UNIVERSIDAD DE CÓRDOBA
Escuela Politécnica Superior
Ingeniería Informática
Metaheurísticas
Francisco José Rodríguez Ramírez
Víctor Monserrat Villatoro

1. Índice Página 1.

2. Objetivo. Página 2.

3. El problema de la mochila.
- **3.1. Descripción.** Página 2.
- **3.2. Análisis.** Página 2.
- **3.3. Desarrollo.** Página 3.
- **3.4. Mejor valor de aptitud para cada iteración (primera mejora).** Página 5.
- **3.5. Mejor valor de aptitud para cada iteración (mejor mejora).** Página 7.
- **3.6. Configuración de la mejor solución (primera mejora).** Página 9.
- **3.7. Configuración de la mejor solución (mejor mejora).** Página 9.

4. El problema del viajante de comercio.
- **4.1. Descripción.** Página 10.
- **4.2. Análisis.** Página 10.
- **4.3. Desarrollo.** Página 11.
- **4.4. Mejor valor de aptitud para cada iteración (primera mejora).** Página 13.
- **4.5. Mejor valor de aptitud para cada iteración (mejor mejora).** Página 15.
- **4.6. Configuración de la mejor solución (primera mejora).** Página 17.
- **4.7. Configuración de la mejor solución (mejor mejora).** Página 17.

2. Objetivos.

El objetivo de esta práctica es resolver dos problemas clásicos, el problema de la mochila (KP) y el del viajante de comercio (TSP), a través de metaheurísticas basadas en trayectorias y particularmente, ejerciendo la búsqueda local. Para ello se leerán instancias de estos problemas desde un archivo, se generarán soluciones aleatorias, se optimizarán localmente cada una de estas soluciones y se evaluarán.

3. El problema de la mochila.

3.1. Descripción.

El problema de la mochila consiste en introducir en una mochila con una capacidad C una serie de objetos con beneficio p_i y peso w_i, el peso de cada objeto es el espacio que ocupa en la mochila y el beneficio el valor que tiene cada objeto individualmente. El problema consiste en maximizar la suma de beneficios de combinaciones de objetos sin superar el peso máximo que soporta la mochila.

3.2. Análisis.

Para resolver problemas de la mochila leemos ficheros en los que se exponen varios problemas de este tipo y almacenaremos cada una de sus configuraciones. Cada problema consta de una mochila, de la cual se nos especifica su capacidad, y de una lista de objetos de los que se nos proporciona su beneficio y peso individual.

Nuestro trabajo consistirá en, una vez cargadas las instancias de este tipo de problema, generar soluciones aleatorias de este, optimizarlas y evaluarlas, teniendo en cuenta las restricciones propias del problema.

Para generar estas soluciones, consideramos la búsqueda aleatoria. Llenaremos la mochila con elementos aleatorios hasta que ningún otro objeto de los que nos quedan por introducir quepa en nuestra mochila, esta será nuestra condición de parada. De esta forma, se generará una solución que llena al máximo la mochila de nuestro problema, aleatoriamente.

Para optimizar estas soluciones, consideramos la búsqueda local. Sacaremos un elemento de la mochila e introduciremos otro de manera aleatoria, sin que el peso del nuevo elemento sobrepase la capacidad de nuestra mochila. De esta manera, obtendremos una nueva solución vecina a la generada en primera instancia de forma totalmente aleatoria.

Para la optimización, tendremos que generar soluciones vecinas y evaluarlas. En nuestro caso, hemos elegido dos estrategias de optimización, primera y mejor mejora. La primera estrategia consiste en generar soluciones vecinas hasta encontrar una vecina que obtenga mayor beneficio. La segunda estrategia consiste en generar un número de soluciones vecinas y almacenar la mejor de todas ellas.

Finalmente, se generarán mil soluciones del problema, se optimizarán a través de los dos tipos de estrategia y evaluaremos todas para poder elegir la que nos interese. En este caso, nuestro interés reside en maximizar el beneficio de la mochila, con lo que iremos almacenando la solución aleatoria que obtenga mayor beneficio.

3.3. Desarrollo.

Para la resolución de este tipo de problemas se han implementado una serie de clases para representar la información de las instancias de estos.

La clase KP::Instance se ha implementado como un vector de KP::Problem, que son problemas de este tipo. Esta clase se instancia a través de un constructor que recibe como único parámetro el nombre del fichero del cual queremos leer un conjunto de problemas de este tipo.

