Universidad Central “Marta Abreu” de Las Villas

Facultad Matemática, Física y Computación

INGENIERÍA INFORMÁTICA

TRABAJO DE DIPLOMA

Integración de técnicas de Visualización con el algoritmo de Búsqueda metaheurística Recocido Simulado

Autor:

Juan Daniel Pedraza Díaz

Tutores:

Ms.C. Andy Morfa Hernández

Cuba, 2017

Hago constar que el presente trabajo de diploma fue realizado en la Universidad Central “Marta Abreu” de Las Villas como parte de la culminación de estudios de la especialidad de Ingeniería en Informática, autorizando a que el mismo sea utilizado por la Institución, para los fines que estime conveniente, tanto de forma parcial como total y que además no podrá ser presentado en eventos, ni publicados sin autorización de la Universidad.

_____

Firma del Autor

Los abajo firmantes certificamos que el presente trabajo ha sido realizado según acuerdo de la dirección de nuestro centro y el mismo cumple con los requisitos que debe tener un trabajo de esta envergadura referido a la temática señalada.

_____ _____

Firma del Autor

Firma del Jefe de Departamento donde se defiende el trabajo

_____

Firma del Responsable de Información Científico-Técnica

Agradecimientos

- A mi madre por estar siempre conmigo, brindándome su apoyo incondicional y su amor, inspirándome confianza y mostrándome que el sacrificio suele ser la clave del éxito.
- A mi padre Nelson Díaz por apoyarme y enseñarme la importancia de sembrar para poder recoger.
- A mis abuelos en especial a Domingo, que Dios lo tenga en la gloria.
- A mi novia Clau por ser tan paciente y comprensiva, apoyándome noche y día, en las buenas y las malas, brindándome su amor y su esencia como persona. Gracias por regalarme tu presencia.

  A sus padres por apoyarme siempre y acogerme como un hijo más.
- A mis hermanos por guiarme, y demostrarme que todo tiene su momento, gracias por ser mis hermanos (Armando, Yoan y Mary) los amo.
- A mi primo Emilio Alejandro por apoyarme sin esperar nada a cambio, y enseñarme que no podemos ahogarnos en un vaso de agua, siempre hay tiempo.
- A tía Caridad, siempre me ha dicho que uno es quien quiere ser.
- A mi familia por el mero hecho de ser FAMILIA y estar siempre conmigo.
- A mi tutor Andy Morfa por creer en mí y ser parte de mi formación como estudiante, demostrándome que si se puede, solo hay que proponérselo.
- A mi mejor amiga Maura Elena por estar siempre y ayudarme a crecer como persona, es la persona más valiente que conozco.
- Gracias a Realce y su familia por apoyarme.
- A Francisco por su apoyo incondicional.
- A todas mis amistades, sin Uds. sería imposible salir adelante, gracias a Yailen y Jennifer por apoyarme sin poner objeciones.
- A Dinora por confiar siempre en mí y brindarme su apoyo.
- A aquellos que han contribuido en mi formación
- A todos los que han creído en mí, Gracias.

- A todos mis compañeros de estudio Gracias, no escogería ningún otro grupo.

Resumen

El algoritmo de búsqueda metaheurística Recocido Simulado unido a las técnicas de visualización es una de las variantes potentes para solucionar problemas de optimización. En la presente investigación se propone un modelo de integración de las técnicas de visualización con el algoritmo Recocido Simulado; esquema que refleja notoriamente las interacciones entre el usuario y el algoritmo en tiempo de ejecución. Además, acorde al modelo se implementa un software para resolver el problema del Viajero Vendedor, a través del que se realizó un análisis, que demostró la eficiencia del modelo propuesto.

Abstract

The Simulated Annealing metaheuristic search algorithm coupled with visualization techniques is one of the powerful variants to solve optimization problems. In the present research we propose a model of integration of visualization techiques with the Simulated Annealing algorithm; scheme that notably reflects the interactions among the user and the execution time algorithm. In addition, according to the model, a software is implemented to solve the Travelling Salesman Problem, through which an analysis was effectuate, which demonstrated the efficiency of the proposed model.

Tabla de Contenidos

Introducción

En matemáticas, estadísticas, ciencias empíricas, ciencia de la computación, o ciencia de la administración el término de optimización matemática (optimización o programación matemática) se refiere al proceso de selección del mejor elemento de un conjunto de elementos que se encuentren disponibles, teniendo en cuenta algún criterio de elección.

En el caso más simple, un problema de optimización consiste en maximizar o minimizar (dependiendo del contexto) una función real, eligiendo sistemáticamente valores de entrada (tomados de un conjunto permitido) y computando el valor de la función. De forma general, la optimización enfoca sus esfuerzos, al descubrimiento de los "mejores valores" de alguna función objetivo dado un dominio definido, incluyendo una variedad de diferentes tipos de funciones objetivo y diferentes tipos de dominios.

Para resolver estos problemas, los investigadores se auxilian de algoritmos que terminen en un número finito de pasos, o métodos iterativos que convergen a una solución (en alguna clase específica de problemas), o heurísticas que pueden proveer soluciones aproximadas a algunos problemas (aunque sus iteraciones no convergen necesariamente). Ejemplo de lo antes expuesto, se encuentran los algoritmos combinatorios que para la investigación poseen gran valor.

Los algoritmos de optimización combinatoria, por lo general, solucionan instancias de aquellos problemas que se entienden como difíciles de resolver, para ello exploran el espacio de soluciones (usualmente grande) de estas instancias. Estos, logran reducir considerablemente y de modo efectivo el tamaño del espacio de soluciones, así como, explorar el espacio de búsqueda eficientemente.

La variedad de estos problemas que pertenecen a la clase NP-Completos, tientan al interés científico y tecnológico, y justifican el creciente desarrollo de intentos de soluciones. En pugna a las dificultades existentes para resolverlos, se han

diseñado diversos métodos aproximados que se basan en algoritmos metaheurísticos, los cuales encuentran buenas soluciones en tiempos razonables.

El algoritmo de Recocido Simulado (cristalización simulada o enfriamiento simulado) es uno de los métodos que se proponen para hacerle frente a tales problemas. Como es característico en los algoritmos metaheurísticos, el recocido simulado se basa en un proceso físico, proceso conocido como recocido del acero y cerámicas, de allí su nombre e inspiración; una práctica que consiste en calentar y luego enfriar lentamente el material para variar sus propiedades físicas. Puesto que la temperatura es elevada considerablemente, el calor propicia que los átomos aumenten su energía, haciendo posible desplazarlos de sus posiciones iniciales (un mínimo local de energía); el enfriamiento lento, aumenta las probabilidades de que los átomos logren recristalizar en configuraciones con menor energía que la inicial (mínimo global).

Desde que se muestra dicho algoritmo, es notable la efectividad del mismo frente a problemas de optimización combinatoria. Al analizarlo detalladamente, resalta su capacidad para encontrar soluciones factibles en un marco de tiempo relativamente corto, pero no quedan desapercibidas algunas lagunas procedimentales implícitas en el algoritmo, que, en dependencia del problema en cuestión, pudiesen provocar grandes tiempos computaciones antes de llegar a una solución buena y tal vez óptima.

Para pulir estas dificultades y optimizar al máximo el algoritmo, se puede tener en cuenta la interacción del usuario con el procedimiento en cuestión, dándole facultades al mismo para modificar visualmente las posibles fallas del código; entonces se hace necesario mantener visible la trazabilidad del código, desde el inicio de los parámetros, hasta una supuesta solución. Es importante poseer acceso a la información e interactuar con el algoritmo, dando por sentado que esto sería una variante poderosa para mejorar su eficiencia y eficacia.

De lo antes expuesto se deriva el **Problema de Investigación** que se debe resolver:

¿Cuáles son los mecanismos adecuados que permitan interactuar en tiempo de ejecución con el algoritmo de búsqueda metaheurística Recocido Simulado?

Al observar la problemática definida anteriormente y las preguntas de investigación podemos arribar al siguiente **objetivo general:**

Valorar la efectividad de la integración de Técnicas de Visualización en el algoritmo de búsqueda metaheurística Recocido Simulado, teniendo en cuenta la interacción en tiempo de ejecución del usuario con el algoritmo, para mejorar su eficiencia.

Objetivos Específicos:

1. Identificar los tipos de interacciones que el usuario puede realizar con la visualización del procedimiento del algoritmo de búsqueda metaheurística Recocido Simulado.
2. Crear un modelo de integración de técnicas de visualización para el algoritmo de búsqueda metaheurística Recocido Simulado.
3. Implementar un software que refleje el modelo creado.
4. Evaluar los beneficios del modelo creado, teniendo en cuenta las comparaciones experimentales.

Se formularon además varias **Preguntas de Investigación:**

1. ¿Qué interacciones pueden implementarse en el algoritmo de búsqueda metaheurística Recocido Simulado que permitan al usuario guiar la búsqueda de la solución?
2. ¿Cómo podrían integrarse técnicas de visualización en el algoritmo de búsqueda metaheurística Recocido Simulado?

3. ¿Qué beneficios se podrían obtener de la integración de las técnicas de visualización en algoritmos de optimización?

Justificación de la investigación:

La presente investigación, tiene como centro de estudio 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.

En la práctica se encuentran infinidades de problemas que son ramificaciones de los NP-Completos, los mismos, son muchas veces casos de estudio de la vida real que hasta el momento no se ha demostrado cuál es la vía más eficiente para obtener al menos una solución buena y en ocasiones óptima; por tanto, es necesario encontrar la forma más eficiente posible de resorberlos. Ante la realidad de que una imagen es mucho más intuitiva que una descripción escrita, surge la idea a partir de un trabajo de diploma desarrollado en la Universidad de las Villas en el curso 2007-2008 por Evelyn Menéndez Alonso, en el que la autora, implementó un prototipo de software para resolver el problema del Viajero Vendedor con el algoritmo Sistema Colonia de Hormigas, de confeccionar una aplicación de escritorio utilizando como tecnología Java, mediante la cual se facilite la comprensión del algoritmo de búsqueda metaheurística Recocido Simulado, para ello se pretende visualizar este procedimiento en la aplicación de escritorio, dándole facilidades al usuario de interactuar con el algoritmo en tiempo de ejecución.

Estructura del documento:

El trabajo de diploma cuenta con cuatro capítulos:

El primer capítulo se nombra “Fundamentación Teórica”, tiene como objetivo principal realizar un análisis crítico y evidenciar los elementos teóricos de la investigación, así como, plantear el objeto de estudio.

El segundo capítulo se titula “Modelo del negocio y requisitos”, contiene aquellos elementos que son típicos de nuestra solución, es decir, los elementos que identifican la solución. Se plantean las reglas del negocio y se exponen los requisitos funcionales y no funcionales identificados. Además, se describen los casos de usos más significativos.

El tercer capítulo se titula “Descripción de la propuesta de solución. En este capítulo se propone un modelo que integra las técnicas de visualización con el algoritmo Recocido Simulado, se describe la arquitectura del sistema, el diagrama de clases de diseño, los principios de diseños utilizados y se mencionan las clases mediante las que se realizó el tratamiento de errores del software.

El cuarto capítulo y último se titula “Pruebas y análisis de factibilidad”, en él se presenta el análisis de costo del sistema y se realiza una descripción de las pruebas de software realizadas.

Capítulo 1. Fundamentación Teórica

1.1 Introducción

Este capítulo ofrece elementos teóricos del algoritmo de búsqueda metaheurístico Recocido Simulado, así como, generalidades del término de visualización, haciéndose énfasis en la visualización de grafos y el problema del Viajero Vendedor. Para ello, se tienen en cuenta algunos de los conceptos a los que diferentes investigadores han llegado como resultado de sus investigaciones científicas.

1.2 Aplicación del Recocido Simulado en problemas de optimización combinatoria

No son pocos los avances que se han realizado en la ciencia teniendo como objeto de estudio la vida cotidiana. Desde inicios de la tecnología computacional, los científicos intentan resolver aquellos problemas del diario, para algunos se han encontrados soluciones relativamente fáciles, sin embargo, otros han hecho que cada día los criterios de sus soluciones difieran, y en muchos casos, no se han resuelto. Como ejemplo de estos problemas, se encuentran: 1) la creación de un plan de mínimo coste, que tenga como objetivo, repartir mercancías a diferentes clientes; 2) la asignación óptima de trabajadores a un conjunto de tareas; 3) encontrar dentro de una cadena o industria de producción una secuencia óptima de trabajos; 4) encontrar una distribución óptima de tripulaciones de aviones; 5) encontrar la configuración óptima una red de telecomunicaciones; 6) la creación de un calendario de exámenes que minimice la probabilidad de solapamientos, entre otros(Pilar Moreno Díaz and Manso 2007).

Es necesario, que cuando se esté en presencia de estos problemas, se sea capaz de formular el modelo matemático que los resuelva, de manera que este cumpla con todas las restricciones del problema, para así maximizar o minimizar sus

beneficios o gastos, dependiendo del caso que sea, entonces se estará hablando de optimizar.

En el lenguaje coloquial, optimizar significa más que mejorar. En el contexto científico, la optimización constituye un proceso que trata de encontrar la mejor solución posible para un determinado problema. El procedimiento que lleva a cabo la búsqueda es denominado programa(Schweickardt 2010).

Al analizar un programa de optimización se pueden apreciar tres características fundamentales:

1. El programa posee un conjunto de posibles soluciones o como también se conoce, conjunto de soluciones factibles, al que es común referirse por el nombre de espacio de búsqueda.
2. El programa cuenta con una estrategia de búsqueda, la cual pude ser expresada en términos generales o particulares, dependiendo de cómo fue formulada esta estrategia, se tendrá un algoritmo o se trabajará con un principio, el cual dependerá del contexto del problema abordado.
3. El programa tiene un criterio de decisión, como su nombre lo indica, será el criterio para excluir del espacio de búsqueda aquellas soluciones que sean peores, la intención es determinar cuáles son las mejores soluciones.

El problema de optimización puede formularse como “encontrar el valor de ciertas variables de decisión para los que una determinada función objetivo (problema mono-objetivo) o un conjunto de funciones objetivo (problema multi-objetivo) alcanza su valor máximo o mínimo, según se pretenda” (Schweickardt 2010).

Los problemas a los que se hace mención a inicios del capítulo son considerados de optimización combinatoria, ya que todos cumplen con las siguientes características: 1) existe un conjunto de objetos (clientes, exámenes, tripulaciones, etc.) que se deberán situar en diferentes posiciones; 2) Existen determinados lugares en los que se deben colocar dichos objetos. Cada colocación de objetos

en un lugar determinado se denomina configuración (Schweickardt 2010). Ejemplo de estos problemas, se encuentra el problema del viajero vendedor, tratado en próximas secciones.

La optimización combinatoria es una rama de la optimización en matemáticas aplicadas y en ciencias de la computación, relacionada a la investigación de operaciones, teoría de algoritmos y teoría de la complejidad computacional (Jesús del Carmen Peralta Abarca 2011).

Su objetivo principal, consiste en buscar una configuración que posibilite la solución de un problema determinado, y maximizar o minimizar sus ganancias o gastos. En la mayoría de los casos y debido a las diferencias existentes entre estos problemas, no se permite el uso de un mismo esquema de solución; por lo cual se han diseñado varios métodos para hacerles frente (Jesús del Carmen Peralta Abarca 2011).

1.3 Problemas y Complejidad

Cuando se trata de resolver un problema, existen determinados patrones que indican que en su mayoría no presentan el mismo grado de dificultad. Es válido que se plantee la pregunta ¿Cómo se puede decidir si se está en presencia de un problema fácil o difícil? O mejor aún, ¿Qué significa que un problema sea fácil o difícil? Una rama de las matemáticas se encuentra estrechamente relacionada con las preguntas anteriores, la cual se denomina *complejidad algorítmica*. "La complejidad algorítmica establece una clasificación de los distintos tipos de problemas por su grado de dificultad de acuerdo con la complejidad computacional del algoritmo más sencillo que permite asegurar su resolución" (Pilar Moreno Díaz and Manso 2007).

Los problemas se pueden clasificar en dos grandes conjuntos: los tratables y los intratables. Los problemas intratables incluyen los que son formalmente indecidibles, para los que existe una demostración de que no existe ningún

algoritmo que permita resolverlos en todos los casos. En estos problemas también son incluidos aquellos que, si cuentan con un algoritmo para resolverlos, pero que aún con tamaños razonables, el costo computacional de hallar sus soluciones los convierte en inabordables, y ello es independientemente de la capacidad computacional con la que se cuente. Se pude afirmar que para un problema intratable no existe ningún algoritmo que permita resolverlo en un número de pasos que sea una función polinomial del tamaño de entrada del problema (Pilar Moreno Díaz and Manso 2007).

Un problema tratable, un problema de la clase P, es un problema que se puede resolver siempre utilizando un algoritmo que conlleva un número de pasos que es función polinomial del tamaño de entrada del problema. Los problemas de la clase P se pueden resolver en tiempo polinomial y los intratables no se pueden resolver en tiempo polinomial (Pilar Moreno Díaz and Manso 2007).

Se puede establecer una clasificación adicional para aquellos problemas decidibles, pero intratables. Estos problemas junto con los de la clase P forman la clase NP. Los problemas de la clase NP son los que se pueden resolver utilizando una máquina imaginaria llamada máquina de Turing no determinista (NDTM, Non-Deterministic Turing Machine), en un número de pasos polinomial (Pilar Moreno Díaz and Manso 2007).

