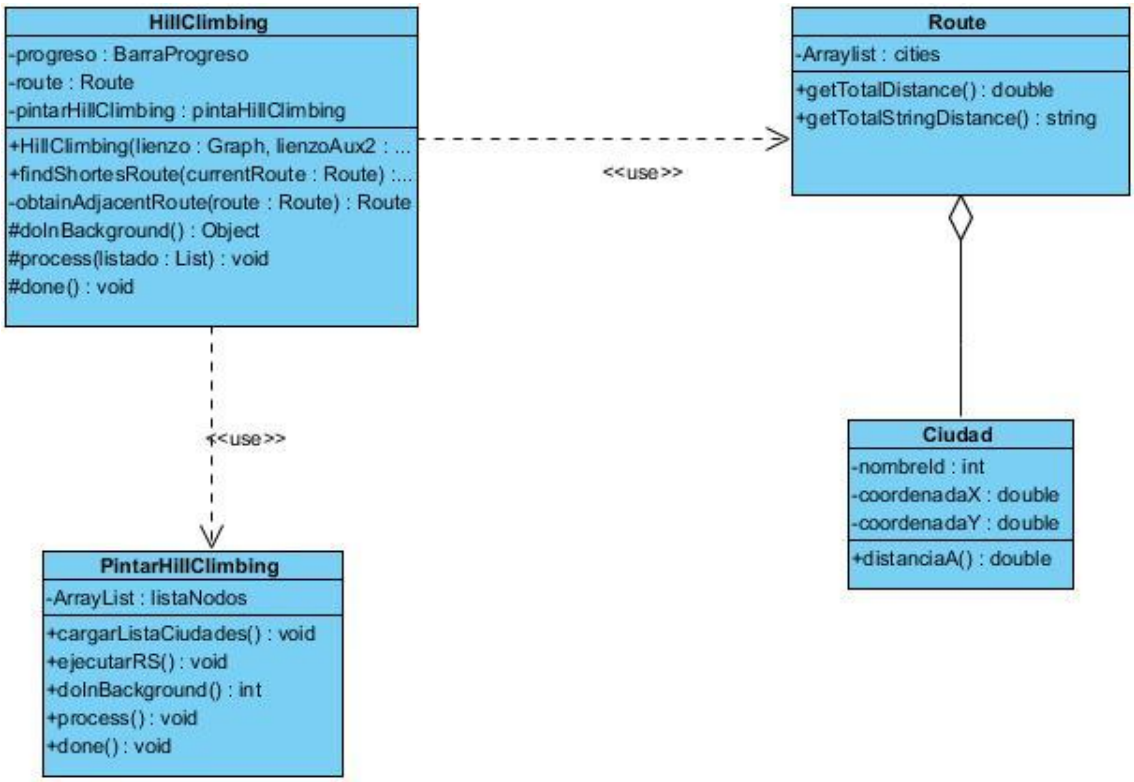


Figura 6: Diagrama de clase del diseño para el caso de uso Hallar solución inicial por Hill Climbing.

Diagrama de clase del diseño para el caso de uso Añadir nodos a Lista de Exclusión.

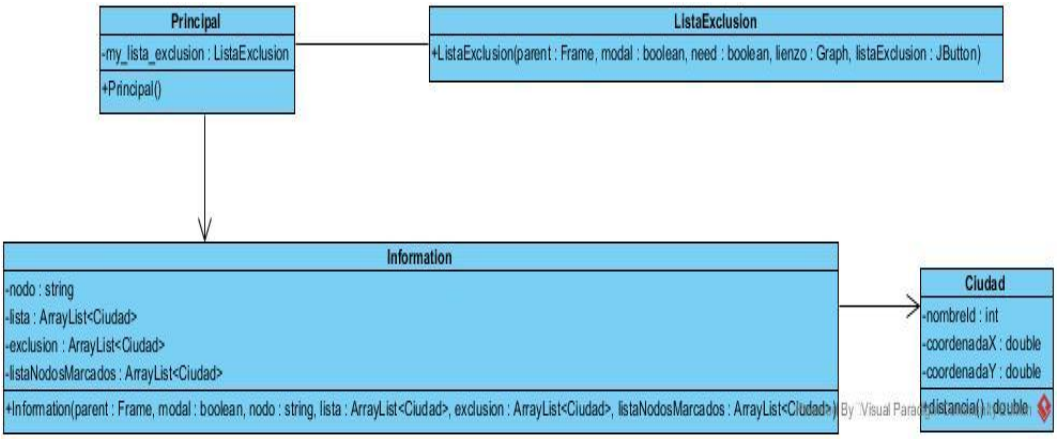


Figura 7: Diagrama de clase del diseño para caso de uso Añadir nodos a Lista de Exclusión.

3.4 Diagrama de secuencia.

Un diagrama de secuencias muestra la interacción de un conjunto de objetos de una aplicación a través del tiempo, en el cual se indicarán los módulos o clases que formaran parte del programa y las llamadas que se hacen cada uno de ellos para realizar una tarea determinada, por esta razón permite observar la perspectiva cronológica de las interacciones. Es importante recordar que el diagrama de secuencias se realiza a partir de la descripción de un caso de uso.

Diagrama de secuencia para el caso de uso Hallar solución inicial por Hill Climbing.

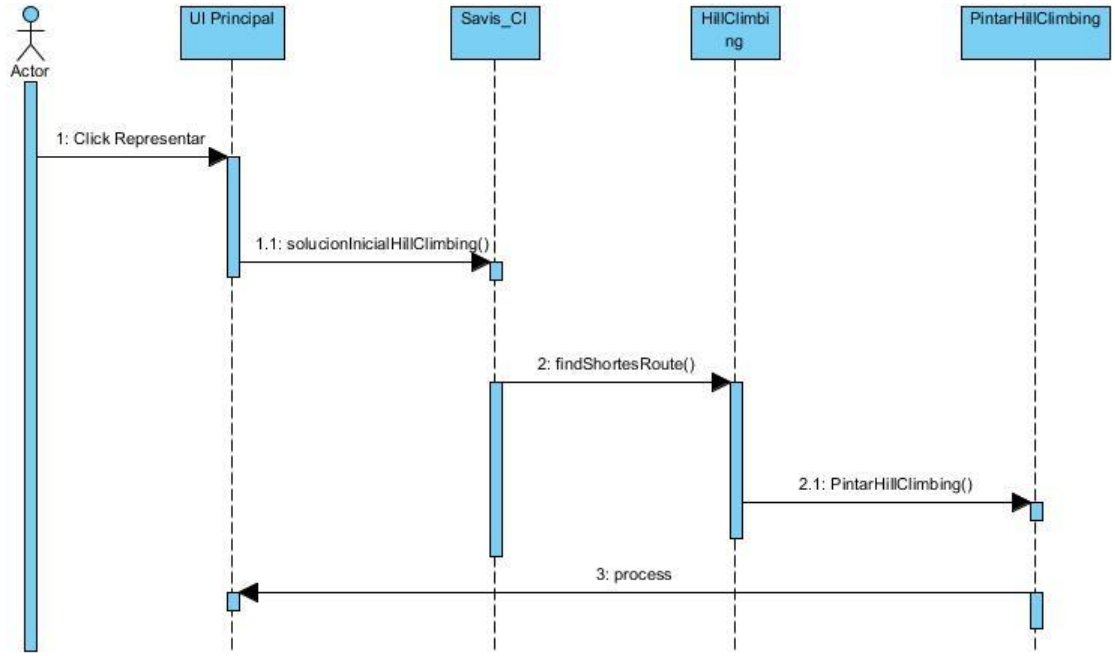


Figura 8: Diagrama de secuencia para el caso de uso Hallar solución inicial por Hill Climbing.

Diagrama de secuencia para el caso de uso Añadir nodos a Lista de Exclusión.

