UNIVERSIDAD DE EXTREMADURA

Escuela Politécnica

Grado en Ingeniería Informática en Ingeniería de Computadores

Trabajo Fin de Grado

Optimización de Rutas Basada en Computación Bio-Inspirada

Carlos Muñoz Zapata

Junio 2019

UNIVERSIDAD DE EXTREMADURA

Escuela Politécnica
Grado en Ingeniería Informática en Ingeniería de Computadores

Trabajo Fin de Grado
Optimización de Rutas Basada en Computación Bio-Inspirada

Autor: Carlos Muñoz Zapata

Tutor: Antonio Manuel Silva Luengo

Tribunal Calificador

Presidente: José Moreno Del Pozo

Secretario: Francisco Andrés Hernández

Vocal: Pablo Bustos García De Castro

Índice General de Contenidos

Índice de Tablas

Índice de Figuras

1 Resumen

El presente trabajo consiste en la obtención de soluciones subóptimas al problema del trazo de rutas en sistemas de recogida (ej.: recogida de residuos).

Este tipo de problema combina problemas clásicos de la historia de la computación tales *como El Problema del Viajante* y *El Problema de la Mochila*, así como otros problemas secundarios como la *segmentación* y la *optimización de costes*.

La forma de abordar estos problemas se realizará mediante dos algoritmos bio-inspirados, Algoritmos Genéticos y Algoritmos de Enjambres, los cuáles se pondrán a prueba en un estudio.

2 Historia del Arte

2.1 Problema del Viajante

El *Problema del Viajante* es un problema NP-Completo muy famoso en el mundo de la computación, debido a sus diversas aplicaciones y apariciones como subproblema a resolver en multitud de campos del saber.

El origen histórico de este problema no esta bien definido y se desconoce por el momento, aunque hay ciertas teorías que están siendo estudiadas.
Por ejemplo, se tiene constancia de una mención al problema en 1832, pero este no fue formulado matemáticamente hasta finales de siglo.

El problema comenzó a ser estudiado más exhaustivamente a comienzos del siglo XIX, aumentando en gran medida su popularidad a mediados de siglo. Esto hizo que multitud de investigadores y expertos en distintos campos estudiaran este problema con puntos de vista totalmente diferentes.

A partir de la década de los 90s y, gracias a los avances de la informática, el problema comenzó a ser resuelto para un número elevado de nodos.

Actualmente continua el estudio de dicho problema, así como la búsqueda de un algoritmo que permita resolver un numero muy elevado de nodos (millones) en un tiempo razonable.

2.2 Problema de la Mochila

El *Problema de la Mochila* es un problema NP-Completo de optimización combinatoria.
Dicho problema, en su versión original, trata de maximizar el valor del conjunto de objetos que entren dentro de "la mochila" sin exceder la máxima capacidad de esta.

Al igual que muchos otros problemas matemáticos e informáticos, su origen no está del todo definido y establecido.
Se tiene constancia de un documento que hace referencia a dicho problema a finales del siglo XIX. Pero no es hasta la década de los 70s que nos encontramos una definición formal de dicho problema, formulada por el informático Richard Karp.

Actualmente, el *Problema de la Mochila*, al ser un subproblema de la teoría de conjuntos, es ampliamente estudiado y adaptado a diversos problemas de actualidad en busca de nuevas soluciones.

2.3 Algoritmos Bio-Inspirados

2.3.1 Algoritmos Genéticos

Los *Algoritmos Genéticos* son un tipo de algoritmo bio-inspirado. Como su nombre indica, es un algoritmo que basa su funcionamiento en el comportamiento del proceso genético evolutivo natural, presente en los seres vivos.

Este tipo de algoritmos fue propuesto y estudiado por Alan Turing a mediados del siglo XX.

Al principio, en la década de los 50s, los *Algoritmos Genéticos* no tuvieron un impacto notable en el mundo científico.
Unos años después, a principios de los 60s, este tipo de algoritmos y sus propiedades comenzaron a ser ampliamente estudiados.

En 1970 se publicó una obra de Fraser y Burnell donde se definían los elementos esenciales de los algoritmos genéticos utilizados actualmente. A lo largo de la década fueron publicadas también otras obras de diversos autores definiendo el problema.

Debido a la creciente fama de los *Algoritmos Genéticos*, en la década de los 80s se llevó a cabo la primera conferencia sobre algoritmos genéticos en Pensilvania.

En los años siguientes este tipo de algoritmo comenzó a ser probado en multitud de problemas existentes, tanto clásicos como novedosos, recibiendo en muchos casos resultados sorprendentes.

Los *Algoritmos Genéticos* son conocidos hoy en día como una forma de resolver cualquier tipo de problema de optimización, pudiendo utilizarse su versión más clásica para problemas sencillos, o ser adaptada específicamente a cualquier problema de mayor complejidad.

2.3.2 Algoritmos de Enjambres

Los *Algoritmos de Enjambres* son un tipo de algoritmo bio-inspirados que tratan de resolver problemas de optimización basándose en el comportamiento de sistemas colectivos autoorganizados.

Al igual que los *algoritmos genéticos*, los *Algoritmos de Enjambres* comenzaron a ser estudiados en la segunda mitad del siglo XX.

En este caso, estos tipos de algoritmos derivaron del estudio de aplicar mecanismos naturales alternativos (a la evolución) en la resolución de problemas lógicos y matemáticos de la época.
Al igual que todos los algoritmos bio-inspirados, estos tipos de algoritmos tuvieron su auge a finales del siglo XX, debido a los avances tecnológicos y computacionales.

En la actualidad, los *Algoritmos de Enjambres* son ampliamente usados en la resolución de problemas de optimización. Esto es debido a que hay una amplia gama de algoritmos dentro de este grupo. Dicha gama aumenta con el tiempo, añadiendo nuevos y sofisticados algoritmos dedicados a la resolución de problemas específicos.

3 Planteamiento del Problema

3.1 Problema Inicial

Comenzaremos este apartado definiendo el **problema** al que deberemos dar solución como un punto de partida.

Nos encontramos ante un conjunto de puntos distribuidos por un espacio (finito de 2 dimensiones), los cuales llamaremos **nodos** a partir de ahora. Podemos indicar que el número aproximado de nodos a tratar varía desde 100 hasta 1.000 nodos.

Estos nodos poseen los siguientes atributos:

1. Un número identificativo del nodo.
2. Un par de coordenadas representando la posición espacial del nodo en un espacio de 2 dimensiones.
3. Un peso en representación de la carga que estos nodos contienen.

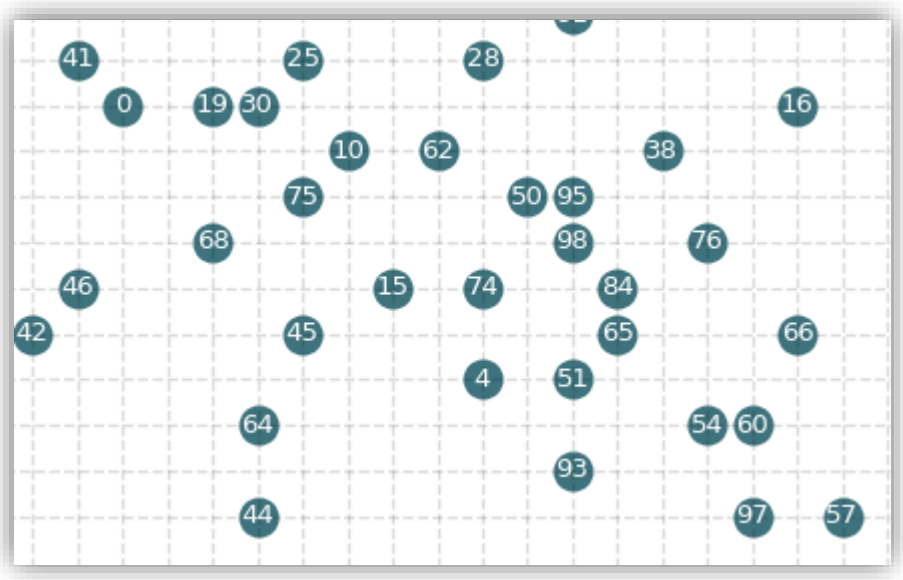


Ilustración 1: Nodos distribuidos *por un espacio bidimensional.*

4 Objetivo

Una vez definido el problema, establecemos como objetivo a cumplir la construcción de un **programa informático** capaz de resolver el problema planteado en el apartado anterior cumpliendo los siguientes **criterios**:

- El programa debe ser capaz de reducir los costes que supone el uso de una determinada **flota de vehículos** al intentar resolver un caso particular del problema (solución a una situación determinada).
- El programa debe ser capaz de reducir los costes en **transporte**. Estos costes derivan directamente de la **distancia** que estos vehículos recorran.
- El programa debe ser capaz de elaborar **rutas** coherentes que conecten los diversos nodos que cada vehículo deba recorrer, tratando de cumplir el criterio anterior.
- El programa debe ser capaz de **segmentar** el conjunto completo de nodos en agrupaciones de nodos cercanos, usando como criterio la distancia entre ellos. Esto permitirá generar una solución para cada grupo de forma independiente.
- El programa debe ser capaz de generar **problemas específicos** aleatorios con los que poder trabajar (llamados **casos**).
- El programa debe ser capaz de **mostrar gráficamente** tanto el problema inicial como los resultados.

5 Planificación

Este apartado reflejará las distintas fases del proyecto, así como la distribución de tiempo utilizada y una breve descripción sobre estas fases.

Tabla 1: Distribución temporal de las diversas fases del proyecto.

Fecha	Tarea	Descripción	Horas
01/02/2019 a 01/03/2019	Descripción de Proyecto	Establecer el Objetivo a alcanzar y las bases para realizar el trabajo.	20
01/03/2019 a --/--/2019	Implementación y Documentación.	Planificar el estilo de trabajo, plantear los problemas directos y estudiar sus posibles soluciones, así como su implementación.	151
16/02/2019 a 19/03/2019	Búsqueda de un algoritmo que resuelva el problema del viajante de forma rápida	Después de estudiar varias opciones se ha optado por desarrollar un algoritmo propio que trace rutas relativamente buenas en poco tiempo. (pudiendo ser mejoradas estas soluciones más adelante) Llamaremos a este algoritmo **TSP_MST**.	25
15/03/2019 a 16/03/2019	Búsqueda de un algoritmo de segmentación para agrupar nodos cercanos	Se ha comenzado obteniendo información sobre algoritmos de segmentación y clustering. Al final se ha optado por elegir el algoritmo **k-means** con la ampliación **k-means++** debido a su velocidad, pues ofrece resultados rápidamente. Esto nos permitirá más adelante elaborar **N** soluciones y compararlas en un tiempo razonable.	5
16/03/2019 a 19/03/2019	Búsqueda y adaptación de una solución al problema de la mochila	Se ha comenzado pensando en elaborar un algoritmo genético para la resolución del problema. Antes de trabajar con el algoritmo, se está adaptando el problema para representar el problema reflejado por este trabajo.	16
19/03/2019 a 21/03/2019	Búsqueda de un algoritmo que mejores los caminos ofrecidos por **TSP_MST**	Como la mejora se basa en eliminar cruces entre caminos, se ha optado por intercambiar el orden en que se visitan nodos contiguos y comprobar si hay mejoría.	10
22/03/2019 a --/03/2019	Limpieza y reorganización del proyecto	Reorganizar el proyecto para que se adapte correctamente a la solución planteada y limpieza de los módulos y utilidades no necesarias.	14

25/03/2019 a 16/03/2019	Implementación de paralelismo	Búsqueda de una forma de incluir paralelismo real en el proyecto de forma que se ejecuten varias soluciones de forma completamente paralela e independiente. Estas soluciones serán posteriormente comparadas. RESULTADO: imposible debido a que el gil es ineludible incluso en ejecuciones independientes, haciendo impredecible la ejecución si este es desactivado.	13
29/03/2019	Cargar y guardar mapas a ficheros	Poder cargar y guardar los datos de un mapa en un fichero para poder ser exportados e importados. Para ello se ha hecho uso de la librería **pickle** y **Cpickle**	2
01/04/2019 a 05/04/2019	Definición e implementación del algoritmo genético de optimización.	Definir e implementar un algoritmo de optimización basado en algoritmos genéticos que trabaje con soluciones crudas como individuos en busca de un óptimo global. Sera uno de los algoritmos a comparar.	17
16/04/2019 a 25/04/2019	Definición e implementación del algoritmo optimización por enjambre de abejas.	Definir e implementar un algoritmo de optimización basado en algoritmos de enjambres que trabaje con soluciones crudas como individuos en busca de un óptimo global. Sera uno de los algoritmos a comparar.	14
16/04/2019 a 15/05/2019	Documentación del proyecto	Documentación interna y externa del proyecto.	35

6 Solución

6.1 Programa Base

Comenzaremos desarrollando la base del programa que albergará todas las funcionalidades que se expresaran a lo largo de este apartado.

6.1.1 Código base

Lo primero a establecer antes de comenzar a desarrollar cualquier tipo de aplicación es que **lenguaje de programación** va a elegirse para desarrollar dicha aplicación y porque motivo.
En este caso decidiremos utilizar el lenguaje de programación **Python** debido a la versatilidad y sencillez que este ofrece frente a la resolución de cualquier tipo de problema estándar y al gran número de librerías disponibles para solventar todo tipo de problemas.
Escogemos Python como lenguaje debido en especial a la existencia de la librería **MatPlotLib** la cual nos permite crear y manipular graficas de forma avanzada y completa.
También cabe destacar como motivo de la elección la familiaridad con dicho lenguaje y con sus librerías, lo cual permitirá un desarrollo fluido del proyecto.

Ilustración 3: Logo de Python.

Ilustración 2: Logo de MatPlotLib.

Una vez tenemos el lenguaje de programación, procedemos a buscar una librería que nos permita desarrollar elementos de uso sencillo e **intuitivo** para el usuario.
Para ello decidimos utilizar la librería de **Qt** para Python, la cual ofrece las funcionalidades de Qt tales como herramientas, diseño de interfaz gráfica, gestión de concurrente de ventanas, etc.
De nuevo destacamos como motivo de la elección la familiaridad con dicha librería.

Para crear la base de nuestro proyecto, primero necesitamos crear un archivo de definición de interfaz gráfica de Qt. Estos archivos tienen la extensión **.ui** y nos

permiten especificar en lenguaje XML todos los aspectos de la interfaz gráfica. Para crear y modificar dicho archivo utilizaremos la herramienta gratuita Qt Creator.

Ilustración 4: Logo de QT Creator.

Una vez hayamos creado el archivo .ui, podemos, mediante una herramienta de Py Qt, transformar el archivo .ui a un código Python que, al ser ejecutado, muestre la interfaz gráfica con todos sus elementos y funcionalidades.

```
pyuic5 -x interfaz.ui -o ventana_ui.py
```

Ilustración 5: Comando Shell que transforma un archivo .ui en código Python.

Después de esto, procederemos a crear un nuevo archivo Python donde comenzara la ejecución de nuestro programa. Este nuevo archivo contendrá una clase que heredará de la clase creada en el archivo que se generó automáticamente.
Esto nos ofrece una gran ventaja, pues nos permite separar la lógica del programa del diseño, pudiendo realizar cambios en la interfaz gráfica sin que esto afecte directamente al código y viceversa.
De esta forma, estableceremos dicho segundo archivo como archivo principal, el cual contendrá la clase principal del programa.

Ilustración 6: Archivos iniciales del proyecto.

6.1.2 Interfaz Gráfica

La interfaz gráfica del proyecto se compondrá de una ventana principal dividida en cuatro secciones:

- Una para mostrar datos de la ejecución.
- Otra para controlar los parámetros y acciones del programa.
- Otras dos, una donde podremos ver la gráfica con los resultados gráficos del programa.
- Y una barra superior donde podremos controlar la gráfica en tiempo de ejecución.

Cada una de estas secciones alberga los elementos de una funcionalidad genérica del proyecto, donde se agrupan los elementos de control, de información y la gráfica donde poder visualizar los resultados.

De esta manera podemos tener control completo de las funcionalidades del programa en una sola ventana, pudiendo alterar los distintos parámetros y realizar las ejecuciones con diversos algoritmos, pudiendo observar los resultados de la ejecución con dichos parámetros en la gráfica y en el cuadro de texto.

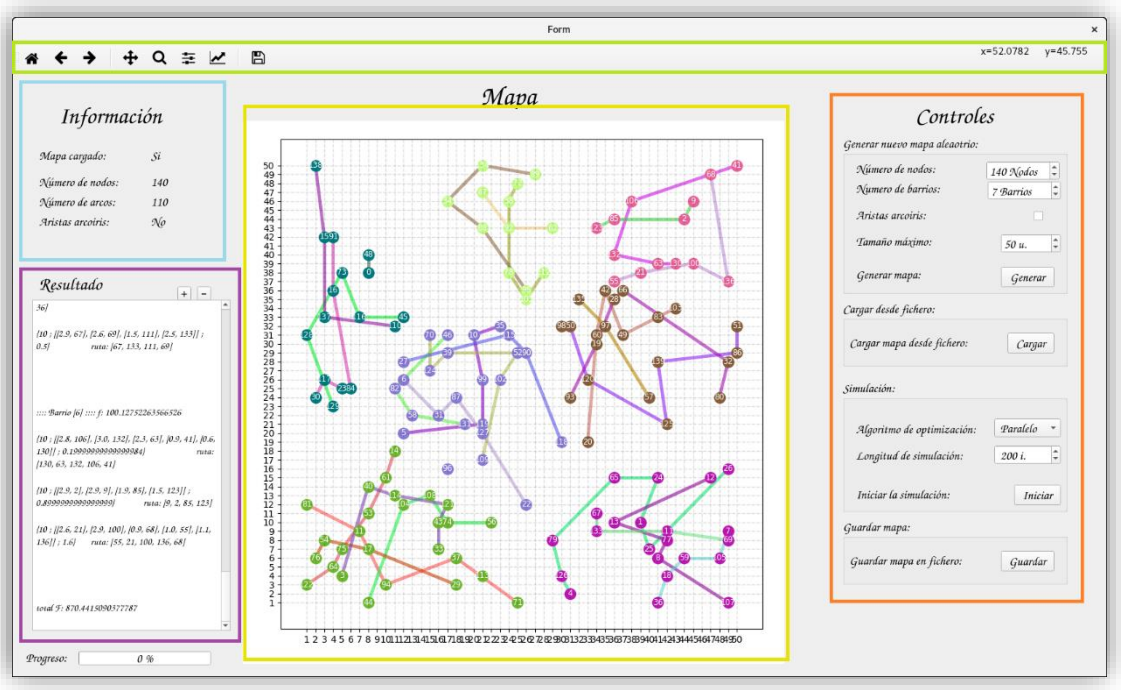


Ilustración 7: Interfaz gráfica de usuario separada en secciones:

- *Verde: Panel de control de la gráfica.*
- *Naranja: Panel de control del programa.*
- *Azul: Panel de información.*
- *Morada: Panel de resultados.*
- *Amarilla: Gráfica.*

6.2 Definición de Datos

Una vez tenemos construido el programa base, pasamos a diseñar el **modelo de datos** que representara un caso.

Puesto que tenemos un conjunto de nodos y, dada la definición de nodo ofrecida en apartado 3 Planteamiento del Problema, vamos a crear una estructura de datos para cada atributo de los nodos.

Estas estructuras apuntarán a ser accesibles de forma genérica, pues lo normal sería acceder a varios nodos o no en función de las necesidades, por lo que se almacenarán los datos en estructuras indexables a las que poder acceder eficientemente.
Usaremos el ID numérico de cada nodo para poder acceder a los datos de un nodo dentro de las estructuras, las cuales estarán ordenadas ascendentemente por estos ID.

Con esta información, procedemos a crear dos listas:

- La primera contendrá los pares de coordenadas de cada nodo, pudiendo acceder a cada par con el ID del nodo.
- La segunda contendrá la carga de cada nodo.

También tendremos un contador donde almacenaremos el número de nodos que hay.
Con esto tenemos definido todos los nodos con sus atributos, pudiendo acceder a los datos de cualquier nodo o a todos ellos de forma ordenada.

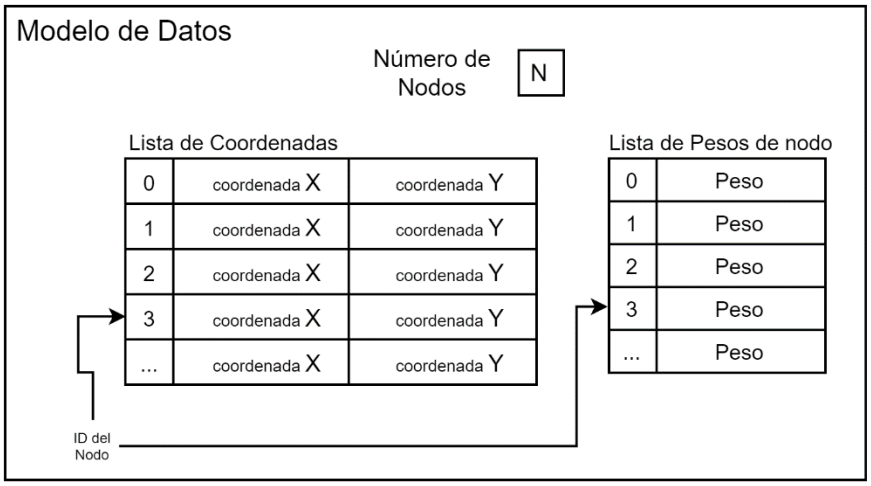


Ilustración 8: Modelo de Datos.