De todos los problemas de la clase NP se puede distinguir un conjunto de ellos denominado NP-completos, que son los más difíciles de resolver. El Teorema de Cook nos permite determinar si un determinado problema NP pertenece a la clase NP-completo. La propiedad de la clase NP-completo es que todo problema de la clase NP se puede transformar polinomialmente en él (Pilar Moreno Díaz and Manso 2007).

Los problemas de optimización no se encuentran, en general, en la clase NP y, por tanto, puede que no sean NP-completos. Es así porque, en general, no es posible comprobar si se ha conseguido una solución óptima en un número de

pasos que sea función polinomial del tamaño del problema. En la mayoría de los casos sólo es posible comprobarlo comparándola con todo el conjunto de soluciones del problema. Si el conjunto de soluciones crece exponencialmente con el tamaño del problema resulta evidente que la comprobación no se puede llevar a cabo en tiempo polinomial. Estos problemas de optimización se encuentran en una clase de problemas que se denominan NP-duros (Pilar Moreno Díaz and Manso 2007).

Por lo anterior se puede afirmar, que para los problemas de optimización NP-duros no existe ningún algoritmo en tiempo polinomial que permita determinar la solución óptima al problema, por ello en los últimos años ha ocurrido un incremento en el desarrollo y uso de métodos aproximados que han sido confeccionados mediante procedimientos heurísticos, con el fin de resolver estos problemas. Este incremento, es posible justificarlo debido a la necesidad actual de poder contar con herramientas, además de disponer de ellas para ofrecer soluciones rápidas a problemas reales. A estos procedimientos se les conoce comúnmente como técnicas heurísticas y metaheurísticas.

1.4 Técnicas heurísticas y metaheurísticas

La heurística “es un procedimiento simple, a menudo basado en el sentido común, que se supone ofrecerá una buena solución (aunque no necesariamente la óptima) a problemas difíciles, de un modo fácil y rápido” (Jesús del Carmen Peralta Abarca 2011).

En la actualidad existen diversos tipos de metaheurísticas. Una visión general de ellas se presenta a continuación:

Metaheurísticas de relajación: son procedimientos que relajan las condiciones del problema original, es decir, modifican el problema de forma que consiguen que el problema original sea más fácil de resolver (Pilar Moreno Díaz and Manso 2007).

• Metaheurísticas constructivas: se orientan a procedimientos que obtienen una solución al problema a partir del análisis y selección paulatina de las componentes que la forman. Los algoritmos más utilizados son los algoritmos voraces (greedy algorithms) que construyen soluciones buscando lo mejor de partes de la misma sin buscar óptimos globales (Pilar Moreno Díaz and Manso 2007).

• Metaheurísticas evolutivas: tratan el problema utilizando conjuntos de soluciones que evolucionan sobre el espacio de soluciones. Entre ellas se encuentran los algoritmos genéticos, meméticos, estimación de distribuciones, reencadenamiento de caminos, colonias de hormigas, etc. (Pilar Moreno Díaz and Manso 2007).

• Metaheurísticas de búsqueda: utilizan procedimientos mediante transformaciones o movimientos para recorrer el espacio de soluciones y explotar las estructuras de entornos asociadas. Entre ellas se encuentran los algoritmos de Hillclimbing, de arranque múltiple, Simulated Annealing, búsqueda tabú, etc. (Pilar Moreno Díaz and Manso 2007). Para esta investigación se utiliza este tipo de metaheurística, haciendo énfasis en Simulated Annealing.

Se ha de tener en cuenta, que existen situaciones en las que se necesitan aplicar más de una metaheurística para conformar la solución de un problema determinado. En estos casos, por lo general, se utiliza una metaheurística para hallar una solución factible y a continuación utilizamos otra para intentar mejorar la solución. Aun en estos escenarios no se garantiza que se encuentre un óptimo, pero si se logran encontrar mejores soluciones que haciendo uso de una sola metaheurística.

1.5 Algoritmo de búsqueda metaheurística Recocido Simulado

El algoritmo de búsqueda metaheurística Recocido Simulado aparece a principios de los años 80 (Kirkpatrick, Gelatt y Vecchi 1983). Este tipo de algoritmo es uno de los tipos de modelos de resolución existentes para aquellos problemas conocidos como combinatorios con mínimos locales. A grandes rasgos, el procedimiento genera de manera aleatoria una solución, intentando que esta sea lo más cercana

a la solución actual o que este situada en el entorno de la solución, entonces se evalúa en una función de coste, en caso de que la función de coste no se minimice (solución peor) se determina si se mantiene la solución actual, logrando así, que la solución encontrada no esté en un mínimo local como suele suceder en los algoritmos de Hillclimbing. La probabilidad de aceptar un estado de solución se reduce con el número de iteraciones y está estrechamente relacionada con el grado en que empeore el coste.

Simulated annealing (SA) (Su término en inglés) (*recocido simulado*, *cristalización simulada* o enfriamiento simulado) se define como un método de búsqueda por entornos, caracterizado por un criterio de aceptación de *soluciones vecinas* que se adapta a lo largo de su ejecución (Jesús del Carmen Peralta Abarca 2011)(es una adaptación del algoritmo Metrópolis-Hastings, un método de Montecarlo utilizado para generar muestras de estados de un sistema termodinámico). El Recocido Simulado intenta acercarse cada vez más al valor óptimo de una función en un espacio de búsqueda grande, al cual se le denomina "óptimo global". Fue creado para minimizar funciones de costo, pero existen contextos en los que este algoritmo se aplica en problemas de maximización.

El nombre e inspiración viene del proceso de recocido del acero y cerámicas, una técnica que consiste en calentar y luego enfriar lentamente el material para variar sus propiedades físicas. El calor causa que los átomos aumenten su energía y que puedan así desplazarse de sus posiciones iniciales (un mínimo local de energía); el enfriamiento lento les da mayores probabilidades de recristalizar en configuraciones con menor energía que la inicial (mínimo global). La probabilidad de movimiento de una molécula de su estado viene dada por la función de Boltzmann $P(\Delta E) = e^{-\Delta E/kT}$, que depende de la temperatura, de la diferencia de energía y de una constante K (la constante de Boltzmann) (Pilar Moreno Díaz and Manso 2007).

Respecto a otros algoritmos de optimización global, los algoritmos de enfriamiento lento simulado poseen ventajas notables. Ejemplos de estas ventajas se citan a continuación:

- Lo relativamente sencillo que resulta implementar este tipo de problemas.
- Su aplicabilidad a la mayoría de los problemas de optimización con una estructura combinatoria.
- Su capacidad para ofrecer soluciones razonablemente buenas a la mayoría de los problemas, aunque hay una cierta dependencia de la planificación del enfriamiento y los movimientos que se realicen.
- La facilidad con la que se puede combinar este tipo de algoritmos con otras técnicas heurísticas como los sistemas expertos, los algoritmos genéticos, las redes neuronales, etc., consiguiendo sistemas híbridos que pueden resultar de gran potencia en la resolución de problemas muy complejos (Pilar Moreno Díaz and Manso 2007).

Como es de suponer, existen características de estos algoritmos que pueden limitar su uso, por ejemplo:

- Se necesita elegir con mucho cuidado los movimientos que se realizan, así como los parámetros que se van a utilizar para tratarlo, como por ejemplo la tasa de enfriamiento.
- Una ejecución del problema puede requerir mucho tiempo de cálculo.
- Puede que sea necesario realizar muchas ejecuciones para encontrar una solución satisfactoria.
- Dependiendo de los parámetros elegidos, las soluciones que se van encontrando pueden ser poco estables, “saltando” mucho de unas a otras sin encontrar una solución buena con la rapidez suficiente lo que obliga a retocar los parámetros con las distintas ejecuciones (Pilar Moreno Díaz and Manso 2007).

1.6 Pseudocódigo del Recocido Simulado

Debido a que el seudocódigo es una descripción desde el punto de vista funcional de un algoritmo, es importante analizarlo. El seudocódigo del algoritmo del Recocido Simulado puede ser observado en la Figura1:

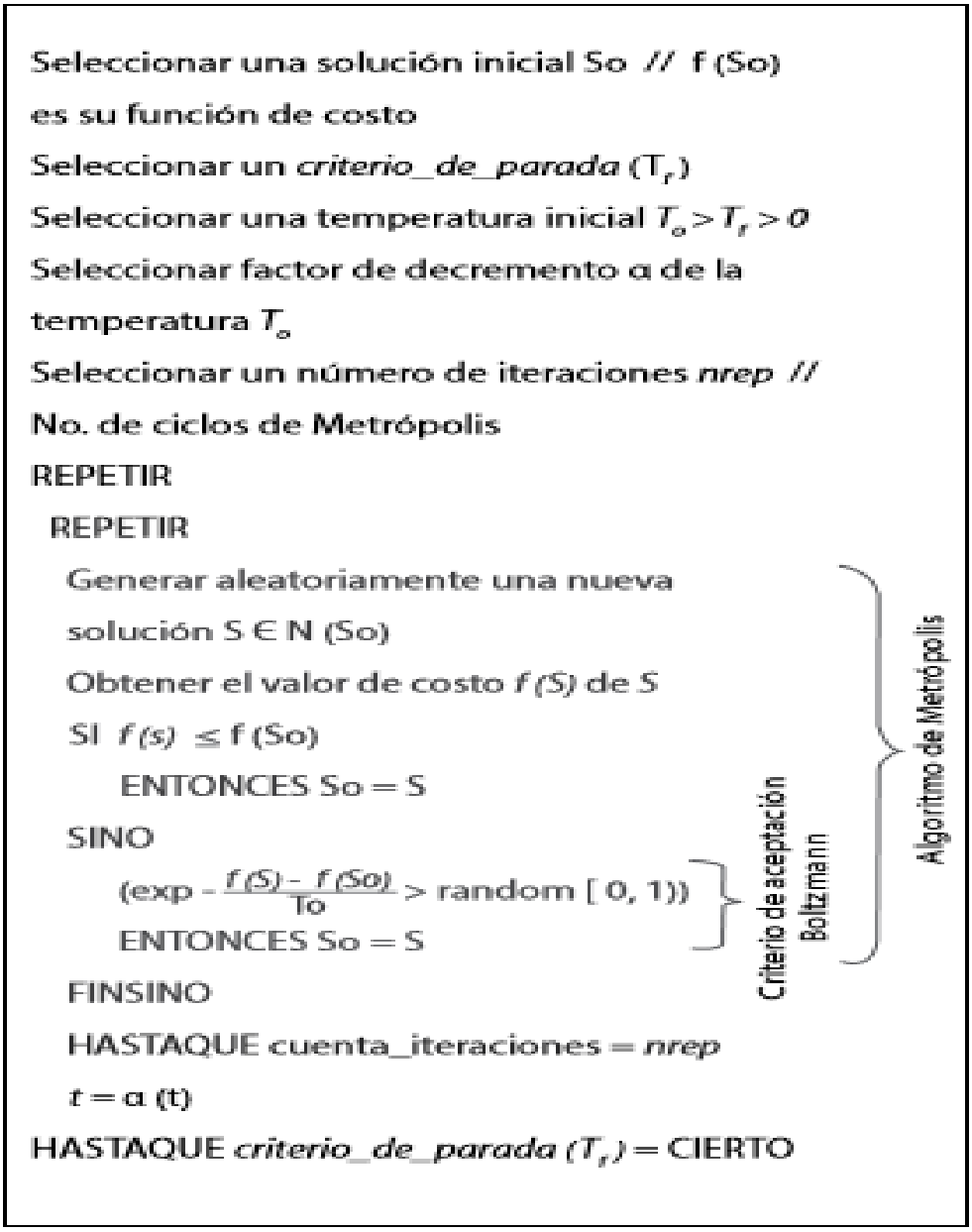


Figura 1: Algoritmo Recocido Simulado

Para que la ejecución de este algoritmo sea la más eficaz, se debe realizar un análisis intensivo del algoritmo y se ha de tener en cuenta aquellos elementos mediante los cuales se puede mejorar considerablemente el Recocido Simulado.

- Temperatura inicial: to.
- Proceso de enfriamiento: α(t).
- Número de repeticiones.
- Condición de parada.
- Espacio de soluciones: S.
- Estructura de vecindad: Si.
- Función de coste: f(s).
- Solución inicial: s0.(Pilar Moreno Díaz and Manso 2007)

En los siguientes acápites se realiza una descripción detallada de estos elementos.

1.6.1 Temperatura inicial

La temperatura inicial juega un papel fundamental en el algoritmo, por ende, debe ser escogida con mucho cuidado. Esta establece el momento en el que debe comenzar el proceso de enfriamiento. La temperatura inicial por lo general es elevada, de forma tal, que permita el libre cambio de soluciones vecinas, ofreciendo mayores posibilidades para explorar el espacio de búsqueda eficientemente, y, sobre todo, permitiendo que la solución final sea independiente de la solución inicial. Lógicamente si se elige una temperatura inicial extremadamente alta, el algoritmo realizará demasiados movimientos hacia estados que carecerán de valor para la solución del problema, provocando así, pérdida de tiempo. En caso de que la temperatura inicial elegida sea demasiada baja, el algoritmo se desplazará muy poco de su solución inicial, descartando regiones del espacio de búsqueda donde se encuentre alguno de los óptimos globales del sistema; existirán riesgos de que el algoritmo se quede estancado en

una región desde la que sólo se pueda llegar a un óptimo local (Pilar Moreno Díaz and Manso 2007).

Una regla práctica que se le suele poner a la temperatura inicial es que debe tener un valor tal que la proporción inicial de vecinos aceptados (tanto de mejora como de no mejora) tenga un alto valor (90% - 95%). Es decir, inicialmente todos los vecinos se aceptan con una probabilidad cercana a 1 (Pilar Moreno Díaz and Manso 2007).

1.6.2 Proceso de enfriamiento

El proceso de enfriamiento es el mecanismo que posee el algoritmo para que la temperatura tienda a 0. Con esto se logra que la probabilidad de aceptar soluciones peores tienda a 0, es decir, este mecanismo garantiza que cada vez se acepten menos soluciones desfavorables. El proceso de enfriamiento, por tanto, determina cómo se modifica la temperatura durante la ejecución del algoritmo (Pilar Moreno Díaz and Manso 2007). Se necesita que en el proceso de enfriamiento en un principio exista un gran porciento de aceptación, logrando una exploración inicial en el espacio de búsqueda, para en su etapa final ir reduciendo progresivamente este porciento, de manera que se consolide el proceso de explotación de vecindad, alcanzando una solución buena u óptima.

1.6.3 Número de repeticiones

El número de repeticiones ofrecen el número de vecinos que se visitan antes de reducir la temperatura del sistema. Este proceso no debe ser estático, se necesita que según se reduzca la temperatura, aumente el número de repeticiones. Es así como se conseguirá explotar la vecindad hacia el final del proceso.

Para que se pueda explorar con cierta completitud todos los vecinos el número de repeticiones debe ser del orden del tamaño de la vecindad. (Pilar Moreno Díaz and Manso 2007)

1.6.4 Condición de parada

La condición de parada indica cuando se debe terminar el algoritmo. Como la temperatura tiende a 0 progresivamente, la probabilidad de aceptar peores soluciones disminuye, es decir se acerca a 0, siendo proporcional a la probabilidad de no encontrar mejores soluciones, por tanto, se ha de tener presente que se han realizado un determinado número de iteraciones sin mejorar la solución, por consecuencia, es necesario que se deje de insistir en la vecindad de una solución para la que ya se lleva tiempo intentando explotar.

1.6.5 Espacio de soluciones

Casi todos los problemas combinatorios poseen diferentes soluciones, para poder resolverlos correctamente, es necesario plantear un modelo que se aproxime a la perfección. Mediante el mismo será posible determinar la estructura del espacio de soluciones, facilitando entonces, conocer cómo se recorrerá este espacio, además de los algoritmos que se pueden utilizar para establecer soluciones.

Resulta de gran importancia que el modelo elegido para crear el modelo del problema disponga, por comodidad algorítmica, de algunas propiedades como: el espacio de soluciones no genere soluciones degeneradas, no permita la generación de soluciones imposibles, la estructura del espacio de costos no sea excesivamente escarpado o llano, etc. (Pilar Moreno Díaz and Manso 2007).

1.6.6 Estructura de vecindad

Si se tiene una solución inicial, es de suma importancia para resolver el problema, que partiendo de ella se pueda obtener una nueva solución realizando diminutos cambios a la inicial. Independientemente de esto, pero no menos significativo, debe ser la respuesta a la siguiente pregunta: ¿Cómo generar una vecindad? El procedimiento de generar una vecindad debe garantizar que al partir de una solución cualquiera se pueda llegar directa o indirectamente a cualquier otra.

1.6.7 Función de coste

La función de coste mide la bondad de una solución. La función de coste es una medida asociada a todos los elementos de una determinada solución. Uno de los objetivos que deben guiar el desarrollo de la función de coste es que esta sea rápida de calcular al generar una solución de la vecindad, creando un cálculo incremental que tenga en cuenta sólo los cambios que generan la vecindad. De esta forma se podrán calcular más soluciones por unidad de tiempo. De hecho, el cálculo de la función de coste suele ser uno de los aspectos más costosos en tiempo de cómputo del algoritmo (Pilar Moreno Díaz and Manso 2007).