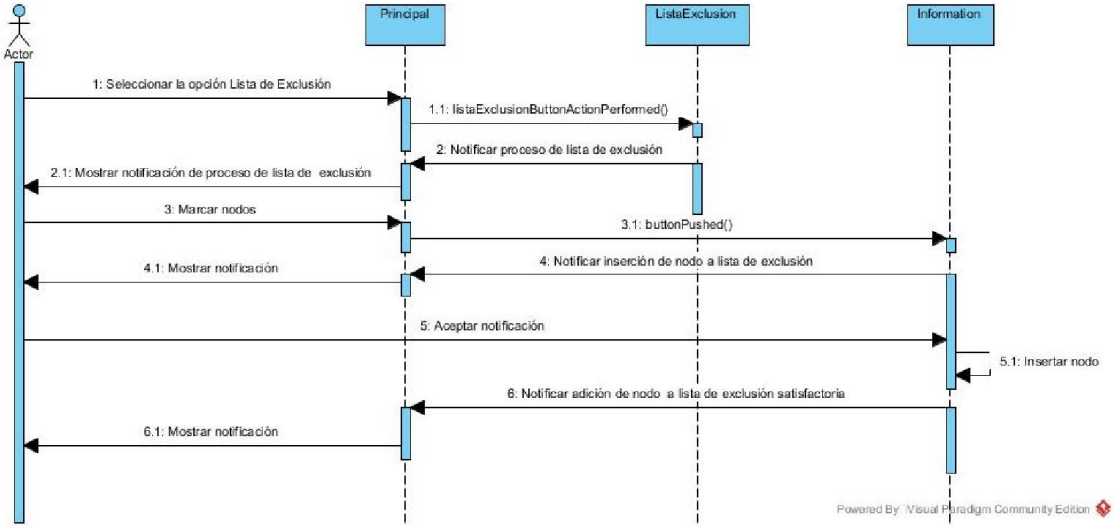


Figura 9: Diagrama de secuencia para el caso de uso Añadir nodos a Lista de Exclusión.

3.5 Patrón de Arquitectura.

El concepto de Arquitectura de Software tiene mucho tiempo de antigüedad, pero no fue hasta la década de los 90 que comenzó a utilizarse de manera formal; del mismo modo, analizando los sistemas se observó que existen patrones que se repiten conformando lo que se conoce como estilos arquitectónicos. (Editor, 2012).

Un patrón de arquitectura de software describe un problema particular y recurrente del diseño, que surge en un contexto específico, y presenta un esquema genérico y probado de su solución. (Editor, 2012).

3.6 Patrón de diseño.

Los patrones resuelven problemas de diseño específicos y hacen el diseño flexible y reusable. Cada patrón describe un problema que ocurre una y otra vez en nuestro entorno y describe también el núcleo de la solución al problema, de forma que puede utilizarse un millón de veces sin tener que hacer dos veces lo mismo.

En este trabajo se aplican los siguientes patrones de diseño:

- Patrón Iterator
- Patrón Singelton

Patrón Iterator: Proporciona acceso secuencial a los elementos de una estructura de datos, sin exponer su representación interna. Los iteradores simplifican la interfaz del contenedor lo que permite hacer más de un recorrido a la vez sobre una misma estructura de datos.

Ejemplo de su uso en el software:

```
public void completarLista() {
    int i = 0;
    Node tempVecinoAnterior =lienzo.getNode(0);
    Node tempInicial=lienzo.getNode(0);
    while (i < lienzo.getNodeCount() - 1) {
        Iterator<? extends Node> nodesArray=myNodes.get(myNodes.size()- 1).getNeighborNodeIterator();
        if (myNodes.size() == 1) {
            myNodes.add(nodesArray.next());
        } else {
            tempVecinoAnterior = myNodes.get(myNodes.size() - 2);
            while (nodesArray.hasNext()) {
                Node node = nodesArray.next();
                if (node != tempVecinoAnterior && node != tempInicial) {
                    myNodes.add(node);

                }
            }
            i++;
        }
    }this.llenarTour();
}
```

Figura 10: Ejemplo de utilización del Patrón Iterator en el software.

Patrón Singelton: Restringe la creación de instancias de una clase para que solo cree una instancia de la clase que lo implementa. Puede ser utilizado de forma global en aplicaciones así que su uso puede ser muy variado, desde acceder a constantes y acumuladores o contadores, hasta la gestión de parámetros de la aplicación.

Ejemplo de uso en el software:

```java
public class Route {

    private ArrayList<Ciudad> cities = new ArrayList<>();
    private static Route instancia;

    private Route(ArrayList<Ciudad> cities) {
        this.cities = cities;
        Collections.shuffle(this.cities);
    }

    public static Route getInstancia(ArrayList<Ciudad> pcit
    {
        if (instancia == null) {
            instancia = new Route(pcities);
            System.out.println("El objeto ha sido creado");
        }
        else {
            System.out.println("Ya existe el objeto");
        }

        return instancia;
    }
```

```java
Route route = Route.getInstancia(auxiliar);
```

Figura 11: Ejemplo de utilización del Patrón Singelton en el software.

3.7 Tratamiento de errores.

El tratamiento de errores es un tema de vital importancia en el desarrollo de una aplicación, para que un producto tenga calidad es indispensable que su funcionamiento sea el correcto sin presentar inconvenientes de ningún tipo, el buen funcionamiento es la primera línea a medir para tener la aceptación de los usuarios, por esto, en esta aplicación se trabajó cuidadosamente este tema en el aspecto de implementación, estas son algunas de las clases y funciones utilizadas para el tratamiento de errores.

Bloques Try…Catch: se utilizan en Java para capturar las excepciones que se hayan podido producir en el bloque de código delimitado por try y catch.

IOException: se utiliza para evitar errores que no puede evitar el programador, generalmente relacionados con la entrada/salida del programa.

Ejemplo:

```
public String leerFichero (String nombreFichero)
                throws IOException
...
```

Figura 12: Ejemplo de utilización IOException en Java.

3.8 Interacciones visuales que se implementaron.

Las interacciones visuales que se implementaron en el software Savis V2.0 son las siguientes:

- ✓ Encontrar una solución inicial eficiente por el algoritmo Hill Climbing Simple en 100 iteraciones para el inicio del algoritmo RS.
- ✓ Eliminar arista en cualquier situación que determine el usuario que es factible cambiar un camino por otro.
- ✓ Dibujar arista para complementar la interacción anterior y cerrar el camino.
- ✓ Contabilizar el tiempo de la corrida del algoritmo RS para tener un balance del tiempo que se utilizó para encontrar una determinada solución.
- ✓ Exportar imagen de la solución final encontrada, ya sea la mejor solución o la peor, como evidencia del resultado obtenido.
- ✓ Añadir nodos a Lista de Exclusión permitiendo restringir el espacio de búsqueda del algoritmo a los nodos no marcados en la lista.
- ✓ Eliminar nodos de la Lista de la Exclusión por si un nodo seleccionado fue marcado sin desearse.
- ✓ Conocer distancia entre dos ciudades, esto es una herramienta útil si existen dos nodos cercanos donde la percepción humana no puede determinar cuál está más cerca de un nodo dado.
- ✓ Hacer zoom sobre todos los lienzos para determinar el espacio de exploración sobre archivos que su representación es tan cercana que no se puede determinar el posicionamiento de los nodos.
- ✓ Ventanas informativas para describir las acciones Eliminar Arista, Dibujar Arista, Añadir nodos a Lista de Exclusión.

3.9 Esquema general del modelo.

El siguiente modelo muestra las iteraciones visuales implementadas para el algoritmo Recocido Simulado en el software Savis v2.0.