6.2.1 ¿Qué es exactamente un Caso?

Una vez hemos creado el modelo de datos, podemos especificar cualquier problema que albergue las exigencias cumplidas, pudiendo variar el número de nodos, su posición y sus cargas a voluntad.

Por ello, a partir de ahora denominaremos **caso** a cualquier combinación de datos que pueda ser almacenada en el modelo de datos definido anteriormente.

Podemos decir que un caso es un problema concreto que resolver, dentro del abanico de acción del programa.

Un símil de un caso sería la instancia de una clase, donde la clase es el modelo de datos y el caso la instancia particular de la clase.

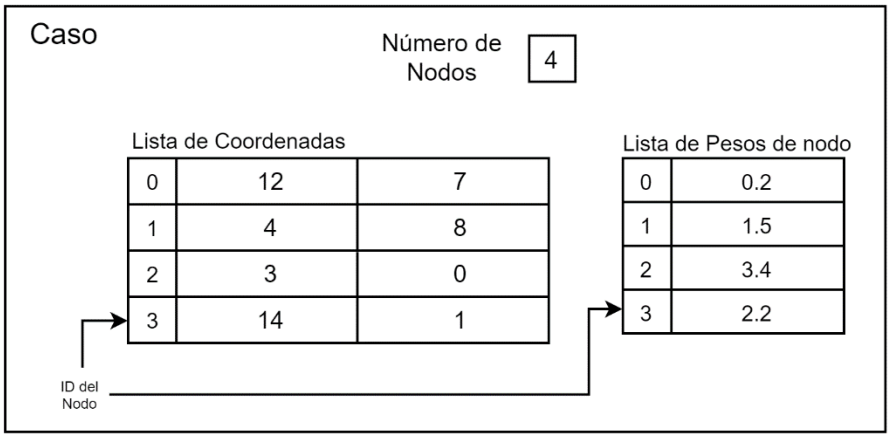


Ilustración 9: Ejemplo de Caso.

6.3 Creación de Casos

Para trabajar con casos distintos y poder probar nuestro programa necesitaremos desarrollar una herramienta que nos permita crear casos representativos del problema a resolver.

Para ello necesitaremos recopilar ciertos parámetros de la interfaz gráfica (configurados previamente por el usuario) y crear un caso acorde a estos parámetros, estableciendo aleatoriamente el resto de parámetros.

Los parámetros a recopilar son:

- Número de nodos a crear.
- Tamaño máximo del mapa.

Con estos datos, la herramienta generadora de casos creará un caso con el **número de nodos** establecido.

Estos nodos se esparcirán aleatoriamente por un espacio cuadrado con lado del **tamaño máximo de mapa**, sin repetir posiciones para evitar que los nodos se solapen. También establecerá un tamaño aleatorio a cada nodo, que representará las **cargas** que estos contienen a la hora de resolver el Problema de la Mochila.

El resultado final es un caso con las estructuras de datos llenas de la información de los nodos.

6.4 Segmentación

Antes de comenzar a resolver un caso, debemos realizar una división de los nodos para poder crear tantas soluciones parciales al problema como grupos hayamos formado.
Posteriormente estas soluciones se recombinarán en una solución completa.

Para llevar esto a acabo necesitamos desarrollar una herramienta que, dada las estructuras de información de un caso y el **número de grupos** a crear, nos devuelva una nueva estructura donde se especifique a que grupo pertenece cada nodo.

La herramienta que utilizaremos se llama **Kmeans++** y pertenece al paquete de algoritmos **scikit-learn**.

Ilustración 10: Logo de scikit-learn.

Esta herramienta, basándose en el algoritmo K-means con la variante K-means++, permite segmentar un conjunto de puntos distribuidos por el espacio, agrupándolos según la distancia de estos puntos, minimizando la **dispersión** de los grupos.

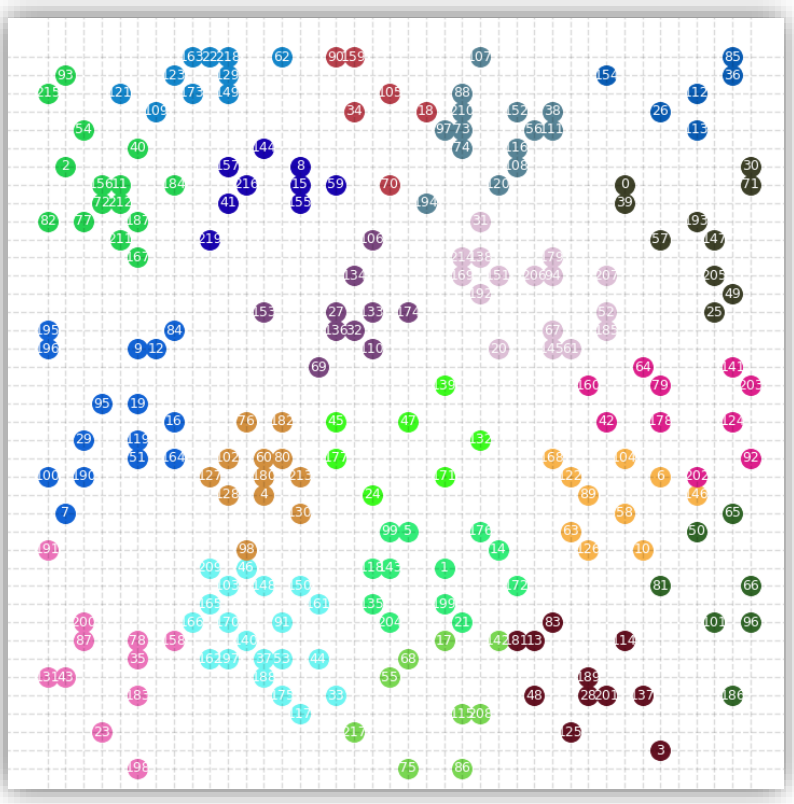


Ilustración 11: Nodos segmentados utilizando el algoritmo K-means++.

Como resultado ofrece una estructura que indica a que grupo pertenece cada nodo, en el mismo orden en el que estos nodos fueron introducidos.

```
Nodos y sus coordenadas:
  -Nodo 0, coordenadas 1, 2.
  -Nodo 1, coordenadas 1, 4.
  -Nodo 2, coordenadas 1, 0.
  -Nodo 3, coordenadas 10, 2.
  -Nodo 4, coordenadas 10, 4.
  -Nodo 5, coordenadas 10, 0.
  -Nodo 6, coordenadas 7, 7.
  -Nodo 7, coordenadas 1, 9.
  -Nodo 8, coordenadas 0, 7.
  -Nodo 9, coordenadas 8, 3.


Resultado del algoritmo KMeans() al segmentar en 3 grupos:
[0 0 0 1 1 1 1 2 2 1]


Esta lista indica a que grupo pertenece cada nodo, donde
la posicion es el ID del nodo y el resultadoel id del grupo:

  -El nodo 0 pertenece al grupo 0
  -El nodo 1 pertenece al grupo 0
  -El nodo 2 pertenece al grupo 0
  -El nodo 3 pertenece al grupo 1
  -El nodo 4 pertenece al grupo 1
  -El nodo 5 pertenece al grupo 1
  -El nodo 6 pertenece al grupo 1
  -El nodo 7 pertenece al grupo 2
  -El nodo 8 pertenece al grupo 2
  -El nodo 9 pertenece al grupo 1
```

Ilustración 12: Resultado estructural del algoritmo K-means++.

Esta estructura pasará a formar parte de los elementos de un caso.

6.5 Creación de Soluciones Iniciales

Llegados a este punto, donde tenemos la base, la interfaz gráfica y los datos del problema (caso), podemos comenzar a construir la parte del programa que generará las primeras soluciones.

Estas primeras soluciones trataran de resolver el problema de la mochila de forma independiente para cada grupo segmentado previamente, los cuales llamaremos a partir de ahora barrios.

Cada barrio, al igual que un caso, está compuesto por un conjunto de nodos con sus ID, sus pesos, sus coordenadas, etc. La diferencia recae en que los barrios son solo una parte de los nodos del caso, concretamente todos los nodos que pertenecen a un mismo grupo, según nos indica la herramienta Kmeans++ anteriormente mencionada.

La forma de generar las soluciones a un barrio se puede desglosar en los siguientes pasos:

1. Crearemos un conjunto de soluciones candidatas, generadas aleatoriamente. Cada solución intentará repartir los nodos por un conjunto de vehículos de diferentes tamaños según el siguiente criterio:

    a. Para cada nodo del barrio obtenido aleatoriamente:
        i. Buscamos en que vehículo del conjunto podemos introducir dicho nodo (haya hueco).
        ii. Si encontramos dicho vehículo, introducimos el nodo y repetimos. Si no, añadimos al conjunto de vehículos un nuevo vehículo. Este nuevo vehículo debe tener espacio suficiente para albergar todos los nodos que queden por repartir en el barrio, o ser el de mayor tamaño posible en otro caso. Cumpliendo este criterio, siempre se ofrecerá el vehículo más pequeño posible.

    b. Una vez repartidos los nodos del barrio, pasamos a evaluar dicha solución, obteniendo una puntuación.
        i. Evaluamos el número de vehículos utilizados en la solución. Usar vehículos más grandes debe salir más caro, pero más rentable si estos se llenan.
        ii. Evaluamos el espacio vacío dejado en cada vehículo. Cuanto menos espacio vacío haya en un vehículo, mejor puntuación recibirá.
        iii. Evaluamos la ruta que dicho vehículo debe realizar para llegar a todos los nodos. Es aquí donde entra en juego la solución al

Problema del Viajante. Necesitaremos crear una herramienta que nos permita obtener rutas subóptimas al conjunto de nodos de cada vehículo. Explicaremos esto en detalle en el siguiente apartado, 6.5.1 Generador de Rutas.

2. Una vez generado el conjunto de soluciones, procederemos a ordenarlas por su puntuación, quedándonos con la mejor de ellas.

Este procedimiento será repetido para cada barrio, obteniendo una solución para cada uno y siendo almacenadas en una estructura de la que se compondrá una **solución completa**.

Una *solución completa*, a parte de la estructura anteriormente mencionada, también estará compuesta de los datos del caso, necesarios para poder llevar a cabo la creación de soluciones parciales; y de la **puntuación total** de la solución, la cual será la suma de las puntuaciones de cada solución parcial.

6.5.1 Generador de Rutas

Como dijimos anteriormente, necesitaremos crear una herramienta que nos permita obtener **rutas** subóptimas a un conjunto de nodos.
Debido a que el programa está enfocado a crear **muchas** soluciones distintas de variada calidad e ir optimizándolas y descartando las peores, podemos decir que el problema debe enfocarse en ofrecer soluciones aceptables en un **tiempo reducido**.

Como solución al problema del viajante utilizaremos un **algoritmo aproximado** de invención propia basado en algoritmos rápidos, como, por ejemplo, el *Algoritmo del Vecino Más Próximo.*

Esta variante, llamada *The Salesman Problem – Minimal Spanning Tree* (*TSP-MST*), toma ejemplo de algoritmos de resolución de árboles de mínima expansión. Dado un conjunto de nodos, el algoritmo *TSP-MST* realiza los siguientes **pasos** explicados brevemente:

- El algoritmo obtiene todos los arcos o conexiones entre nodos existentes.
- Ordena el conjunto de arcos por peso en orden ascendente.
- Después comienza extraer arcos del conjunto:
    - Si se cumplen una serie de condiciones se inserta el arco en la solución.
    - Repite el proceso hasta que todos los nodos estén conectados entre sí.
- Por último, obtiene la ruta como un conjunto de nodos ordenados y el coste de recorrer esta ruta, ofreciendo ambos elementos como solución.

El algoritmo *TSP-MST* funciona considerablemente rápido para un conjunto reducido de nodos, ofreciendo soluciones aceptables.

El problema de este algoritmo es que genera, en determinadas situaciones locales, caminos incoherentes que, aun siendo perfectamente válidos, son evidentemente inapropiados al realizar trazos innecesarios como "*cruces de caminos*" o "*codos de ángulo fino*".

Para solucionar esto, desarrollaremos una mejora del algoritmo *TSP-MST*, llamada *TSP-MST++*, que consistirá en mejorar la solución ofrecida por el algoritmo antes de terminarla. Con esta mejora podremos reducir incoherencias locales, mejorando el resultado.

Mas información sobre el funcionamiento de los algoritmos *TSP-MST* y *TSP-MST++* en el Anexo 1.

6.6 Algoritmos de Optimización

En este apartado trataremos de mejorar las soluciones obtenidas hasta ahora en busca de soluciones mejores.

Para ello vamos a desarrollar dos algoritmos de optimización bio-inspirados que, dado un conjunto de soluciones, realicen diversas operaciones sobre estas soluciones tratando de alcanzar una solución mejorada.

6.6.1 Algoritmo Genético

El primer algoritmo se trata de un Algoritmo Genético.

Este algoritmo, al igual que cualquier algoritmo genético, trabaja con un grupo de individuos que trata de mejorar, basándose en las reglas de la evolución biológica y su base genético-molecular.

En este caso, los individuos que el algoritmo necesita son las soluciones completas, candidatas a mejorar.

El algoritmo genético que procedemos a construir deberá seguir los siguientes pasos:

1. Creamos un conjunto inicial de soluciones. A este conjunto lo llamaremos *población*.
2. Para un número determinado de épocas, repetimos los siguientes pasos, los cuales representan el transcurso de una época:
    a. Ordenamos la población en función de su puntuación (puntuación total de una solución completa) en orden ascendente.

b. Realizamos la **reproducción** de la población, la cual consisten en obtener soluciones nuevas a partir de las actuales mediante los siguientes métodos:
    i. Un 5 % de la nueva población se compondrá de las **mejores** soluciones de la población anterior, las cuales se mantendrán inalteradas.
    ii. Otro 5 % serán soluciones **nuevas**.
    iii. El resto de las soluciones nuevas serán producto del **cruce** de dos soluciones. Para elegir a los padres, escogeremos en cada caso a cada una de las soluciones de la población, siendo el otro padre el mejor de dos soluciones escogidas aleatoriamente. Este cruce consiste en crear dos nuevas soluciones, una con los mejores barrios y otra con los peores, dando como resultado una solución con lo mejor de los padres y una con lo peor.
    iv. Una vez creado cada solución hija de la nueva población, estas tienen un pequeño porcentaje de sufrir un proceso de **mutación**, mediante el cual se cambiará una pequeña característica de la solución. La mutación de una solución consiste en, para un barrio aleatorio, escogeremos un nodo aleatorio de uno de sus vehículos y lo intercambiaremos por otro nodo similar aleatorio de otro de los vehículos del mismo barrio (entendiendo por nodos similares todo par de nodos cuya diferencia de peso del nodo de mayor peso respecto del nodo de menor peso se menor a un 10% del tamaño del nodo mayor). De esta forma se ven alteradas tanto las rutas de ambos vehículos como sus puntuaciones y, por ende, la puntuación del barrio.
    v. Como el proceso de cruce se realiza para cada solución de la población, devolviendo dos soluciones nuevas, obtenemos demasiadas soluciones que superan el tamaño de nuestra población. Como solución y, para mantener el tamaño de la población constante, eliminaremos los peores hijos generados para que el número de hijos obtenidos coincida con el 90 % de la población.

3. Tras procesar todas las épocas, ordenamos la población una última vez, obteniendo la **mejor** solución que ha podido producirse en todo el proceso.

Esta solución obtenida mediante el algoritmo genético pasa a ser una **solución mejorada** y se convierte en un resultado factible que poder ofrecer al **usuario**.

El número de épocas que el algoritmo realizará y, por tanto, el **tiempo de ejecución**, dependerán de un parámetro que el usuario podrá especificar en la interfaz gráfica.

6.6.2 Algoritmo de Enjambres

El segundo algoritmo está basado en un Algoritmo de Enjambres.

Concretamente, en un *Algoritmo de Colonia de Abejas* (*ABC* por sus siglas en inglés). Estos algoritmos tratan de moverse por el espacio n-dimensional de soluciones de un problema en busca de un óptimo global.

El algoritmo, simulando el comportamiento natural de los enjambres de abejas, trata de encontrar siempre una solución mejor buscándola en un entorno local a la mejor solución actual. Si al realizar un número determinado de intentos no lo consigue, pasa a explorar otro espacio local distinto, lo que equivaldría a escoger una nueva solución y comenzar a buscar mejores soluciones cercanas.

En nuestro caso, el algoritmo parte de una solución inicial:

1. El algoritmo, durante un determinado **número** de intentos, comienza a **modificar** la solución en busca de mejorarla. El proceso de modificación es similar al proceso de **mutación** mencionado en el apartado de 6.6.1 Algoritmo Genético.
    a. Si consigue **mejorar** la solución, volverá a empezar, reiniciando el número de iteraciones realizadas e intentando ahora mejorar la solución mejorada.
    b. Si se realizan el máximo número de intentos de modificación, **descarta** la solución y genera una **nueva** solución inicial, volviendo a intentar mejorarla.
    c. En cualquiera de los casos, el algoritmo siempre guarda la mejor solución que haya encontrado, actualizándola cuando alguna solución sea mejor.
2. Al terminar, devuelve la **mejor** solución que ha encontrado en todo el proceso de búsqueda.

El número de soluciones que el algoritmo tratara de mejorar y, por tanto, el **tiempo de ejecución**, dependerán de un parámetro que el usuario podrá especificar en la interfaz gráfica.

6.7 Ejecución paralela

Como tipo de ejecución **adicional** ofrecida al usuario, se encuentra la Ejecución Paralela.

Este tipo de ejecución combina el **paralelismo computacional** con las dos soluciones anteriores para ofrecer un numero plural de soluciones mejoradas en un tiempo reducido, pudiendo obtener la mejor de ellas.

En computación, el paralelismo permite ejecutar varias tareas (procesos) a la vez sin que el **rendimiento temporal** de estas tareas decaiga. El número de tareas que cualquier computador puede ejecutar simultáneamente está estrechamente ligado al número de núcleos del procesador.

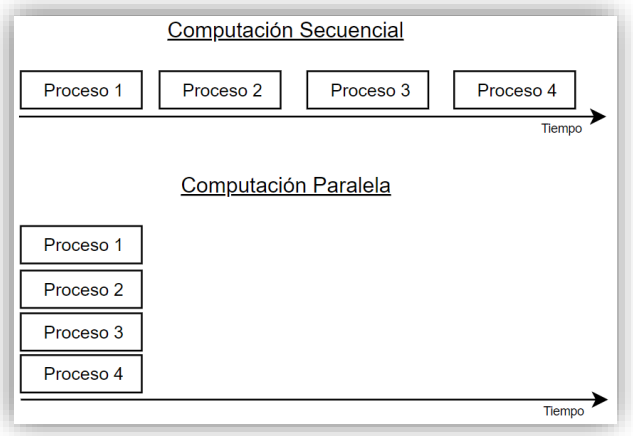


Ilustración 13: Computación secuencial frente a computación paralela.

Dicho esto, pueden obtenerse dos ventajas:

- Realizar una misma tarea en **menor tiempo**. Para ello, el problema debe fragmentarse, si es que es posible, y deben resolverse los fragmentos por separado, conminándolos después para forjar la solución.
- Realizar **varias tareas** en el tiempo de una sola. Esto permite obtener mayores cantidades de resultados en el tiempo que tardaría un solo resultado, aproximadamente.

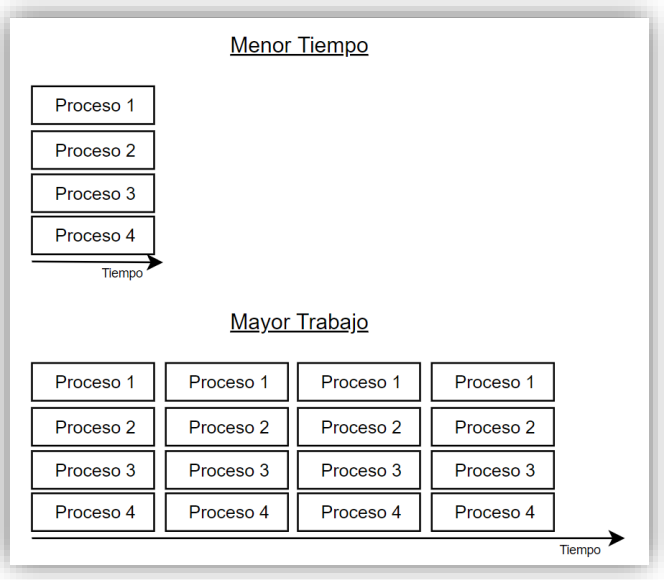


Ilustración 14: Ventajas de la computación paralela.

En nuestro caso aprovecharemos la segunda ventaja, produciendo varias soluciones mejoradas en tiempo reducido y seleccionando la mejor de ellas.