La clase anterior instancia todos los problemas del fichero a través de KP::Problem que se ha implementado como una representación de un problema en sí a través de una mochila KP::Knacksack y un vector de objetos KP::Object.

La mochila KP::Knacksack tiene una capacidad y los objetos KP::Object tienen un beneficio y un peso.

Con estas clases y a través del constructor de KP::Instance se instancian todos los problemas de un fichero en nuestro programa.

Para generar las soluciones se ha creado una clase KP::Solution cuyo constructor recibe un problema KP::Problem y genera una solución aleatoria para este, metiendo elementos del problema aleatoriamente hasta que la mochila no soporta más y así no sobrepasar la capacidad de esta, esta será nuestra condición de parada.

De esta forma, y con esta clase, seremos capaces de generar el número de soluciones que deseemos a un determinado problema de este tipo.

Para la elección de nuestra estrategia de optimización local hemos creado dos nuevas clases, KP::FirstNeighExplorator para la estrategia de primera mejora, y KP::BestNeighExplorator para la estrategia de mejor mejora. Estas dos clases tienen un límite que indica a la búsqueda local el número de soluciones vecinas a generar de máximo hasta encontrar una mejor para la estrategia de primera mejora o como condición de parada para la estrategia de mejor mejora.

KP::FirstNeighExplorator y KP::BestNeighExplorator crean soluciones como se ha comentado en el análisis del problema.

La clase KP::LocalSearch recibe una estrategia de búsqueda local y tiene un método search(), que recibirá una solución aleatoria y aplicará la optimización correspondiente a la estrategia correspondiente. Este método devolverá una solución vecina de mejor beneficio o la propia solución si no ha sido capaz de encontrar una mejor sin sobrepasar el límite de la estrategia.

Finalmente, a través de un programa principal que hace uso de las clases ya descritas se generan soluciones aleatorias para un problema determinado que se introduce a través de un fichero y el número de problema que se quiere solucionar. Este programa genera las soluciones deseadas, se optimizan todas ellas a través de la estrategia correspondiente y se almacena siempre la de mayor beneficio total, con lo que al final de su ejecución se nos informará de esta solución óptima encontrada, pudiendo acceder a su configuración y poder ver qué objetos componen esta solución.

3.4. Mejor valor de aptitud para cada iteración (primera mejora).

El problema que se representa es la instancia número 50 del fichero knapPI_1_200_10000.csv, la capacidad de la mochila es de 499304 unidades y se consta de 200 objetos.

A continuación se representa el valor de aptitud de cada solución inicial, sus soluciones optimizadas y de la mejor solución con una estrategia de primera mejora.

	Solución inicial	Solución optimizada	Mejor solución
Iteración 100	**499695**	**501049**	**501049**
Iteración 200	**444291**	**463582**	**501049**
Iteración 300	**461157**	**463582**	**501049**
Iteración 400	**533716**	**537981**	**537981**
Iteración 500	**516769**	**521786**	**537981**
Iteración 600	**488456**	**488562**	**537981**
Iteración 700	**551161**	**554798**	**554798**
Iteración 800	**489947**	**495339**	**554798**
Iteración 900	**539080**	**539508**	**554798**
Iteración 1000	**524571**	**524785**	**554798**

Como podemos observar en la tabla, la maximización del beneficio se ha conseguido en la iteración 700.

Por norma general, para un número de iteraciones parecido, se suelen obtener soluciones óptimas parecidas, aunque en algunos casos se encuentran diferencias significativas.

Podemos observar que conforme se avanza el número de iteración se reduce la probabilidad de conseguir una solución mejor a las ya generadas aleatoriamente. Es decir, al comienzo de la ejecución conseguimos mejorar rápidamente la solución óptima. Sin embargo, al ir avanzando tendremos que usar mayor número de iteraciones para conseguir mejorar esta solución óptima actual.

A continuación, se representa de forma gráfica los valores de aptitud de algunas de las soluciones generadas por la búsqueda aleatoria junto con los valores de aptitud de las soluciones que ha generado la búsqueda local a partir de estas mismas soluciones anteriores.

Ver de forma interactiva

En rojo, se representan las soluciones generadas en la búsqueda aleatoria, y alrededor su vecindario explorado por la búsqueda local a través de la estrategia correspondiente. Se puede apreciar una estimación de la curva de nuestro problema, los óptimos locales, los vecindarios y el óptimo global.