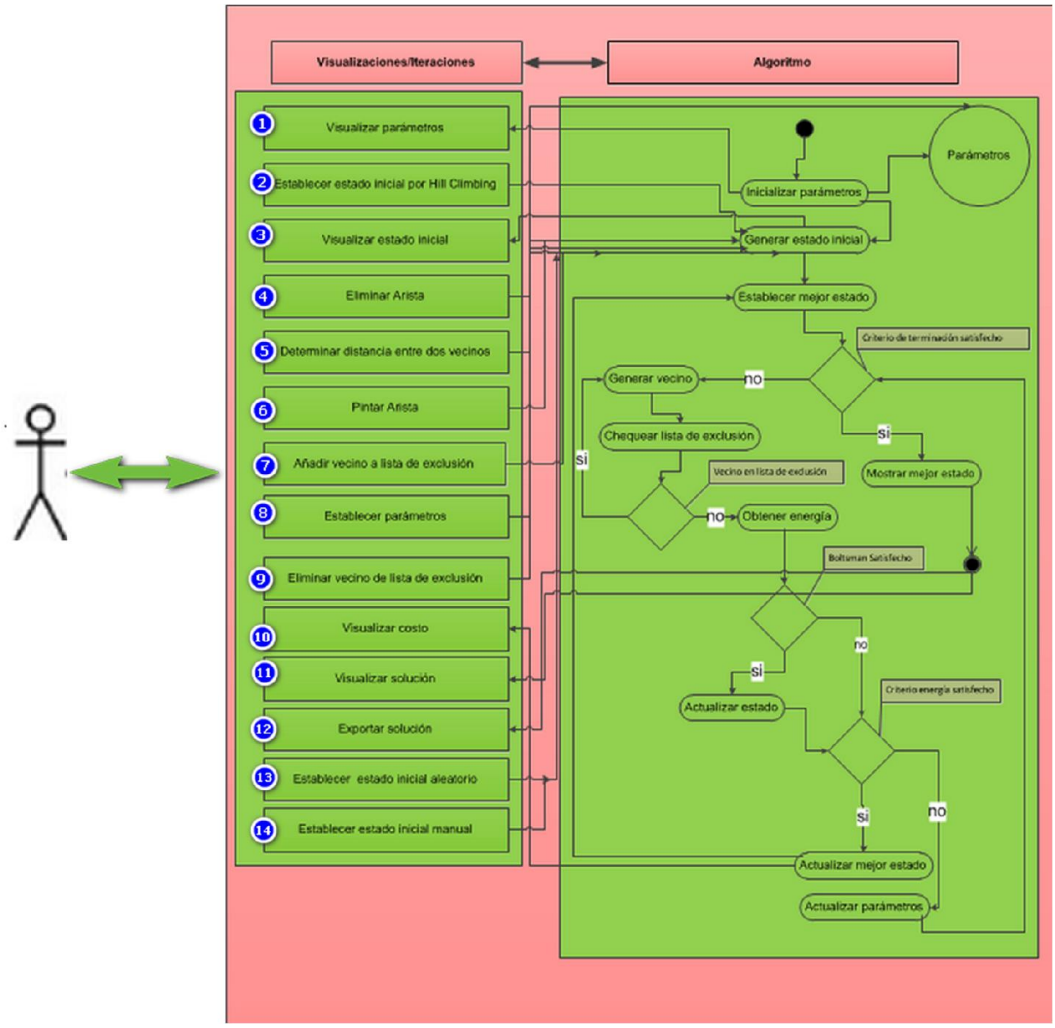


Figura 13: Modelo de interacciones visuales para el algoritmo Recocido Simulado.

3.10 Implementación general del esquema del modelo.

Basado en el modelo visualizado en el epígrafe anterior, se presenta la interfaz principal del software Savis V2.0 implementado con todas las funcionalidades del modelo.

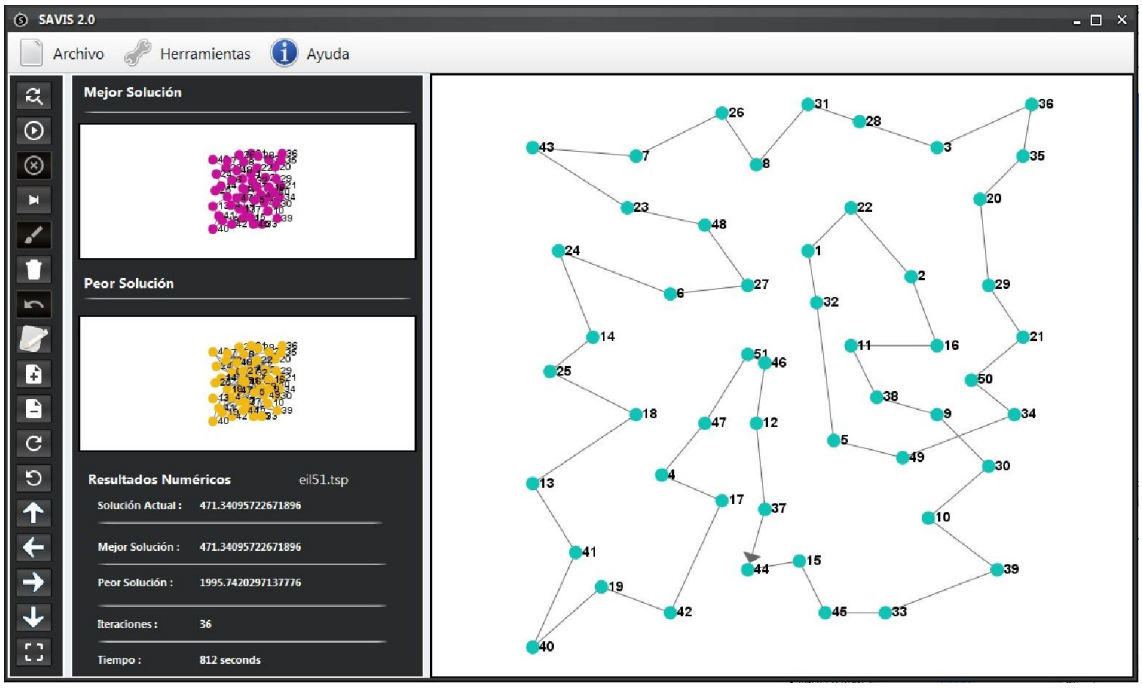


Figura 14: Software Savis v2.0.

3.11 Estudio Experimental.

Para este estudio experimental se utilizaron los siguientes ficheros tsp:

- eil51.
- berlin52.
- st70.
- eil76.
- kroA100.
- ch150.
- a280.

El objetivo es establecer una comparación entre los resultados encontrados ejecutando el software Savis v2.0 sin asistencia del usuario, y con asistencia del usuario a través de las interacciones implementadas para valorar la efectividad de este modelo.

Para la corrida sin asistencia del usuario se tomaron 500 iteraciones sin pausa, con un coeficiente de enfriamiento de 0.003 y una temperatura de 100000, siempre partiendo de una solución inicial aleatoria. Además, se delimitaron 10 ejecuciones con cada fichero para escoger el mejor resultado.

Para la corrida con asistencia del usuario se va a utilizar la variación de los parámetros de enfriamiento y temperatura, el total de iteraciones será 300, se empleará la solución inicial por Hill Climbing y se podrá pausar, continuar eliminar arista, dibujar arista y añadir nodos a la Lista de Exclusión siempre que se determine, el total de ejecuciones será 10 y se seleccionará el mejor resultado encontrado.

Sin asistir por el usuario			Asistido por el usuario		
Fichero	**Mejor Solución**	**Iteraciones**	**Fichero**	**Mejor Solución**	**Iteraciones**
eil51	593	500	eil51	428	300
berlin52	9889	500	berlin52	7544	300
st70	1192	500	st70	678	300
eil76	950	500	eil76	558	300
kroA100	45292	500	kroA100	21420	300
ch150	20055	500	ch150	6545	300
a280	14901	500	a280	3288	300

Figura 15: Comparación experimental.

3.12 Análisis de la comparación.

Para realizar la comparación entre las muestras se estableció un test estadístico, en este caso se utilizó el test no paramétrico de los rangos con signo de Wilcoxon que presenta el software SPSS, el cual evalúa si existen diferencias significativas entre dos muestras pareadas.

Hipótesis:

H0: No hay diferencias entre las observaciones pareadas

H1: Sí hay diferencias entre las observaciones pareadas

Se plantea como **hipótesis** fundamental la búsqueda guiada con asistencia del usuario es más eficiente que la búsqueda sin asistencia.