1.6.8 Solución inicial

La solución inicial, podrá ser cualquier solución aleatoria del problema, se debe tener en cuenta que sea válida.

Una alternativa mucho mejor y justificando una de las ventajas escritas en el capítulo, es la facilidad de combinar este algoritmo con otros, se puede utilizar una heurística para generar una solución inicial suficientemente buena, en lugar de generarla aleatoriamente, como se dice, a ciegas.

Otra alternativa puede ser, apoyarse en interacciones visuales facilitando por medio de una interfaz de usuario una solución inicial, la cual en dependencia de la intuición del usuario será lo suficientemente buena para comenzar.

1.7 Historia de la visualización

Cuando se intenta comprender, analizar y dar una explicación contundente de cualquier fenómeno de la vida cotidiana, la visualización es inminente, siempre está presente. A pesar de ello, la historia demuestra que el precipitado desarrollo tecnológico y computacional ha favorecido a un florecimiento de los gráficos (Ponjuán 2009).

En octubre de 1986, la División de Computación Científica Avanzada de la *National Science Foundation* (NSF) de los Estados Unidos favoreció la realización de una reunión en la que fue premisa el desarrollo de herramientas computacionales visuales. Para esto, se incluirían en los centros que se encontraban realizando computación científica avanzada y sobre todo en los de la NSF, hardware, software, herramientas de interfaces visuales para gráficos y técnicas de procesamiento de imágenes, todo bajo la justificación de que las herramientas existentes no satisfacían sus necesidades. De esta reunión se derivó como recomendación el establecimiento de una nueva iniciativa: la *visualización en informática científica.* La alternativa de solución propuesta fue la tecnología de la visualización, dirigida a facilitar a los científicos la integridad del análisis de los datos, provocar una nueva perspectiva de análisis y revelar un nuevo conocimiento que se pueda comunicar a otros (Ponjuán 2009).

Desde este momento la tecnología de la visualización comienza apreciarse como centro de los procesos científicos. La visualización tomada de la mano de la computación gráfica conllevaría al desarrollo de disímiles de sistemas.

1.7.1 Visualización ¿Disciplina o Ciencia?

Muchos han sido los criterios emitidos por los investigadores con respecto a la pregunta en cuestión. Algunos se basan en sus análisis para afirmar la importancia de la visualización en el estudio de los gráficos, la ciencia de la computación y la psicología cognitiva (Ponjuán 2009).

Aunque los criterios difieren, la inmensa mayoría de los investigadores concuerdan en que la visualización, ciencia o no, necesita nutrirse de muchas ciencias diferentes, y que posee déficit en cuanto a la cantidad de métodos que ofrece, siendo una necesidad su aporte de teorías, leyes y modelos que respalden el fenómeno que estudia.

En general se acepta la visualización como una disciplina científica, y se puede afirmar que tiene una comunidad académica asociada, un conjunto de modelos, métodos, principios, y prácticas aceptados desde distintas perspectivas; constituye una disciplina científica en evolución, dinámica, cambiante e innovadora (Ponjuán 2009).

1.7.2 Categorías de la visualización

La visualización puede dividirse en tres categorías: Visualización Científica, Visualización del Software y Visualización de Información (Ponjuán 2009).

Visualización Científica: destinada a comprender de manera más eficiente los fenómenos físicos a partir de grandes volúmenes de datos (Ponjuán 2009). Mediante ella es posible extraer información relevante de un conjunto de datos de gran tamaño para luego realizar su representación gráfica.

Visualización del Software: dirigida a comprender y utilizar el software con efectividad, investiga dos tipos fundamentales: herramientas de visualización de programas (código fuente) para mantener, comprender, perfeccionar y depurar el software y algoritmos de animación, empleados fundamentalmente en la educación para motivar el aprendizaje (Ponjuán 2009).

Visualización de Información: tiene como objetivo principal identificar patrones, correlaciones o agrupamientos de un volumen grande de información compleja, estructurada o no (Ponjuán 2009).

La visualización de información es el proceso de comunicación a partir de la cual es posible percibir hechos y fenómenos que de otra forma pasarían inadvertidos (Félix de Moya Anegón 2006).

Algunas de las subáreas fundamentales de la Visualización de Información son: Visualización de Arquitecturas de Software, Data Mining Visual y Visualización de Grafos, la cual se utilizó en la presente investigación (Alonso 2007).

La Visualización de Arquitecturas de Software proporciona sistemas interactivos que permiten extraer y visualizar datos sobre dichas arquitecturas, de forma que complementen la documentación existente acerca del software (Alonso 2007).

Data Mining Visual: este término se refiere a la integración del acceso a las bases de datos con minería de datos y visualización.

La Visualización de Grafos soluciona el problema de visualizar información estructural o relacional construyendo representaciones de grafos o redes que son los modelos subyacentes en una gran cantidad de datos abstractos (Alonso 2007).

1.8 Visualización de Grafos

Los grafos son estructuras abstractas utilizadas para representar información que pueda ser modelada como objetos y conexiones entre los mismos. Permiten expresar de una forma visualmente muy sencilla y efectiva las relaciones que se dan entre elementos de diversa índole, y resultan especialmente útiles en la representación visual de estructuras de datos en red y jerárquicas (Alonso 2007).

Es común que un grafo se represente con el objetivo de ser visualizado. Su visualización suele realizarse como un conjunto de vértices y aristas colindantes entre los puntos que en no muy pocos casos, representan un escenario de la vida real. Se debe tener en cuenta que un grafo tiene infinidades de formas de ser esbozado, esto depende del contexto en el que sea utilizado.

1.9 El problema de viajero vendedor

Un ciclo Hamiltoniano en un grafo G = (V, E) es un ciclo que recorre exactamente una vez cada vértice de V. Análogamente, un camino Hamiltoniano es un camino que recorre exactamente una vez cada vértice de V. Si consideramos un grafo completo G = (V, E) de n vértices y con costos no-negativos en los arcos $c \in R^E$, entonces el problema del viajero vendedor o TSP consiste en encontrar un ciclo Hamiltoniano en G de costo mínimo (Vargas 2012).

Informalmente el problema del viajero vendedor radica en, partiendo de una lista de ciudades y teniendo como dato el costo de viajar entre cualquier par de ellas, se debe encontrar el viaje que visita todas las ciudades una vez, y que su punto de fin sea el mismo que el de partida teniendo en cuenta que debe ser el menos costoso.

El TSP es un problema NP-duro [25] y por lo tanto no existe ningún algoritmo conocido que lo resuelva en tiempo polinomial (Vargas 2012). En la presente investigación se utiliza el algoritmo Recocido Simulado para resolver este problema.

1.10 Conclusiones parciales

Los métodos aproximados constituyen una vía fiable cuando se trata de resolver problemas de optimización, pero en su totalidad para ofrecer buenas soluciones en tiempos razonables dependen del contexto del problema en el que son utilizados. Las representaciones visuales de programas tomadas de la mano de los métodos aproximados, sin duda, contribuyen a la eficiencia y eficacia de estos procedimientos, incluso propician la comunicación fluida entre el desarrollador y los expertos. Las visualizaciones pueden utilizarse como la base para el entendimiento y aprendizaje de los algoritmos.

Capítulo 2. Modelo del Negocio y Requisitos

2.1 Introducción

En este capítulo se describen las características que el sistema posee, para ello se muestran un conjunto de diagramas que manifiestan parte de la solución encontrada. Además, se definen los requisitos funcionales y no funcionales del sistema, pilares importantes para su implementación.

2.2 Actores del sistema a automatizar

Cualquier usuario que haga uso de la aplicación será actor del sistema. Para esta aplicación de escritorio no existen roles, es decir no se definen clases desde el punto de vista social, que por su desempeño se les asigne privilegios diferentes.

2.3 Definición de los requisitos funcionales

Los requisitos funcionales son aquellas condiciones que el sistema debe cumplir; mirándose desde otra perspectiva, serían las actividades a ser realizadas por el sistema (James Rumbaugh 1998). En la Tabla 1 se muestra una lista de los requisitos funcionales más importantes, y para cada uno de ellos se realiza una breve descripción.

RF1	Importar archivo
Descripción	El sistema tiene que permitir cargar los ficheros con estructura <nombre fichero>.tsp, en los que se encuentra la información relacionada con cada ciudad.
RF2	**Ejecutar algoritmo**
Descripción	El sistema tiene que permitir ejecutar el algoritmo, es decir, el algoritmo debe

	comenzar a buscar una solución, ya sea, partiendo de una solución inicial generada aleatoriamente por él, o teniendo en cuenta una solución creada por el usuario.
RF3	**Cancelar algoritmo**
Descripción	El sistema, una vez iniciado el algoritmo tiene que permitir su cancelación, previendo que el interés del usuario sea prohibir que el algoritmo continúe ejecutándose.
RF4	**Organizar ciudades**
Descripción	El sistema tiene que permitir mover las ciudades, ya sea porque el usuario lo desea o ante la necesidad de observar algún detalle en la solución representada en el lienzo.
RF5	**Dibujar caminos**
Descripción	El sistema tiene que permitir dibujar el camino entre dos nodos; es una acción muy necesaria cuando se desea construir una solución inicial, solución asistida por el usuario del sistema.
RF6	**Centrar ciudades**
Descripción	El sistema tiene que permitir centrar las ciudades trazadas, ofreciendo así, que el usuario en caso de haber explorado la solución pueda regresar a la vista

	mostrada en un principio.
RF7	**Configurar parámetros**
Descripción	El sistema tiene que permitir configurar los parámetros relacionados con el algoritmo, los cuales influyen en su comportamiento, en este caso se tienen el coeficiente de enfriamiento, la temperatura y el número de veces que desea ejecutar el algoritmo(Iteraciones).
RF8	**Exportar solución**
Descripción	El sistema debe permitir almacenar los resultados obtenidos. En este caso la visualización gráfica se almacena en un archivo de estructura <nombre archivo>. jpeg.
RF10	**Dibujar representación**
Descripción	El sistema tiene que permitir la representación gráfica de la solución encontrada. La representación puede realizarse Automática o Manual, en dependencia de los objetivos que persigue el usuario.
RF11	**Mostrar soluciones**
Descripción	El sistema tiene que permitir mostrar la mejor y peor solución encontrada hasta el momento, así como el costo de ambas soluciones, permitiendo un

	análisis coherente al usuario.
RF12	**Deshacer cambios**
Descripción	El sistema debe permitir volver a estados anteriores. Esta opción es muy utilizada cuando se está construyendo una solución inicial asistida por el usuario, pues puede suceder que se halla trazado caminos que no se deseen.
RF13	**Aumentar lienzo**
Descripción	El sistema debe permitir aumentar la vista del lienzo, importante cuando se intenta realizar un análisis de la solución representada.
RF14	**Disminuir lienzo**
Descripción	El sistema debe permitir disminuir la vista del lienzo.
RF15	**Rotar lienzo**
Descripción	El sistema tiene que permitir rotar la vista del lienzo tanto a la derecha como a la izquierda.
RF16	**Recomenzar**
Descripción	El sistema en dependencia de la configuración asignada por el usuario, puede ejecutar el algoritmo más de una vez, por tanto, si el algoritmo es pausado en algún momento, debe

	permitirse que comenzar donde se pauso.
RF17	**Ayuda**
Descripción	El sistema tiene que permitir acceso algún manual de usuario, posibilitando que el usuario pueda comprender mejor el sistema.

Tabla 1: Requisitos funcionales

2.4 Definición de los requisitos no funcionales

Los requisitos no funcionales son condiciones del sistema que por lo general se encuentran vinculados con los requisitos funcionales. Cumplen con la particularidad de poder hacer de nuestro sistema un buen producto o no, definiendo que tan seguro o agradable puede ser el software en cuestión (James Rumbaugh 1998). En la Tabla 2 se muestra una lista de los requisitos no funcionales para este sistema.

RNF1	**Software**
Plataforma	• Es necesario que en la computadora en la que se desee utilizar el sistema, se encuentre disponible la máquina virtual de Java que se corresponda con el sistema operativo.
RNF2	**Usabilidad**
Exploración	• El sistema ofrecerá iconos y textos salientes en los botones, con el fin de lograr la mejor comprensión del sistema. • El sistema ofrecerá un menú principal que constará con los requisitos funcionales y algunas tareas adicionales, como por ejemplo:

	aumentar vista del lienzo, disminuir vista del lienzo, mover hacia la derecha o izquierda la vista el lienzo, entre otras opciones.
RNF3	**Apariencia o interfaz externa**
Vista	• El sistema ofrecerá una interfaz con apariencia de Windows 10, de manera que resulte atractiva para el usuario. • Su color debe ser en matices de negro buscando profesionalidad.
RNF4	**Rendimiento**
Tiempo de respuesta	• Ante una petición del usuario, el sistema no debe tardarse más de 5 segundos en ofrecer alguna respuesta.
RNF5	**Estandarización**
Lienzo	• El sistema permitirá exportar los resultados obtenidos en el lienzo a imágenes en formato jpeg.
RNF6	**Escalabilidad**
Descripción	• La aplicación permitirá incorporar nuevas funciones sin afectar a las ya existentes.
RNF7	**Documentación y ayuda**
Descripción	• El sistema tendrá un manual de ayuda al que se podrá acceder haciendo uso de un botón del sistema.

Tabla 2: Requisitos no funcionales

2.5 Paquetes y sus relaciones

Un paquete es una parte de un modelo. Cada parte de un modelo debe pertenecer a un paquete. El modelador puede asignar el contenido de un modelo a un conjunto de paquetes. Pero, para ser funcional, la asignación 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 muestran los paquetes del sistema con una breve descripción.

Paquete	Descripción
vistas	Almacena todas las vistas del sistema, entiéndase por vistas, las interfaces gráficas de usuario.
iconos	Recopila todas las imágenes utilizadas por las vistas.
controladores	Almacena las clases encargadas de gestionar los eventos de los usuarios; clases intermediarias entre las vistas y los modelos.
modelos	Almacena las clases que representan la información del negocio al que el sistema hace referencia, por tanto, en estas clases se gestionan todos los accesos a la información.
servicios	Posee clases que nos facilitan la realización de muchas tareas que se realizan en el modelo, es decir, lo complementan.
clases_recocidoSimulado	Almacenan las clases que intervienen

	en el algoritmo Recocido Simulado.

Tabla 3: Paquetes y sus descripciones

Las dependencias entre los paquetes resumen dependencias entre los elementos internos a ellos, es decir, las dependencias del paquete son derivables a partir de las dependencias entre elementos individuales (James Rumbaugh 1998). En la Figura 2 se muestra el diagrama de paquetes del sistema, poniéndose de manifiesto las dependencias entre ellos y evidenciando el modelo-vista-controlador.

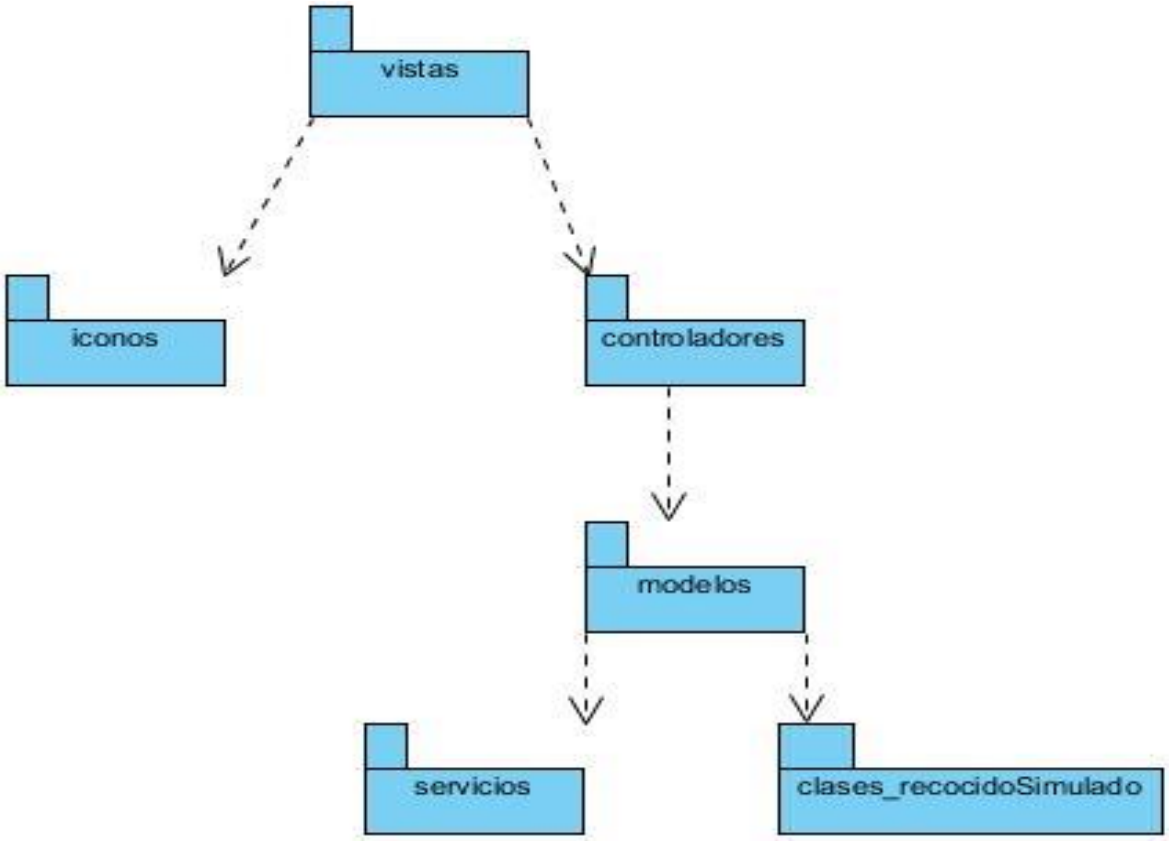


Figura 2: Diagrama de paquetes

2.6 Diagrama de Casos de Uso del Sistema

Mediante el diagrama de Casos de Uso del Sistema es posible representar las interacciones existentes entre los actores y los procesos del sistema (James Rumbaugh 1998). En la Figura 3 se observa el diagrama del sistema.