El algoritmo de ejecución paralela tratará de ejecutar **simultáneamente** los dos algoritmos de optimización anteriores de la siguiente forma:

1. Obtenemos el **número** de núcleos del sistema.
2. **Dividimos** las ejecuciones:
    a. La mitad de las ejecuciones serán de Algoritmos Genéticos.
    b. La otra mitad de las ejecuciones serán Algoritmos de Enjambres.
3. **Preparamos** las ejecuciones para ser lanzadas, rellenando las estructuras de datos que ambos tipos de algoritmos necesitan. Los algoritmos tratan de ejecutase en un tiempo similar, pues se inician con los mismos parámetros.
4. **Ejecutamos**.
5. **Esperamos** a que terminen las ejecuciones.
6. Cuando todas las ejecuciones han terminado, **obtenemos** sus soluciones mejoradas, ordenándolas y devolviendo al usuario la mejor de estas junto con un ranking de puntuaciones de todas las soluciones mejoradas.

6.8 Formato del resultado

La forma en la que una solución mejorada se presenta al usuario se establece de dos formas:

6.8.1 Resultado Gráfico

La solución se presenta al usuario como una gráfica donde se puede visualizar algunos datos del caso, como la posición, numeración de nodos y la segmentación de estos, separadas por colores.

La forma de representar la solución consiste en, para cada vehículo de la solución, se dibujará un **trazo** de un color aleatorio característico del vehículo. Dicho trazo representa la ruta del vehículo, es decir, recorre los nodos que dicho vehículo debe recorrer en el orden en el que debe recorrerlos.

Es importante enfatizar que las rutas mostradas en la solución grafica **NO** representan calles o caminos existente, sino distancias euclídeas entre nodos.

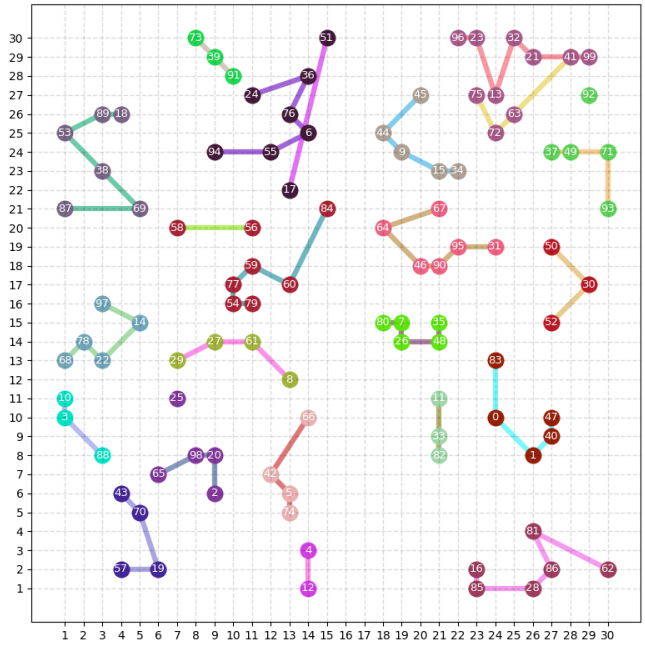


Ilustración 15: Resultado grafico de la ejecución del programa.

6.8.2 Resultado Textual

La solución se representa al usuario como un texto donde aparecen exclusivamente los detalles de la solución.

El formato del texto es el siguiente:

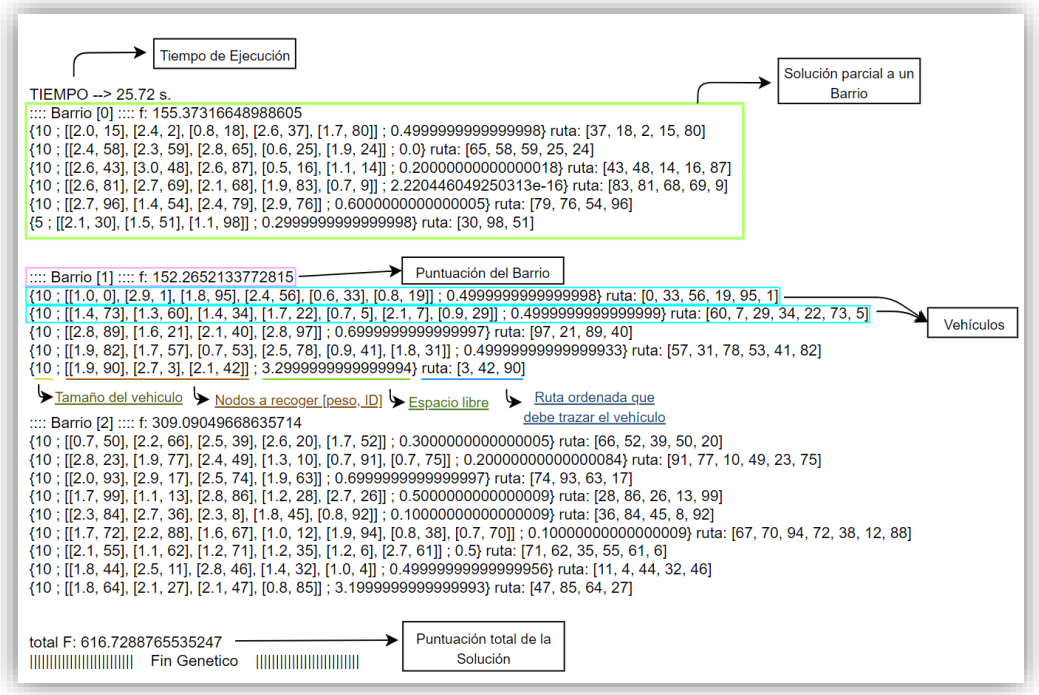


Ilustración 16: Resultado textual del programa explicado.

7 Resultados

En este apartado realizaremos un proceso de ejecución de los tipos de ejecución disponibles, obteniendo los resultados de cada uno de ellos y organizándolos en una tabla.

Para cada uno de los tipos de ejecución se realizarán las siguientes baterías de pruebas:

- Se realizará una ejecución de **corta, media y larga** duración para los siguientes tipos de mapa:
    - Para un mapa de **100** nodos dividido en **2** barrios.
    - Para un mapa de **100** nodos dividido en **8** barrios.
    - Para un mapa de **100** nodos dividido en **20** barrios.
- Se realizará una ejecución de **media** duración para los siguientes tipos de mapa:
    - Para un mapa de **300** nodos dividido en **2** barrios.
    - Para un mapa de **300** nodos dividido en **8** barrios.
    - Para un mapa de **300** nodos dividido en **20** barrios.
    - Para un mapa de **600** nodos dividido en **8** barrios.

El mapa es el mismo para cada grupo de ejecuciones con las mismas propiedades de número de nodos y barrio, pero es diferente si alguno de los parámetros cambia, por lo que no deben compararse las puntuaciones.

Como recordatorio, establecemos que puntuaciones más bajas representan soluciones menos costosas y, por tanto, mejores.

7.1 Resultados del Algoritmo Genético

Tabla 2: Resultados de ejecución del algoritmo genético.

N.º	N.º de nodos	N.º de barrios	Tipo de ejecución	Tiempo de ejecución (segundos)	Puntuación
1	100	2	Corta	15,67	691,4
2	100	2	Media	53,16	661,53
3	100	2	Larga	134,11	647,29
4	100	8	Corta	15,11	424,10
5	100	8	Media	51,81	404,06
6	100	8	Larga	129,3	403,02
7	100	20	Corta	14,07	426,52
8	100	20	Media	46,71	426,52
9	100	20	Larga	111,98	426,52
10	300	2	Media	185,46	4213,46
11	300	8	Media	169,51	2087,70
12	300	20	Media	156,52	1504,32
13	600	8	Media	331,89	5993,41

7.2 Resultados del Algoritmo de Enjambres

Tabla 3: Resultados de ejecución del algoritmo de enjambres.

N.º	N.º de nodos	N.º de barrios	Tipo de ejecución	Tiempo de ejecución (segundos)	Puntuación
1	100	2	Corta	19,41	673,48
2	100	2	Media	67,41	662,45
3	100	2	Larga	169,23	655,62
4	100	8	Corta	15,95	424,23
5	100	8	Media	54,17	426,44
6	100	8	Larga	133,69	423,84
7	100	20	Corta	14,87	426,52
8	100	20	Media	42,33	426,52
9	100	20	Larga	124,73	426,52
10	300	2	Media	231,07	4244,90
11	300	8	Media	178,79	2196,38
12	300	20	Media	158,44	1609,18
13	600	8	Media	351,35	6276,86

7.3 Resultados de la Ejecución Paralela

Tabla 4: Resultados de ejecución de la ejecución paralela.

N.º	N.º de nodos	N.º de barrios	Tipo de ejecución	Tiempo de ejecución (segundos)	Puntuación
1	100	2	Corta	38,69	644,69
2	100	2	Media	136,34	647,56
3	100	2	Larga	346,08	641,59
4	100	8	Corta	32,68	423,19
5	100	8	Media	116,42	407,83
6	100	8	Larga	299,11	397,77
7	100	20	Corta	28,87	426,52
8	100	20	Media	114,02	426,52
9	100	20	Larga	279,53	426,52
10	300	2	Media	477,3	4136,72
11	300	8	Media	388,25	2072,58
12	300	20	Media	346,09	1516,43
13	600	8	Media	770,48	5985,61

8 Conclusiones

En este apartado contemplaremos las conclusiones que pueden sacarse de los resultados obtenidos del apartado anterior.

- Para ejecuciones de corta duración, el algoritmo de enjambres parece ofrecer resultados ligeramente mejores.

- Para ejecuciones más duraderas, el algoritmo genético ofrece mejores resultados que el algoritmo de enjambres.

- La ejecución paralela ofrece soluciones significativamente mejores que los dos algoritmos por separado, a costa de una ejecución más duradera que suele duplicar el tiempo de ejecución de los otros dos algoritmos.

- En la ejecución 7, 8 y 9 puede apreciarse que se ha alcanzado un mínimo global, pues todas las ejecuciones reflejan el mismo resultado, independientemente de la efectividad.

- En ejecuciones más longevas, los algoritmos genéticos y de enjambres difieren, marcándose aún más las diferencias explicadas en los puntos anteriores.

- En una ejecución con un alto número de nodos, la ejecución paralela no parece ofrecer una mejoría pertinente frente al algoritmo genético. Quizás sea necesario mayor tiempo de ejecución para permitir una mayor indagación y búsqueda de mejoras.

- En general, todos los algoritmos ofrecen soluciones relevantes en un tiempo aceptable.

9 Manual de Usuario

9.1 Instalación

Para instalar correctamente el programa y poder ejecutarlo hay que seguir las explicaciones expuestas en los siguientes apartados:

9.1.1 Programa principal

El programa se compone de una serie de archivos que contienen todas las herramientas y funciones que necesita el programa para su correcto funcionamiento. Dichos archivos han de permanecer siempre juntos en el mismo directorio.

Por lo tanto, para realizar una instalación correcta del programa deberemos situar **todos** los archivos de este en un **mismo directorio** cualquiera del sistema de archivos.

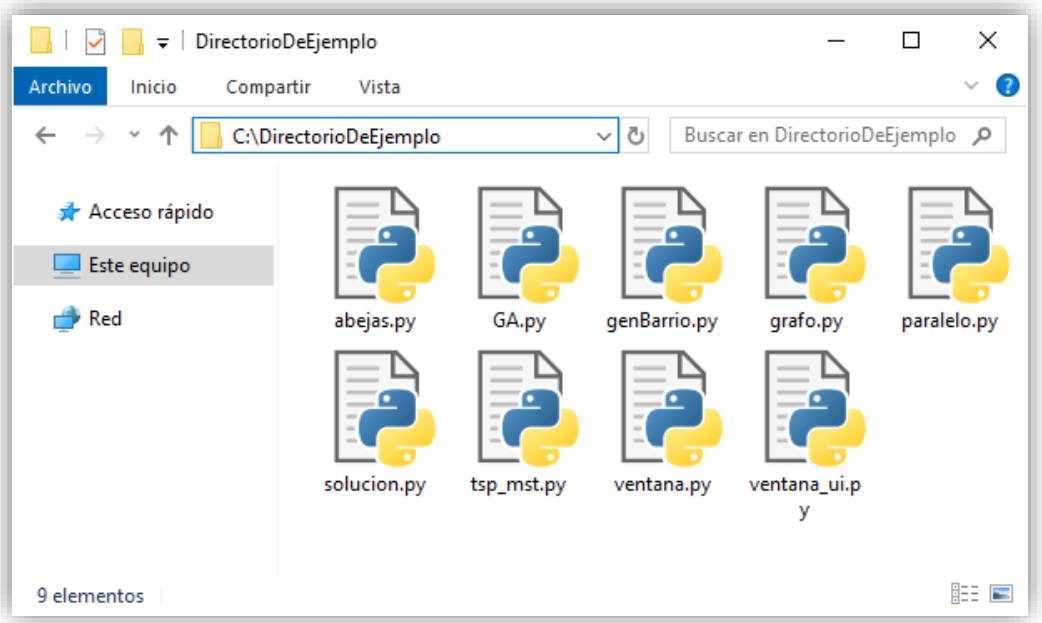


Ilustración 17: Directorio con los archivos del programa.

9.1.2 Librerías

En adición a los archivos del programa, para que el programa funcione correctamente es necesario instalar una serie de librerías, cuya instalación se explica a continuación:

- Python 3.7:
  - El programa ha sido construido y probado utilizando la versión 3.7 de Python. Por este motivo, se recomienda su uso.
  - Para instalar Python de forma correcta en un tipo concreto de sistema, dirigirse a la página WEB principal de Python y seguir la instalación dedicada al tipo de sistema en cuestión.

- PyQt 5:
  - El programa utiliza las funciones de Qt adaptadas a Python (PyQt) en la construcción de la interfaz gráfica, así como la lógica que permite a esta funcionar.
  - Para instalar PyQt 5 deberemos ingresar el siguiente comando en una consola:
  - pip install pyqt5

- MatPlotLib 3:
  - MatPlotLib nos permite crear y modificar graficas con una amplia variedad de herramientas y opciones. En el programa es utilizada para controlar la gráfica principal.
  - Para instalar MatPlotLib deberemos ingresar el siguiente comando en una consola:
  - pip install matplotlib

- Numpy:
  - Numpy es una librería llena de herramientas para trabajar con vectores y matrices N-dimensionales, numero aleatorios, etc.
  - Para instalar Numpy deberemos ingresar el siguiente comando en una consola:
  - pip install numpy

9.2 Ejecución

Para ejecutar el programa deberemos abrir una consola del sistema y dirigirnos al directorio en el cual se encuentre instalado el programa.
Una vez allí, procederemos a ejecutar el código Python del archivo "*ventana.py*"

```
python ventana.py
```

Esto iniciará la ejecución del programa, mostrando la interfaz de usuario.

9.3 Interfaz Gráfica

La interfaz gráfica del proyecto se compondrá de una ventana principal dividida en cinco secciones:

- Una para mostrar datos de la ejecución.
- Otra para mostrar datos textuales del resultado.
- Otra para controlar los parámetros y acciones del programa.
- Otras dos, una donde podremos ver la gráfica con los resultados gráficos del programa.
- Y una barra superior donde podremos controlar la gráfica en tiempo de ejecución.

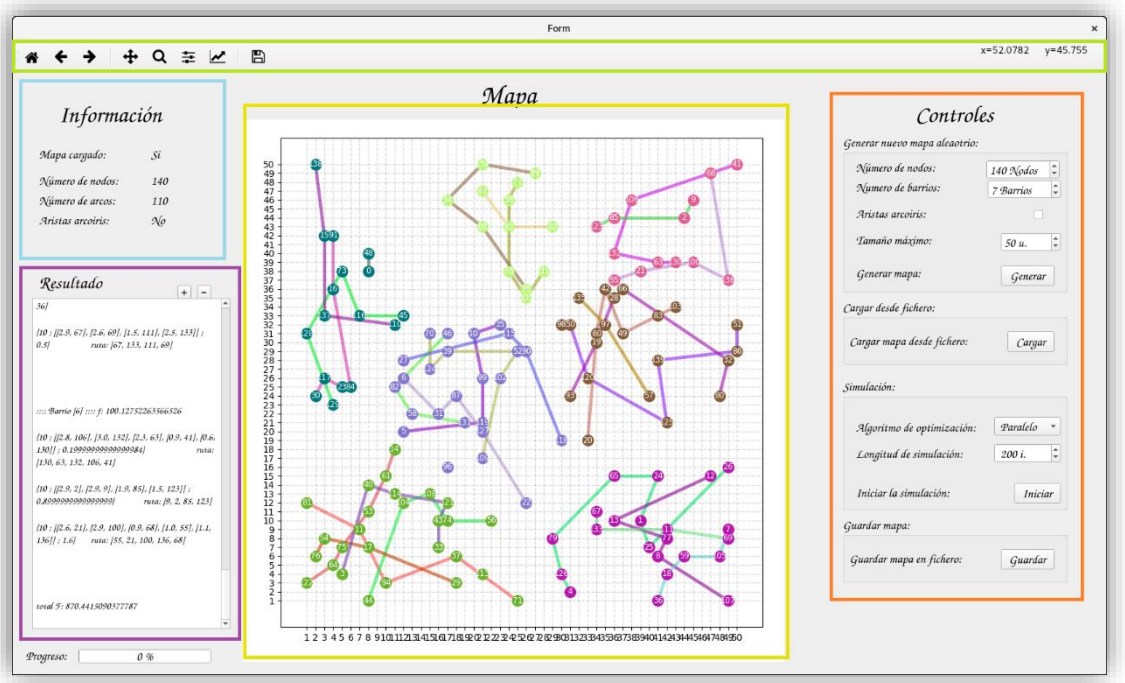


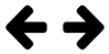
Ilustración 18: Interfaz gráfica de usuario separada en secciones:

- *Verde: Panel de control de la gráfica.*
- *Naranja: Panel de control del programa.*
- *Azul: Panel de información.*
- *Morada: Panel de resultados.*
- *Amarilla: Gráfica.*

9.3.1 Panel de Control de la Grafica

Este panel proporciona las siguientes herramientas para el control de la gráfica:


- **Restablecer**: restablece la gráfica a su posición original.


- **Avanzar/Retroceder**: avanza o vuelve a una configuración posterior o anterior de la gráfica.


- **Mover**: permite mover la vista de la gráfica.


- **Zoom**: Permite ampliar (clic izquierdo) o reducir (clic derecho) la vista al dibujar la sección en la gráfica.


- **Configuración avanzada**: muestra la configuración avanzada numérica de las diversas propiedades de la gráfica.


- **Guardar**: Permite guardar la gráfica como una imagen.

También encontraremos las coordenadas en la gráfica en las que se encuentra nuestro cursor en el extremo derecho de este panel.

9.3.2 Panel de control del Programa

El panel de control del programa se divide en cuatro secciones o paneles, cada uno de ellos dedicado a una funcionalidad concreta.

En el **primer** panel, etiquetado como "*Generar nuevo mapa aleatorio:*", encontramos los diversos parámetros modificables en la generación de un mapa aleatorio. Estos son:

- **Numero de nodos**: Número de nodos del nuevo mapa.
- **Numero de barrios**: Número de grupos en los que se dividirá el conjunto de nodos.
- **Aristas arcoíris**: Cuando esta opción está marcada, todas las aristas dibujadas en la gráfica pasan a tener colores aleatorios.
- **Tamaño máximo**: Tamaño del espacio cuadrado 2D, es decir, longitud del lado. Todos los nodos serán generados dentro del cuadrado.

También encontramos un botón etiquetado como "Generar mapa", el cual, al ser pulsado, inicia el proceso de generación de un nuevo mapa aleatorio configurado con los parámetros anteriormente mencionados.

El **segundo** panel, etiquetado como "*Cargar desde fichero:*", contiene un botón que nos permite cargar un mapa desde un fichero externo, así como los datos del caso.

En el **tercer** panel, etiquetado como "*Simulación:*", encontramos una lista desplegable y un panel numérico.

- En la lista podremos elegir el tipo de ejecución deseada entre los tres tipos disponibles (Algoritmo Genético, Algoritmo de Abejas y Ejecución paralela).
- En el panel numérico podremos indicar la duración de la simulación en individuos creados. La creación de cada individuo tardará una duración fija dependiente del dispositivo que ejecute el programa.

En este panel también encontraremos un botón etiquetado como "*Iniciar la simulación*", el cual, al ser pulsado, iniciara la simulación utilizando el algoritmo seleccionado durante un máximo de tiempo dependiente al número de individuos establecido.

El **cuarto** panel, etiquetado como "*Guardar mapa:*", contiene un botón que nos permite guardar un mapa en un fichero externo junto con los datos de configuración del caso.

9.3.3 Panel de Información

En el panel de información del programa encontramos ciertas etiquetas seguidas de sus datos.
Estos datos nos muestran información útil como el número de nodos del mapa o el número de arcos de la solución.

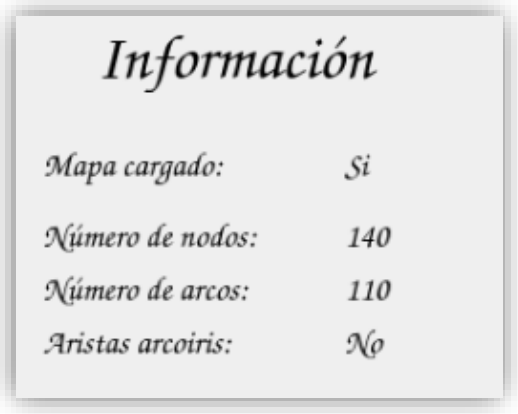


Ilustración 19: panel de información del programa.