Resultados:

Prueba de Wilcoxon de los rangos con signo

Rangos

		N	Rango promedio	Suma de rangos
con_asistencia - sin_asistencia	Rangos negativos	7[a]	4,00	28,00
	Rangos positivos	0[b]	,00	,00
	Empates	0[c]		
	Total	7		

a. con_asistencia < sin_asistencia

b. con_asistencia > sin_asistencia

c. con_asistencia = sin_asistencia

Estadísticos de prueba[a]

	con_asistencia - sin_asistencia
Z	-2,366[b]
Sig. asintótica (bilateral)	,018

a. Prueba de Wilcoxon de los rangos consigno.

b. Se basa en rangos positivos.

Figura 16: Resultados arrojados por el test estadístico.

Interpretación: En la tabla titulada “Rangos” vemos que se analizaron 7 pares (los 7 ficheros que se estudiaron). Hubo siete rangos negativos, cero positivos y ningún empate.

En la tabla titulada “Estadísticos de contrastes” se observa la fila Significación asintótica bilateral y su valor de 0,018.

Podemos decir que, como el valor de la Significación asintótica bilateral es menor que 0,05, entonces se rechaza la hipótesis nula y se concluye que hay evidencias suficientes para plantear que la búsqueda asistida por el usuario es efectiva en la reducción del costo con un nivel de significación del 2%.

3.13 Conclusiones parciales.

En este capítulo se analizó la descripción de la propuesta de solución, se definió un nuevo modelo con las interacciones del software Atreeb v1.0 y las nuevas iteraciones implementadas para este trabajo, se obtuvo como producto final el software Savis v2.0, se realizó un experimento comparativo para evaluar la eficiencia de este nuevo modelo de técnicas de visualización y se mostraron los resultados obtenidos a través de un test estadístico.

Capítulo 4: Análisis de factibilidad.

4.1 Introducción.

En este capítulo se abordan los temas relacionados con el análisis de factibilidad, se analiza la planificación del costo del producto desde la perspectiva económica basado en la planificación a través de los casos de uso determinados y del esfuerzo que lleva su desarrollo teniendo en cuenta diferentes factores y variables, también se muestran los escenarios de pruebas del tipo caja negra realizados al software final para evaluar su rendimiento y funcionamiento.

4.2 Planificación basada en casos de uso.

Para determinar el costo de estimación de un proyecto a partir de la planificación basada en casos de uso lo primero es calcular los puntos de casos de uso sin ajustar a través de la siguiente ecuación:

UUCP = UAW + UUCW donde el significado de cada variable es el siguiente:

UUCP: Puntos de Casos de Uso sin ajustar

UAW: Factor de Peso de los Actores sin ajustar

UUCW: Factor de Peso de los Casos de Uso sin ajustar

Factor de Peso de los Actores sin ajustar (UAW).

El criterio para el cálculo es a partir de la siguiente tabla.

Tipo de Actor	Descripción	Factor de Peso	Número de Actores
Simple	Otro sistema que interactúa con el sistema a desarrollar mediante una interfaz de programación (API, Application Programming Interface)	1	0
Medio	Otro sistema que interactúa con el sistema a desarrollar mediante un protocolo o una interfaz basada en texto	2	0
Complejo	Una persona que interactúa con el sistema mediante una interfaz gráfica	3	1

Tabla 7: Criterio para el cálculo del factor de peso de los actores sin ajustar.

Entonces

UAW = $\sum$ (Actor $_i$ * Factor de Peso$_i$)

UAW = 3 * 1

UAW = 3

Factor de Peso de los Casos de Uso sin ajustar (UUCW).

Criterios para el cálculo.

Tipo de Caso de Uso	Descripción	Factor de Peso	Número de Casos de Uso
Simple	El Caso de Uso contiene de 1 a 3 transacciones	5	7
Medio	El Caso de Uso contiene de 4 a 7 transacciones	10	0
Complejo	El Caso de Uso contiene más de 8	15	0

	transacciones		

Tabla 8: Criterio para el cálculo del factor de peso de los casos de uso sin ajustar.

Caso de Uso	Transiciones	Factor de Peso
Pintar Arista	3 transacciones	5
Eliminar Arista	1-3 transacciones	5
Exportar Imagen	1-3 transacciones	5
Contabilizar el tiempo de la corrida	1-3 transacciones	5
Añadir nodos a Lista de Exclusión	1-3 transacciones	5
Conocer distancia entre dos ciudades	1-3 transacciones	5
Hallar solución inicial por Hill Climbing	1-3 transacciones	5

Tabla 9: Total de transiciones por casos de uso.

Entonces

UUCW = $\sum$ (Caso de Uso $_i$* Factor de Peso$_i$)

UUCW = 7 * 5

UUCW = 35

Luego despejando en la ecuacion principal:

UUCP = UAW + UUCW

UUCP = 3+35

UUCP = 38

Cálculo de Puntos de Casos de Uso ajustados.

Se define por la siguiente ecuación:

UCP=UUCP*TCF*EF

Donde:

UCP: Puntos de Casos de Uso ajustado

UUCP: Puntos de Casos de Uso sin ajustar

TCF: Factor de complejidad técnica

EF: Factor de Ambiente

Factor de complejidad técnica (TCF).

Criterios para el cálculo

Factor	Descripción	Peso	Valor Asignado
T1	Sistema distribuido	2	0
T2	Objetivos de performance o tiempo de respuesta	1	5
T3	Eficiencia del usuario final	1	5
T4	Procesamiento interno complejo	1	4
T5	El código debe ser reutilizable	1	3
T6	Facilidad de instalación	0.5	1
T7	Facilidad de uso	0.5	2
T8	Portabilidad	2	2
T9	Facilidad de cambio	1	1
T10	Concurrencia	1	1
T11	Incluye objetivos especiales de seguridad	1	0
T12	Provee acceso directo a terceras partes	1	0
T13	Se requieren facilidades especiales de entrenamiento a usuarios	1	2

Tabla 10: Factores que determinan la complejidad técnica del proyecto.

Donde

Valor asignado es un valor entero de 0 a 5, donde 0 significa un aporte irrelevante y 5 un aporte muy importante.

Entonces se tiene la siguiete ecuación:

TCF = 0.6+0.01*$\sum$ (Peso $_i$ * valor asignado)

TCF = 0.6 + 0.01* (0+5+5+4+3+0.5+1+4+1+1+0+0+2)

TCF = 0.6 + 0.01* 26.5

TCF = 0.6 + 0.27

TCF = 0.162

Factor de ambiente (EF).

Criterios para el cálculo

Factor	Descripción	Peso	Valor Asignado
E1	Familiaridad con el modelo de proyecto utilizado	1.5	1
E2	Experiencia en la aplicación	0.5	1
E3	Experiencia en orientación a objetos	1	4
E4	Capacidad del analista líder	0.5	3
E5	Motivación	1	3
E6	Estabilidad de los requerimientos	2	1
E7	Personal part-time	-1	2
E8	Dificultad del lenguaje de programación	-1	3

Tabla 11: Criterio para el cálculo del factor ambiente del proyecto.

Para los factores E1 al E4, un valor asignado de 0 significa sin experiencia, 3 experiencia media y 5 amplia experiencia (experto).

Para el factor E5, 0 significa sin motivación para el proyecto, 3 motivación media y 5 alta motivación.

Para el factor E6, 0 significa requerimientos extremadamente inestables, 3 estabilidad media y 5 requerimientos estables sin posibilidad de cambios.

Para el factor E7, 0 significa que no hay personal part-time (es decir todos son full-time), 3 significa mitad y mitad, y 5 significa que todo el personal es part-time (nadie es full-time).

Para el factor E8, 0 significa que el lenguaje de programación es fácil de usar, 3 medio y 5 que el lenguaje es extremadamente difícil.

El Factor de ambiente se calcula mediante la siguiente ecuación:

EF = 1.4 - 0.03 x Σ (Peso i x Valor asignado)

EF = 1.4 - 0.03 * (1.5+0.5+4+1.5+3+2-2-3)

EF = 1.4 - 0.03 * 7.5

EF = 1.4 – 0.225

EF = 1.18

Por tanto, calculando los puntos de casos de uso ajustados tenemos.