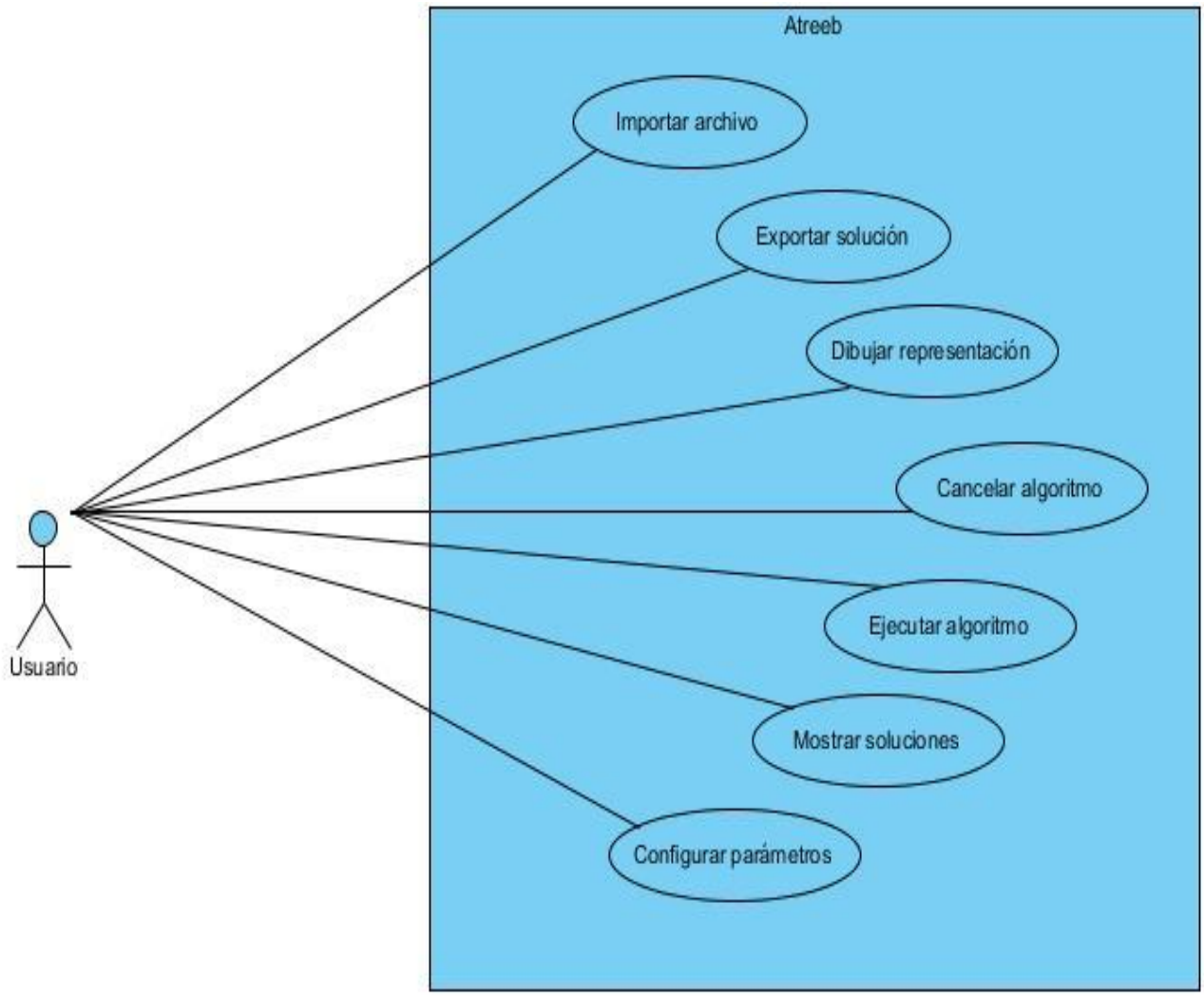


Figura 3: Diagrama de casos de uso del sistema

2.7 Determinación de los casos de Uso del Sistema más significativos

Cuando se está desarrollando un software, el tiempo es una variable importante; el cliente por lo general exige que el software sea desarrollado en un marco de

tiempo relativamente pequeño, por ende, se debe determinar qué casos de usos son más significativos, con el objetivo de priorizarlos.

Los casos de usos pueden ser clasificados en:

- **Críticos**: Casos de uso más relevantes para el usuario que tienden a resumir las principales funciones del sistema que se analiza.
- **Secundarios:** Casos de uso que en muchas ocasiones son utilizados por los casos de uso críticos, por lo que tienen gran impacto en el interés de los usuarios.
- **Auxiliares:** Casos de usos que no son determinantes para la arquitectura del sistema.
- **Opcionales:** Casos de usos que no son indispensables para el sistema, es decir, se pueden dejar a consideración del equipo de trabajo si realmente son necesarios para implementarlos en versiones iniciales.

En la Tabla 4 se pueden observar la clasificación de los casos de usos del sistema.

Clasificaciones	Casos de Uso del sistema
Críticos	• Importar archivo • Dibujar representación • Ejecutar algoritmo
Secundarios	• Cancelar algoritmo • Configurar parámetros
Auxiliares	• Mostrar soluciones
Opcionales	• Exportar soluciones

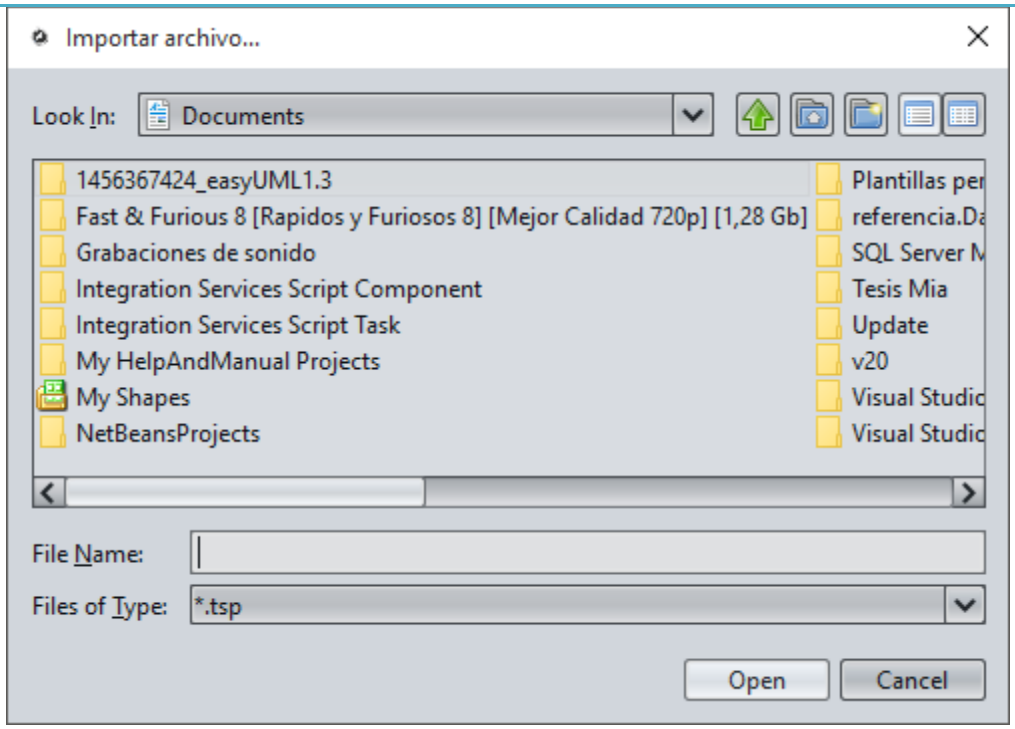
Tabla 4: Clasificación de los casos de usos del sistema

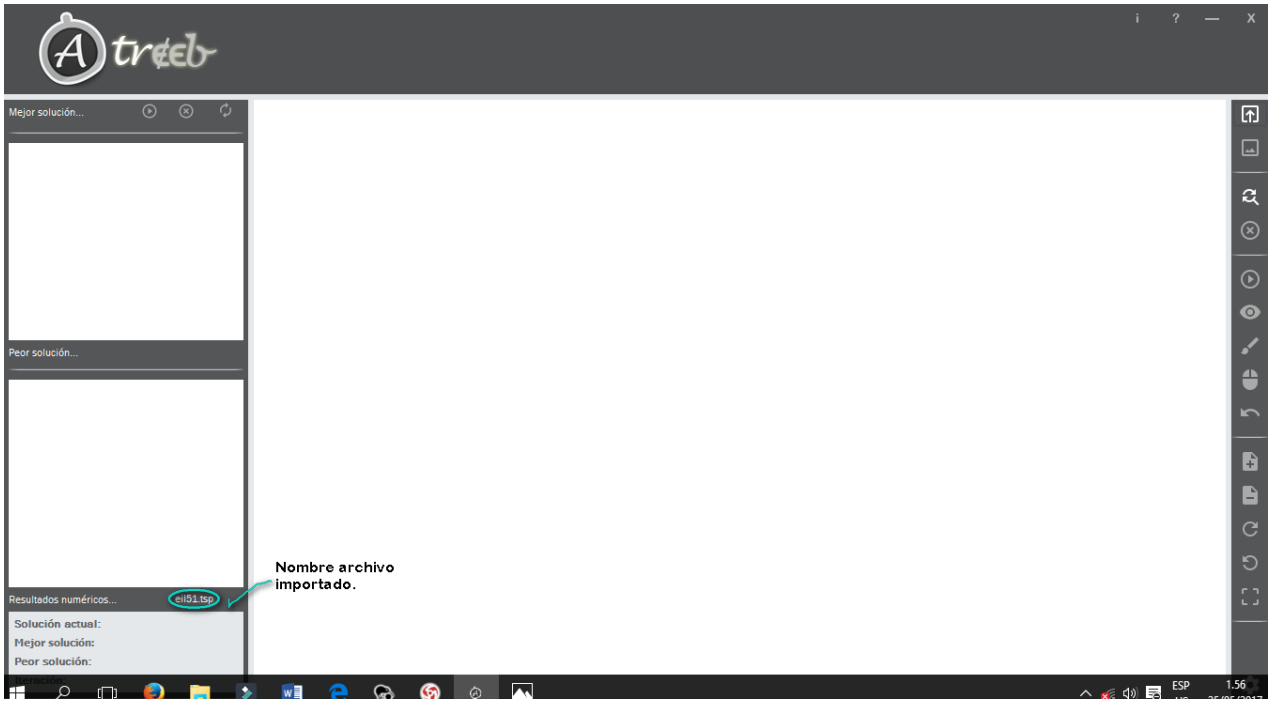
Lo antes expuesto, sirve de argumento a la clasificación que se le asignó a cada caso de uso del sistema realizada anteriormente, y con base a ello, se realizará la descripción de los casos del uso del sistema más significativos.

2.8 Descripción de los casos de uso del Sistema más significativos

Con la descripción de los casos de usos más significativos es posible mostrar el funcionamiento de cada uno de manera más explícita. En las Tablas 5, 6 y 7 se muestran las descripciones de estos casos de usos.

Caso de uso del sistema	Importar archivo
Actores	Usuario
Propósito	Definir el fichero a analizar.
Resumen	El usuario establece el fichero que desea analizar, para ello, hace uso de un selector de archivos, que le permitirá buscar por el sistema el archivo deseado.
Responsabilidades	Cargar ciudades.
Requisitos especiales	_
Precondiciones	_
Descripción	
	



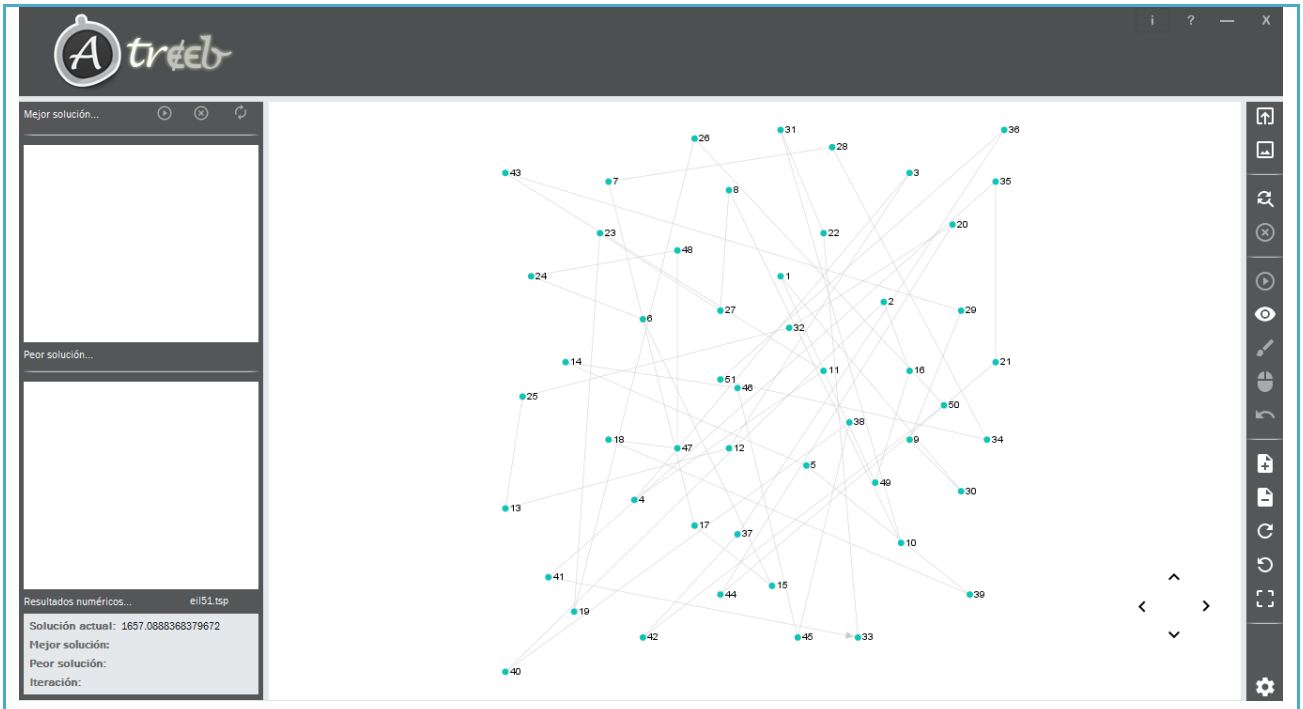


Flujo normal de los eventos	
Acción del actor	**Respuesta del sistema**

1. Elige en la barra de herramientas el botón de Importar archivo.	2. Muestra la vista correspondiente.
3. Selecciona el archivo que desea analizar, busca su ubicación física en el sistema operativo.	4. Muestra la vista correspondiente.
Post condiciones	Queda guardado en memoria un fichero que contiene los datos de todas las ciudades que se desean analizar.

Tabla 5: Descripción del caso de uso del sistema Importar archivo

Caso de uso del sistema	**Dibujar representación**
Actores	Usuario
Propósito	Mostrar en el lienzo la representación elegida.
Resumen	El usuario elige en la configuración el tipo de representación que desea realizar, y ordena al sistema que la muestre.
Responsabilidades	Mostrar ciudades.
Requisitos especiales	_
Precondiciones	El usuario debe haber seleccionado un tipo de representación.
Descripción	
	



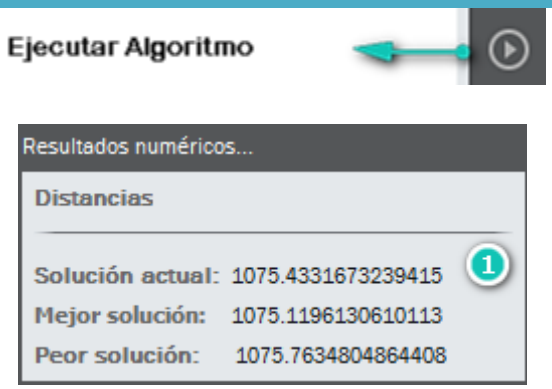
Flujo normal de los eventos	
Acción del actor	**Respuesta del sistema**
1. Elige en la barra de herramientas el botón de Dibujar representación.	2. Comienza a representar en el lienzo el archivo elegido. 3. Muestra la vista correspondiente.
Post condiciones	Queda representada en el lienzo la solución inicial creada a partir del fichero seleccionado; solución generada por el algoritmo o creada por el usuario.