9.3.4 Panel de Resultados

En este panel encontraremos un cuadro de texto. Este cuadro mostrará el resultado textual de las ejecuciones, es decir, las soluciones, cuando estas terminen, así como cierta información útil (Ej.: si un archivo ha sido cargado con éxito o no).

Podemos desplazarnos por todo el cuadro de texto utilizando la barra desplazadora situada a la derecha del cuadro.

El texto podrá ser seleccionado y copiado, permitiendo al usuario marcar ciertas partes (seleccionando) o copiar información fuera del programa (copiando el texto).

También encontraremos dos botones situados en la parte superior derecha del cuadro. Estos botones, etiquetados como "+" y "-", permiten aumentar y reducir el tamaño del texto dentro del cuadro de texto.

9.3.5 Gráfica

En este panel encontraremos la representación gráfica del caso. Dicha representación se compone de:

- Un espacio bidimensional, el cual está dividido, por una rejilla, en unidades de medida.
- Los nodos del caso, distribuidos cada uno en su posición espacial. Dentro del nodo esta dibujado su ID. El color del nodo representa el grupo al que pertenece.
- Las aristas, las cuales unen dos nodos y serán mostradas una vez se haya calculado y mostrado una solución. El color de la arista representa que vehículo la está recorriendo.

9.3.6 Ejecución completa

Para realizar un uso completo del programa, debemos, una vez abierto el programa, dirigirnos al panel de control del programa.

Si deseamos crear un nuevo mapa aleatorio, nos dirigiremos al apartado "*Generar nuevo mapa aleatorio:*". Deberemos establecer los parámetros del caso. Una vez hayamos establecido los parámetros, pulsaremos el botón "*Generar*".

Si deseamos cargar un mapa desde un fichero, pulsaremos el botón "*Cargar*" del apartado "*Cargar desde fichero:*" y seleccionaremos dicho fichero.

Una vez el caso está definido, iremos al apartado "*Simulación:*". En este apartado seleccionaremos el tipo de ejecución que deseemos y el tiempo de ejecución. Una vez seleccionados, pulsamos el botón "*Iniciar*". El programa procederá a iniciar la simulación.

Mientras el programa está ejecutando la solución podemos contemplar la salida por consola, la cual nos brindará información del estado de la ejecución.

Una vez el programa ha terminado, nos mostrará la solución en la interfaz gráfica de usuario dentro de los paneles "*Gráfica*" y "*Panel de resultados*".

10 Manual de Programador

10.1 Estructura del Proyecto

El programa está formado por una serie de archivos que contienen las funciones y herramientas necesarias para un correcto funcionamiento.

Estos archivos son:

- ventana_ui.py
- ventana.py
- abejas.py
- paralelo.py
- GA.py
- tsp_mst.py
- solucion.py
- genbarrio.py

Dichos archivos se relacionan entre ellos de una forma determinada para que el programa funcione. A continuación, se muestra dicha relación en un diagrama UML:

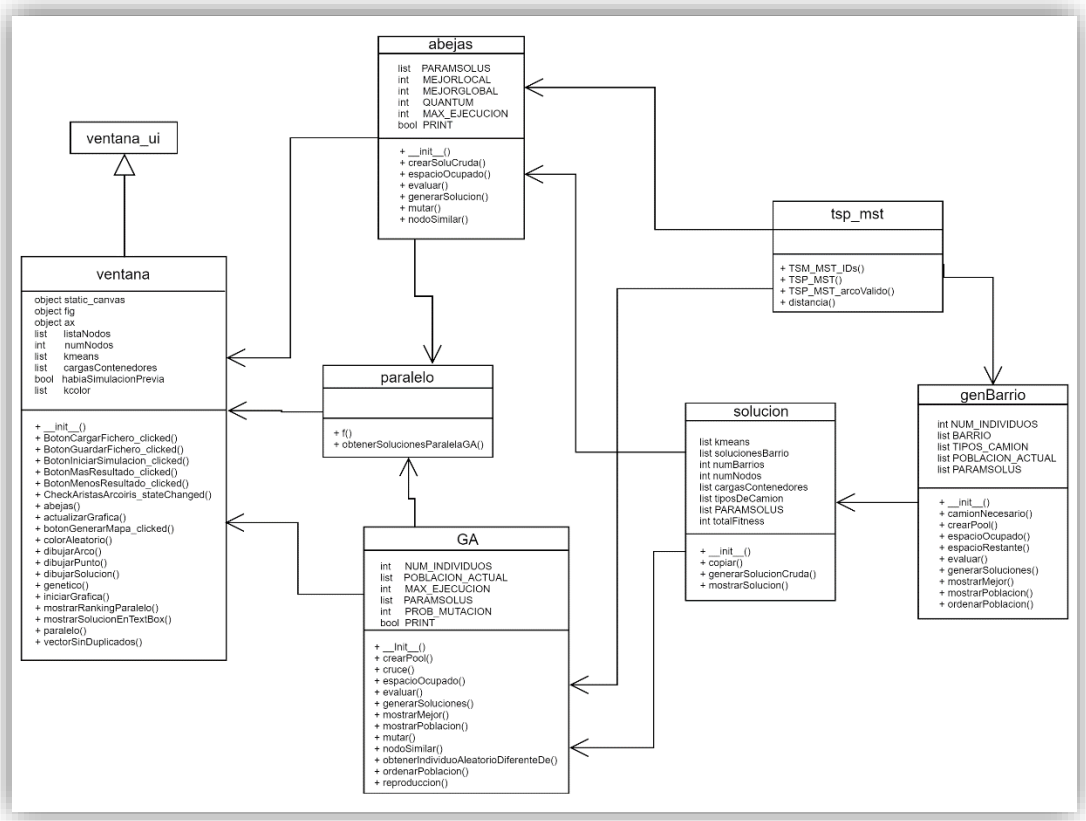


Ilustración 20: Diagrama de Clases del proyecto.

En adición para definir la estructura del proyecto, definiremos las funciones que un usuario puede realizar. Dichas funciones representan las diversas acciones que un usuario puede llevar a cabo utilizando el programa.

A continuación, se muestran las funciones mencionadas anteriormente en un diagrama UML:

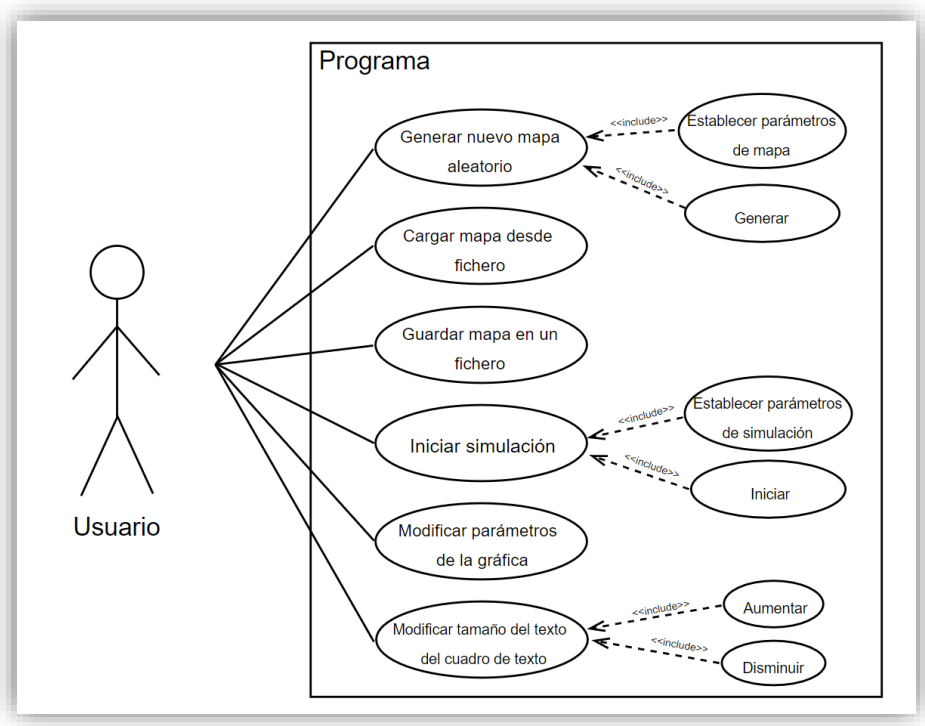


Ilustración 21: Diagrama de casos de uso del programa.

10.2 Documentación

A continuación, se muestra la documentación interna del proyecto en formato PyDoc. Dicha documentación se encuentra separada por módulos (archivos del proyecto) y sigue siempre el mismo formato:

- Descripción del modulo
- Librerías utilizadas
- Clases
    - Descripción de la clase
    - Funciones de la clase
    - Atributos de la clase
    - Extras
- Funciones del modulo

10.2.1 ventana_ui

-*- coding: utf-8 -*-

Modules

PyQt5.QtCore PyQt5.QtGui PyQt5.QtWidgets

Classes

builtins.object
    Ui_MainWindow

class **Ui_MainWindow**(builtins.object)

Methods defined here:

retranslateUi(self, MainWindow)

setupUi(self, MainWindow)

10.2.2 ventana

Clase principal de la interfaz grafica. Diseñada para tratar la funcionalidad de manera aislada al diseño.

Modules

GA	PyQt5.QtWidgets	numpy	random
PyQt5.QtCore	abejas	paralelo	time
PyQt5.QtGui	matplotlib.lines	pickle	

Classes

PyQt5.QtWidgets.QMainWindow(PyQt5.QtWidgets.QWidget)
    MainWindow(PyQt5.QtWidgets.QMainWindow, ventana_ui.Ui_MainWindow)
ventana_ui.Ui_MainWindow(builtins.object)
    MainWindow(PyQt5.QtWidgets.QMainWindow, ventana_ui.Ui_MainWindow)

class **MainWindow**(PyQt5.QtWidgets.QMainWindow, ventana_ui.Ui_MainWindow)

Clase principal que controla la GUI y las funciones iniciales.

Esta clase contiene las funciones utilizadas para:

- El control de la ventana (GUI).
- La creación de nuevos mapas aleatorios.
- La importación y exportación de mapas desde/a archivos.
- Establecimiento y recolccion de parametros necesarios para la ejecucion.
- Selección de tipo de simulación e inicio de la misma.
- Muestra de resultados gráficos y detallados (texto).

Method resolution order:

MainWindow
PyQt5.QtWidgets.QMainWindow
PyQt5.QtWidgets.QWidget
PyQt5.QtCore.QObject
sip.wrapper
PyQt5.QtGui.QPaintDevice
sip.simplewrapper
ventana_ui.Ui_MainWindow
builtins.object

Methods defined here:

BotonCargarFichero_clicked(self)

Carga un mapa desde un fichero elegido por el usuario.

BotonGuardarFichero_clicked(self)

Guarda el mapa en un fichero elegido por el usuario.

BotonIniciarSimulacion_clicked(self)

Función ejecutada al pulsar el botón BotonIniciarSimulacion.

Llama a la funcion conveneiente dependiendo del tipo de ejecucion
que haya decidido el usuario.

BotonMasResultado_clicked(self)

Amplia el tamaño de texto del textBox de la GUI.

BotonMenosResultado_clicked(self)

Reduce el tamaño de texto del textBox de la GUI.

CheckAristasArcoiris_stateChanged(self)

Función que controla el uso de aristas arcoiris.

__init__(self, *args, **kwargs)

Funcion que realiza las operaciones iniciales de la clase MainWindow.

Realiza las conexiones entre los elementos de la GUI y el código
que estos elementos deben ejecutar.
También inicia la gráfica con los parámetros correctos.

abejas(self, devolver=False)

Funcion que inicia una simulacion basada en algoritmos de enjambres, mostrando u
n resultado.

Inicia la simulacion con los parametros indicados por el mapa, la cual devuelve una s
olucion
y esta es mostrada al usuario de forma grafica y detallada (texto).

actualizarGrafica(self)

Metodo que limpia y rellena la grafica con los datos actuales de la lista de nodos.

botonGenerarMapa_clicked(self)

Función ejecutada al pulsar el botón BotonGenerarMapa.

Crea un nuevo mapa aleatorio con los parámetros establecidos por
el usuario en diversos elementos de la GUI.

colorAleatorio(self)

Función que devuelve un color aleatorio.

Devuelve un color aleatorio, ni demasiado claro, ni demasiado
oscuro, en el siguiente formato:
- '#RRGGBBFF' (donde el ultimo FF es el canal alfa siempre opaco).

dibujarArco(self, Xo, Yo, Xd, Yd, col='dodgerblue')

Método que dibuja una línea desde un origen a un destino.

parámetros:
Xo - coordenada X del origen.
Yo - coordenada Y del origen.
Xd - coordenada X del destino.
Yd - coordenada Y del destino.
col - color de la línea en un formato de MatPlotLib valido.

dibujarPunto(self, x, y, ID, col='black')

Metodo que dibuja un punto en la grafica con su numero identificador.

parámetros:
x - coordenada X del punto.
x - coordenada Y del punto.
ID - número identificador que se mostrara dentro del punto.
col - color del punto en un formato de MatPlotLib valido.

dibujarSolucion(self, solu)

Método que dibuja la solucion dada en la gráfica.

Dibuja las rutas de los camiones de la solucion, cada
una con un color aleatorio distinto.

genetico(self, devolver=False)

Funcion que inicia una simulacion basada en algoritmos geneticos, mostrando un re
sultado.

Inicia la simulacion con los parametros indicados por el mapa, la cual devuelve una s
olucion
y esta es mostrada al usuario de forma grafica y detallada (texto).

iniciarGrafica(self)
Función que inicia la gráfica para su posterior uso.

Inicia la grafica con los parametros preestablecidos para cumplir con las necesidades del programa.
SOLO PUEDE SER LLAMADO UNA VEZ.

limpiarGrafica(self)
Función que limpia la gráfica, dejándola vacía pero configurada.

mostrarRankingParalelo(self, solus)
Función que muestra el ranking de soluciones.

Imprime por consola y por GUI textBox las posiciones y el fitness de las soluciones dadas por parametros. Estas soluciones deben estar pre-ordenadas de mejor a peor.

mostrarSolucionEnTextBox(self, solu)
Función que muestra una solucion en el textBox de la GUI.

Muestra una solucion en el formato preestablecido en el textBox de la GUI.

paralelo(self)
Funcion que inicia una simulacion mixta basada en todos los algoritmos, mostrando un resultado.

Inicia tantas simulaciones como nucleos tenga el sistema, devolviendo las soluciones de cada simulacion
ordenadas de mejor a peor, mostrando un ranking de estas y mostrando la mejor al usuario de forma grafica
y detallada (texto).

vectorSinDuplicados(self, tam, rangoDe, rangoA)
Genera coordenadas no repetidas.

Genera "tam" pares de coordenadas sin que estas se repitan (en conjunto) para evit ar que algun nodo caiga encima de otro.

Data and other attributes defined here:
ax = 0
cargasContenedores = []
fig = 0
habiaSimulacionPrevia = False
kcolor = []
kmeans = []
listaNodos = 0
numNodos = 0
static_canvas = 0

10.2.3 abejas

Algoritmo genetico diseñado para ofrecer solucion a la recogida de residuos en un barrio. Ataca el problema de la mochila combinado con las distancias entre los nodos y el uso eficiente de los camiones.

Modules

numpy random solucion tsp_mst

Classes

builtins.object
AAbejas

class **AAbejas**(builtins.object)

Clase contenedora del algoritmo de optimizacion por enjambres dedicada a obtener una solucion optimizada.

Methods defined here:

__init__(self, max_ejecucion, paramSolus, imprimir)
Inicializa los parámetros del algoritmo:
max_ejecucion - numero máximo de individuos a mejorar.
paramSolus - lista con los parámetros esenciales del mapa.
imprimir - badera que indica si han de imprimirse estados intermedios por consola.

crearSoluCruda(self)
Crea y devuelve una solucion inicial cruda.

espacioOcupado(self, C)
Devuelve el espacio ocupado de un camión C.

evaluar(self, Barrio)
Evalúa un barrio, devolviendo su fitness.

generarSolucion(self)
Funcion que, tras jugar con el algoritmo de abejas, devuelve la mejor solucion que ha encontrado.

mutar(self, hijo)
Función que altera mínimamente un individuo (solucion).

La alteracion consiste en intercambiar dos nodos de peso similar de un mismo barrio.

nodoSimilar(self, n1, n2, c1)
Dice si dos nodos son similares o no (nodo1, nodo2, camionDeNodo1).

Dice si dos nodos son similares respecto al siguiente criterio:
- Dos nodos son similares si el menor es menos de un 10% menor que el mayo
r.
- No son el mismo nodo.
- No son nodos del mismo camión.

10.2.4 paralelo

Libreria auxiliar que permite procesar varias soluciones en paralelo, devolviendo la mejorobte-
nida al terminar.

Modules

GA abejas multiprocessing

Functions

f(x)

Función genérica que ejecuta un hilo del pool al ser iniciado.

Esta funcion, con los parametros de las soluciones como entrada, ejecuta
uno de los dos algoritmos de optimización en función de:

- si el ID del hilo es PAR ejecuta el Algoritmo Genético
- si el ID del hilo es IMPAR ejecuta el algoritmo de abejas

obtenerSolucionesParalelaGA(paramSolus, numIteracciones)

Funcion que levanta un hilo por cada nucleo del sistema para generar una solucion.

La mitad de los hilos seran dedicados a algoritmos geneticos y la otra mitad a algoritmo
de abejas.
Una vez obtiene las soluciones, las devuelve ordenadas de mejor a peor en una lista.

10.2.5 GA

Algoritmo genetico diseñado para ofrecer solucion a la recogida de residuos en un barrio.
Ataca el problemade la mochila combinado con las distancias entre los nodos y el uso efi-
ciente de los camiones.

definición de individuo:
tipo solucion (solucion.py)

definición de población:
[solucion, solucion, solucion, ...]

Modules

numpy random solucion tsp_mst

Classes

builtins.object
Genetico

class **Genetico**(builtins.object)

Clase contenedora del algoritmo genetico de optimizacion dedicada a obtener una solucion optimizada.

Methods defined here:

__init__(self, num_individuos_in, max_ejecucion, paramSolus, imprimir)

Inicializa los parámetros del algoritmo:
num_individuos_in - tamaño de la poblacion y del pool inicial.
max_ejecucion - numero máximo de individuos a mejorar.
paramSolus - lista con los parámetros esenciales del mapa.
imprimir - badera que indica si han de imprimirse estados intermedios por consola.

crearPool(self, num_individuos)

Crea un pool de n individuos con valores aleatorios y los devuelve en una lista.

cruce(self, padre, madre)

Metodo que realiza el cruce de dos individuos, devolviendo uno con los mejores genes y otro con los peores.

espacioOcupado(self, C)

Devuelve el espacio ocupado de un camión C.

evaluar(self, Barrio)

Evalúa un barrio, devolviendo su fitness.

generarSoluciones(self, numSolus=1)

Funcion que, tras jugar con el algoritmo genetico, devuelve las @numSolus mejores soluciones que ha encontrado.

mostrarMejor(self)

Muestra el mejor individuo de una población ordenada.

La población debe estar ordenada de mejor a peor.

mostrarPoblacion(self)

Muestra la población completa por consola.

mutar(self, hijo)

Función que altera mínimamente un individuo (solucion).

La alteracion consiste en intercambiar dos nodos de peso similar de un mismo barrio.

nodoSimilar(self, n1, n2, c1)

Dice si dos nodos son similares o no (nodo1, nodo2, camionDeNodo1).

Dice si dos nodos son similares respecto al siguiente criterio:
- Dos nodos son similares si el menor es menos de un 10% menor que el mayor.
- No son el mismo nodo.
- No son nodos del mismo camión.

obtenerIndividuoAleatorioDiferenteDe(self, poblacion, individuo)

Dada una poblacion no vacia, devuelve un individuo diferente del parametro @individuo.

ordenarPoblacion(self)

Funcion que ordena la poblacion actual, segun el fitness de cada individuo, de mejor a peor.

reproduccion(self)

Metodo que cambia la poblacion actual por una nueva siguiendo los criterios de reproduccion y mutacion.

Criterios:
- Elitismo de 5 individuos.
- Para cada individuo se escogen 2 aleatorios que no sean él mismo y se compara n, usando en la reproducción aquel que sea mejor.
- El cruce consiste en crear una solu con los mejores barrios de ambos individuos (tambien se genera otra con los peores).
- La mutacion consiste en coger un nodo de uno de los camiones del barrio y cambiarlo por otro de otro de los camiones del barrio si son de peso similar.
  (para todos los barrios barrios ó para uno solo) cada barrio con su prob de mutar.
- De la poblacion resultante nos quedamos con los (NUM_INDIVIDUOS - 5%) mejores. El 5% restante son individuos aleatorios nuevos.
- La población queda ordenada por fitness al finalizar.

10.2.6 tsp_mst

Libreria que contiene TSP_MST.
The Salesman Problem _ Minimal Spanning Tree

- Este algoritmo, basandose en arboles de minima expansion, intenta resolver el problema del viajante de comercio para un conjunto de nodos dados.