UCP = UUCP * TCF * EF
UCP = 38 * 0.162 * 1.18
UCP = 7.26
Estimación del esfuerzo (E).

Ecuación

E = UCP * CF

E = 7.26 * 20

E = 145.3

Donde

E: Esfuerzo estimado en horas-hombre

UCP: Puntos de Casos de Uso ajustados

CF: Factor de conversión (20 horas-hombre por defecto)

Se contabilizan cuántos factores de los que afectan al Factor de ambiente están por debajo del valor medio (3), para los factores E1 a E6.

Se contabilizan cuántos factores de los que afectan al Factor de ambiente están por encima del valor medio (3), para los factores E7 y E8.

Si el total es 2 o menos, se utiliza el factor de conversión 20 horas-hombre/Punto de Casos de Uso, es decir, un Punto de Caso de Uso toma 20 horas-hombre. - Si el total es 3 o 4, se utiliza el factor de conversión 28 horas-hombre/Punto de Casos de Uso, es decir, un

Punto de Caso de Uso toma 28 horas-hombre. - Si el total es mayor o igual que 5, se recomienda efectuar cambios en el proyecto, ya que se considera que el riesgo de fracaso del mismo es demasiado alto.

Estimación del esfuerzo del proyecto

Criterio para la estimación del esfuerzo del proyecto completo.

Actividad	Porcentaje
Análisis	10.00%
Diseño	20.00%
Programación	40.00%
Pruebas	15.00%
Sobrecarga(otras actividades)	15.00%

Tabla 12: Distribución genérica del esfuerzo.

Con este criterio y tomando como entrada la estimación de tiempo calculada a partir de los Puntos de Casos de Uso, se pueden calcular las demás estimaciones para obtener la duración total del proyecto.

El esfuerzo calculado es solamente el de la etapa de programación.

Actividad	Porcentaje	Horas / hombre
Análisis	10.00%	36.325
Diseño	20.00%	72.65
Programación	40.00%	145.3
Pruebas	15.00%	54,49

Sobrecarga(otras actividades)	15.00%	54,49
Total	100%	363,25

Tabla 13: Esfuerzo en la etapa de programación.

Cálculo del esfuerzo total (ETotal).

Ecuación:

Etotal = ∑ **actividades**

Donde:

ETotal: esfuerzo total

Etotal = 363,25

Cálculo del tiempo de desarrollo

Ecuación:

TDesarrollo = ETotal / CHTotal / CHTrabajo

TDesarrollo = 363,25 / 1 / 6=60,54h.

Donde:

TDesarrollo: tiempo de desarrollo total en horas

CHTotal: cantidad de hombres

CHTrabajo: cantidad de horas de trabajo diario

Cálculo del costo

Ecuación:

CostoTotal = ETotal * CHTotal * TH

CostoTotal = 363,25 * 1 * 1.031

CostoTotal = 374.5

Donde:

TH: tarifa horaria (suponga $1.031)

4.3 Casos de pruebas.

Las pruebas de Caja Negra permiten obtener un conjunto de condiciones de entrada que ejerciten completamente todos los requisitos funcionales de un programa. En este caso se analizarán los escenarios de pruebas para los casos de uso más significativos determinados anteriormente.

Escenarios a probar en el caso de uso Añadir nodos a Lista de Exclusión.

Nombre de la sección	Escenarios de la sección	Descripción de la funcionalidad	Flujo Central
SC 1: Añadir nodos a Lista de Exclusión	*EC 1.1: Flujo Normal.*	*El caso de uso se inicia cuando el usuario desea simplificar el espacio de búsqueda del algoritmo RS marcando nodos que el algoritmo no puede intercambiar en su proceso iterativo. Como resultado se obtiene un nodo añadido a la Lista de Exclusión.*	1. *El usuario selecciona en el panel de herramientas la opción Lista de Exclusión.* 2. *El sistema muestra una ventana informativa con la descripción del proceso seleccionado.* 3. *El usuario hace doble click sobre un nodo en el lienzo principal.* 4. *El sistema muestra una ventana de confirmación.* 5. *El usuario confirma en el botón aceptar de la ventana.*

	EC 1.2: Flujo Alternativo. *El nodo ya está añadido en la Lista Exclusión.*	*El usuario desea añadir un nodo a la Lista de Exclusión, el cual ya había marcado con anterioridad, el sistema responde mostrando un mensaje de alerta al usuario informándole que el nodo ya existe en la Lista de Exclusión.*	1. *El usuario selecciona en el panel de herramientas la opción Lista de Exclusión.* 2. *El sistema muestra una ventana informativa con la descripción del proceso seleccionado.* 3. *El usuario hace doble click sobre un nodo en el lienzo principal.* 4. *El sistema muestra un mensaje de alerta al usuario indicándole que el nodo ya existe en la Lista de Excusión.*

Tabla 14: Escenarios a probar en el caso de uso Añadir nodos a Lista de Exclusión.

Sesiones a probar en el caso de uso Añadir nodos a Lista de Exclusión.

[La Variable 3, hace referencia al nodo marcado y la Variable 4 referencia al escenario que el nodo no se encuentra en la Lista de Exclusión.]

Id del escenario	Escenario	Variable 3	Variable 4	Resultado del Sistema	Resultado de la Prueba
EC1	*Añadir nodo a Lista de Exclusión.*	*Válido*	*Válido*	*El sistema inserta el nodo en la Lista de Exclusión.*	*Prueba superada exitosamente.*

		Válido	*Inválido*	*El sistema lanza un mensaje de alerta*	*fallo*
		Inválido	*Válido*	-	
		Inválido	*Inválido*	-	

Tabla 15: Sesiones a probar en el caso de uso Añadir nodos a Lista de Exclusión.

Figura 16: Ejemplo de mensaje de alerta mostrado por el software para el caso de prueba Añadir nodos a Lista de Exclusión.

Escenarios a probar en el caso de uso Eliminar Arista.

Nombre de la sección	Escenarios de la sección	Descripción de la funcionalidad	Flujo Central
SC 2: Eliminar Arista	*EC 1.1: Flujo Normal.*	*El caso de uso inicia cuando el usuario desea eliminar una arista para cambiar un camino, como resultado se obtiene una arista eliminada.*	1. *El usuario selecciona en la barra de herramientas la opción Eliminar Arista.* 2. *El sistema muestra una ventana informativa con los pasos a seguir para eliminar una arista.* 3. *El usuario selecciona dos nodos unidos por una arista.*

			4. *El sistema elimina la arista.*
	EC 1.2: Flujo Alternativo. Entre los nodos	*El usuario desea eliminar una arista pero entre los dos nodos seleccionados no existe una arista.*	1. *El usuario selecciona en la barra de herramientas la opción Eliminar Arista.* 2. *El sistema muestra una ventana informativa con los pasos a seguir para eliminar una arista.* 3. *El usuario selecciona dos nodos entre los cuales no existe una arista.* 4. *El sistema muestra un mensaje de alerta informando que entre los nodos seleccionados no existe una arista.*

Tabla 16: Escenarios a probar en el caso de uso Eliminar Arista.

Sesiones a probar en el caso de uso Eliminar Arista.

[La Variable 3, hace referencia a dos nodos seleccionados y la Variable 4 referencia que entre los dos nodos existe una arista.]

Id del escenario	Escenario	Variable 3	Variable 4	Resultado del Sistema	Resultado de la Prueba
EC1	*Eliminar Arista.*	*Válido*	*Válido*	*El sistema elimina la arista.*	*Prueba superada exitosamente.*

		Válido	*Inválido*	*El sistema lanza un mensaje de alerta*	*fallo*
		Inválido	*Válido*		
		Inválido	*Inválido*		

Tabla 17: Sesiones a probar en el caso de uso Eliminar Arista.

Figura 17: Ejemplo de mensaje de alerta mostrado por el software para el caso de prueba Eliminar Arista.

4.4 Conclusiones parciales.

En este capítulo se realizó un análisis de factibilidad para determinar la estimación del costo del proyecto desarrollado, para esto se tuvo en cuenta diferentes variables que muestra el esfuerzo total realizado por el desarrollador, además se probaron algunos escenarios de pruebas para evaluar la respuesta del sistema.

