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

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). Para ello se leerán instancias de estos problemas desde un archivo, se generarán soluciones aleatorias 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 y evaluar soluciones aleatorias de este, 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.

Finalmente, se generarán mil soluciones del problema 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.

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 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.

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. Se han realizado cinco ejecuciones y evaluado la mejor solución para cada iteración.

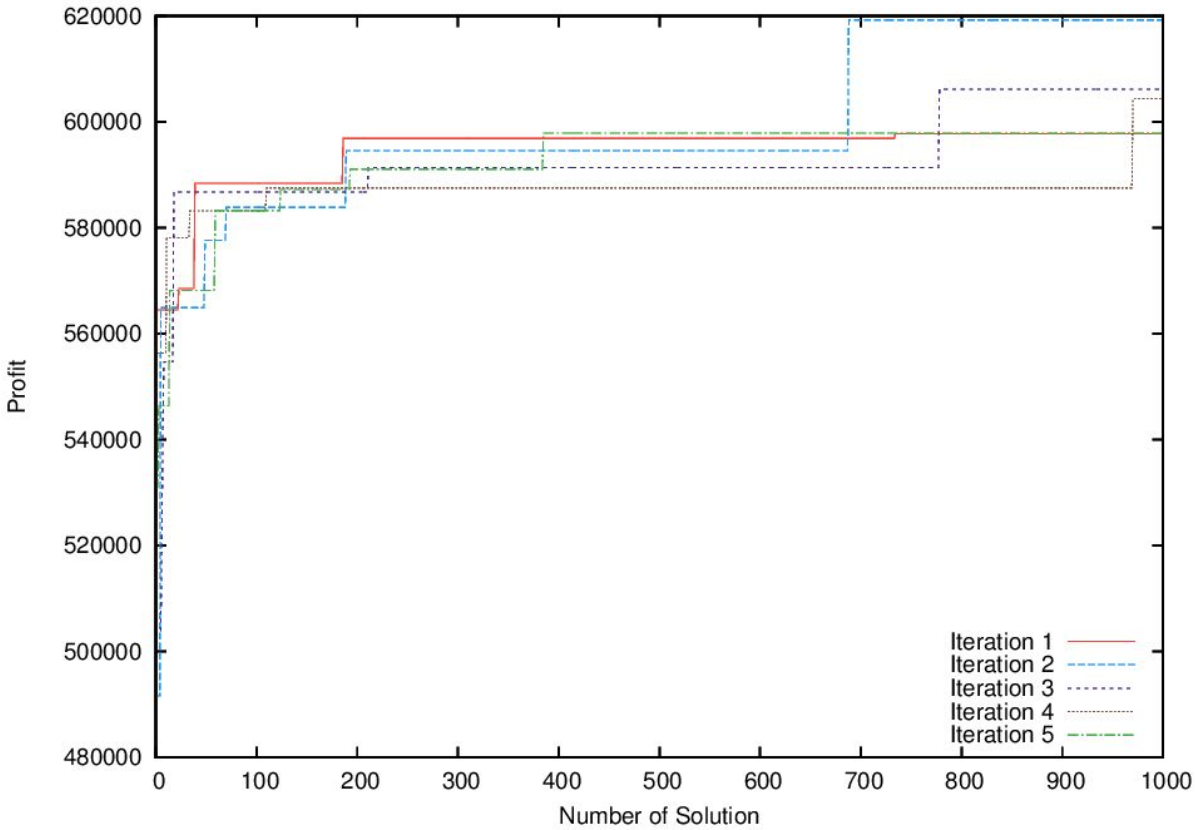
	1 ejecución	2 ejecución	3 ejecución	4 ejecución	5 ejecución
Iteración 100	**588403**	**583892**	**586810**	**583183**	**583275**
Iteración 200	**596894**	**594592**	**586810**	**587569**	**591098**
Iteración 300	**596894**	**594592**	**591402**	**587569**	**591098**
Iteración 400	**596894**	**594592**	**591402**	**587569**	**597905**
Iteración 500	**596894**	**594592**	**591402**	**587569**	**597905**
Iteración 600	**596894**	**594592**	**591402**	**587569**	**597905**
Iteración 700	**596894**	**619250**	**591402**	**587569**	**597905**
Iteración 800	**597757**	**619250**	**606162**	**587569**	**597905**
Iteración 900	**597757**	**619250**	**606162**	**587569**	**597905**
Iteración 1000	**597757**	**619250**	**606162**	**604411**	**597905**

Como podemos observar en la tabla, la maximización del beneficio se ha conseguido en la segunda ejecución.

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 aleatoria mejor a las ya generadas anteriormente. 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.

Si representamos estos mismos datos de la tabla en una gráfica podremos observar lo ya mencionado rápidamente.



Se puede apreciar que las funciones crecen de forma exponencial inversa, es decir, de forma logarítmica. Por ello su función se asemeja bastante, en las primeras iteraciones la optimización crece de una forma muy rápida pero conforme avanza va disminuyendo esta velocidad y la optimización crece cada vez más lentamente, como se puede observar en la figura.

3.5. Configuración de la mejor solución.

Como ya se ha comentado la mejor solución se consigue en la segunda ejecución, 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 499205 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.

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.

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 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.