Tabla 6: Descripción del caso de uso del sistema Dibujar Representación

Caso de uso del sistema	Ejecutar algoritmo
Actores	Usuario

Propósito	Mostrar los resultados numéricos encontrados por el algoritmo hasta el momento.
Resumen	El usuario ejecuta el algoritmo.
Responsabilidades	Mostrar resultados numéricos.
Requisitos especiales	_
Precondiciones	El usuario debe haber realizado al menos una representación.

Descripción



Flujo normal de los eventos

Acción del actor	Respuesta del sistema
1. Elige en la barra de herramientas el botón de ejecutar algoritmo.	2. Muestra la vista correspondiente.
Post condiciones	Queda representada en los resultados numéricos la información solicitada por el usuario del sistema.

Tabla 7: Descripción del caso de uso del sistema Ejecutar algoritmo

2.9 Conclusiones parciales

En este capítulo se realizó un análisis del sistema, plasmándolo en un conjunto de diagramas que exponen a groso modo como se desempeñarán las principales funcionalidades del software.

Capítulo 3. Descripción de la propuesta de solución

3.1 Introducción

En el siguiente capítulo se propone un modelo de integración de las técnicas de visualización con el algoritmo de búsqueda metaheurística Recocido Simulado. Este expone brevemente, aquellos puntos en el que el algoritmo haciendo uso de las técnicas de visualización pudiese mostrar datos numéricos, propiciando las interacciones que el usuario pudiera tener con el algoritmo para asistir la búsqueda de una mejor solución, mejorando así, la eficiencia y eficacia del Recocido Simulado.

3.2 Arquitectura del sistema

- Interiorizar los conceptos fundamentales de la visualización para lograr un análisis correcto cuando se intente integrar con el Recocido Simulado.
- Identificar las posibles interacciones del usuario con el algoritmo de Recocido Simulado, así como, las visualizaciones que pudiera realizar el algoritmo para un mejor análisis del usuario.
- Proponer un modelo de integración de las técnicas de visualización con el algoritmo de Recocido Simulado.
- Realizar una aplicación de escritorio desarrollada en el lenguaje de programación Java en su edición estándar que refleje el modelo propuesto.

3.3 Diagrama de clases de diseño

Los conceptos de la aplicación son modelados como clases, cada una de las cuales describe un conjunto de objetos discretos que almacenan información y se comunican para implementar un comportamiento (James Rumbaugh 1998). El diagrama de clases influye en la calidad del software, por tanto, es de gran importancia contar con un diagrama de clases impecable que garantice el mantenimiento de nuestro software.

3.3.1 Caso de uso Importar archivo

En la Figura 4 que se muestra a continuación se puede observar el diagrama de clases del caso de uso Importar archivo.

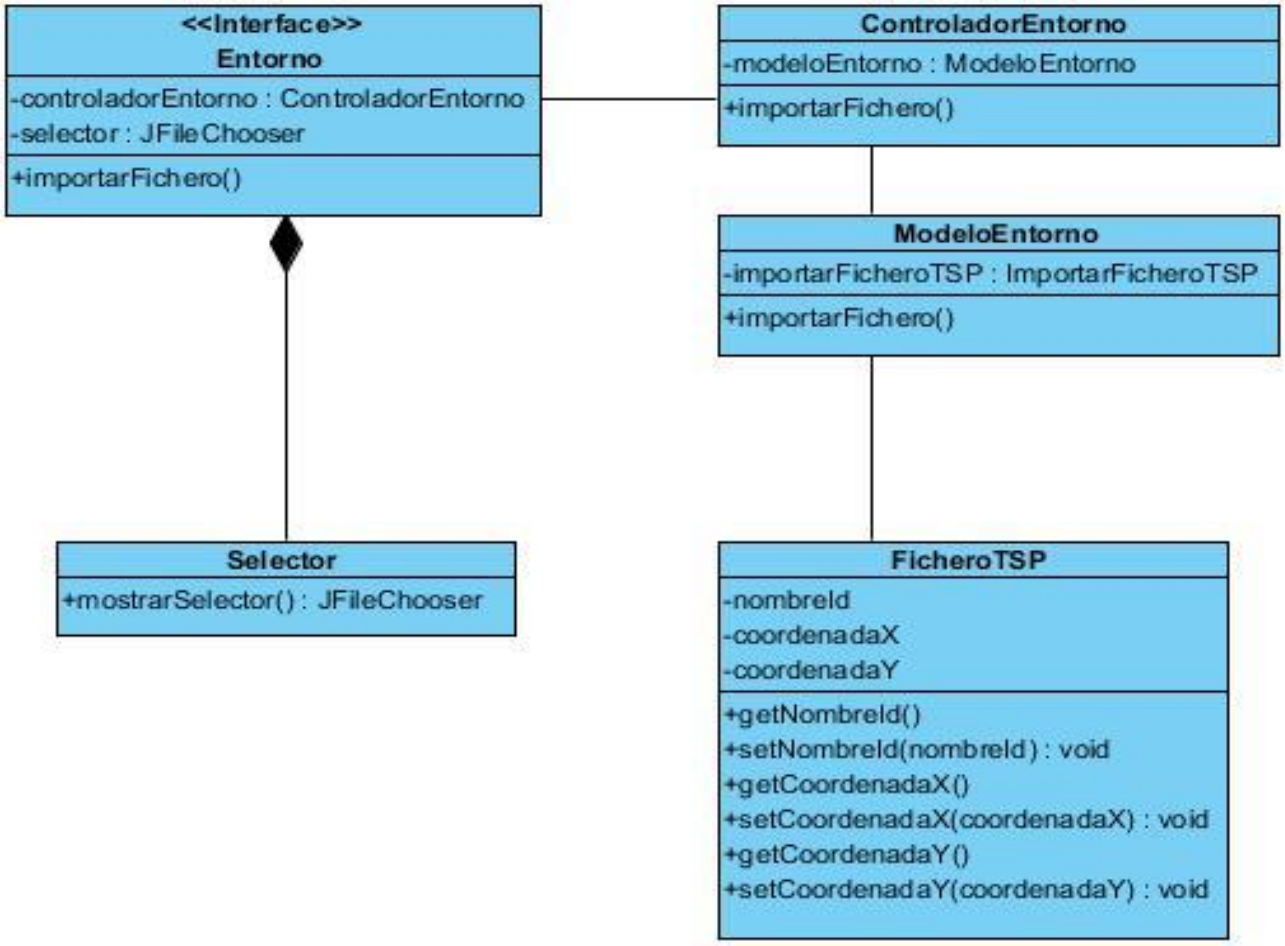


Figura 4: Diagrama de clases del caso de uso del sistema Importar archivo

3.3.2 Caso de uso Dibujar representación

En la Figura 5 que se muestra a continuación se puede observar el diagrama de clases del caso de uso Dibujar representación.

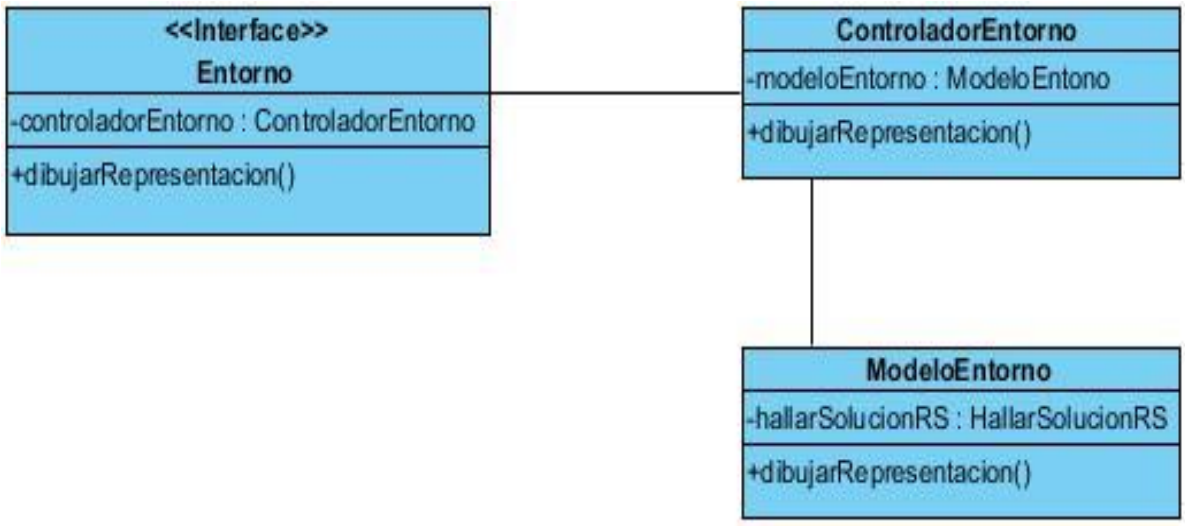


Figura 5: Diagrama de clases del caso de uso Dibujar representación

3.3.3 Caso de uso Ejecutar algoritmo

En la Figura 6 que se muestra a continuación se puede observar el diagrama de clases del caso de uso Ejecutar algoritmo.

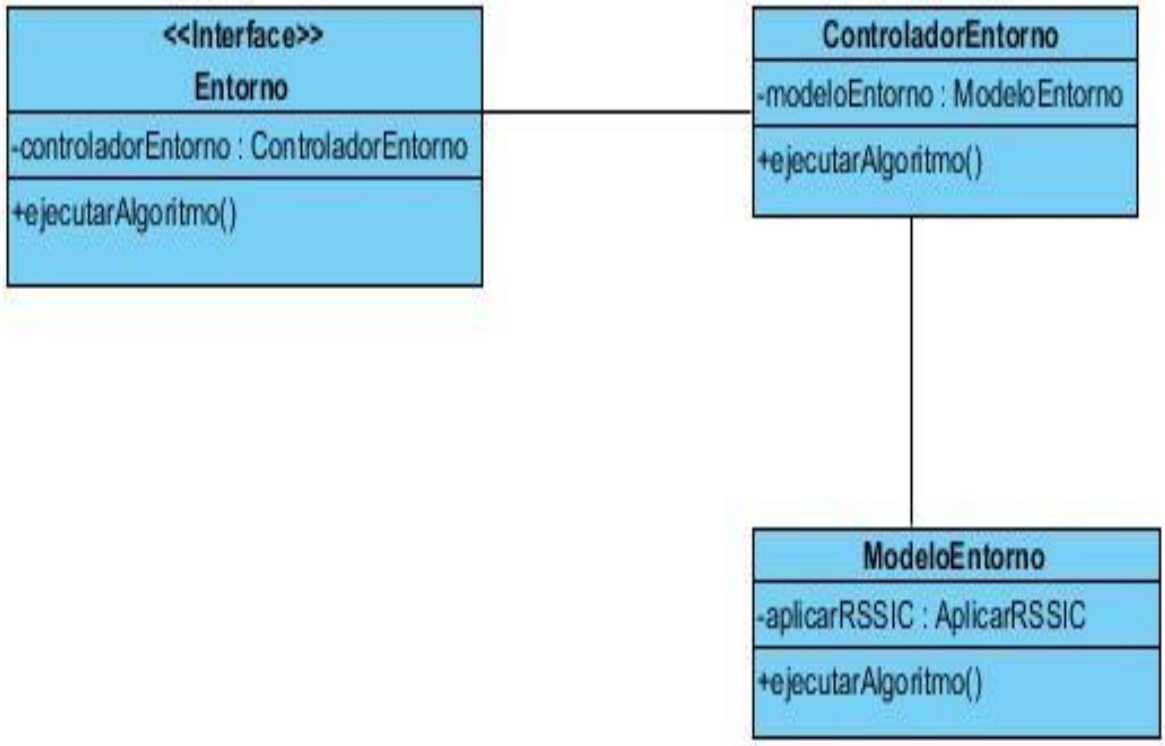


Figura 6: Diagrama de clases de uso del sistema Ejecutar algoritmo

3.4 Diagrama de secuencia

El diagrama de secuencias, muestra la forma en que los objetos se comunican entre sí al transcurrir el tiempo (Danny Felipe Vergel Paba 2008).

3.4.1 Caso de uso Importar Archivo

En la Figura 7 se muestra el diagrama de secuencias del caso de uso Importar archivo.

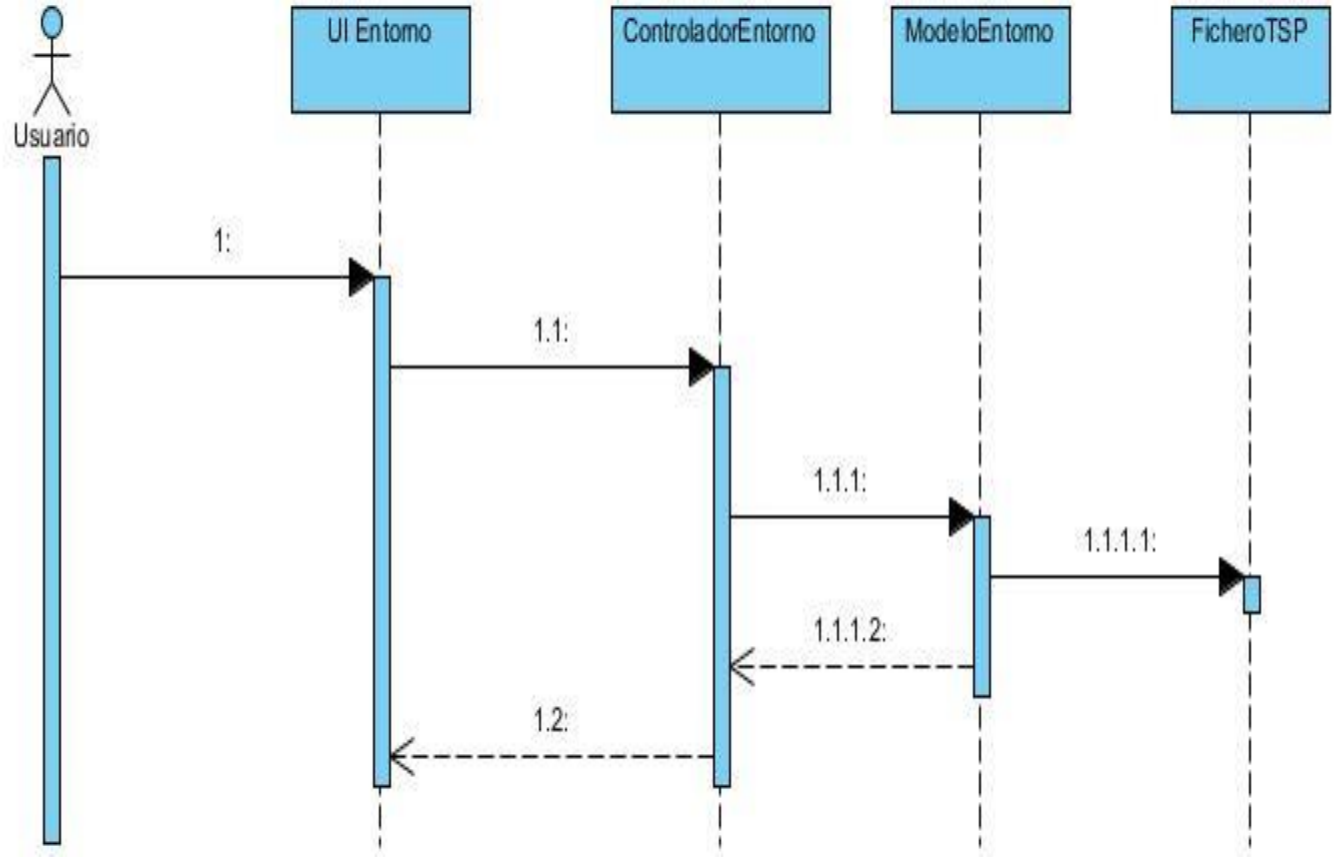


Figura 7: Diagrama de secuencia del caso de uso del sistema Importar archivo

1: Ingresar a la interfaz de Importar archivo.

1.1: Consultar los datos del archivo (importar archivo).

1.1.1: Consulta los datos del archivo en el fichero.

1.1.1.1: Carga los datos del fichero.

1.1.1.2: Redireccionar al controlador.

1.2: Redireccionar a la vista.

3.4.2 Caso de uso Dibujar representación

En la Figura 8 se muestra el diagrama de secuencias del caso de uso Dibujar representación.