3.5. Mejor valor de aptitud para cada iteración (mejor mejora).

El problema que se representa es la instancia número 50 del fichero knapPI_1_200_10000.csv, la capacidad de la mochila es de 499304 unidades y se consta de 200 objetos.

A continuación se representa el valor de aptitud de cada solución inicial, sus soluciones optimizadas y de la mejor solución con una estrategia de mejor mejora.

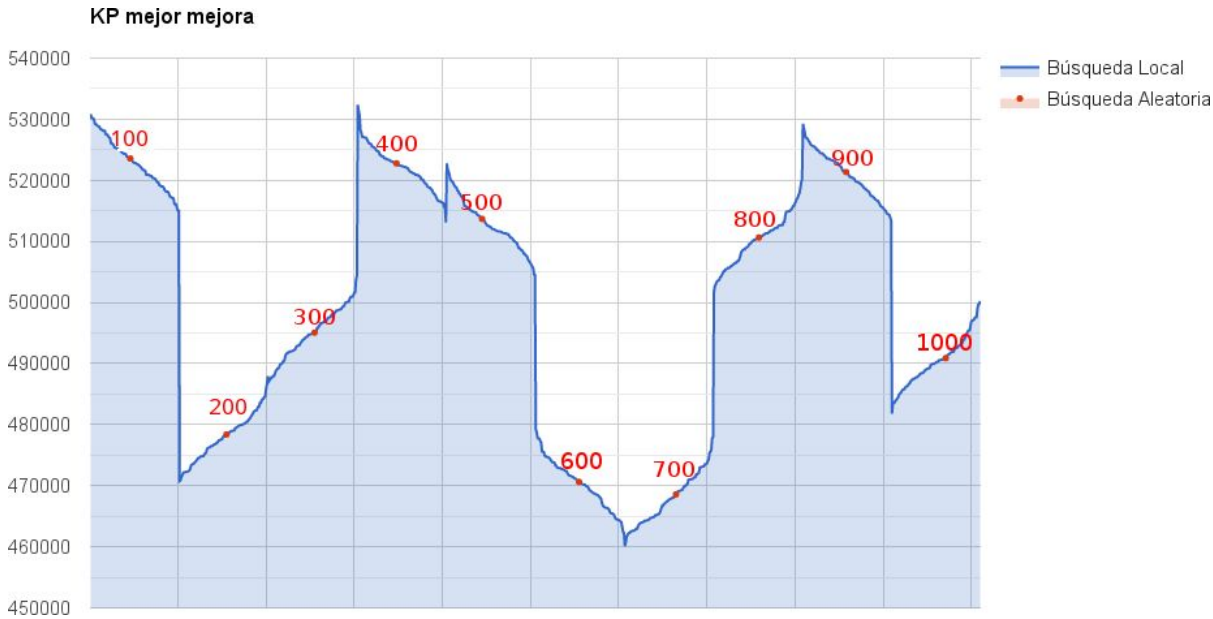
	Solución inicial	Solución optimizada	Mejor solución
Iteración 100	**523569**	**530839**	**530839**
Iteración 200	**478373**	**487947**	**530839**
Iteración 300	**495056**	**504386**	**530839**
Iteración 400	**522769**	**532352**	**532352**
Iteración 500	**513681**	**522770**	**532352**
Iteración 600	**470620**	**479170**	**532352**
Iteración 700	**468572**	**477963**	**532352**
Iteración 800	**510631**	**520214**	**532352**
Iteración 900	**521356**	**529254**	**532352**
Iteración 1000	**490883**	**500244**	**532352**

Como podemos observar en la tabla, la maximización del beneficio se ha conseguido en la iteración 400.

Por norma general, para un número de iteraciones parecido, se suelen obtener soluciones óptimas parecidas, aunque en algunos casos se encuentran diferencias significativas.

Podemos observar que conforme se avanza el número de iteración se reduce la probabilidad de conseguir una solución mejor a las ya generadas aleatoriamente. Es decir, al comienzo de la ejecución conseguimos mejorar rápidamente la solución óptima. Sin embargo, al ir avanzando tendremos que usar mayor número de iteraciones para conseguir mejorar esta solución óptima actual.

A continuación, se representa de forma gráfica los valores de aptitud de algunas de las soluciones generadas por la búsqueda aleatoria junto con los valores de aptitud de las soluciones que ha generado la búsqueda local a partir de estas mismas soluciones anteriores.