Utilizar la función TSM_MST_IDs(listaDeIDsDeNodos)

Modules

math numpy

Functions

TSM_MST_IDs(listaNodos, paramSolus)

Metodo que dado una lista de ID's de nodos y los parametros de las soluciones te devuel ve la distancia y ruta obtenida de aplicar TSP_MST++ sobre ellos.

TSP_MST(listaArcos, numNodos, paramSolus, TSP_MST_plusplus=True)

Algoritmo de The Salesman Problem con Minimal Spannig Tree.

Devuelve la distancia del recorrido y una lista con los IDs de la ruta ordenada, respectiva mente.
TSP_MST_plusplus -> bandera que indica si se usara la funcionalidad TSP_MST++. Dicha f uncionalidad
recorre la solucion para mejorarla, eliminando ciertas incoherencias como cr uces
de caminos o codos de Angulo fino.

TSP_MST_arcoValido(actual, conjunto)

Método que dice si el arco es válido en el algoritmo TSP_MST.

distancia(Xo, Yo, Xd, Yd)

Distancia entre dos puntos dadas sus coordenadas (float).

10.2.7 Solucion

Clase cuyas instancias representan una solucion al problema.

Modules

genBarrio

Classes

builtins.object
Solucion

class **Solucion**(builtins.object)

Clase que representa una solucion al problema global.

Estas soluciones guardan sus parametros como la solucion en si misma o el fitness.
Nada mas ser creada una instancia, no representara una solucion valida a menos que se iniciali ce correctamente llamando
a la funcion generarSolucionCruda().

Methods defined here:

__init__(self, paramSolus, tiposDeCamion_in=[10, 5, 2])

Constructor de la clase Solucion.

Argumentos:
paramSolus -> lista con los parametros de la simulacion en el siguiente orden:
[self.**listaNodos**, self.**kmeans**, self.**SpinBoxNumeroBarrios**.value(), self.**numNodos**, self.**cargasContenedores**]
tiposDeCamion_in -> tipos de camion disponibles a usar en formato [cargaTipo1, cargaTipo2, cargaTipo3, ...] ordenados descendentemente.

copiar(self)
Devuelve un objeto nuevo equivalente al invocador.

generarSolucionCruda(self)
Genera una solucion cruda, de forma que la instancia invocante pasa a ser una solucion cruda.

Una solucion cruda es aquella que producida por la libreria genBarrio.py.
Estas soluciones, como su nombre indica, son imperfectas se utilizan como el recurso fuente de los algoritmos de optimizacion del programa, en busca de una solucion mejorada.

mostrarSolucion(self)
Muestra la solucion junto con su fitness total.

10.2.8 genBarrio

Algoritmo genetico diseñado para ofrecer solucion a la recogida de residuos en un barrio.
Ataca el problema de la mochila combinado con las distancias entre los nodos y el uso eficiente de los camiones.

definición de individuo:
lista de Python -> [genes, fitness]

donde genes = [camion, camion, ...]
donde camion = [Tam, [nodos...], espacioLibre, ruta]
donde nodos = [peso, ID] ; ruta = [ID, ID, ID, ...] en orden

Modules

numpy random tsp_mst

Classes

builtins.object
GenBarrio

class **GenBarrio**(builtins.object)
Clase contenedora del algoritmo pseudo-genetico que resuelve el problema de la mochila, devolviendo una solucion cruda.

Methods defined here:

__init__(self, paramSolus, num_individuos, Barrio, tiposDeCamion)

Constructor de la clase GenBarrio.

Argumentos:
paramSolus -> lista con los parametros de la simulacion en el siguiente orden:
[self.**listaNodos**, self.**kmeans**, self.**SpinBoxNumeroBarrios**.value(), self.**numNodos**, self.**cargasContenedores**]
num_individuos -> numero de individuos del algoritmo pseudo-genetico.
Barrio -> datos del barrio en formato [peso, id].
tiposDeCamion -> tipos de camion disponibles a usar en formato [cargaTipo1, cargaTipo2, cargaTipo3, ...] ordenados descendentemente.

camionNecesario(self, B, T)

Funcion que devuelve el tipo de camion necesario para seguir rellenando la solucion al barrio.

crearPool(self, num_individuos)

Crea un pool de individuos con valores aleatorios y los devuelve en una lista.

Argumentos:
num_individuos -> número de individuos a devolver.

Los individuos poseen su fitness al ser devueltos.

espacioOcupado(self, C)

Devuelve el espacio ocupado del camión.

espacioRestante(self, C)

Devuelve el espacio libre del camión.

evaluar(self, Barrio)

Evalúa un barrio, devolviendo su fitness.

La forma de evaluación tiene en cuenta:
Los camiones utilizados.
La carga que llevan (y por ende su espacio libre).
La distancia del recorrido que debe hacer el camión.

generarSoluciones(self, numSolus=1)

Funcion que, tras jugar con el algoritmo pseudo-genetico, devuelve las n mejores soluciones que ha encontrado.

Argumentos:
numSolus -> número de soluciones a devolver.

mostrarMejor(self)

Muestra el mejor individuo por consola.

La población debe estar ordenada.

mostrarPoblacion(self)

Muestra la población por consola.

ordenarPoblacion(self)
Función que ordena la población principal según su fitness

11 Ejemplo de ejecución

En este apartado se mostrarán tres ejemplos de ejecución del programa.
Cada ejemplo ira dedicado a uno de los tipos de ejecución posible mostrando los pasos necesarios para configurar y ejecutar el programa, así como los resultados que este nos ofrece al terminar.

11.1 Algoritmo Genético

Primero comenzamos iniciando el programa como se explica en el apartado 9.2 Ejecución.

Se nos mostrará una ventana con la interfaz gráfica de usuario:

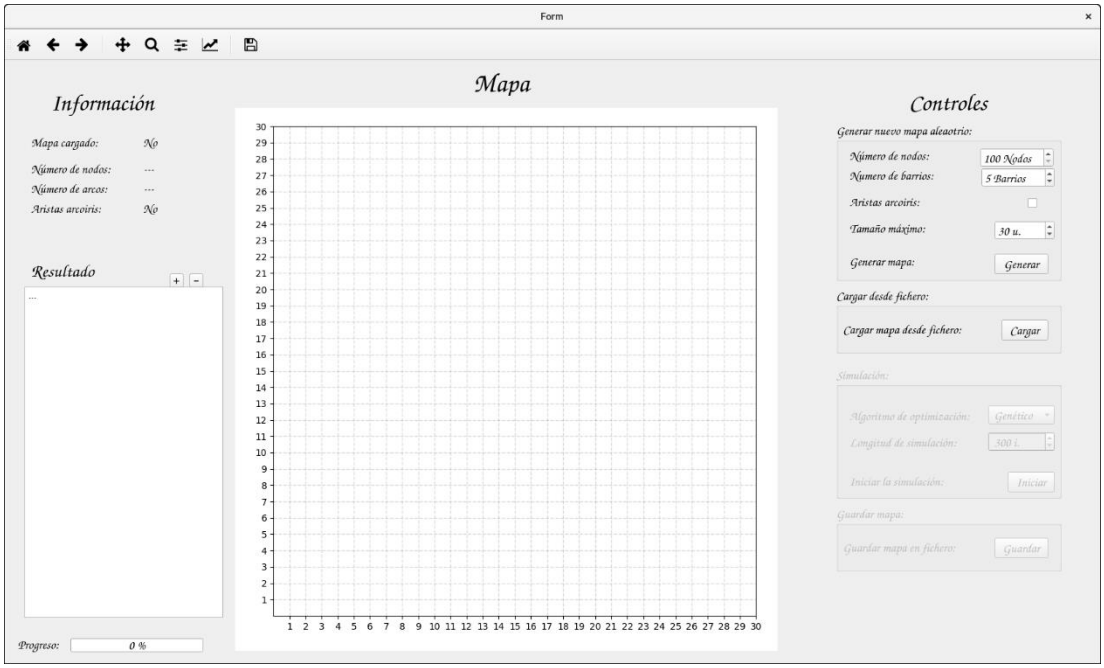


Ilustración 22: Programa recién iniciado.

Nos dirigiremos al panel de controles, al apartado de "*Generar nuevo mapa aleatorio:*", donde estableceremos los parámetros del caso a gusto.
En este caso dejaremos los parámetros por defecto.

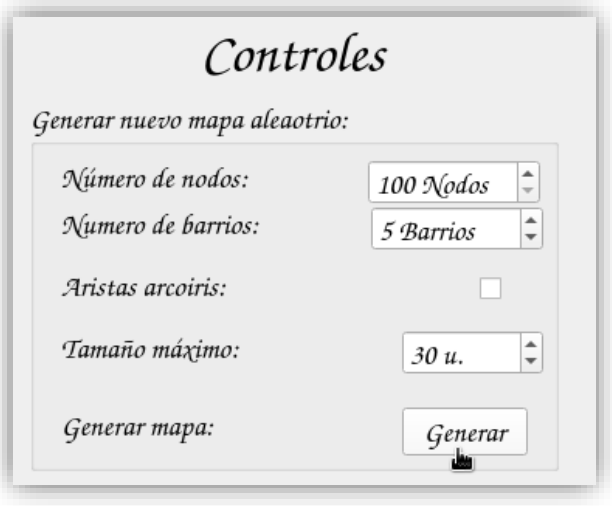


Ilustración 23: Panel de controles, apartado "Generar nuevo mapa aleatorio".

Pulsamos el botón "*Generar*" y se nos generará el nuevo mapa aleatorio con el nuevo caso.

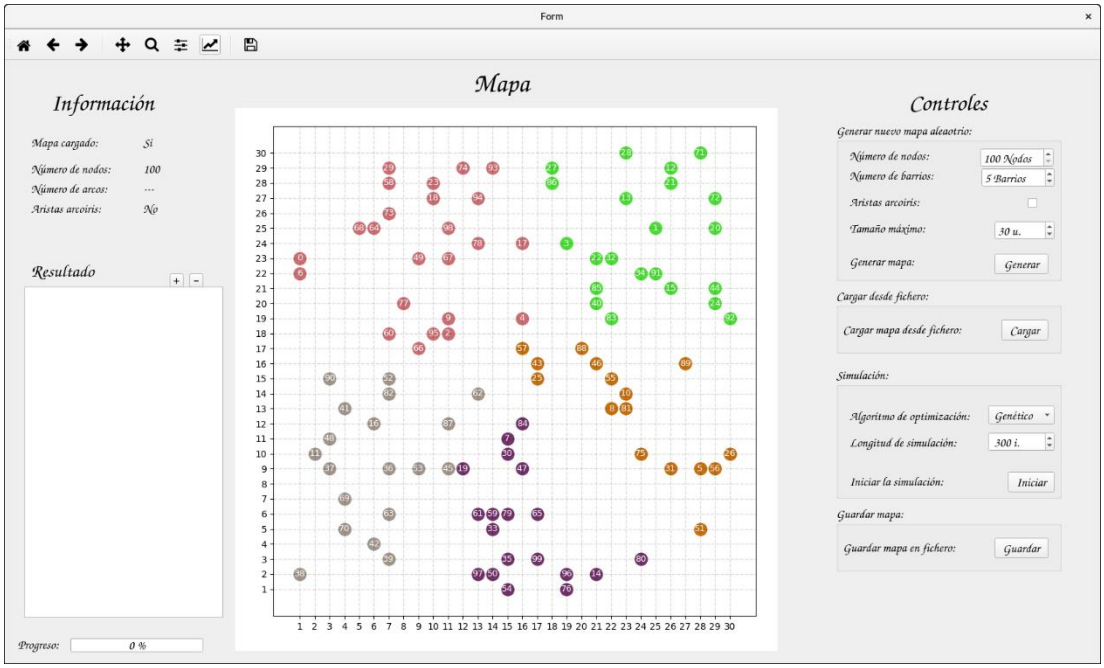


Ilustración 24: Programa con un mapa (caso) definido.

Ahora nos dirigiremos al apartado "*Simulación:*". Dentro de este apartado seleccionaremos el tipo de ejecución como "*Genético*" y, dejando el resto de parámetros por defecto, pulsaremos en "*Iniciar*".

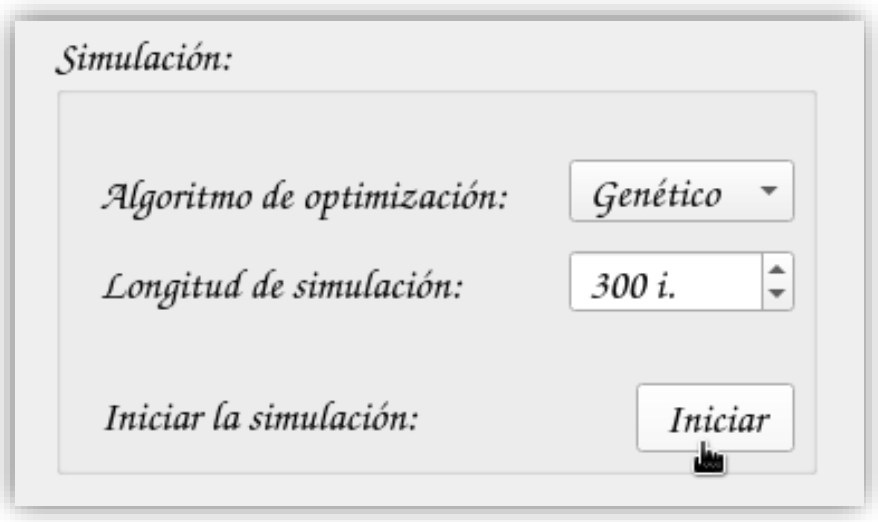


Ilustración 25: Inicio de simulación usando algoritmo genético.

Al instante de pulsar, el programa comenzará a trabajar utilizando el algoritmo seleccionado.

Todos los progresos intermedios del algoritmo pueden verse en la consola que utilizamos para iniciar el programa.

Este algoritmo comienza generando el pool de individuos inicial, el cual tiene una longitud de 100:

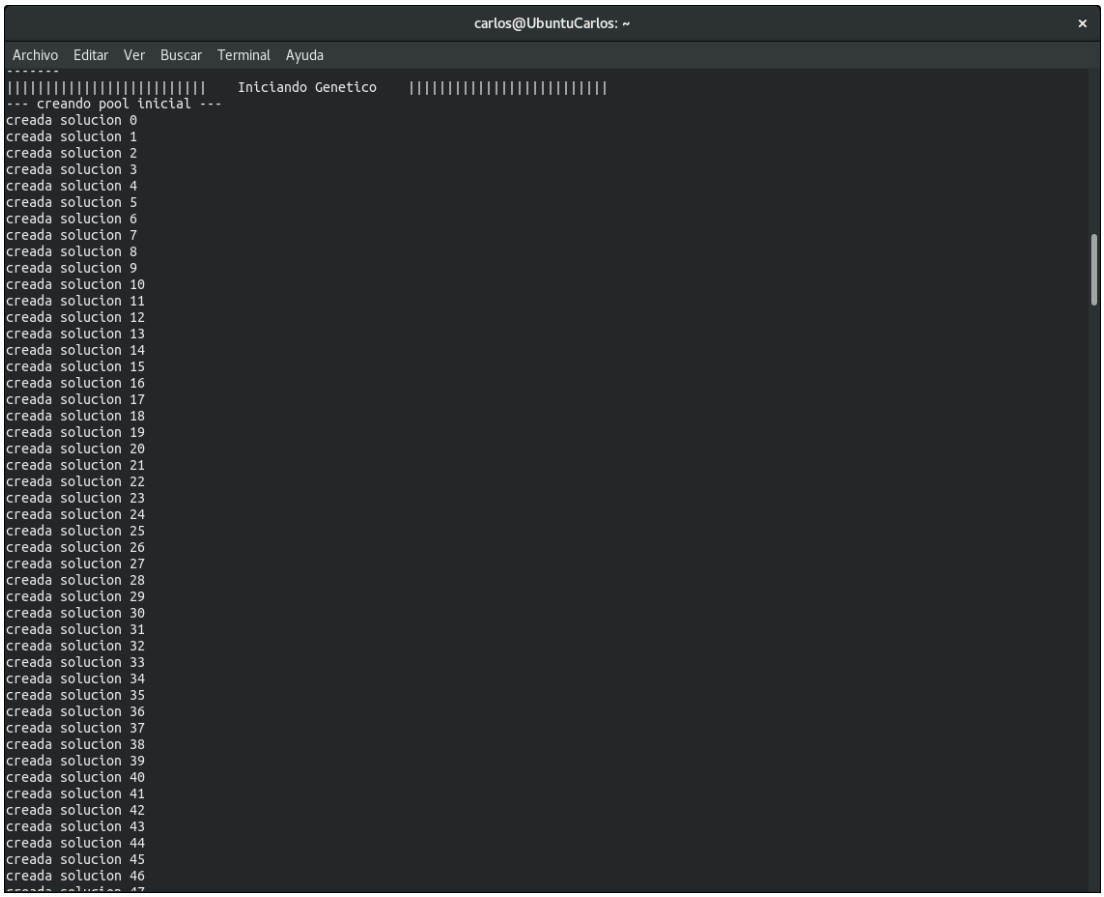


Ilustración 26: Generación de pool inicial en el algoritmo genético.

Una vez creado el pool, comenzarán a iterarse las diversas épocas del algoritmo genético.

En cada época el algoritmo mostrará cuales la mejor solución que ha obtenido hasta esa época, mostrando su puntuación y marcándola con el carácter "*" si esta ha mejorado.

Al principio el algoritmo mejorará fácilmente las soluciones.

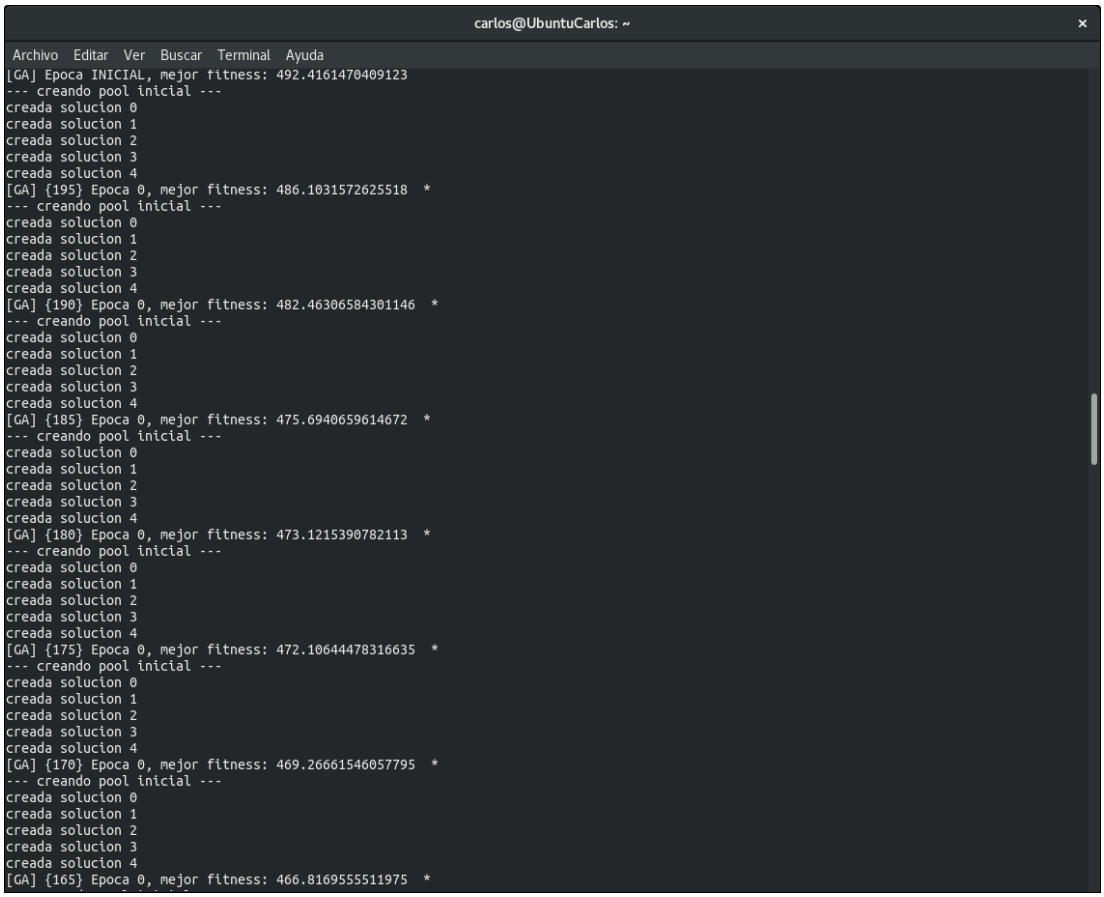


Ilustración 27: salida temprana del algoritmo genético.

Conforme nos acercamos a un óptimo, el algoritmo necesitará más iteraciones para mejorar.