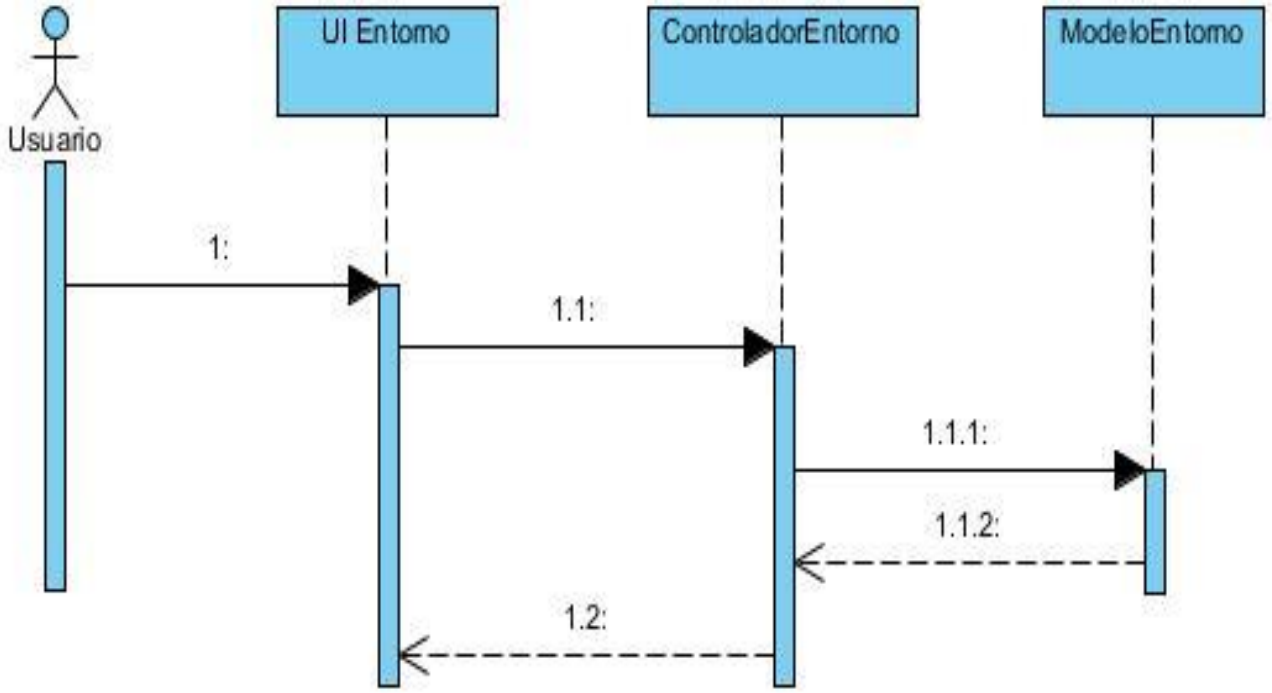


Figura 8: Diagrama de secuencia del caso de uso del sistema Dibujar representación

1: Solicitud de dibujar representación.

1.1: Dibuja la representación.

1.1.1: Dibuja representación.

1.1.2: Redireccionar al controlador.

1.2: Redireccionar a la vista.

3.4.3 Caso de uso Ejecutar algoritmo

En la Figura 9 se muestra el diagrama de secuencias del caso de uso Ejecutar algoritmo.

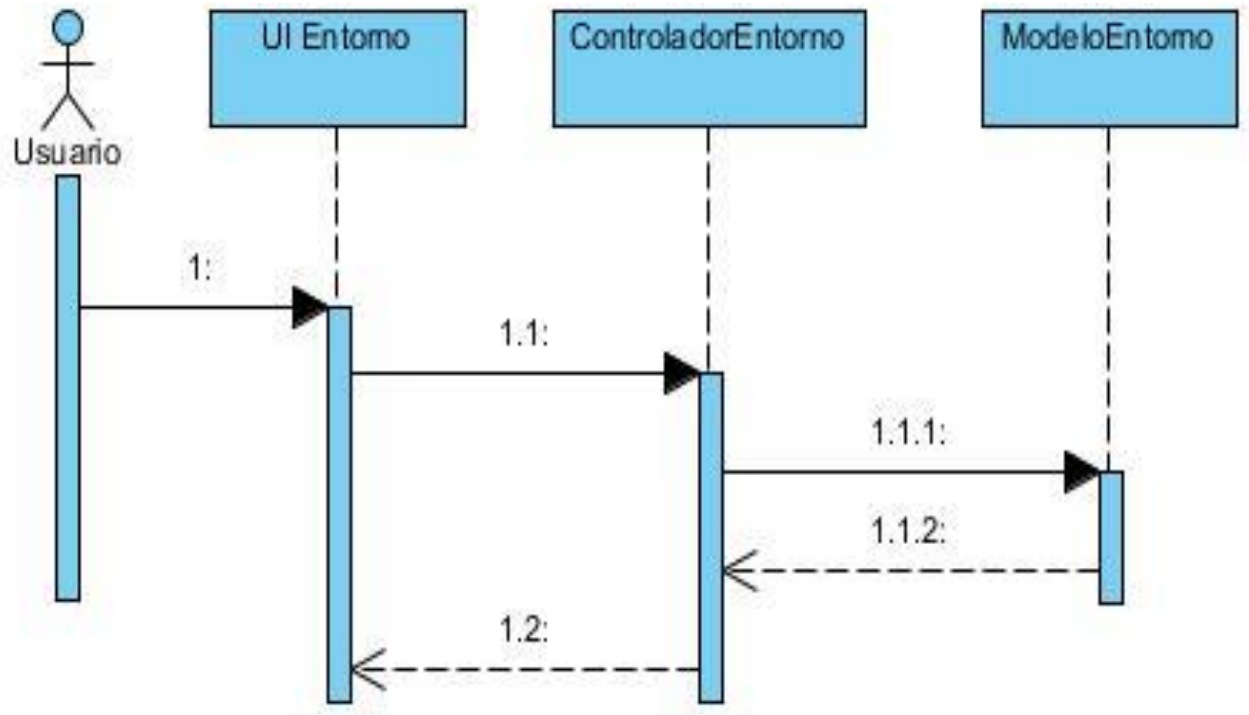


Figura 9: Diagrama de secuencia del caso de uso del sistema Ejecutar algoritmo

1: Solicitad de ejecutar algoritmo.

1.1: Ejecuta el algoritmo.

1.1.1: Ejecutar algoritmo.

1.1.2: Redireccionar al controlador.

1.2: Redireccionar a la vista.

3.5 Patrón arquitectura

Los patrones de arquitectura brindan soluciones a problemas de arquitectura de software en ingeniería de software. Estos gestionan cómo debe ser organizado un sistema, apoyándose en un esquema de organización estructural básica, en el que se reflejan las responsabilidades e interacciones de los elementos del sistema que

se analiza. Se ha de tener en cuenta que los patrones arquitectónicos poseen un nivel de abstracción superior a los patrones de diseño.

Para la creación de este sistema se utiliza el patrón de arquitectura conocido por el nombre de modelo-vista-controlador (MVC). El esquema que propone este patrón se basa en la creación de tres componentes:

- Modelo: Representa la información del sistema, por tanto, se encarga de gestionar todos los accesos a la información.
- Vista: Presenta la información del modelo. Las vistas suelen ser las interfaces gráficas de usuario.
- Controlador: Responde a todos los eventos del usuario, es conocido como el intermediario entre la vista y el modelo.

En la Figura 10 siguiente se muestra el funcionamiento del modelo-vista-controlador:

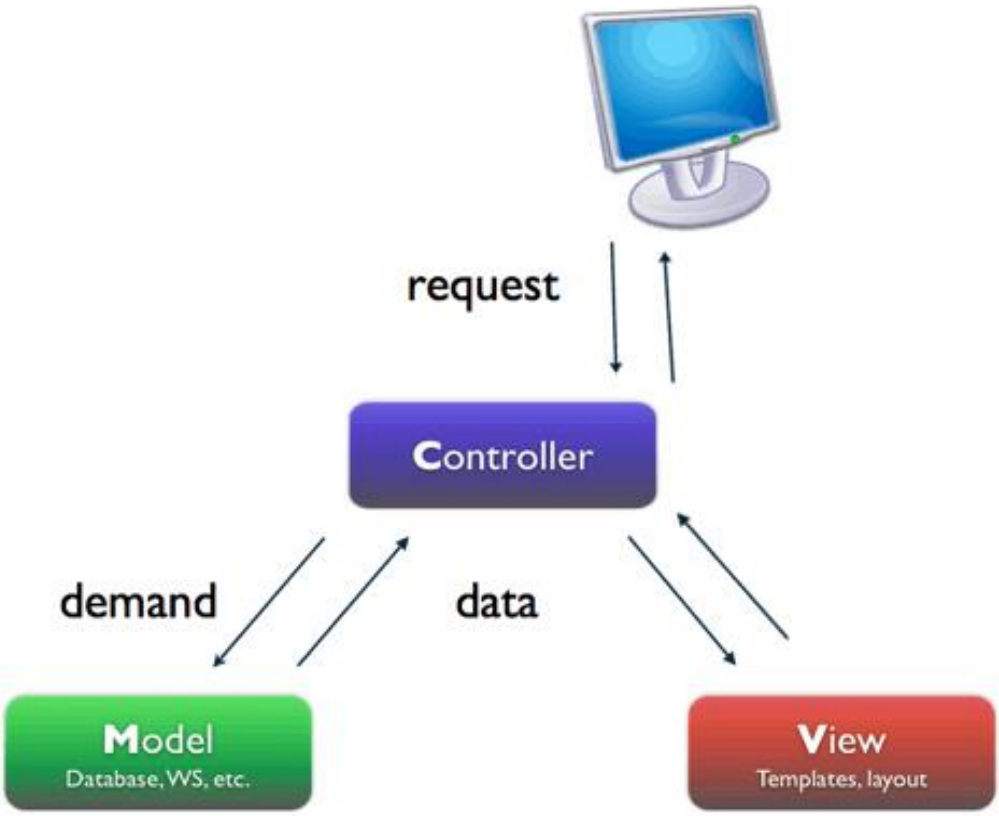


Figura 10: Patrón arquitectónico MVC

3.6 Patrón de diseño

En el desarrollo de software, existen problemas que aparecen una y otra vez, aun en diferentes contextos. Por esto, se han creado variantes para resolverlos sin tener que idear nuevas soluciones, a las que se le conoce como patrones de diseño.

El uso de patrones de diseño asegura la calidad del software, además de propiciar el mantenimiento del sistema, por esto, en el sistema que se implementó se utiliza el patrón conocido como **Iterator,** el cual permite realizar recorridos sobre objetos

compuestos sin tener en cuenta sus implementaciones. En la Figura 11 se muestra un fragmento de código en el que se aprecia su utilización:

- **Iterator:**

```
// Obtiene la distancia total del tour.
public double getDistancia() {
    Iterator iterator = tour.iterator();
    int contador = -1;
    if (distancia == 0) {
        double tourDistancia = 0;
        while (iterator.hasNext()) {
            contador++;
            // Obtiene la ciudad en la que nos encontramos actualmente.
            Ciudad ciudadActual = (Ciudad) iterator.next();
            // Ciudad a la que se debe ir.
            Ciudad ciudadDestino;
            if (contador + 1 < tour.size()) {
                ciudadDestino = getCiudad(contador + 1);
            } else {
                ciudadDestino = getCiudad(0);
            }
            tourDistancia = (int) (tourDistancia + ciudadActual.distanciaA(ciudadDestino));
        }
        distancia = tourDistancia;
    }
    return distancia;
}
```

Figura 11: Patrón Iterator evidenciado en el código

3.7 Tratamiento de errores

Los errores en un sistema además de generan comportamientos indeseables, afectan la calidad del software y elevan su costo de mantenimiento. Es de vital importancia realizar un sistema libre de errores. En el software implementado, el tratamiento de errores se manejó a través de la captura de excepciones predefinidas del lenguaje de programación Java.

Excepciones predefinidas de Java que se utilizaron:

- HeadlessException
- IOException
- NumberFormatException
- InterruptedException
- ExecutionException

- EdgeRejectedException: predefinida de la biblioteca graphStream.
- Exception

3.8 Interacciones que pueden implementarse en el Recocido Simulado

Cuando se ejecuta el Recocido Simulado, en muchas ocasiones se aleja de las soluciones medianamente buenas, esto se debe a cómo funciona el algoritmo. Un sistema interactivo, posibilita que en algún momento el algoritmo muestre los parámetros que están siendo procesados, dando la posibilidad al usuario de observar su comportamiento y tomar el control externo sobre la información que se visualiza; así puede iniciarse un intercambio entre el algoritmo y el usuario. A continuación se proponen aquellas interacciones que han sido identificadas para interactuar con el Recocido Simulado:

- Visualizar en cada instante de la ejecución del algoritmo el recorrido del viajero por las ciudades. (Véase Figura 12)

  Permite observar la trayectoria que sigue el vendedor para construir la solución. Solución que en una primera instancia refleja el tour a partir del cual el algoritmo comenzará a buscar soluciones mejores.

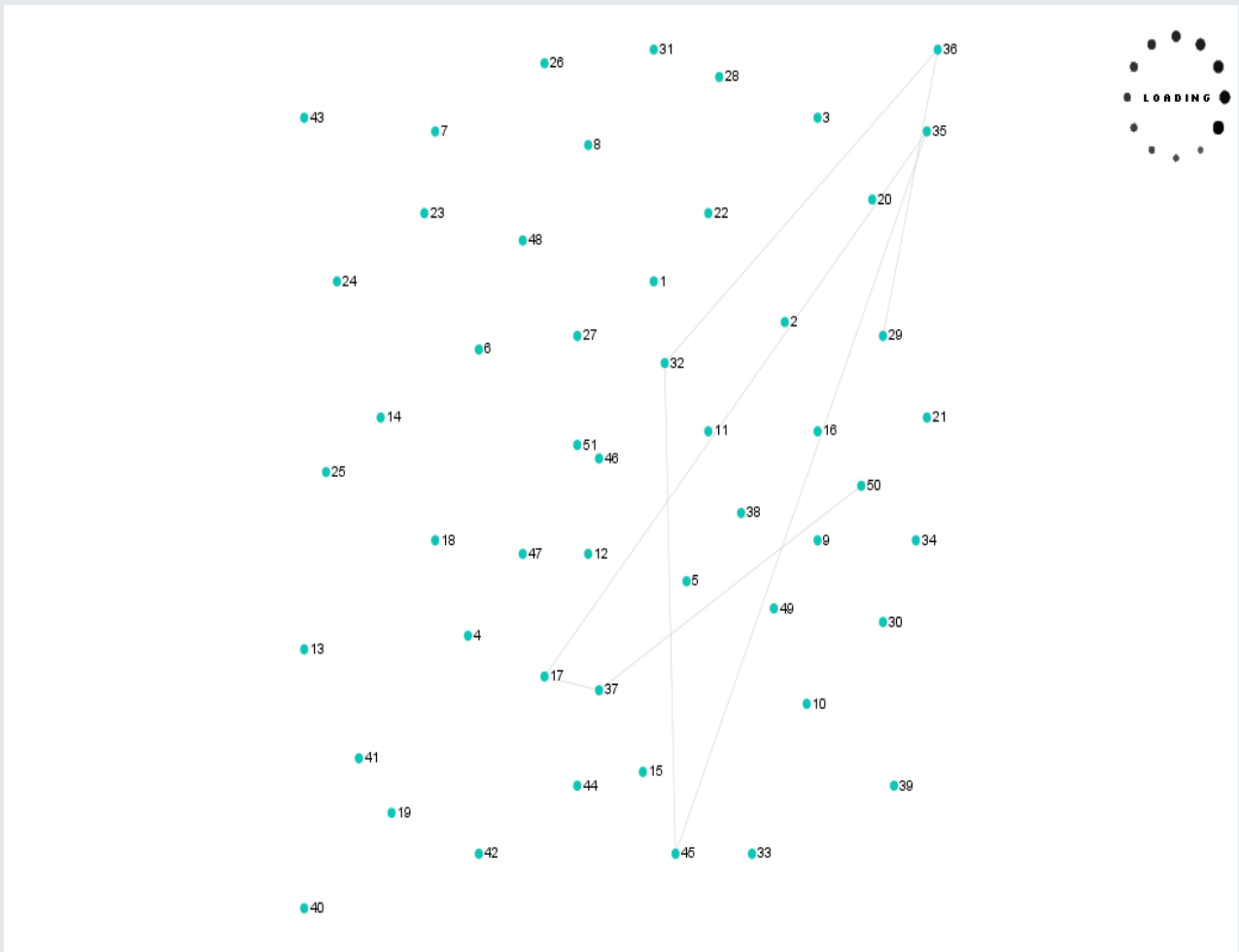


Figura 12: Ejecución del algoritmo

- Mostrar las mejore y peores soluciones encontradas por el algoritmo cada vez que el usuario lo desee o en el instante en que el algoritmo culmine. (Véase Figura 13)

  Ofrece la posibilidad al usuario de realizar un análisis de las mejores y peores soluciones encontradas hasta ese momento. De este análisis se puede inferir que, en caso de que el usuario cree una solución inicial, debe parecerse a la mejor encontrada hasta el momento y bajo ningún concepto acercarse a la más mala.