Ver de forma interactiva

En rojo, se representan las soluciones generadas en la búsqueda aleatoria, y alrededor su vecindario explorado por la búsqueda local a través de la estrategia correspondiente. Se puede apreciar una estimación de la curva de nuestro problema, los óptimos locales, los vecindarios y el óptimo global.

3.6. Configuración de la mejor solución (primera mejora).

Como ya se ha comentado la mejor solución se consigue en la iteración 700, pero ahora hablaremos de la configuración de esta.

Para esta solución se utilizan 101 de los 200 objetos disponibles, ocupando así con un peso total de 498234 unidades de la capacidad total de 499304 unidades que tiene la mochila en esta instancia del problema.

3.7. Configuración de la mejor solución (mejor mejora).

Como ya se ha comentado la mejor solución se consigue en la iteración 400, pero ahora hablaremos de la configuración de esta.

Para esta solución se utilizan 113 de los 200 objetos disponibles, ocupando así con un peso total de 499232 unidades de la capacidad total de 499304 unidades que tiene la mochila en esta instancia del problema.

4. El problema del viajante de comercio.

4.1. Descripción.

El problema del viajante de comercio consiste en recorrer un grafo totalmente conexo de N nodos de forma cíclica, de forma que se recorran todos los nodos volviendo al punto de partida. Los caminos entre nodos tienen una distancia d_{ij}. El problema consiste en minimizar la distancia total para recorrer todos los nodos del grafo, seleccionando el camino de menor coste.

4.2. Análisis.

Para resolver problemas del viajante de comercio leemos ficheros en los que se exponen varios problemas de este tipo y almacenaremos cada una de sus configuraciones. Cada problema consta de una serie de nodos, con unas coordenadas y una distancia entre cada uno de ellos.

Nuestro trabajo consistirá en, una vez cargadas las instancias de este tipo de problema, generar y evaluar soluciones aleatorias de este, teniendo en cuenta las restricciones propias del problema.

Para generar estas soluciones, consideramos la búsqueda aleatoria. Recorreremos el grafo con un orden totalmente aleatorio sin repetir ningún nodo hasta que lleguemos al nodo inicial, esta será nuestra condición de parada. De esta forma, se generará siempre una solución que recorre el grafo por completo sin pasar dos veces por el mismo nodo a excepción del nodo inicial que será también el nodo final.

Para optimizar estas soluciones, consideramos la búsqueda local. Elegiremos dos nodos al azar del vector trayectoria y daremos la vuelta a todos los comprendidos, de forma que solo cambian dos distancias. De esta manera, obtendremos una nueva solución vecina a la generada en primera instancia de forma totalmente aleatoria.

Para la optimización, tendremos que generar soluciones vecinas y evaluarlas. En nuestro caso, hemos elegido dos estrategias de optimización, primera y mejor mejora. La primera estrategia consiste en generar soluciones vecinas hasta encontrar una vecina que obtenga mayor beneficio. La segunda estrategia consiste en generar un número de soluciones vecinas y almacenar la mejor de todas ellas.

Finalmente, se generarán mil soluciones del problema y evaluaremos todas para poder elegir la que nos interese. En este caso, nuestro interés reside en minimizar la distancia total que habrá que recorrer para pasar por todos los nodos del grafo, con lo que iremos almacenando la solución aleatoria que obtenga una menor distancia total.

4.3. Desarrollo.

Para la resolución de este tipo de problemas se han implementado una serie de clases para representar la información de las instancias de estos.

La clase TSP::Instance se ha implementado como un vector de TSP::Problem, que son problemas de este tipo. Esta clase se instancia a través de un constructor que recibe como único parámetro el nombre del fichero del cual queremos leer un problema de este tipo.

La clase anterior instancia el problema del fichero a través de TSP::Problem que se ha implementado como una representación de un problema en sí a través de un vector de nodos TSP::Node.

Los nodos TSP::Node tienen unas coordenadas *x* e *y* que representan su situación.

Con estas clases y a través del constructor de TSP::Instance se instancian todos los problemas de un fichero en nuestro programa.

Para generar las soluciones se ha creado una clase TSP::Solution cuyo constructor recibe un problema TSP::Problem y genera una solución aleatoria para este, recorriendo el grafo aleatoriamente hasta que se alcanza el nodo inicial pasando sólo una vez por cada nodo intermedio.

De esta forma, y con esta clase, seremos capaces de generar el número de soluciones que deseemos a un determinado problema de este tipo.