Conclusiones

En el presente trabajo se abordaron los temas referentes al marco teórico, se puntualizaron conceptos y definiciones importantes para entender el comportamiento del producto final, se detalló la ingeniería de software manejada en la confección del mismo, se identificaron las interacciones visuales para guiar la búsqueda del usuario de una forma eficiente, se creó un modelo de integración con las interacciones identificadas y el algoritmo metaheurístico Recocido Simulado, se implementó el modelo creado y como resultado se obtuvo el software Savis v2.0, se evaluó la eficiencia del modelo definido a través de un test estadístico estableciendo comparaciones entre la búsqueda asistida por el usuario y sin asistencia del usuario, el test arrojó resultados positivos que determinan la eficiencia del modelo. Como análisis final se llega a la conclusión de que la aplicación de las nuevas técnicas integradas en el software mejora la eficiencia de este.

Recomendaciones.

Se recomienda:

- La explotación del software en ficheros TSP de más de 500 ciudades.
- Utilizar el software Savis v.2.0 para apoyar el aprendizaje de los estudiantes, en temáticas como la búsqueda heurística y metaheurística en la asignatura Inteligencia Artificial.
- Implementar una funcionalidad para que el marcado de ciudades en la lista de exclusión se pueda realizar de forma masiva.
- Ampliar la búsqueda de soluciones iniciales por algoritmos metaheurísticos de mayor complejidad que permitan encontrar soluciones más eficientes que el algoritmo de búsqueda local Hill Climbing.
- Profundizar en el estudio de nuevas técnicas de visualización para heurísticas reconocidas que han demostrado buenos resultados resolviendo el TSP.

Bibliografía

ADRIÁN BONILLA PETRICOLET, M. G. L. P. Y. C. A. S. B. 2005. APLICACIÓN DEL MÉTODO DE OPTIMIZACIÓN DE RECOCIDO SIMULADO EN LAREGRESIÓN DE ISOTERMAS DE ADSORCIÓN.

ALEJANDRA ORTIZ, C. V. *Diagrama de paquetes* [Online]. Available: prezi.com.

EDITOR 2012. Patrones de arquitectura de software

GARCÍA, S. P. 2015. El problema del viajante. Métodos de resolución y un enfoque hacia la Teoría de la Computación.

JESÚS DEL CARMEN PERALTA ABARCA, J. Y. J. C., BEATRIZ MARTÍNEZ BAHENA 2015. Aplicación de recocido simulado en problemas de optimización combinatoria.

LÓPEZ, I. B. S. 2016. PROBLEMA DEL AGENTE VIAJERO - TSP.

MARIO CÉSAR VÉLEZ, J. A. M. 2007. METAHEURÍSTICOS: UNA ALTERNATIVA PARA LA

SOLUCIÓN DE PROBLEMAS COMBINATORIOS EN ADMINISTRACIÓN DE OPERACIONES.

MORENO, B. P. D. V. 2015. Resolución del Problema del Viajante de Comercio (TSP) y su variante con Ventanas de Tiempo (TSPTW) usando métodos heurísticos de búsqueda local.

PONJUÁN, D. T. 2009. Aproximaciones a la visualización como disciplina científica.

PONTIS, S. 2007. La historia de la esquemática en la visualizacián de datos.

REVECO, C. 2010. Arquitectura, Diseño y Construcción de un Negocio con Apoyo TI.

RIBERA, C. S.-B. Y. M. 2014. Visualización de la información en la democratización de los datos: propuestas desde el periodismo y la narratiVidad.

ROXANA CURINO, S. M., SILVIA CASTRO 2009. Visualización de Grafos.

RUIZ, F. Requisitos. *INGENIERÍA DEL SOFTWARE I,* Tema 3.

SHANAHAN, E. 2017. Un repaso del tiempo: la historia de la visualización de datos.

TEAM, G. 2010. *Getting Started Some words about GraphStream* [Online]. Available: http://graphstream-project.org/.

VARGAS, O. A. L. 2012. El problema del vendedor viajero en grafos cúbicos.

Anexos

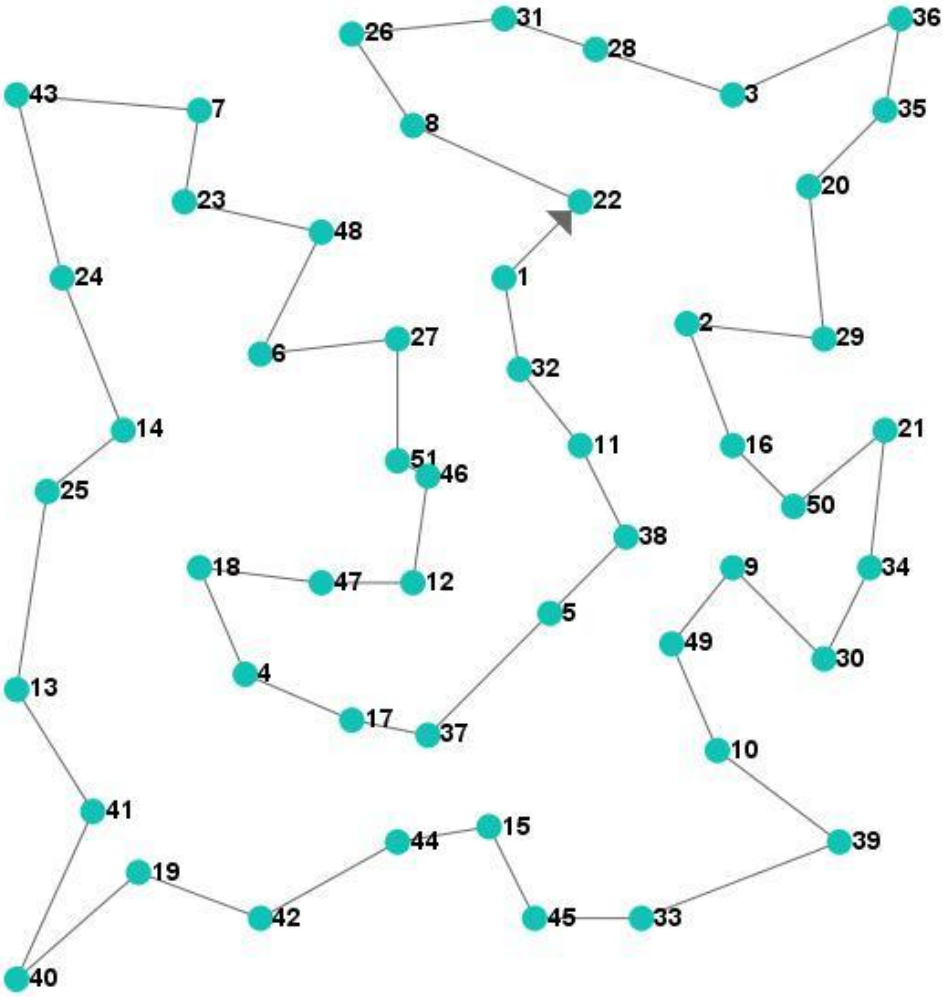


Figura 18: Mejor Solución encontrada por el software Savis v2.0 para el fichero eil51.tsp (428).