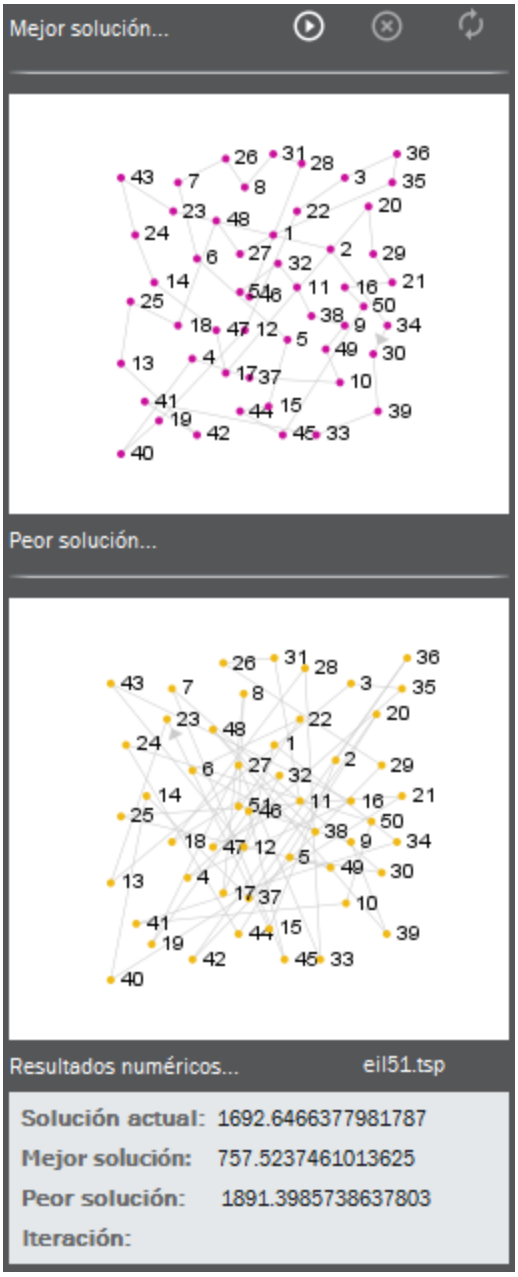


Figura 13: Soluciones

- Crear una solución inicial. (Véase Figura 14)

  Muchas veces como consecuencia de que el algoritmo genera aleatoriamente la solución inicial, no suele ser una solución buena. Esta interacción permite al usuario crear la solución inicial mediante la cual el

algoritmo comenzará. Para ello, se recomienda observar la mejor y peor solución encontrada en otras ocasiones.

Figura 14: Solución inicial

- Visualizar los resultados numéricos de las soluciones encontradas. (Véase Figura 15)

  En dependencia de lo que se muestre en los resultados numéricos, se pueden modificar los parámetros que están siendo procesados por el algoritmo.

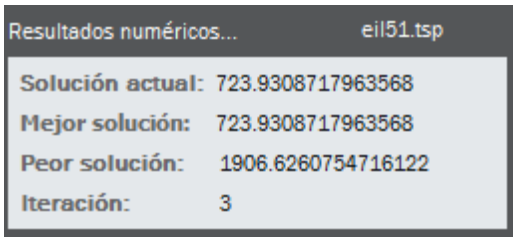


Figura 15: Resultados

- Representar información de los nodos.

  Se muestra en el lienzo los números de todos los nodos representados, estos concuerdan con los nombres de las ciudades que se almacenan en el fichero importado en el inicio.

- Visualizar la distancia real entre las ciudades.

  Cuando se realiza la representación de las ciudades, se muestra en pantalla la información importada del fichero. En el fichero se almacena los nombres de las ciudades y sus coordenadas. Para que el usuario pueda crear una solución inicial para el algoritmo, las ciudades representadas en la pantalla deben mantener una distancia lógica entre ellas, según sus coordenadas debe ser la ubicación de ellas en el lienzo. El sistema a través de esta visualización gestiona que todas las ciudades se representen en su ubicación real, dejando a la intuición del usuario la creación de la solución.

- Variar la velocidad de corrida del algoritmo.

  Posibilita observar con más detalle la representación en el lienzo de cualquier solución encontrada por el algoritmo.

- Detener el algoritmo.

  En el momento en que se detiene el algoritmo, se representa en pantalla la solución inicial desde que se partió, así como, la mejor y peor solución asociada ella. Además, se reflejan en los datos numéricos los resultados obtenidos por cada solución y en que instante fueron adquiridos.

- Establecer la cantidad de veces que se realizará el algoritmo.

  Permite ejecutar el algoritmo todas las veces que el usuario desee, y observar en cada momento los resultados numéricos encontrados hasta ese instante.

- Modificar parámetros. (Véase Figura 16)

  Puede suceder que el algoritmo este dejando de analizar variantes en las que se encuentra una mejor solución que la actual. Modificando los parámetros que intervienen en el algoritmo se puede intensificar la búsqueda, logrando explotar de manera más eficiente el espacio de búsqueda.

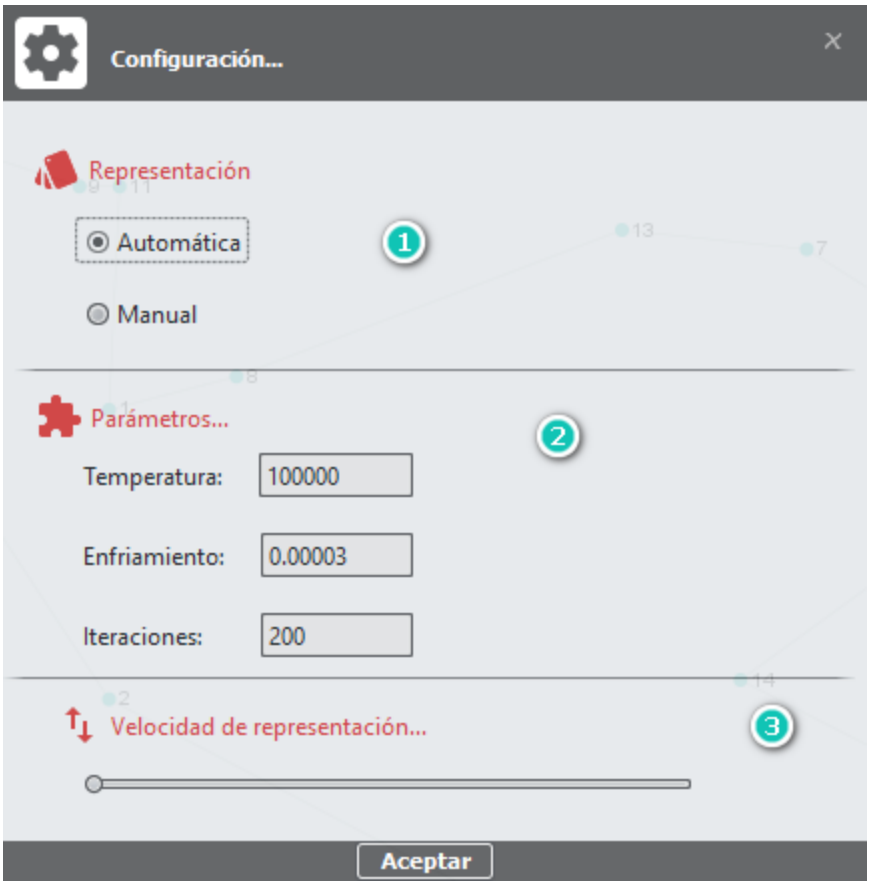


Figura 16: Configuración

- Ejecutar algoritmo a partir de la mejor solución encontrada.

  Permite ejecutar el algoritmo a partir de la mejor solución encontrada hasta el momento. Es muy probable que partiendo de una solución bastante buena se obtengan mejores en un tiempo razonable.

- Restringir el espacio de búsqueda.

  Como el algoritmo genera las nuevas soluciones aleatoriamente se corre el riesgo de no encontrar nuevas soluciones, por esto, esta interacción permite al usuario establecer trayectorias que a criterio de él son buenas, y obliga al Recocido Simulado a encontrar nuevas soluciones que respeten

las trayectorias que el usuario considera buenas, logrando restringir el espacio de búsqueda a soluciones que cuenten con estos caminos.

3.9 Esquema general del modelo

En la Figura 17 se muestra un modelo que resume las interacciones visuales entre el algoritmo de búsqueda metaheurística Recocido Simulado y el sistema interactivo.

El modelo en su panel izquierdo almacena las visualizaciones que el algoritmo puede realizar al sistema, y las interacciones que el usuario puede efectuar que influyen directamente en el comportamiento del Recocido Simulado. Su panel derecho muestra el funcionamiento del algoritmo.

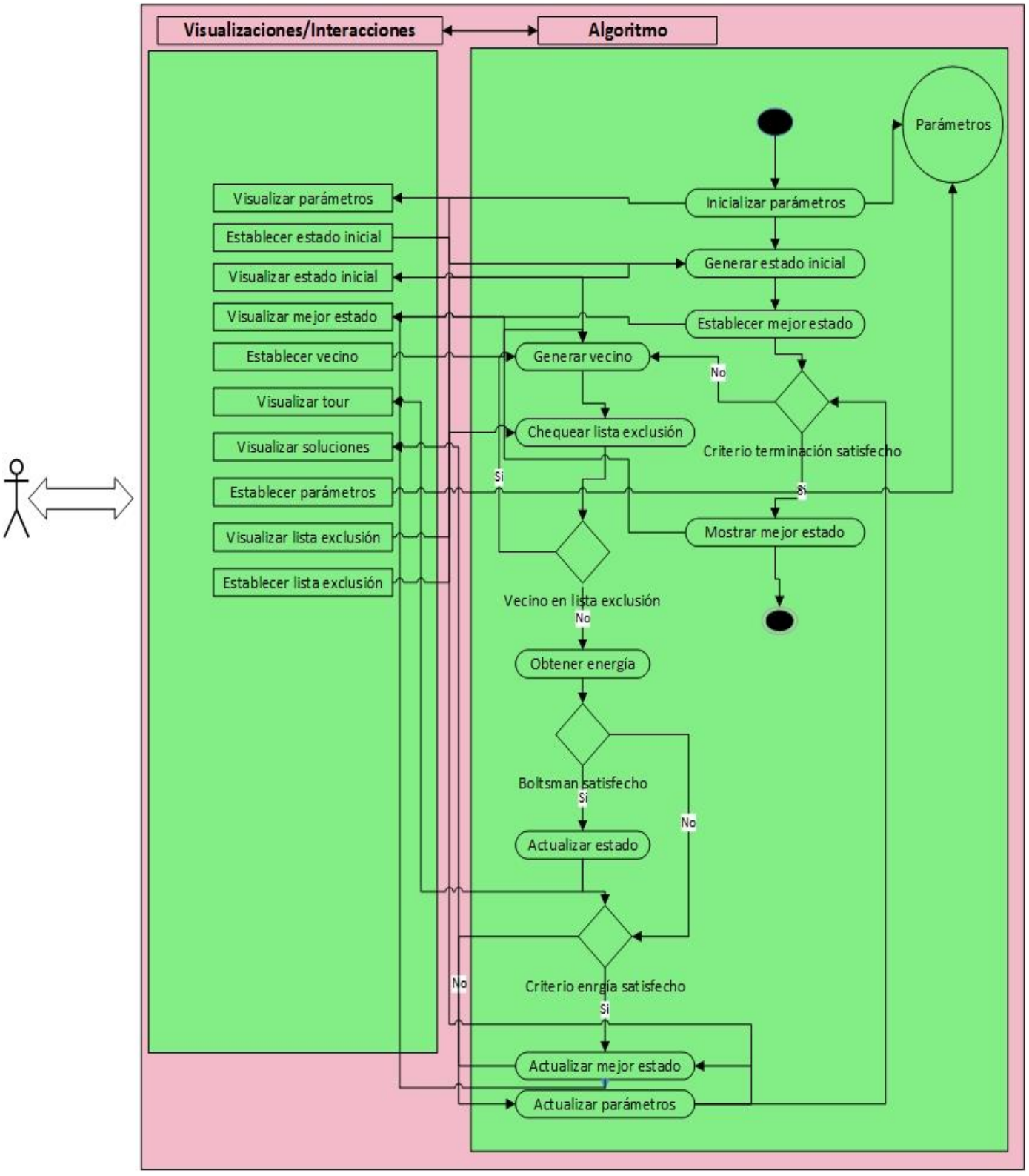


Figura 17: Modelo de integración de técnicas de visualización

3.10 Implementación del esquema general del modelo

El modelo de integración propuesto en el acápite anterior sirvió como guía para la creación de una aplicación de escritorio implementada en el lenguaje de programación Java. En la primera versión del software, se implementan la mayoría de las interacciones representadas en el modelo. A continuación se presenta en la Figura 18 una imagen del sistema Atreeb o Berta.

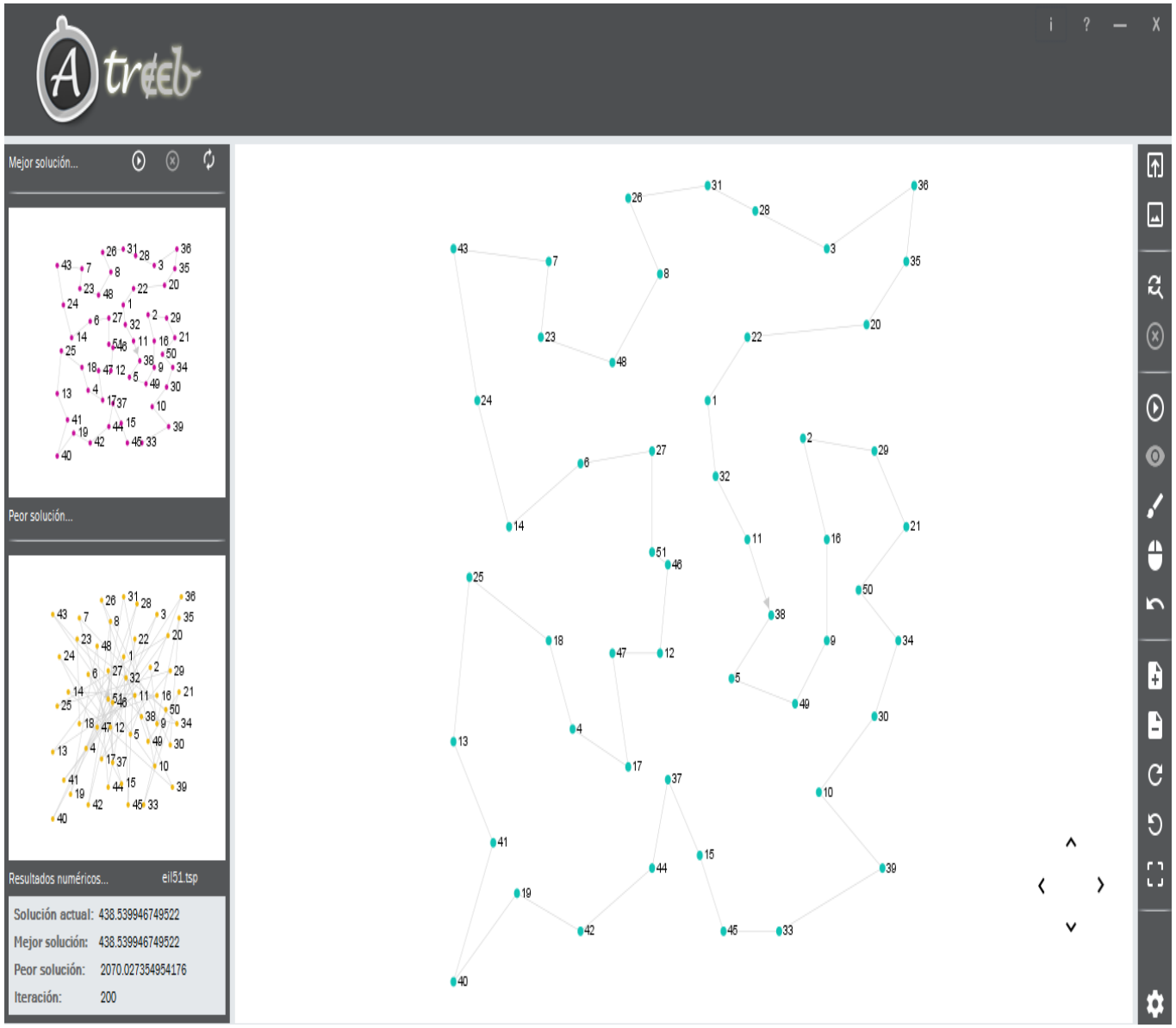


Figura 18: Interfaz principal

3.11 Estudio experimental

El comportamiento del algoritmo Recocido Simulado sin asistencia del usuario y del Recocido Simulado con interacción del usuario mediante la aplicación de escritorio creada, es una comparación que se tuvo presente en la investigación. El sistema se ejecutó en un ordenador con las siguientes propiedades:

- Procesador: Intel(R) Core(TM) i3-5005U, 2.0GHz (4CPUs).
- Memoria: 4096MB RAM.
- Sistema Operativo: Windows versión 10.0.10240.

Para la comparación se utilizaron los siguientes ficheros tsp:

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

En las ejecuciones del algoritmo sin asistencia del usuario la temperatura fue utilizada con valor de 100000 y el coeficiente de enfriamiento con valor de 0.003.

En las ejecuciones del algoritmo guiadas por el usuario, la modificación de los parámetros fue la principal interacción aplicada. Aunque también se utilizó la creación de la solución inicial del algoritmo.

En la Tabla 8 se resumen los resultados obtenidos en la comparación. De izquierda a derecha las columnas de la tabla representan: nombre del problema TSP, cantidad de ciudades, para el Recocido Simulado con y sin interacción: los costos de las ejecuciones y la cantidad de iteraciones en cada una, la última columna ofrece el costo de la mejor solución registrada en el sitio http://www1.iwr.uni-heidelberg.de/groups/comopt/software/TSPLIB95/index.html.