Para la elección de nuestra estrategia de optimización local hemos creado dos nuevas clases, TSP::FirstNeighExplorator para la estrategia de primera mejora, y TSP::BestNeighExplorator para la estrategia de mejor mejora. Estas dos clases tienen un límite que indica a la búsqueda local el número de soluciones vecinas a generar de máximo hasta encontrar una mejor para la estrategia de primera mejora o como condición de parada para la estrategia de mejor mejora.

TSP::FirstNeighExplorator y TSP::BestNeighExplorator crean soluciones como se ha comentado en el análisis del problema.

La clase TSP::LocalSearch recibe una estrategia de búsqueda local y tiene un método search(), que recibirá una solución aleatoria y aplicará la optimización correspondiente a la estrategia correspondiente. Este método devolverá una solución vecina de menor distancia o la propia solución si no ha sido capaz de encontrar una mejor sin sobrepasar el límite de la estrategia.

Finalmente, a través de un programa principal que hace uso de las clases ya descritas se generan soluciones aleatorias para un problema determinado que se introduce a través de un fichero. Este programa genera las soluciones deseadas, se optimizan todas ellas a través de la estrategia correspondiente y se almacena siempre la de menor distancia total, con lo que al final de su ejecución se nos informará de esta solución óptima encontrada, pudiendo acceder a su configuración y poder ver qué camino compone esta solución.

4.4. Mejor valor de aptitud para cada iteración (primera mejora).

El problema que se representa es la instancia del fichero berlin52.tsp, la dimensión del grafo es de 52 nodos.

A continuación se representa el valor de aptitud de cada solución inicial, sus soluciones optimizadas y de la mejor solución con una estrategia de primera mejora.

	Solución inicial	Solución optimizada	Mejor solución
Iteración 100	**30318,3**	**30310,4**	**30310,4**
Iteración 200	**28879**	**28312,3**	**28312,3**
Iteración 300	**32648,3**	**325586**	**28312,3**
Iteración 400	**32979,4**	**32909,7**	**28312,3**
Iteración 500	**29224,3**	**29045,3**	**28312,3**
Iteración 600	**29961,3**	**29774**	**28312,3**
Iteración 700	**27199**	**27052,5**	**27052,5**
Iteración 800	**27910,5**	**27420,2**	**27052,5**
Iteración 900	**31109,2**	**30582**	**27052,5**
Iteración 1000	**30361,4**	**30355,8**	**27052,5**

Como podemos observar en la tabla, la minimización de la distancia total se ha conseguido en la iteración 700.

Por norma general, para un número de iteraciones parecido, se suelen obtener soluciones óptimas parecidas, aunque en algunos casos se encuentran diferencias significativas.

Podemos observar que conforme se avanza el número de iteración se reduce la probabilidad de conseguir una solución mejor a las ya generadas aleatoriamente. Es decir, al comienzo de la ejecución conseguimos mejorar rápidamente la solución óptima. Sin embargo, al

ir avanzando tendremos que usar mayor número de iteraciones para conseguir mejorar esta solución óptima actual.

A continuación, se representa de forma gráfica los valores de aptitud de algunas de las soluciones generadas por la búsqueda aleatoria junto con los valores de aptitud de las soluciones que ha generado la búsqueda local a partir de estas mismas soluciones anteriores.

Ver de forma interactiva

En rojo, se representan las soluciones generadas en la búsqueda aleatoria, y alrededor su vecindario explorado por la búsqueda local a través de la estrategia correspondiente. Se puede apreciar una estimación de la curva de nuestro problema, los óptimos locales, los vecindarios y el óptimo global.

4.5. Mejor valor de aptitud para cada iteración (mejor mejora).

El problema que se representa es la instancia del fichero berlin52.tsp, la dimensión del grafo es de 52 nodos.

A continuación se representa el valor de aptitud de cada solución inicial, sus soluciones optimizadas y de la mejor solución con una estrategia de mejor mejora.