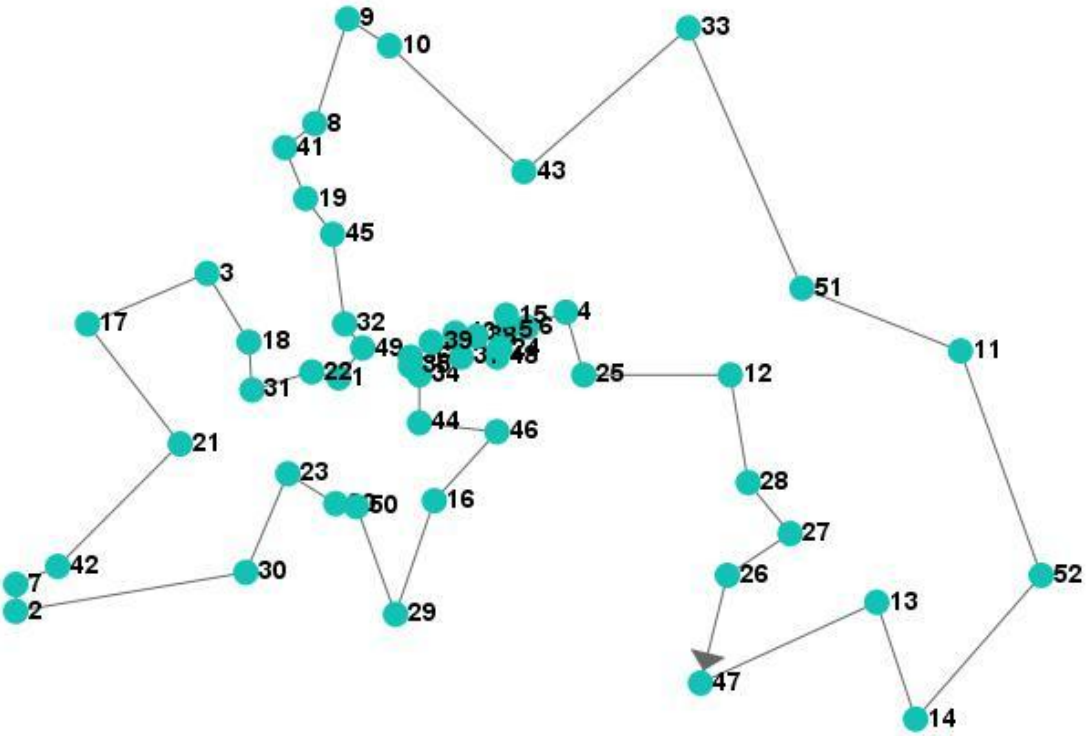


Figura 19: Mejor Solución encontrada por el software Savis v2.0 para el fichero berlin52.tsp (7544).

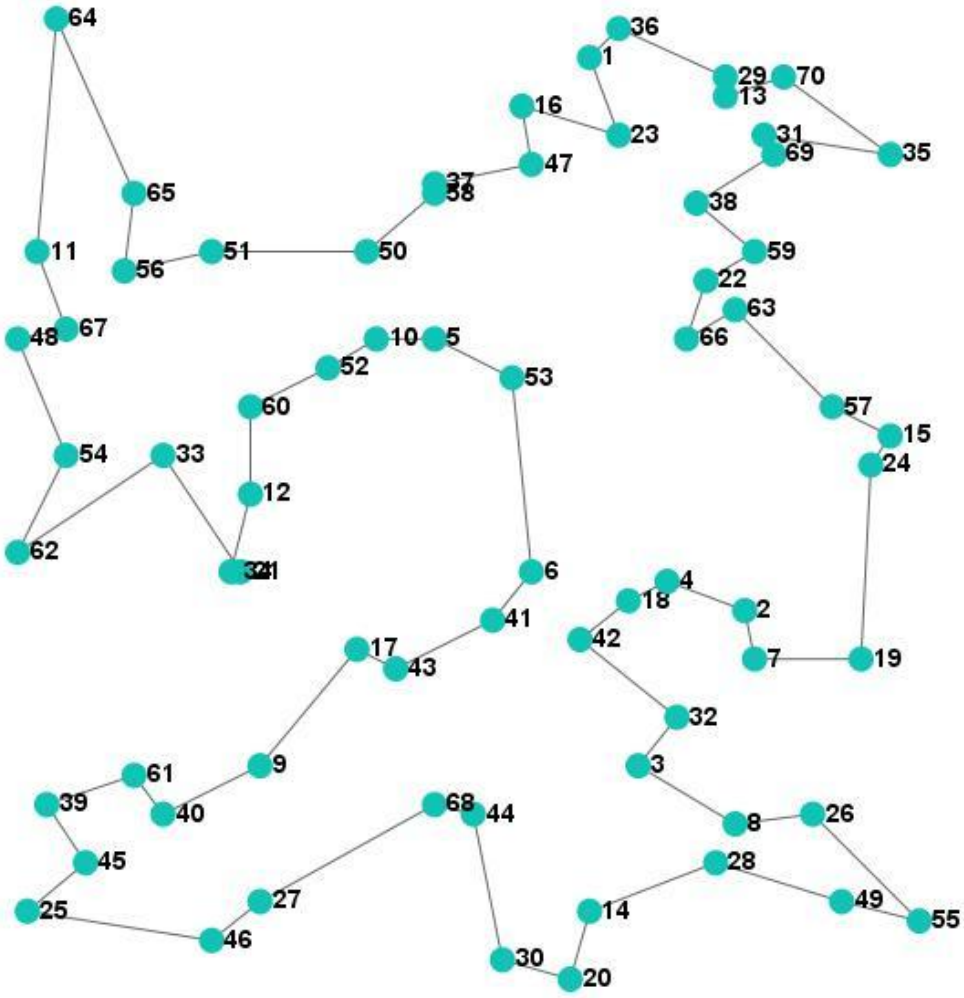


Figura 20: Mejor Solución encontrada por el software Savis v2.0 para el fichero st70.tsp (678).

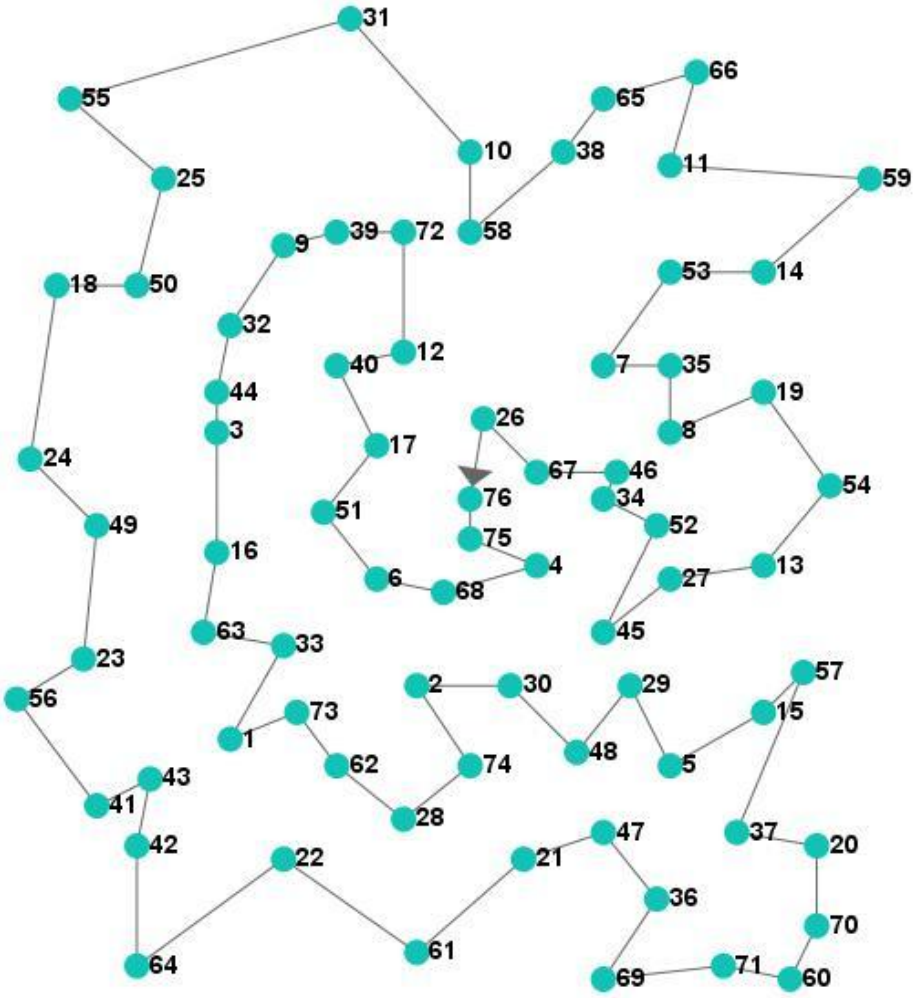


Figura 21: Mejor Solución encontrada por el software Savis v2.0 para el fichero eil76.tsp (558).