El problema que se representa es la instancia del fichero berlin52.tsp, la dimensión del grafo es de 52 nodos. Se han realizado cinco ejecuciones y evaluado la mejor solución para cada iteración.

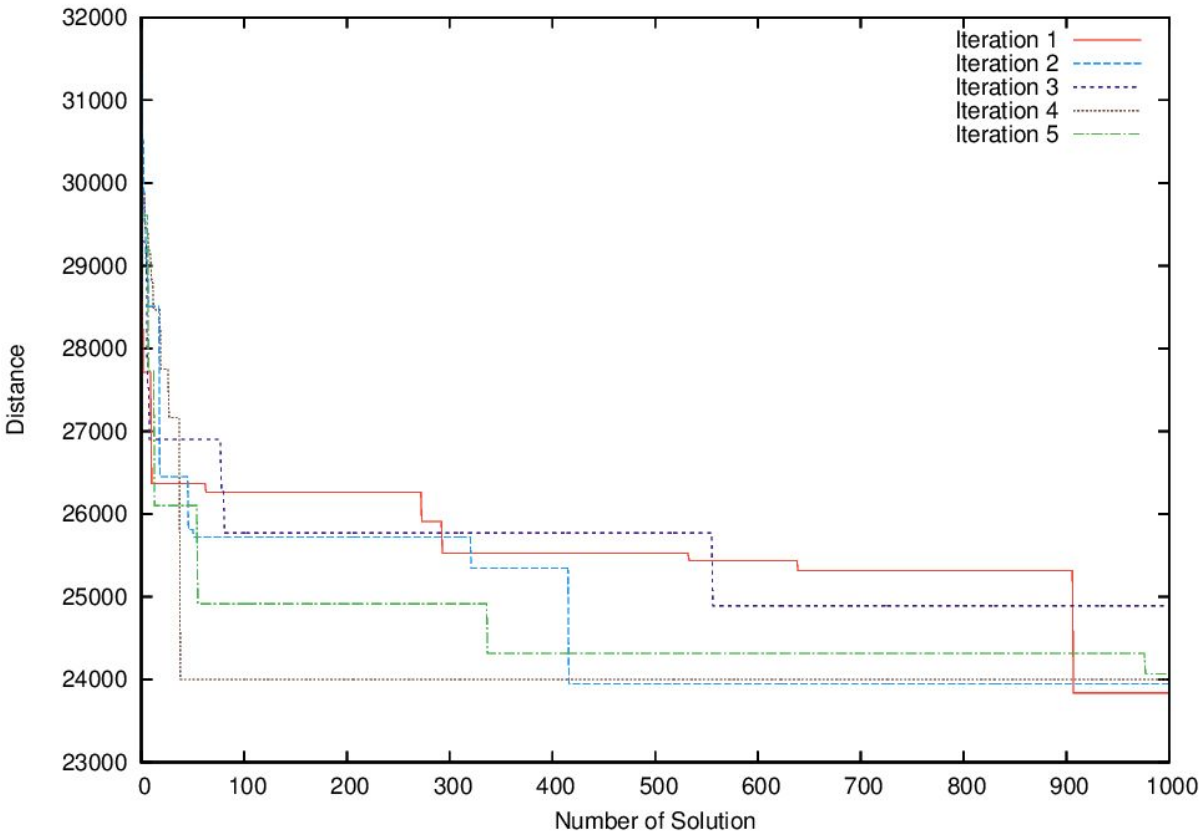
	1 ejecución	2 ejecución	3 ejecución	4 ejecución	5 ejecución
Iteración 100	**26261.3**	**25720.2**	**25768.4**	**23999.9**	**24913.5**
Iteración 200	**26261.3**	**25720.2**	**25768.4**	**23999.9**	**24913.5**
Iteración 300	**25524.2**	**25720.2**	**25768.4**	**23999.9**	**24913.5**
Iteración 400	**25524.2**	**25345.9**	**25768.4**	**23999.9**	**24314.4**
Iteración 500	**25524.2**	**23944.3**	**25768.4**	**23999.9**	**24314.4**
Iteración 600	**25433.4**	**23944.3**	**24888.3**	**23999.9**	**24314.4**
Iteración 700	**25316**	**23944.3**	**24888.3**	**23999.9**	**24314.4**
Iteración 800	**25316**	**23944.3**	**24888.3**	**23999.9**	**24314.4**
Iteración 900	**25316**	**23944.3**	**24888.3**	**23999.9**	**24314.4**
Iteración 1000	**23835.7**	**23944.3**	**24888.3**	**23999.9**	**24066.3**

Como podemos observar en la tabla, la minimización de la distancia total se ha conseguido en la primera ejecución.

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 aleatoria mejor a las ya generadas anteriormente. 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.

Si representamos estos mismos datos de la tabla en una gráfica podremos observar lo ya mencionado rápidamente.



Se puede apreciar que las funciones decrecen de forma logarítmica inversa. Por ello su función se asemeja bastante, en las primeras iteraciones la optimización crece de una forma muy rápida pero conforme avanza va disminuyendo esta velocidad y la optimización crece cada vez más lentamente, como se puede observar en la figura.

4.5. Configuración de la mejor solución.

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

Para esta solución se recorre una distancia total de 23835.7 unidades a través de los 52 nodos del problema.