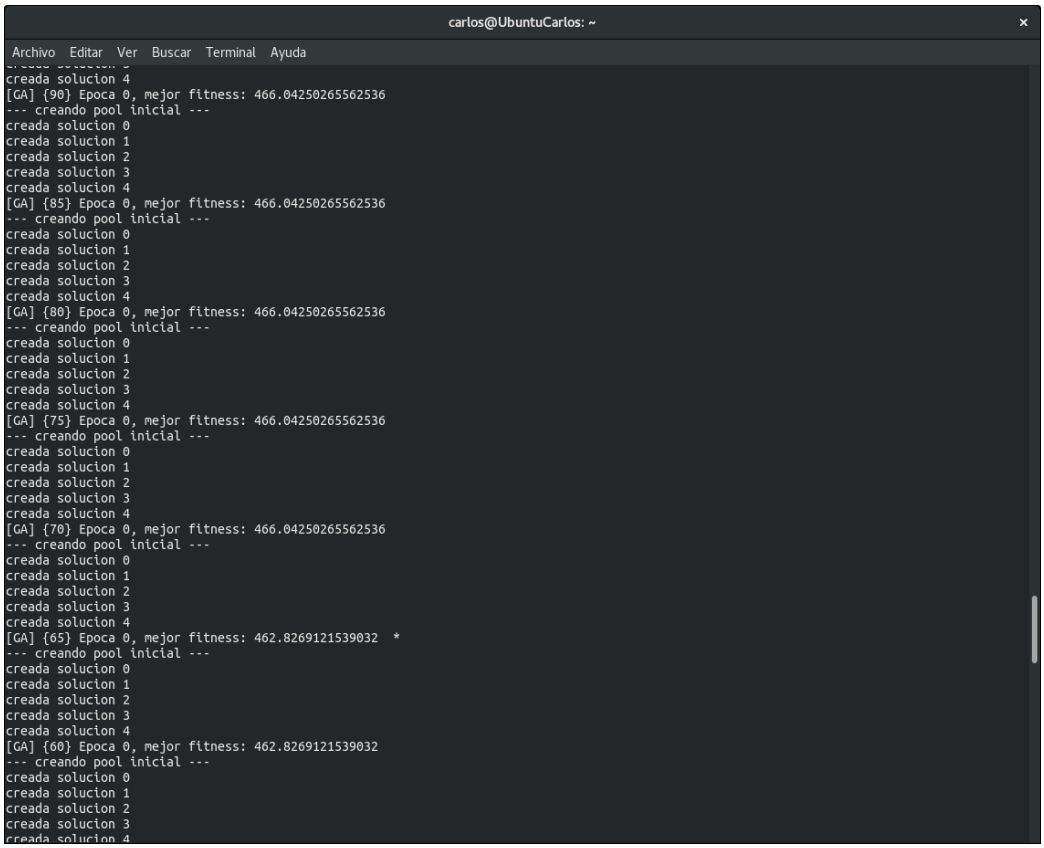


Ilustración 28: Salida tardía del algoritmo genético.

Cuando el algoritmo termine, mostrará por consola el resultado textual siguiendo el formato del apartado 6.8.2 Resultado Textual.

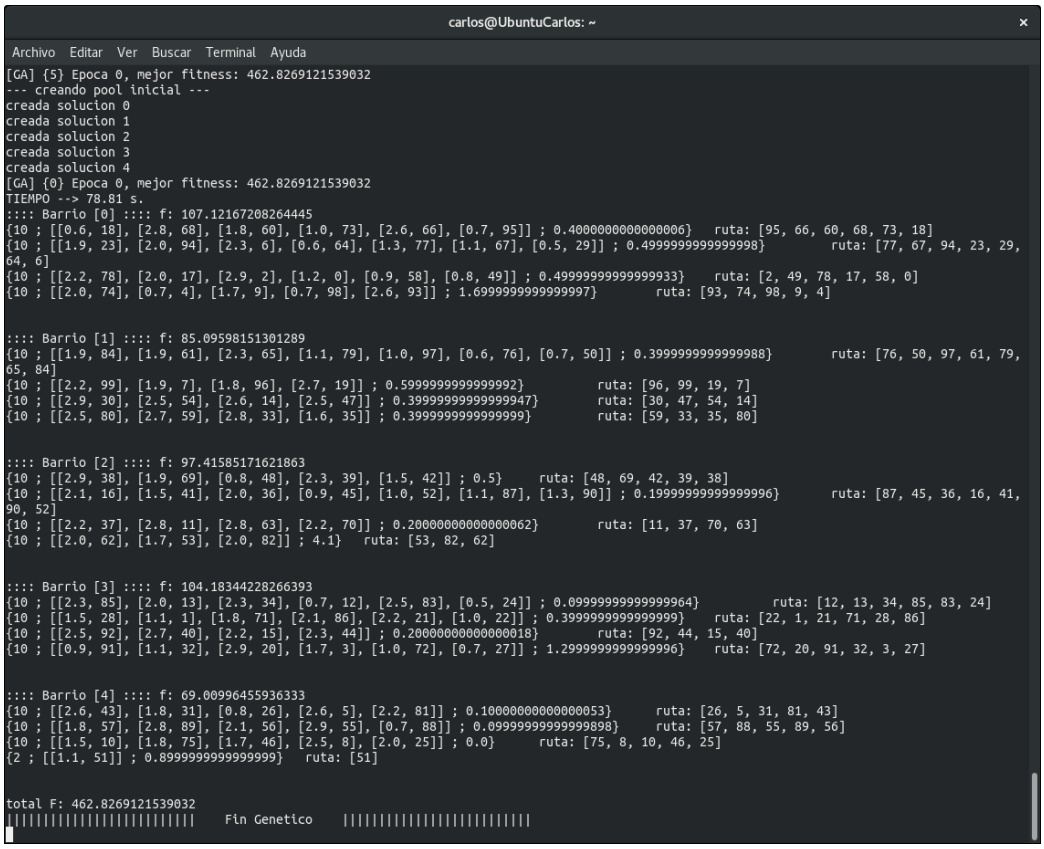


Ilustración 29: Resultado textual del algoritmo genético.

También mostrará el resultado textual en el TextBox de la interfaz gráfica, así como el resultado grafico en la gráfica principal del programa (Según el formato del apartado 6.8.1 Resultado Gráfico).

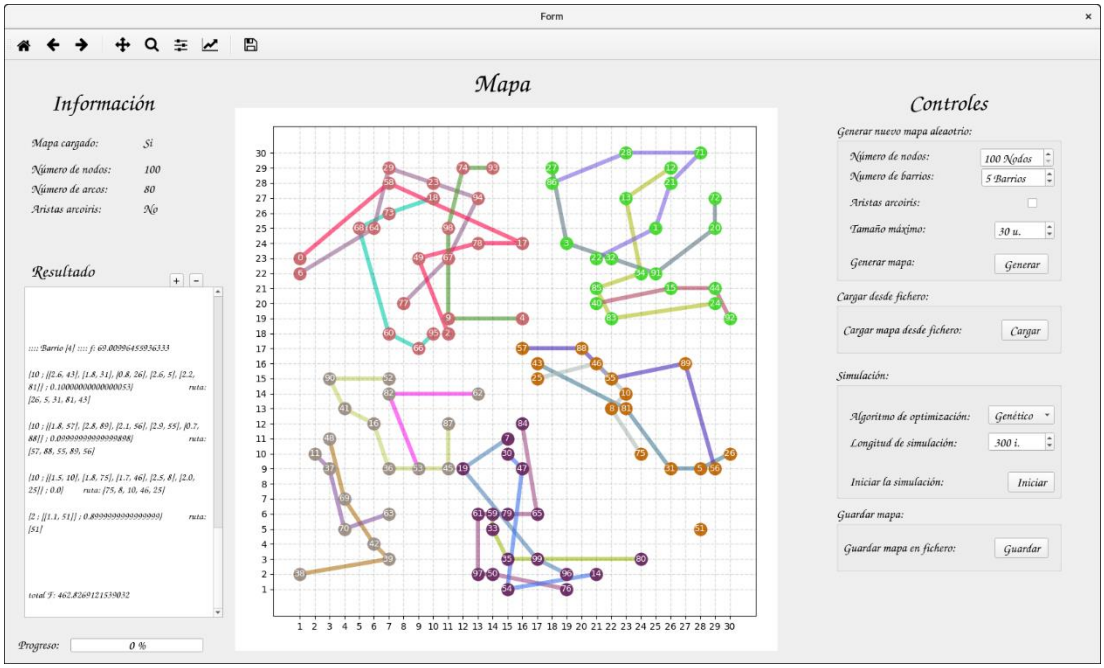


Ilustración 30: Resultado gráfico del algoritmo genético.

Por ultimo y, para mostrar dicha funcionalidad, guardaremos el mapa utilizado para usarlo en las dos ejecuciones siguiente.
Para ello, nos dirigimos al panel de controles, apartado "*Guardar mapa:*" y pulsamos el botón "*Guardar*".

Ilustración 31: Guardado de mapa (caso) en un fichero.

Seleccionamos el nombre y ruta del archivo en el cuadro de dialogo y guardamos.

El fichero debe tener la extensión .mapa para poder ser leídos por el programa.

11.2 Algoritmo de Enjambres

Primero comenzamos iniciando el programa como se explica en el apartado 9.2 Ejecución.

Se nos mostrará una ventana con la interfaz gráfica de usuario:

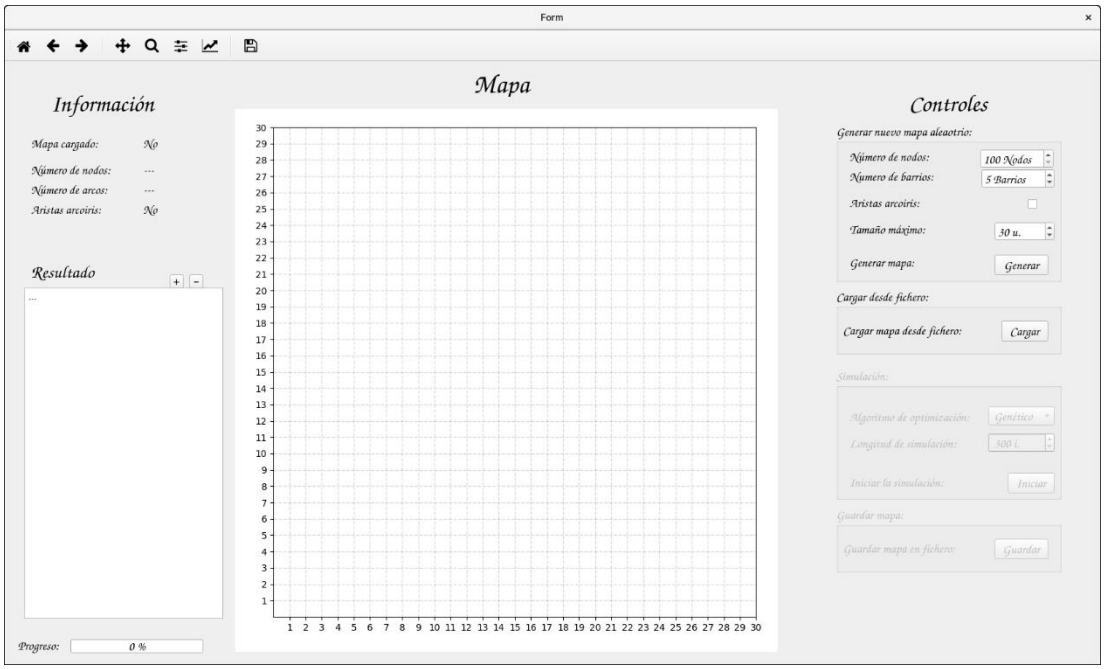


Ilustración 32: Programa recién iniciado.

En este caso, cargaremos un mapa.

El mapa a cargar es el utilizado y guardado previamente en el apartado anterior.

Para ello nos dirigiremos al apartado "*Cargar desde fichero:*" y pulsamos el botón "*Cargar*".

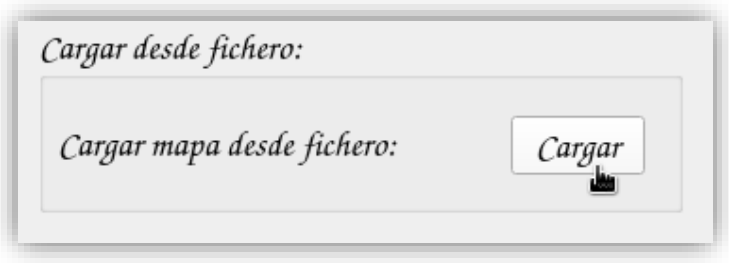


Ilustración 33: Carga de un mapa (caso) desde un fichero.

Nos aparecerá un cuadro de dialogo donde deberemos seleccionar el archivo. Una vez cargado el mapa, nos aparecerá el caso del apartado anterior en la interfaz gráfica de usuario.

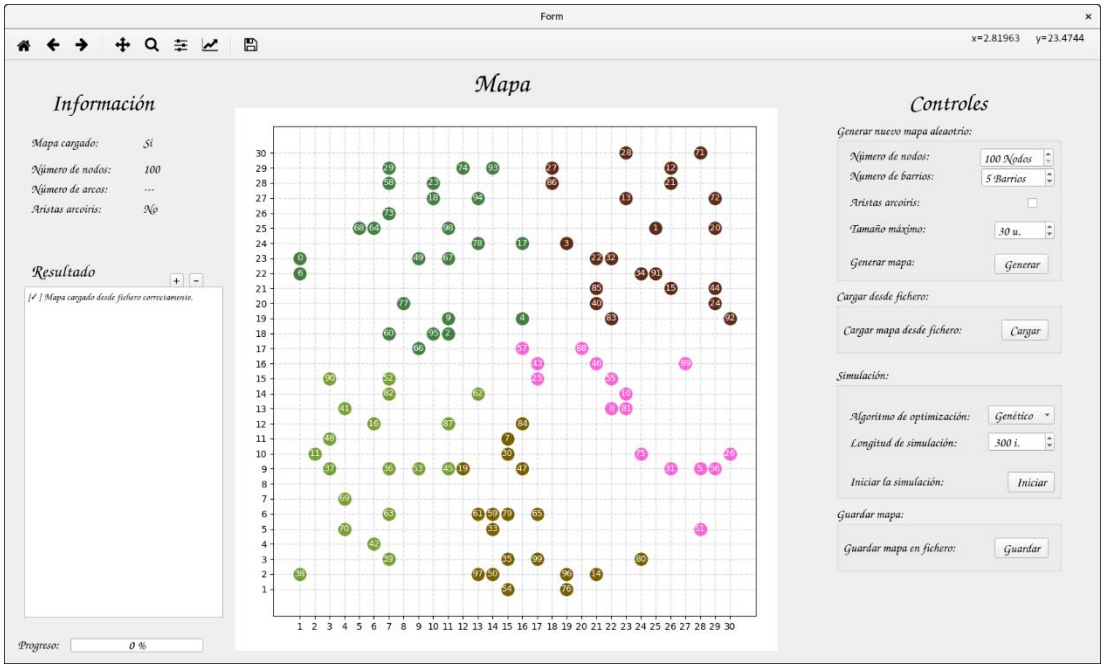


Ilustración 34: Mapa cargado en el programa.

Ahora nos dirigiremos al apartado "*Simulación:*". Dentro de este apartado seleccionaremos el tipo de ejecución como "*Abejas*" y, dejando el resto de parámetros por defecto, pulsaremos en "*Iniciar*".

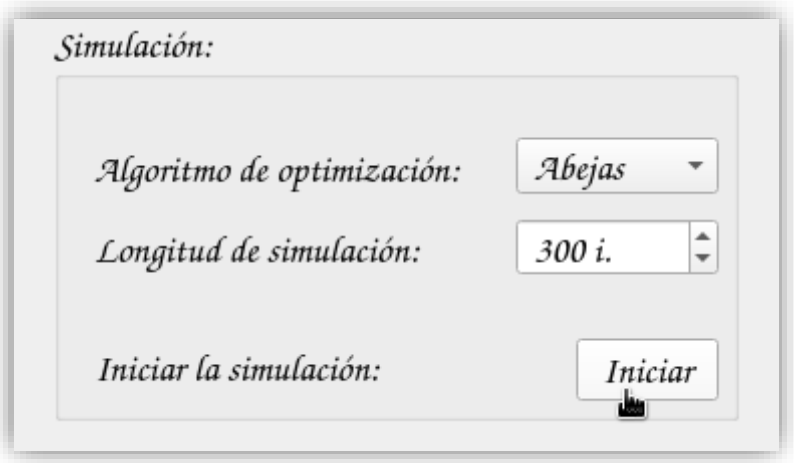


Ilustración 35: Inicio de simulación usando algoritmo de enjambres.

Al instante de pulsar, el programa comenzará a trabajar utilizando el algoritmo seleccionado.

Todos los progresos intermedios del algoritmo pueden verse en la consola que utilizamos para iniciar el programa.

El algoritmo de enjambres comenzará a iterar al instante de comenzar la simulación. En cada iteración, el algoritmo nos muestra el número de intentos por mejorar dicha solución, así como el valor de esta (candidato), el mejor valor que la solución candidata a obtenido (local) y el valor de la mejor solución obtenida por el algoritmo hasta entonces (global).

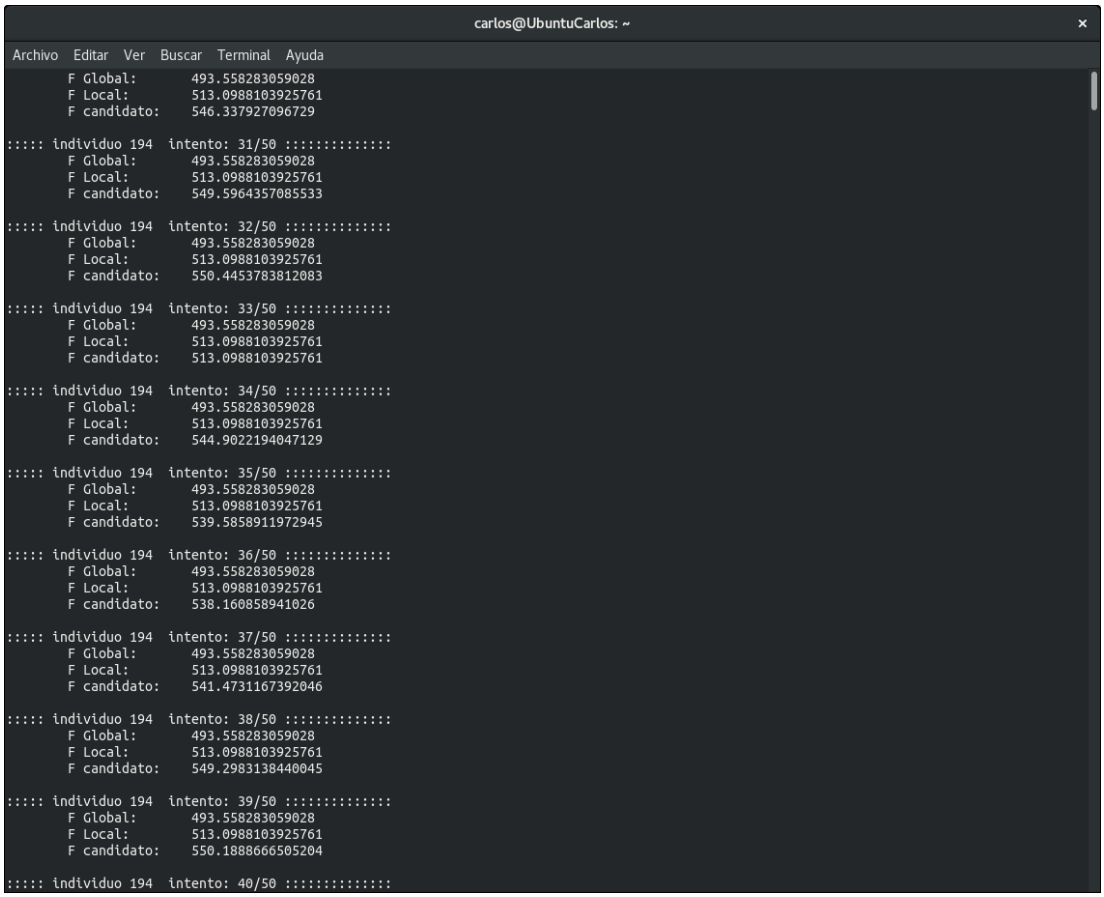


Ilustración 36: salida de ejecución del algoritmo de enjambres.

Al igual que en el algoritmo anterior, cuando la solución global mejora, esta es marcada con el carácter "*".

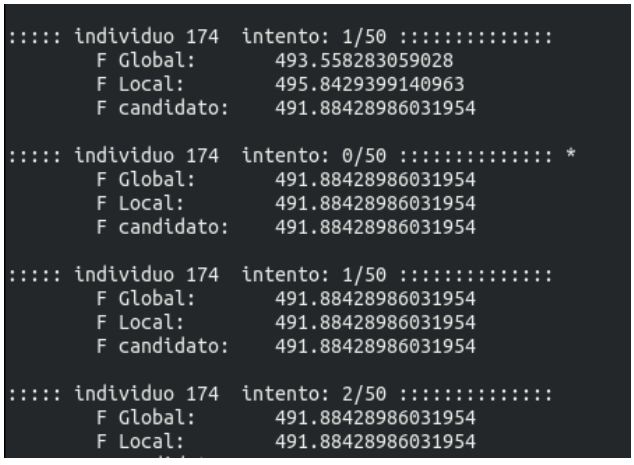


Ilustración 37: Ejemplo de mejoría del individuo en el algoritmo de enjambres.