Nombre	Nro.	Recocido Simulado		Recocido Simulado		Mejor Solución
		Costo	Iteraciones	Costo	Iteraciones	
eil51	51	618.84	500	426.7	300	426
berlin52	52	10100.52	500	7543.3	300	7542
st70	70	1192.11	500	676.1	300	675
eil76	76	1005.93	500	538	300	538
kroA100	100	47342.35	500	21312.5	300	21282
ch150	150	7908.48	500	6528	300	6528
a280	280	14878.14	500	2599.3	300	2579

Tabla 8: Comparación con soluciones sin asistencia del usuario

3.12 Análisis de la comparación

Para realizar este análisis, se ejecutó mediante la herramienta implementada, el algoritmo sin asistencia del usuario y el algoritmo siendo guiado por el usuario. En cada instancia se tomó el costo obtenido y utilizando el test no paramétrico de los rangos con signo de Wilcoxon se determinó según los costos de las ejecuciones, si existían diferencias significativas entre las soluciones obtenidas por ambas pruebas.

En el Anexo 1, se aprecia los resultados obtenidos por el SPSS, como el valor de significación asintótica es de 0.018<0.05 se demuestra que existen diferencias significativas entre las soluciones obtenidas por ambas pruebas. Como consecuencia de lo antes expuesto es posible afirmar que, la interacción visual del usuario en tiempo de ejecución con el algoritmo Recocido Simulado conduce a mejores resultados en márgenes de tiempos más cortos.

3.13 Conclusiones parciales

En este capítulo se realizó una breve descripción de la propuesta de solución, asiéndose énfasis en las interacciones visuales entre el algoritmo y el sistema. Estas interacciones se reflejan en el modelo propuesto en el acápite 3.9.

Capítulo 4. Análisis de factibilidad

4.1 Introducción

Este capítulo es esencial para demostrar que el sistema además de funcionar, lo hace de manera correcta. En él se detalla el análisis de factibilidad y las pruebas realizadas a la aplicación, lo que se utilizará para corroborar que el software implementado ofrece resultados sorprendentes.

4.2 Planificación basada en casos de uso

La planificación basada en casos de usos se encuentra estrechamente relacionada con las variables más importantes del proyecto. Ella enfoca sus esfuerzos en determinar los valores de costo, tiempo y recursos requeridos para realizar un software, ofreciendo la posibilidad de estimar los resultados del proyecto.

Para realizar la planificación existen varios métodos, en el caso de esta investigación se utiliza la técnica presentada a continuación:

• Planificar tareas, asignando recursos y determinando puntos de control.

4.2.1 Cálculo de Puntos de Casos de Uso sin ajustar

Para determinar la estimación de un proyecto, el primer paso consiste en calcular los Puntos de Casos de Uso sin ajustar, para ello, se emplea la siguiente fórmula:

Ecuación:

UUCP = UAW + UUCW

Donde:

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

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

Este valor se determina en dependencia de la cantidad de actores presentes en el sistema y la complejidad correspondiente a cada uno de ello. Existen diferentes criterios para establecer la complejidad, a continuación son mostrados en la Tabla 9:

Criterios para el cálculo:

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	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 9: Criterios para establecer la complejidad de los actores del sistema

Como consecuencia de que existe solo un actor (Usuario) que interactúa con el sistema, se ha identificado un actor complejo.

Por tanto:

$UAW = \sum (Actor_i * Factor\ de\ Peso_i)$

UAW = 3 * 1 = 3

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

Para calcular UUCW se realiza un análisis de la cantidad de Casos de Uso del sistema y la complejidad de cada uno de ellos. La complejidad se encuentra asociada a la cantidad de transacciones efectuadas por cada caso de uso. (Véase Tabla 10 y 11)

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 transacciones	15	0

Tabla 10: Criterios para establecer la complejidad de los casos de uso del sistema

Cantidad de transacciones por casos de uso:

Caso de uso	Cantidad de transacciones	Peso
Importar archivo	2	5
Exportar solución	2	5
Dibujar representación	1	5
Cancelar algoritmo	1	5
Ejecutar algoritmo	1	5

Mostrar soluciones	1	5
Configurar parámetros	2	5

Tabla 11: Cantidad de transacciones por caso de uso

Por tanto:

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

UUCW= 5 * 7 = 35

Por consiguiente, UUCP = 3 + 35 = 38

4.2.2 Cálculo de Puntos de Casos de Uso ajustados

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

4.2.2.1 Factor de complejidad técnica (TCF)

El TCF puede calcularse teniéndose en cuenta una lista de factores que determinan la complejidad técnica del sistema. Para trabajar con estos valores se instaura un número de cero a cinco, siendo cero un aporte irrelevante y cinco el valor de mayor aporte.

En la Tabla 12 se muestra la relación entre los factores utilizados y el peso correspondiente a cada uno de ellos.

Factor	Descripción	Peso
1	Sistema distribuido	2
2	Objetivos de performance o tiempo de respuesta	1
3	Eficiencia del usuario final	1
4	Procesamiento interno complejo	1
5	El código debe ser reutilizable	1
6	Facilidad de instalación	0.5
7	Facilidad de uso	0.5
8	Portabilidad	2
9	Facilidad de cambio	1
10	Concurrencia	1
11	Incluye objetivos especiales de seguridad	1
12	Provee acceso directo a terceras partes	1
13	Se requieren facilidades especiales de entrenamiento a usuarios	1

Tabla 12: Factores que determinan la complejidad técnica del sistema

De la tabla mostrada anteriormente se realizó un análisis mediante el cual se arribaron a los siguientes resultados: (Véase Tabla 13)

Factor	Valor asignado	Valor asignado*Peso
1	4 (es distribuido)	8
2	5 (debe responder de forma rápida a los pedidos)	5

3	3 (los usuarios no tienen por qué ser eficientes)	3
4	4 (procesamiento complejo)	4
5	5 (código reutilizable)	5
6	0	0
7	5 (fácil de usar)	2,5
8	5 (funciona en cualquier sistema operativo)	10
9	5 (adaptable al cambio)	5
10	0	0
11	0	0
12	0	0
13	0	0

Tabla 13: Valor asignado a los factores que determinan la complejidad técnica del sistema

Por tanto:

$TCF = 0,6 + 0,01 * \sum (Peso_i * valor\ asignado)$

$TCF = TCF = 0,6 + 0,01 * (8+5+3+4+5+2,5+10+5) = 1,025$

4.2.2.2 Factor de ambiente (EF)

Para calcular EF se utilizan las variables de entrenamiento y la de actividades realizadas por el grupo en el desarrollo.

La Tabla 14 muestra el significado y el peso de cada uno de estos factores:

Factor	Descripción	Peso
1	Familiaridad con el modelo de proyecto utilizado	1.5
2	Experiencia en la aplicación	0.5
3	Experiencia en orientación a objetos	1
4	Capacidad del analista líder	0.5
5	Motivación	1
6	Estabilidad de los requerimientos	2
7	Personal part-time	-1
8	Dificultad del lenguaje de programación	-1

Tabla 14: Listado de factores

Consideraciones a tener en cuenta:

Para los factores del 1 al 4, un valor asignado de 0 significa sin experiencia, 3 experiencia media y 5 amplia experiencia (experto).

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

Para el factor 6, 0 significa requerimientos extremadamente inestables, 3 estabilidad media y 5 requerimientos estables sin posibilidad de cambios.

Para el factor 7, 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 8, 0 significa que el lenguaje de programación es fácil de usar, 3 medio y 5 que el lenguaje es extremadamente difícil. (Véase Tabla 15)

Factor	Valor asignado	Valor asignado* Peso

1	3	4,5
2	3	1.5
3	3	3
4	4	2
5	5	5
6	4	8
7	0	0
8	3	-3

Tabla 15: Valor asignado a cada factor

Por tanto:

EF = 1,4 – 0,03 * Σ (Peso i x Valor asignado)

EF = 1,4 – 0,03 *(4,5+1.5+3+2+5+8-3) = 0,77

Por consiguiente, UCP = 38 * 1,025 * 0,77= 29,9915

4.2.3 Estimación del esfuerzo por los Puntos de Casos de Uso

El esfuerzo en horas-hombre viene dado por:

E = UCP * CF

Dónde:

E: Esfuerzo estimado en horas-hombre

UCP: Puntos de Casos de Uso ajustados

CF: Factor de conversión.

- Para el cálculo se siguen los siguientes pasos:
  1. Se contabilizan cuántos factores de los que afectan al Factor de ambiente están por debajo del valor medio (3), para los factores del 1 al 6.
  2. Se contabilizan cuántos factores de los que afectan al Factor de ambiente están por encima del valor medio (3), para los factores 7 y 8.
  3. 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.
  4. 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.
  5. 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.

Dado que quedó un total de 0 se toma como Factor de conversión 20 horas-hombre.

Por consiguiente, E = 29,9915 * 20 = 599,83

Este método proporciona una estimación del esfuerzo en horas-hombre contemplando sólo el desarrollo de la funcionalidad especificada en los casos de uso.

4.2.4 Estimación del esfuerzo total del proyecto

Si a la estimación del esfuerzo obtenida por los Puntos de Casos de Uso se le añade las estimaciones de esfuerzo de las demás actividades que intervienen en el desarrollo del software, se obtiene una estimación más completa de la duración total del proyecto.

El siguiente criterio refleja cómo fue distribuido el esfuerzo entre las diferentes actividades por las que pasa un proyecto: (Véase Tabla 16)

Actividad	Porcentaje
Análisis	10,00%
Diseño	20,00%
Programación	40,00%
Pruebas	15,00%
Sobrecarga (otras actividades)	15,00%

Tabla 16: Actividades del proyecto

Dado que ya se posee la estimación del esfuerzo en horas para la actividad de programación de los casos de usos, se puede calcular a través de esta, las estimaciones para las demás actividades. En la Tabla 17 se muestran los resultados.

Actividad	Porcentaje	Horas / hombre
Análisis	10,00%	149,9575
Diseño	20,00%	299,915
Programación	40,00%	**599,83**
Pruebas	15,00%	224,93625
Sobrecarga (otras actividades)	15,00%	224,93625
Total	100%	1499,575

Tabla 17: Cantidad de horas/hombre por cada una de las actividades

Ecuación:

ETotal = ∑ actividades

ETotal = 1499,575 HH

Donde:

ETotal: esfuerzo total

4.2.5 Estimación del tiempo del desarrollo del proyecto

Para calcular el tiempo de desarrollo aproximado del proyecto se posee la siguiente fórmula:

TDesarrollo = ETotal / CHTotal

Donde:

TDesarrollo: tiempo de desarrollo total de horas.

CHTotal: cantidad de hombres que desarrollan el proyecto.

Por consiguiente, TDesarrollo = 1499,575/1 = 1499,575 h

4.2.6 Estimación del costo de desarrollo del proyecto

Para realizar la estimación del costo monetario del proyecto, se cuenta con la siguiente fórmula:

CTotal = ETotal * CHH

Donde:

CHH: costo por hombre-hora

CHH = K * THP

Donde:

K: coeficiente que tiene en cuenta los costos indirectos (1,5 y 2,0). Ejemplo de estos costos puede ser el costo del equipamiento, etc.

THP: Tarifa Horaria Promedio. La Tarifa Horaria Promedio se calcula por la siguiente fórmula:

THP = ST/CHM

Donde:

ST: Representa el salario del trabajador.

CHM = 190.6: Cantidad de horas mensuales.

El valor de 190.6 refleja la cantidad de horas que un trabajador labora en un mes. Esta cantidad de horas se calcula teniendo en cuenta que un trabajador labora 7.33 horas por día, lo que significa que, si en un mes se trabajan 26 días, la CHM = 7.33 * 26.

Por tanto:

Si a cada trabajador cuenta con un salario de $1000.

THP = 1000/190.6 = 5.25

Entonces:

CTotal = ETotal * K * THP

CTotal = 1499,575 * 2 * 5,25 = $15745,5375

4.3 Casos de pruebas

La prueba de caja negra se refiere a las pruebas que se llevan a cabo sobre la interfaz del software. O sea, los casos de prueba pretenden demostrar que las funciones del software son operativas, que la entrada se acepta de forma

adecuada y que se produce un resultado correcto, así como que la integridad de la información externa se mantiene.

Características fundamentales:

- Verifican las especificaciones funcionales y no consideran la estructura interna del programa.
- Es hecha sin el conocimiento interno del producto.
- No validan funciones ocultas (por ejemplo funciones implementadas pero no descritas en las especificaciones funcionales del diseño) por tanto los errores asociados a ellas no serán encontrados.

4.4 Conclusiones parciales

En este capítulo se realizó el análisis de factibilidad, mediante el que se determinó el tiempo necesario para la confección del sistema y su costo de desarrollo.

Conclusiones Generales

En la presente investigación:

- Se presentaron los aspectos teóricos de las variables abordas. Haciéndose énfasis en el algoritmo de búsqueda metaheurística Recocido Simulado y las visualizaciones, en especial, la visualización de grafos. Además se describió en qué consisten los problemas de optimización, prestando mayor atención, al problema del viajero vendedor.
- Fueron identificadas las interacciones mediante las que el usuario puede interactuar con el algoritmo de búsqueda metaheurística Recocido Simulado.
- Se propuso un modelo de integración entre las técnicas de visualización y el algoritmo de búsqueda metaheurística Recocido Simulado.
- Se realizó la implementación de un sistema que permite la interacción entre el usuario y el algoritmo de búsqueda metaheurística Recocido Simulado en la solución del TSP.
- Se demostró la efectividad del modelo propuesto utilizando las comparaciones con la herramienta implementada y certificando el resultado con el SPSS.

Al integrarse las técnicas de visualización con cualquier algoritmo, se supone que la información gestionada por el algoritmo sea visualizada. Esta, se presenta al usuario con el fin de que interactúe con ella y la modifique a conveniencia de sus intereses. La integración abordada, además de facilitar la comprensión del algoritmo, favorece a su eficiencia y eficacia; se ha de tener en cuenta que, al usuario interactuar con un algoritmo de optimización en tiempo de ejecución, puede obtener soluciones de mejor calidad y en menor tiempo, todo depende de su intuición.

Recomendaciones

- Se recomienda implementar interacciones para restringir el espacio de búsqueda.
- Se recomienda extender el análisis a ficheros TSP con más de 280 ciudades.
- Se recomienda implementar la funcionalidad de eliminar aristas.
- Se recomienda estudiar la búsqueda guiada por usuario para resolver problemas de optimización, pues es un tema que ofrece ventajas considerables.

Anexos

Prueba de Wilcoxon de los rangos con signo

Rangos

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

a. costo_de_ejecuciones_del_algoritmo_sin_asistencia_del_usuario < costo_de_ejecuciones_del_algoritmo_con_asistencia_del_usuario

b. costo_de_ejecuciones_del_algoritmo_sin_asistencia_del_usuario > costo_de_ejecuciones_del_algoritmo_con_asistencia_del_usuario

c. costo_de_ejecuciones_del_algoritmo_sin_asistencia_del_usuario = costo_de_ejecuciones_del_algoritmo_con_asistencia_del_usuario

Estadísticos de prueba[a]

	costo_de_ejecuciones_del_algoritmo_sin_asistencia_del_usuario - costo_de_ejecuciones_del_algoritmo_con_asistencia_del_usuario
Z	-2,366[b]
Sig. asintótica (bilateral)	,018

a. Prueba de Wilcoxon de los rangos con signo

b. Se basa en rangos negativos.

Anexo 1: Resultados de la Prueba de los rangos con signo de Wilcoxon.

Bibliografía

Danny Felipe Vergel Paba, H. M. C. (2008). ANÁLISIS, DISÑO, IMPLEMENTACION E IMPLEMENTACIÓN DEL MÓDULO DE EGRESADOS Y ENCUENTAS PARA EL SITIO WEB DE LA ESCUELA DE INGENIERIA DE SISTEMAS E INFORMATICA DE LA UNIVERSIDAD INDUSTRIAL DE SANTANDER, UNIVERSIDAD INDUSTRIAL DE SANTANDER 127.

Félix de Moya Anegón, B. V. Q., Zaida Chinchilla Rodríguez, Elena Corera Álvarez, Antonio González Molina,Francisco José Muñóz Fernández,Víctor Herrero Solana (2006). "Visualización y análsis de la estructura científica española: ISI Web of sciencie 1990-2005."

James Rumbaugh, I. J., Grady Booch (1998). El Lenguaje Unificado de Modelado. Manual de Referencia.

Jesús del Carmen Peralta Abarca, J. Y. J. C., Beatriz Martínez Bahena (2011). "Aplicación del recocido simulado en problemas de optimización combinatoria ".

Pilar Moreno Díaz, G. H. F.-T., Jesús Sánchez Allende, and A. G. Manso (2007). "METAHEURÍSTICAS DE OPTIMIZACIÓN COMBINATORIA:

USO DE SIMULATED ANNEALING PARA UN PROBLEMA DE

CALENDARIZACIÓN." Tecnologí@ y Desarrollo **V**.

Ponjuán, D. T. (2009). "Aproximaciones a la visualización como disciplina científica."

Schweickardt, G., Miranda,Vladimiro (2010). "Metaheurística FEPSO aplicada a problemas de Optimización Combinatoria: Balance de Fases en Sistemas de Distribución Eléctrica " CIENCIAS EXACTAS Y NATURALES: 135.

Vargas, O. A. L. (2012). EL PROBLEMA DEL VENDEDOR VIAJERO EN GRAFOS CÚBICOS. Ingeniería Industrial, Universidad de Chile**:** 68.

Alonso, E. M. (2007). Integración a técnicas de Visualización a algoritmos de Optimización de la metaheuristica Colonia de Hormigas. Ciencia de la Computación, Universidad Central Marta Abreu de las Villas **:** 81.