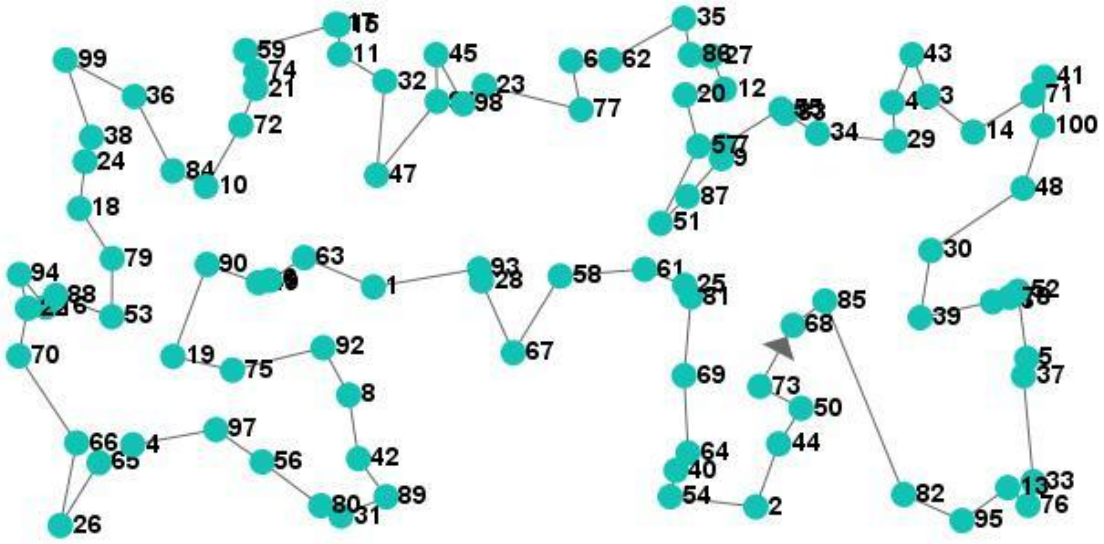


Figura 22: Mejor Solución encontrada por el software Savis v2.0 para el fichero kroA100.tsp (21420).

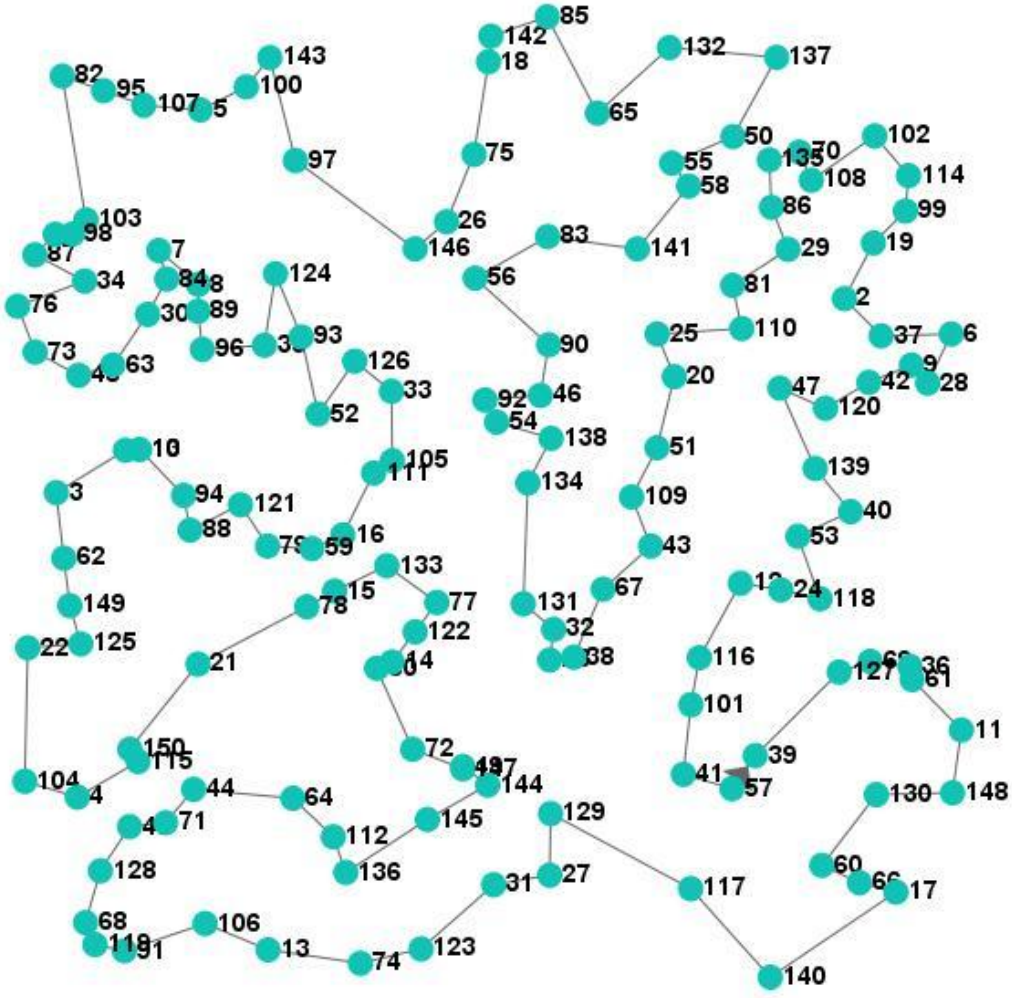


Figura 23: Mejor Solución encontrada por el software Savis v2.0 para el fichero ch150.tsp (6545).

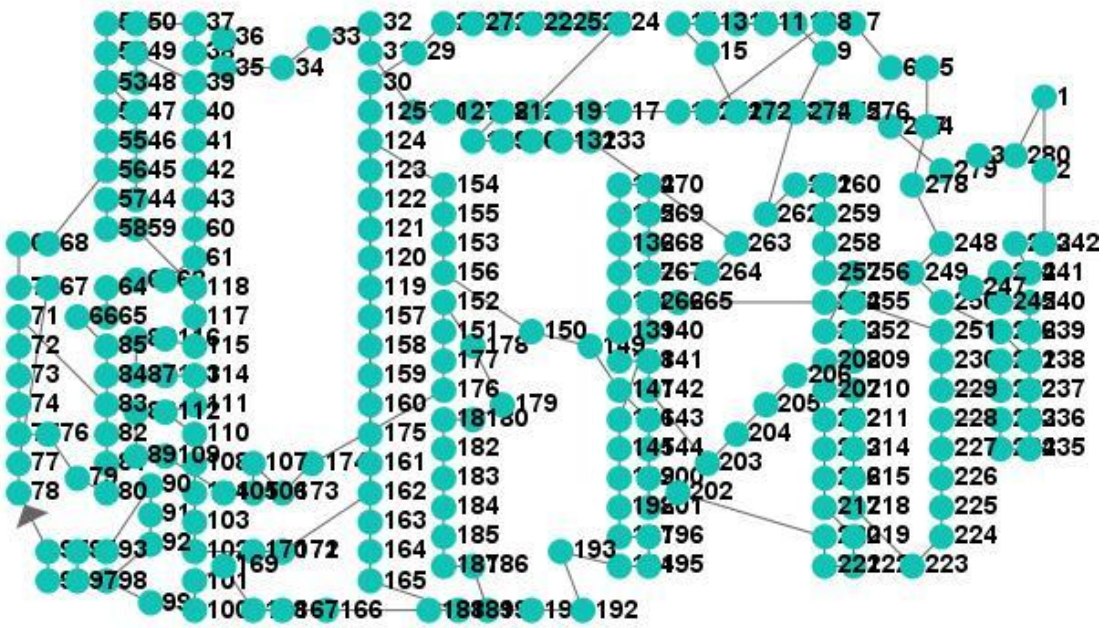


Figura 24 : Mejor Solución encontrada por el software Savis v2.0 para el fichero a280.tsp (3288).