Cuando el algoritmo termine, mostrará por consola el resultado textual siguiendo el formato del apartado 6.8.2 Resultado Textual.

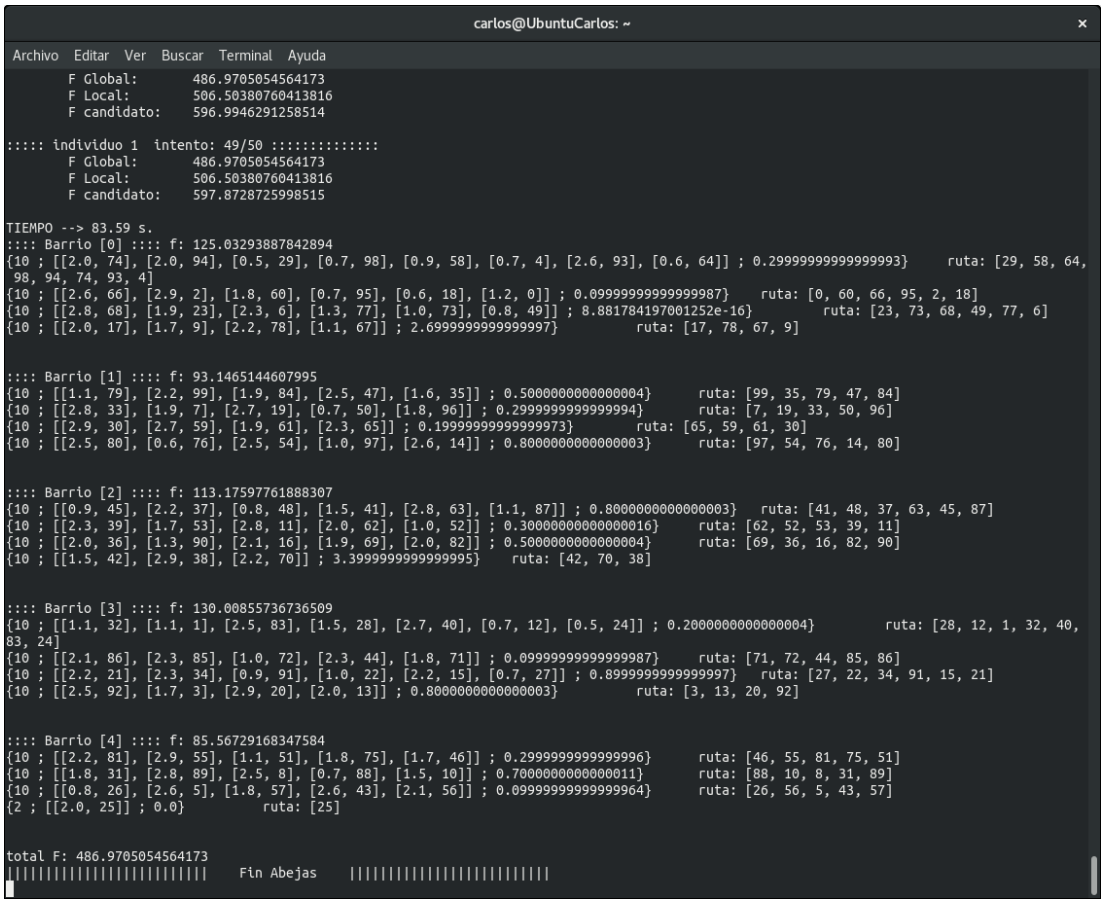


Ilustración 38: Resultado textual del algoritmo de enjambres.

También mostrará el resultado textual en el TextBox de la interfaz gráfica, así como el resultado grafico en la gráfica principal del programa (Según el formato del apartado 6.8.1 Resultado Gráfico).

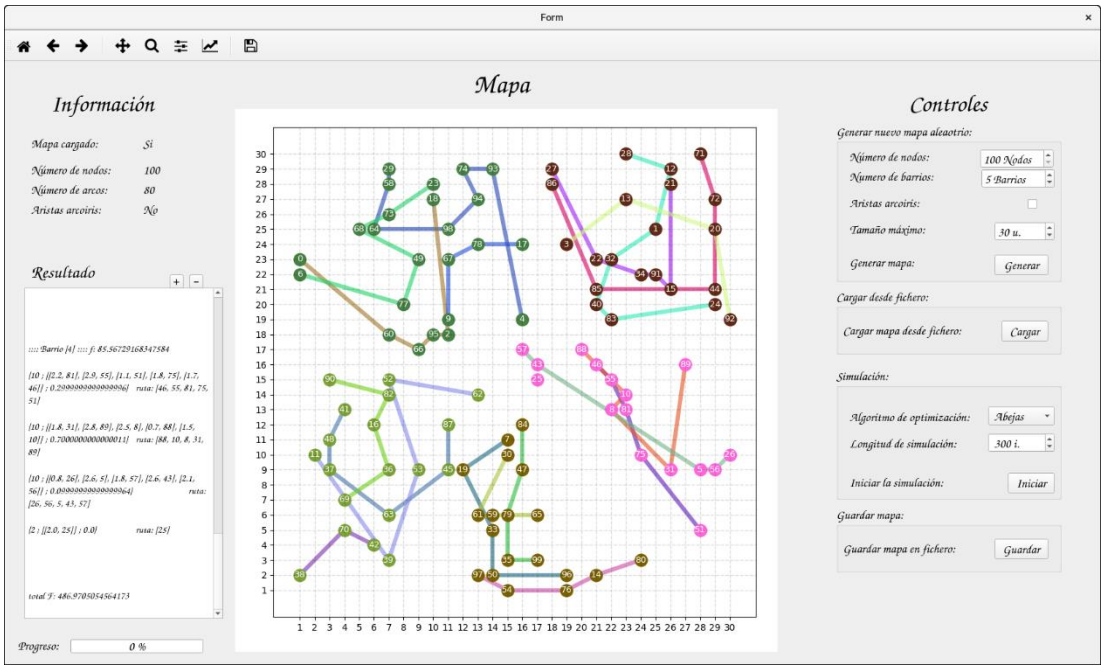


Ilustración 39: Resultado gráfico del algoritmo de enjambres.

11.3 Ejecución Paralela

Primero comenzamos iniciando el programa como se explica en el apartado 9.2 Ejecución.

Se nos mostrará una ventana con la interfaz gráfica de usuario:

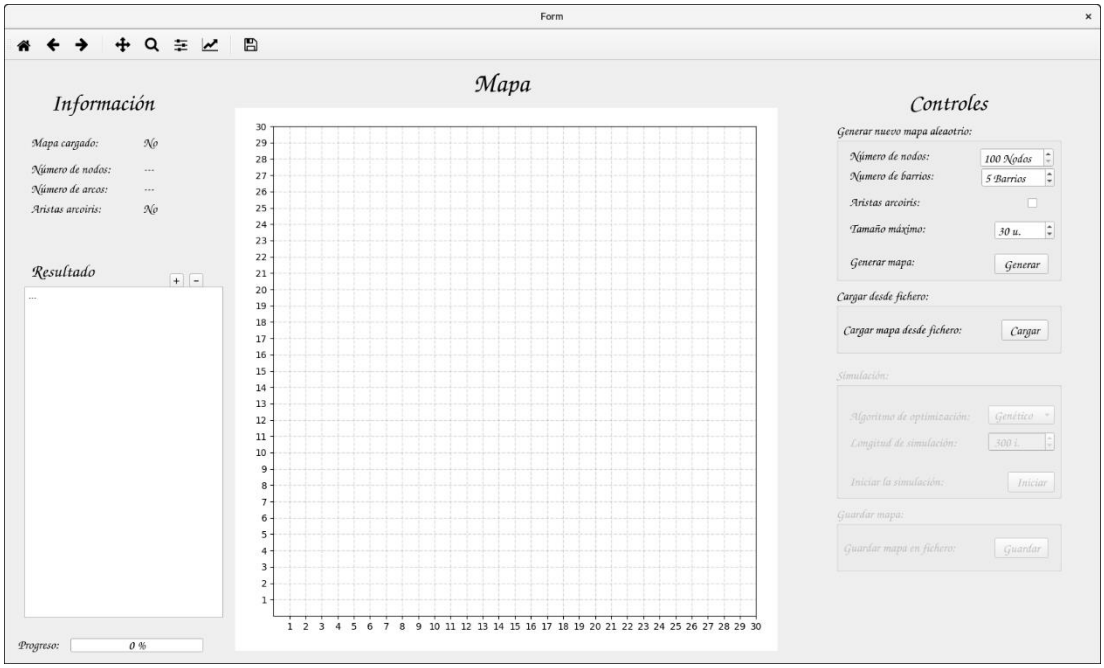


Ilustración 40: Programa recién iniciado.

En este caso, cargaremos un mapa.

El mapa a cargar es el utilizado y guardado previamente en el apartado anterior.

Para ello nos dirigiremos al apartado "*Cargar desde fichero:*" y pulsamos el botón "*Cargar*".

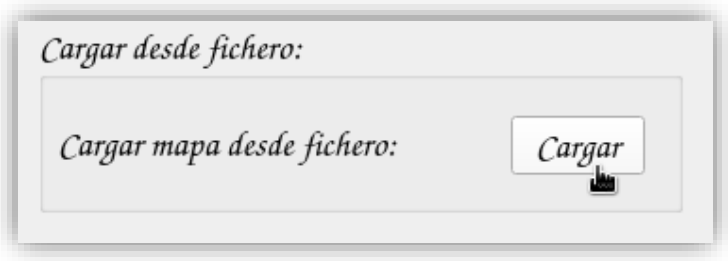


Ilustración 41: Carga de un mapa (caso) desde un fichero.

Nos aparecerá un cuadro de dialogo donde deberemos seleccionar el archivo. Una vez cargado el mapa, nos aparecerá el caso del apartado anterior en la interfaz gráfica de usuario.

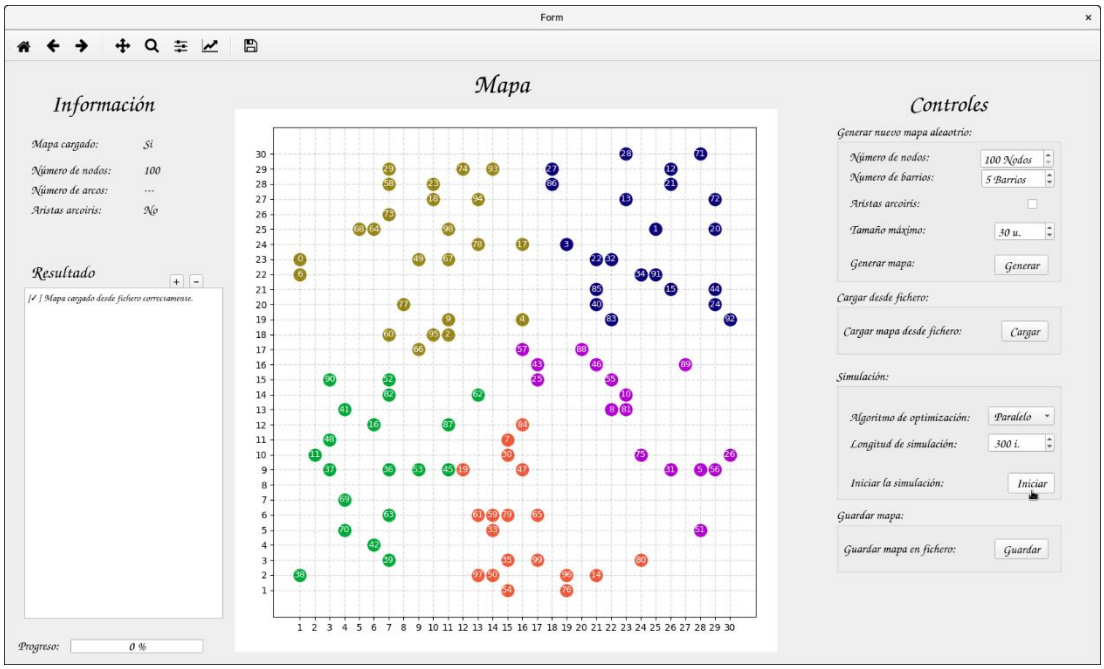


Ilustración 42: Mapa cargado en el programa.

Ahora nos dirigiremos al apartado "*Simulación:*". Dentro de este apartado seleccionaremos el tipo de ejecución como "*Abejas*" y, dejando el resto de parámetros por defecto, pulsaremos en "*Iniciar*".

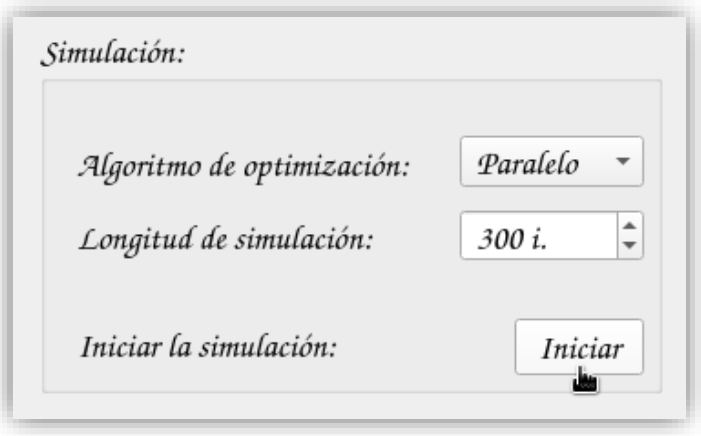


Ilustración 43: Inicio de simulación usando ejecución paralela.

Al instante de pulsar, el programa comenzará a trabajar utilizando el algoritmo seleccionado.

El algoritmo de ejecucion paralela comenzará a ejecutar tantos algoritmos como nucleos haya disponible en el sistema, siendo la mitad de ellos algoritmos geneticos y la otra mitad algoritmos de enjambres.

Nada mas comenzar, el programa muestarpor consola el siguiente fragmento, indicando la cantidad de algoritmos iniciados.

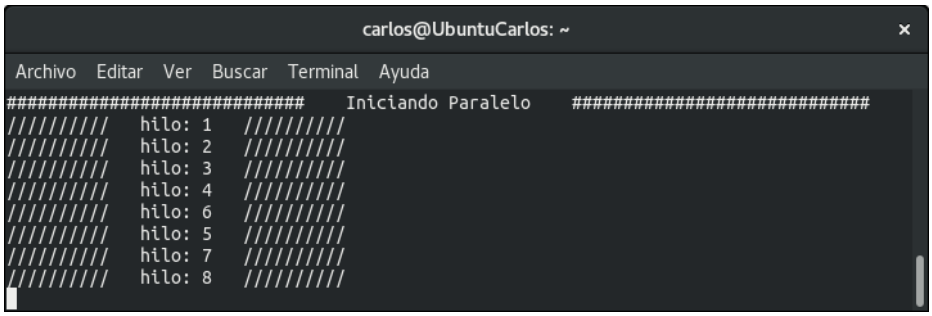


Ilustración 44: Comienzo de la ejecución paralela.

En este caso, el programa no ofrecerá salida por pantalla, por lo que permanecerá aparentemente congelado hasta que terminen las ejecuciones de los algoritmos.

Para asegurarse de la continuidad del programa, el usuario puede visualizar el monitor de recursos de su sistema, el cual mostrará que todos los núcleos del sistema están siendo utilizados.

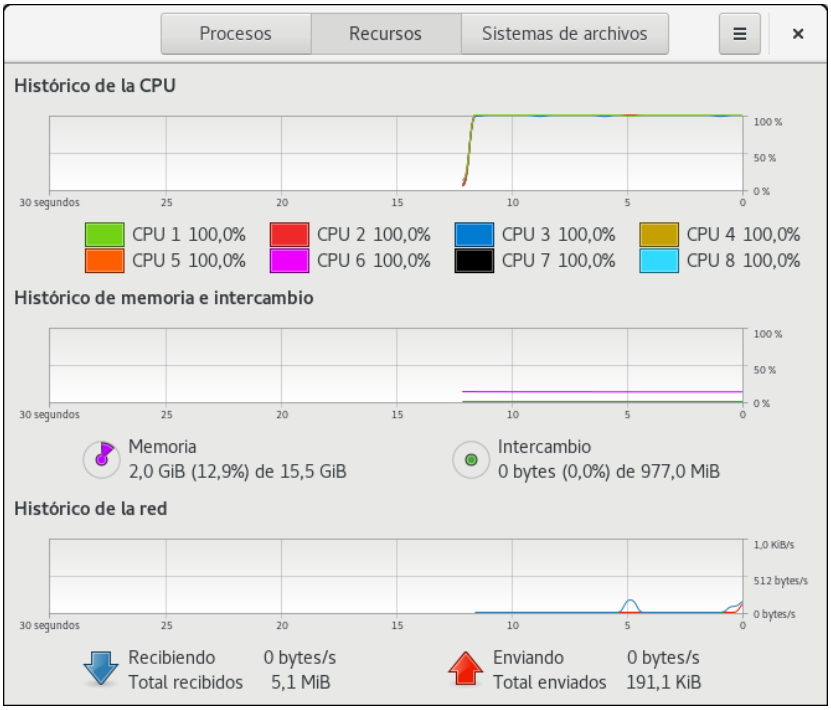


Ilustración 45: Monitor de recursos del sistema mostrando el uso completo del procesador.

Cuando el algoritmo termine, mostrará por consola el resultado textual siguiendo el formato del apartado 6.8.2 Resultado Textual.

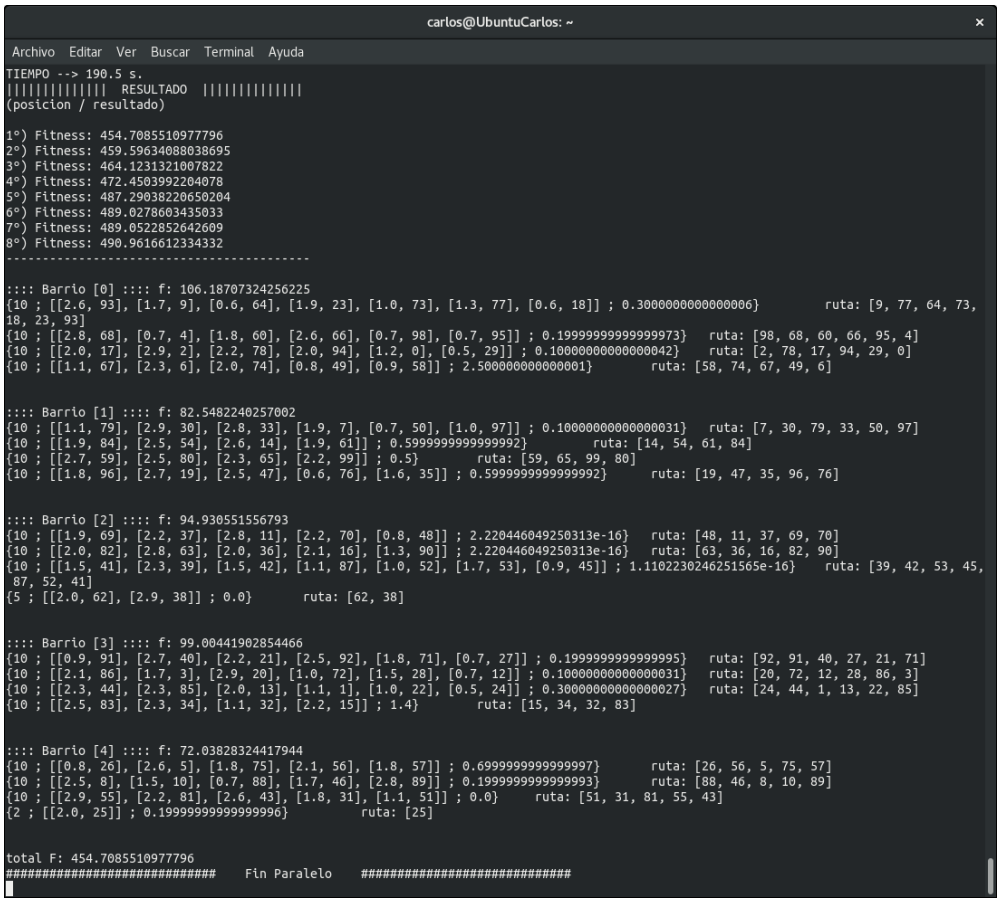


Ilustración 46: Ranking y resultado de la mejor solución obtenida en la ejecución paralela.

También mostrará el resultado textual en el TextBox de la interfaz gráfica, así como el resultado grafico en la gráfica principal del programa (Según el formato del apartado 6.8.1 Resultado Gráfico).