	Solución inicial	Solución optimizada	Mejor solución
Iteración 100	**30721,5**	**29568,2**	**29568,2**
Iteración 200	**29662,2**	**28864,8**	**28864,8**
Iteración 300	**31571,2**	**30477,5**	**28864,8**
Iteración 400	**28852,2**	**27688,4**	**27688,4**
Iteración 500	**26833,3**	**25966,1**	**25966,1**
Iteración 600	**31757**	**29785,9**	**25966,1**
Iteración 700	**27348,8**	**25798,3**	**25798,3**
Iteración 800	**32269,6**	**31130,4**	**25798,3**
Iteración 900	**30203,8**	**28710,9**	**25798,3**
Iteración 1000	**28379,7**	**27223,4**	**25798,3**

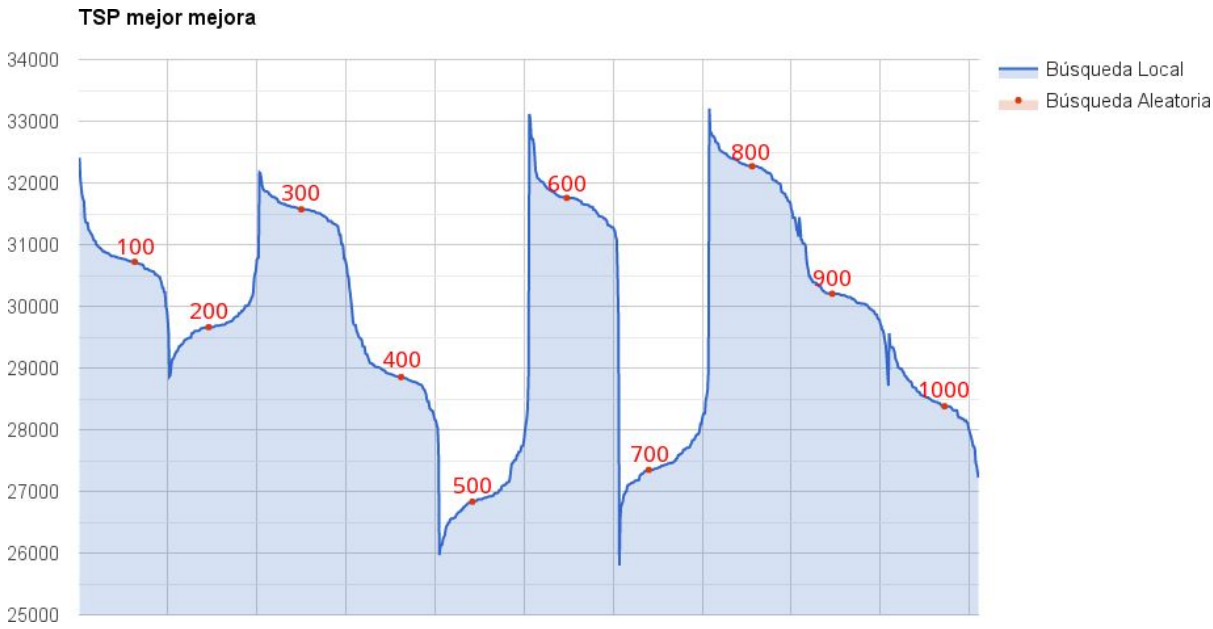
Como podemos observar en la tabla, la minimización de la distancia total se ha conseguido en la iteración 700.

Por norma general, para un número de iteraciones parecido, se suelen obtener soluciones óptimas parecidas, aunque en algunos casos se encuentran diferencias significativas.

Podemos observar que conforme se avanza el número de iteración se reduce la probabilidad de conseguir una solución mejor a las ya generadas aleatoriamente. Es decir, al comienzo de la ejecución conseguimos mejorar rápidamente la solución óptima. Sin embargo, al

ir avanzando tendremos que usar mayor número de iteraciones para conseguir mejorar esta solución óptima actual.

A continuación, se representa de forma gráfica los valores de aptitud de algunas de las soluciones generadas por la búsqueda aleatoria junto con los valores de aptitud de las soluciones que ha generado la búsqueda local a partir de estas mismas soluciones anteriores.



Ver interactivamente

En rojo, se representan las soluciones generadas en la búsqueda aleatoria, y alrededor su vecindario explorado por la búsqueda local a través de la estrategia correspondiente. Se puede apreciar una estimación de la curva de nuestro problema, los óptimos locales, los vecindarios y el óptimo global.

4.6. Configuración de la mejor solución (primera mejora).

Como ya se ha comentado la mejor solución se consigue en la iteración 700, pero ahora hablaremos de la configuración de esta.

Para esta solución se recorre una distancia total de 27052.5 unidades a través de los 52 nodos del problema.

4.7. Configuración de la mejor solución (mejor mejora).

Como ya se ha comentado la mejor solución se consigue en la iteración 700, pero ahora hablaremos de la configuración de esta.

Para esta solución se recorre una distancia total de 25798.3 unidades a través de los 52 nodos del problema.