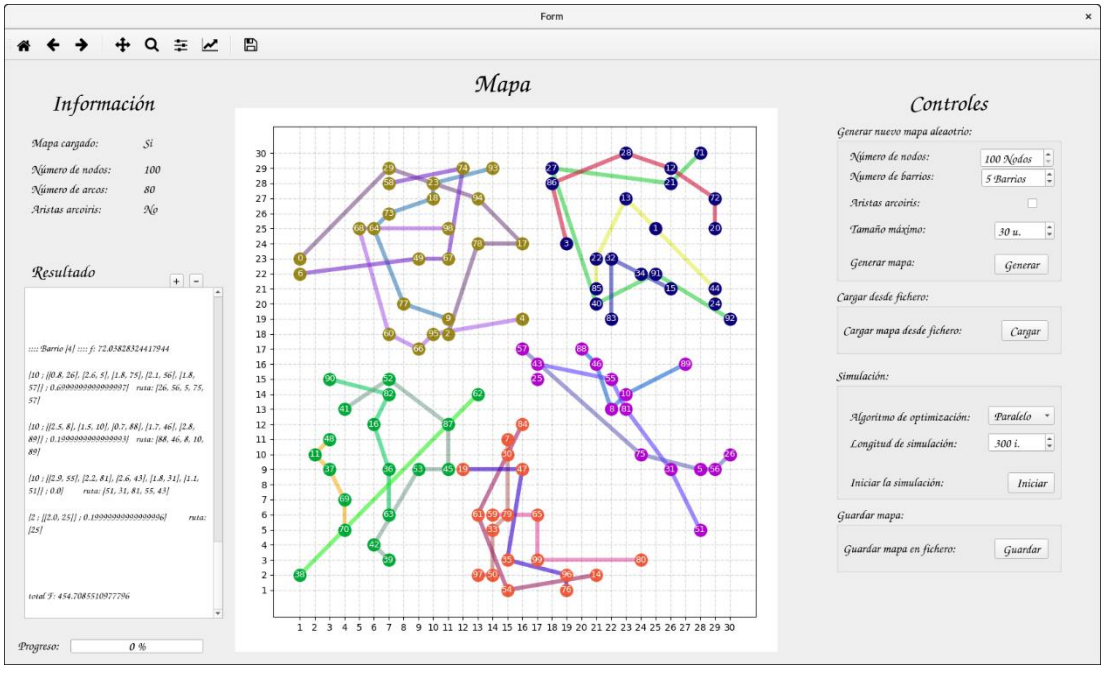


Ilustración 47: Resultado grafico de la mejor solución obtenida en la ejecución paralela.

12 Líneas futuras

En este apartado se encuentran las posibles ampliaciones o modificaciones que se proponen para este proyecto.

12.1 Algoritmo asimétrico

Se propone modificar el proyecto para que este sea capaz de trabajar con grafos asimétricos.

Habría que estudiar si se puede modificar el algoritmo *TSP-MST* para adaptarlo al problema o, si resultaría mejor diseñar otro algoritmo.

12.2 Distintos tipos de cargas

Se propone modificar el caso para que los nodos puedan albergar distintos tipos de materiales a recoger. Esto complicaría el problema de la mochila, teniéndose que modificar el actual método de resolución del problema de la mochila, para que este se adaptara a varias cargas.

Se propone también, si es relevante, incluir dicha búsqueda de la mejor combinación de recogida en los algoritmos de optimización, siendo un punto más a optimizar.

12.3 Ejecución dinámica

Se propone modificar el programa para que la ejecución sea dinámica.

Esto quiere decir que, en vez de realizar una sola ejecución, el programa deberá trabajar continuamente, intentando resolver un caso.

El hecho de que dicho caso sea solucionado cambiará el estado del mapa, el cual deberá volver a ser solucionado.

Dicho mapa generará nuevos problemas (nodos) aleatoriamente, ofreciendo trabajo al algoritmo en continua ejecución.

12.4 Tipos de camino

Se propone incluir distintos tipos de caminos. Estos nuevos caminos poseerán restricciones que harán que diversos tipos de vehículos no puedan recorrerlos.

El programa deberá estudiar, mediante los algoritmos de optimización, si es mejor utilizar un vehículo que pueda atravesar dicho camino o si conviene realizar una ruta alternativa.

El programa debe de ser capaz de generar dichas rutas alternativas.

12.5 Creador de mapas

Se propone diseñar una herramienta nueva que permita al usuario construir manualmente nuevos mapas (casos).

Dicha herramienta ha de ser intuitiva y completa, permitiendo al usuario modificar todos los parámetros de un caso (tanto los parámetros modificables como los de obtención aleatoria).

Dicha herramienta puede estar contenida en el programa principal o ser desarrollada independientemente.

La herramienta debe utilizar el mismo formato que los archivos de carga y guardado de ficheros, generando un fichero con la extensión .mapa al completar la construcción del nuevo caso.

12.6 Escalabilidad

Se propone añadir mayor escalabilidad al programa, modificando los algoritmos de optimización para que puedan desarrollar ciertas tareas en paralelo.

También se propone ampliar el paralelismo utilizado en el programa al uso de GPU o de sistemas distribuidos (utilizando, por ejemplo, MPI)

12.7 Estudio de TSP-MST

Se propone realizar un estudio del algoritmo *TSP-MST* y *TSP-MST++*.

Dicho estudio debe evaluar las capacidades del algoritmo, mostrar su escalabilidad, su eficacia y eficiencia, respaldada por datos reales y relevantes.

También se propone estudiar si se puede incluir alguna modificación en el algoritmo que mejore su funcionamiento.

13 Anexo

13.1 Anexo 1

En este apartado se explicará en detalle el funcionamiento de los algoritmos *TSP-MST y TSP-MST++*.

13.1.1 TSP-MST

El algoritmo *TSP-MST* se basa en el uso de listas, tratando de representar la situación grafica real.

El algoritmo tendrá una serie de listas, con ID de nodos. Cada una de estas listas representa un conjunto de nodos conectados. Por lo tanto, el acto de unir dos nodos con un arco consiste en, dependiendo del estado previo de los nodos, en la fusión de dos listas o en la adición de un nodo a alguna lista.
En caso de unir dos nodos aislados, se crearía una nueva lista con ambos nodos.

Sabiendo esto, la explicación del funcionamiento del algoritmo *TSP-MST* es la siguiente:

1) **Obtenemos** una lista de todos los arcos (un arco equivale a un par de nodos).
2) **Ordenamos** la lista de arcos, dejando los de menor coste primero.
3) Iteramos **hasta** que solo nos quede una lista con todos los nodos del caso
    a. Extraemos un arco de la lista (el arco de menor peso)
    b. Comprobamos si el arco es válido para ser utilizado. En caso de no ser válido, cogemos otro arco. Un arco **es válido** si cumple las siguientes condiciones:
        i. Si no forma círculos cerrados, es decir, si el arco no conecta dos nodos de una misma lista.
        ii. Si no forma divisiones de caminos, haciendo que un nodo tenga tres caminos conectados a él. Esto es, el arco no conecta a un nodo que aparece dos veces en una lista.
    c. Si el arco es válido, realizamos el acto de **insertarlo** en la solución. Esto es:
        i. Si el arco conecta dos nodos de dos listas distintas, las fusionamos, añadiendo al resultado dicho arco. Esto es equivalente a conectar dos regiones distintas de la solución.
        ii. Si el arco conecta un nodo aislado (no está en ninguna lista) con un nodo de una lista, introducimos dicho arco en la lista. Esto es equivalente a añadir un nodo a una región de la solución.
        iii. Si el arco conecta dos nodos aislados, creamos una nueva lista con este arco en su interior. Esto es equivalente a conectar dos

nodos aislados en la solución, creando una nueva región de solución.

d. Añadimos el coste del arco a un acumulador, el cual devolveremos al final.

4) Una vez terminado el proceso, analizamos la lista que ha quedado, la cual contiene la información de la ruta:
   a. Buscamos uno de los extremos de la ruta, es decir, un nodo que solo aparezca una vez en la lista. Insertamos dicho nodo en una nueva lista, la cual contendrá la ruta a recorrer de manera ordenada.
   b. Seguimos la pista:
      i. Cogemos el otro nodo del par y buscamos su otra aparición, es decir, con quien se conecta. También lo insertamos en la lista de ruta
      ii. Al encontrarlo, insertamos el nuevo nodo del nuevo par en la lista de ruta.
      iii. Repetimos hasta que encontremos el otro nodo extremo, es decir, hasta que no se encuentren más conexiones.
5) El resultado total es:
   a. Una lista con los ID de los nodos a recorrer ordenados de un extremo a otro
   b. Un Float con el coste total de la ruta.

13.1.2 TSP-MST++

El algoritmo *TSP-MST++*, utilizando los resultados de *TSP-MST*, trata de mejorar la ruta eliminando malformaciones que, a pesar de ser completamente validas, realizan rutas ilógicas que aumentan el coste total de la ruta.

A continuación, se muestran dos ejemplos de malformaciones:

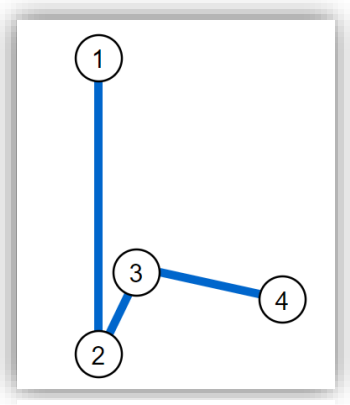


Ilustración 48: Codo de ángulo fino.

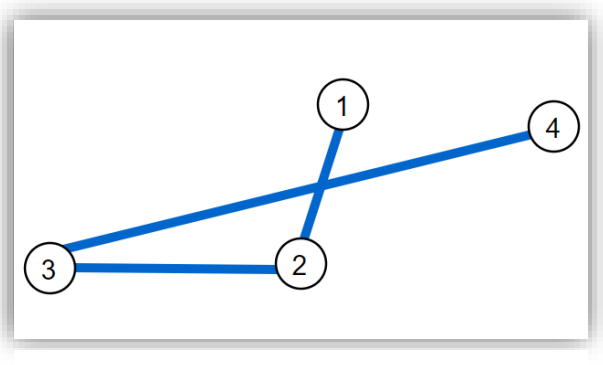


Ilustración 49: Cruce de caminos.

Este tipo de malformaciones pueden arreglarse si se invierte el orden en el que visitamos los dos nodos intermedios:

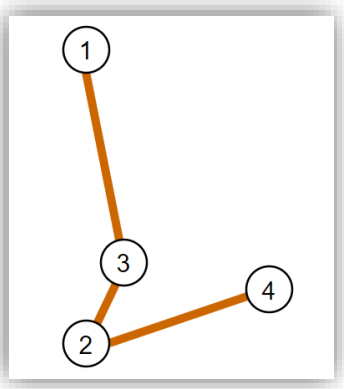


Ilustración 50: Codo de ángulo fino reparado.

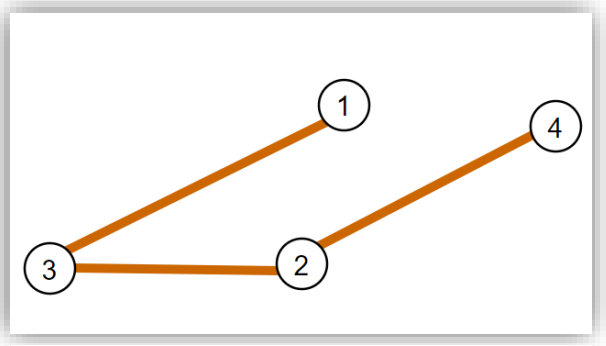


Ilustración 51: Cruce de caminos reparado.

De esta forma, lo que el algoritmo *TSP-MST++* realiza se puede dividir en los siguientes pasos:

1) Para todos los nodos de la lista **agrupados** de 4 en 4, avanzando de 1 en 1 (si la lista es [1 2 3 4 5] primero cogemos [1 2 3 4] y luego [2 3 4 5] y asi sucesivamente) (debe haber al menos 4 nodos en la lista de ruta):
    a. Siguiendo la nomenclatura de las imágenes anteriores, **obtenemos** las siguientes distancias:
        i. Distancia del nodo 1 al nodo 2. (D-12)
        ii. Distancia del nodo 1 al nodo 3. (D-13)
        iii. Distancia del nodo 4 al nodo 3. (D-43)
        iv. Distancia del nodo 4 al nodo 2. (D-42)
    b. **Comparamos** las distancias:
        i. Si D-12 + D-43 es **mayor** que D-13 + d-42; la modificacion sale rentable. Invertimos el orden de los nodos 2 y 3 en la lista de rutas. Modificamos tambien el coste total de la ruta, eliminando la diferencia.
        ii. Si D-12 + D-43 es **menor** que D-13 + d-42; la modificacion **no** sale rentable, por lo que dejamos la lista como esta.
2) Una vez hemos recorrido toda la lista de rutas, esta se ha visto modificada, pudiendo haber generado nuevas malformaciones. Para evitar esto recorremos otra vez la lista realizando de nuevo el paso **1**.
3) Si al realizar el paso **1** no se ha modificado ninguna vez la lista de rutas, consideramos que hemos reparado el mapa por completo y la ejecución termina.
4) Como resultado, obtenemos la lista de rutas reparada junto con su coste reducido.

A continuación, se muestran dos ejemplos locales de reparación del algoritmo *TSP-MST++*, seguido de un ejemplo global.

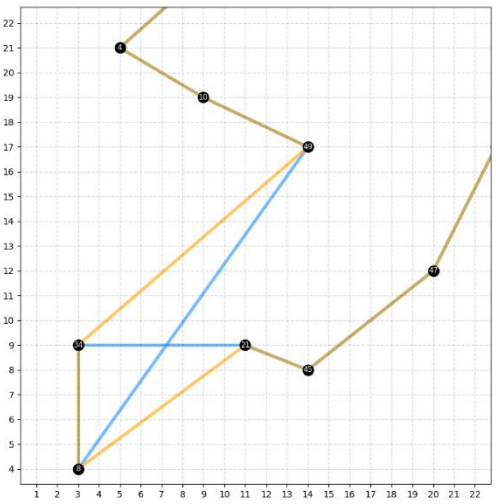


Ilustración 52: Ejemplo de reparación de cruce de caminos.

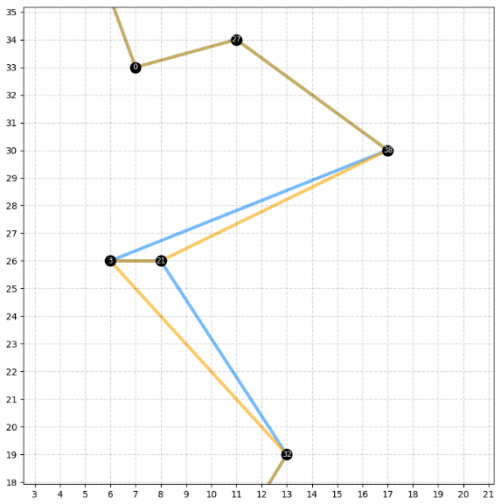


Ilustración 53: Ejemplo de reparación de codo de ángulo fino.

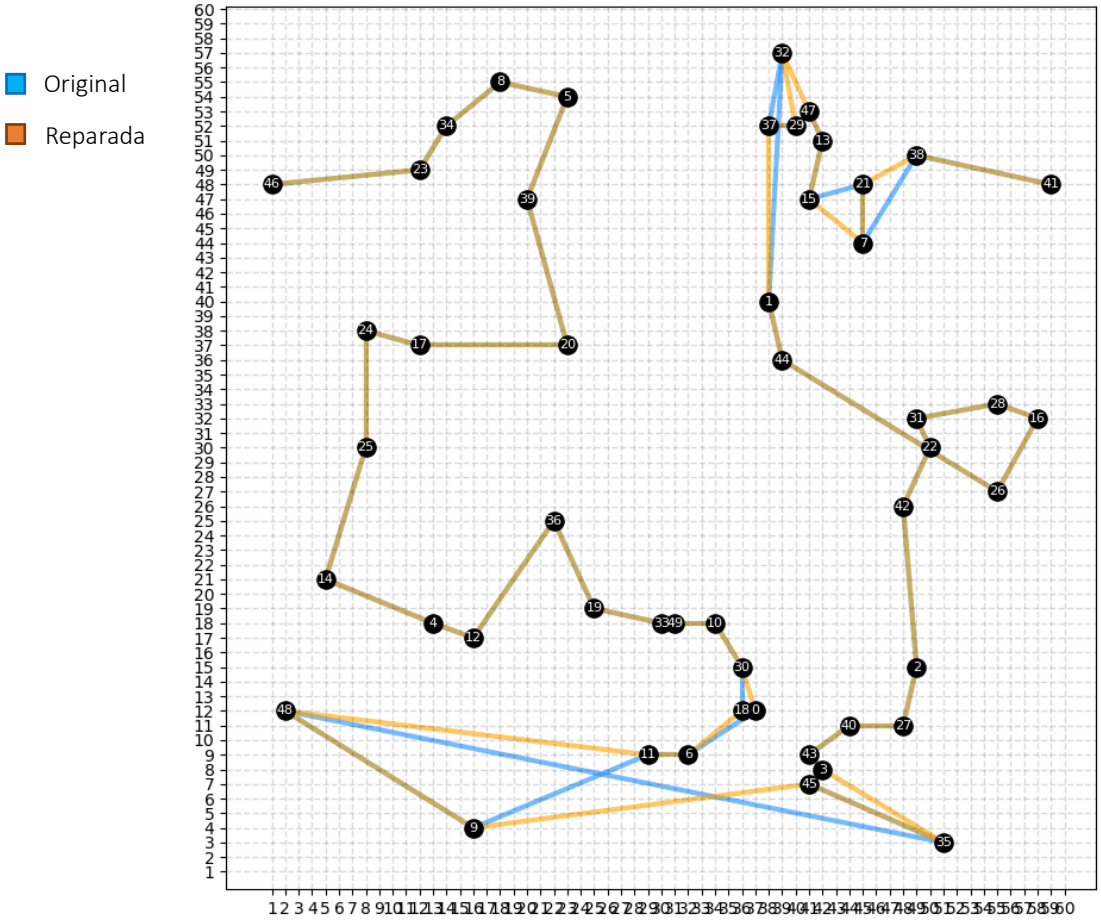


Ilustración 54: Ejemplo de reparación de ruta completa.

14 Bibliografía

14.1 Wikipedia

Colaboradores de Wikipedia. *Problema de la mochila* [en línea]. Wikipedia, La enciclopedia libre, 2018 [fecha de consulta: 16 de febrero del 2019]. Disponible en <https://es.wikipedia.org/w/index.php?title=Problema_de_la_mochila&oldid=106923447>.

Colaboradores de Wikipedia. *Problema de la suma de subconjuntos* [en línea]. Wikipedia, La enciclopedia libre, 2017 [fecha de consulta: 16 de febrero del 2019]. Disponible en <https://es.wikipedia.org/w/index.php?title=Problema_de_la_suma_de_subconjuntos&oldid=99056170>.

Colaboradores de Wikipedia. *Problema del viajante* [en línea]. Wikipedia, La enciclopedia libre, 2019 [fecha de consulta: 16 de febrero del 2019]. Disponible en <https://es.wikipedia.org/w/index.php?title=Problema_del_viajante&oldid=116132674>.

Colaboradores de Wikipedia. *K-medias* [en línea]. Wikipedia, La enciclopedia libre, 2019 [fecha de consulta: 15 de marzo del 2019]. Disponible en <https://es.wikipedia.org/w/index.php?title=K-medias&oldid=115970517>.

Colaboradores de Wikipedia. *Algoritmo genético* [en línea]. Wikipedia, La enciclopedia libre, 2019 [fecha de consulta: 1 de abril del 2019]. Disponible en <https://es.wikipedia.org/w/index.php?title=Algoritmo_gen%C3%A9tico&oldid=114996581>.

Colaboradores de Wikipedia. Inteligencia de enjambre [en línea]. Wikipedia, La enciclopedia libre, 2019 [fecha de consulta: 16 de abril del 2019]. Disponible en <https://es.wikipedia.org/w/index.php?title=Inteligencia_de_enjambre&oldid=116422988>.

14.2 scikit-learn

SCIKIT-LEARN. *Machine Learning in Python*. [fecha de consulta: 16 de marzo del 2019]. Disponible en <https://scikit-learn.org/stable/index.html>

SCIKIT-LEARN. *KMeans*. [fecha de consulta: 16 de marzo del 2019]. Disponible en <https://scikit-learn.org/stable/modules/generated/sklearn.cluster.KMeans.html>

14.3 Numpy

SCIPY. *Numpy and Scipy Documentation*. [fecha de consulta: 10 de febrero del 2019]. Disponible en <https://docs.scipy.org/doc/>

14.4 MatPlotLib

THE MATPLOTLIB DEVELOPMENT TEAM. *MatPlotLib*. [fecha de consulta: 16 de febrero del 2019]. Disponible en <https://matplotlib.org/>

14.5 Cpickle

PYTHON SOFTWARE FOUNDATION. *The Python Standard Library, Pickle – Python object serialization*. [fecha de consulta: 29 de marzo del 2019]. Disponible en <https://docs.python.org/2/library/pickle.html#module-pickle>

PYTHON SOFTWARE FOUNDATION. *The Python Standard Library, Cpickle – A faster Pickle*. [fecha de consulta: 29 de marzo del 2019]. Disponible en <https://docs.python.org/2/library/pickle.html#module-cPickle>

14.6 Py Qt

HEKTOR PROFE. *Primeros pasos en PyQt 5 y Qt Designer: Programas gráficos con Python*. [fecha de consulta: 1 de febrero del 2019]. Disponible en <https://medium.com/@hektorprofe/primeros-pasos-en-pyqt-5-y-qt-designer-programas-gr%C3%A1ficos-con-python-6161fba46060>