VRP-Report

Juan García Santos
alu0101325583@ull.edu.es

Universidad de La Laguna
Diseño y Análisis de Algoritmos

Curso 2021-2022

Índice

Introducción

En este informe se expone una propuesta de resolución, codificada en el lenguaje C#, al problema VRP planteado como práctica 7 de la asignatura. Se explicarán los 3 algoritmos implementados así como las estructuras de datos empleadas, su finalidad y para concluir, se mostraran los resultados obtenidos con las múltiples instancias.

1. Sobre las estructuras de datos, de entornos y otros aspectos generales

En la implementación que en este informe se propone, *respecto a la representación del problema y las posibles soluciones* se identificaron ***dos clases***:

1.1. Clase Problema

Esta clase representa nuestro problema y por tanto almacena la siguiente información:

- Un entero que contiene el numero de vehículos disponibles
- Un entero que contiene el número de clientes a visitar
- una matriz de enteros donde se almacenarán los costes de trasladarse desde el cliente i al cliente j, de esta forma el acceso al coste será tan sencillo y eficiente como añadir matriz[i][j]. Puesto que se tratará de un elemento estático, se prefirió usar esta estructura en lugar de Listas (List int) puesto que son mas ligeras y eficientes en estos casos particulares en los que no se realizará una manipulación intensiva de la estructura

1.2. Clase Solución

Esta clase representa una solución al problema. Para ello contiene los siguientes atributos:

- Un entero con el coste global de esa solución
- Una lista de listas de enteros para representar las rutas. Aquí se optó por emplear las listas pues poseen métodos ya implementados que permiten la inserción y la eliminación de elementos en posiciones concretas (algo muy útil puesto que las soluciones serán los elementos que emplearemos para realizar las tareas de diversificación, intensificación y selección)

Ambas clases son las primitivas empleadas por la gran mayoría de los métodos implementados y por ello resulta vital entender cuál es la utilidad real de cada clase.

Adicionalmente, se definieron ***dos interfaces***

1.3. IAlgorithm

Define el método **Solve** que implementan los tres algoritmos definidos (GRASP, Greedy y GVNS). Este método realiza la ejecución de cada algoritmo y dado un problema, devuelve una solución.

1.4. EnvironmentStructure

Define el método **LocalSearch** que implementan todas las estructuras de entorno (2OPT, Reinserción e Intercambio, intra y entre rutas). El método recibe una solución, una referencia a la matriz de distancias entre clientes y devuelve el óptimo local a la solución dada.

Las estructuras de entorno realizan movimientos de clientes (reinserción o intercambio) entre rutas o dentro de una misma ruta. Para una llamada, respecto de la solución dada simula todos los movimientos posibles (del tipo definido por su estructura) y calculando el coste asociado a cada nueva posible solución. Almacena el coste asociado a la mejor solución así como los movimientos necesarios para obtenerla de forma efectiva.

Al finalizar, construye y devuelve la nueva y mejorada solución a partir del coste y los movimientos necesarios para obtenerla. A pesar de que todo este proceso aumenta la complejidad en términos de cálculos del coste y las posiciones usando índices relativos, se prefirió esta forma respecto a construir la solución asociada a cada movimiento por el alto impacto en la eficiencia que tenía generar tantas instancias. Es en este punto donde el usar Listas

2. Greedy

Este algoritmo se caracteriza por escoger el mejor cliente posible en cada momento. Posee un bucle general cuyo criterio de parada es haber visitado todos los clientes; dentro, para cada vehículo disponible, añadimos a su ruta aquel cliente más cercano al último cliente añadido en esa ruta. Así, se construyen en paralelo las rutas de cada vehículo disponible, lo que garantiza cierta homogeneidad en los tamaños de las rutas y facilita la tarea de comprobar que no se supera la capacidad máxima de cada vehículo.

3. Grasp

La versión del algoritmo GRASP que hemos implementado posee un constructor con el que indicarle que estructura de entorno deseamos que emplee para la búsqueda local y se divide en las siguientes fases:

3.1. Inicialización y Construcción

En esta fase generamos una solución inicial usando el método **ConstructGreedyRandomizedSolution**. Este método hace prácticamente el mismo pro-

ceso que el algoritmo Greedy salvo que como próximo cliente para añadir a cada ruta selecciona uno al azar de la lista de candidatos restantes.

3.2. Mejora

Aplicamos una mejora a la solución inicial usando la estructura de entorno indicada en el constructor. Este proceso se engloba dentro de un bucle que se repite hasta que deje de haber mejora y en el que se actualiza la solución inicial por una mejor solución (si la hubiera) en cada iteración.

3.3. Actualización

De la fase anterior se obtiene una solución que se compara con la mejor solución global almacenada hasta el momento. Si la nueva solución es mejor (posee menor coste), se actualiza la mejor solución global con la nueva solución.

3.4. Construcción

Por la implementación específica realizada, al finalizar el bucle se realiza la fase de construcción para obtener una nueva solución inicial a partir de la cual repetir todo el proceso previamente comentado.

4. GVNS

El algoritmo GVNS (en este caso con multi-arranque) se caracteriza por, a partir de una solución inicial, realizar varias fases:

4.1. Inicialización

Se declara una variable k = 0, se genera una solución inicial usando la fase constructiva del algoritmo GRASP así como una Lista de estructuras de entorno ordenadas para que puedan ser usadas en la fase de intensificación; el orden se estableció por consenso de la siguiente forma: Reinserción Entre - Reinserción Intra - Intercambio Entre - Intercambio Intra - 2OPT.

4.2. Diversificación por Shaking

Se declara un bucle que parará cuando k sea igual al número de estructuras de entorno haya en la lista declarada previamente. Dentro de ese bucle se declara una variable l a 0 y realizan dos acciones: primero, shaking respecto de la solución inicial, y con la nueva solución obtenida se inicia la fase de intensificación, en este caso usando búsquedas locales por medio de VND. La solución obtenida de la fase de intensificación se compara con la solución inicial: si la nueva es mejor, se actualiza la solución inicial a la nueva, se reinicia la k a 0 y se repite el proceso anterior pero partiendo de esta nueva solución; en caso opuesto, se le suma 1

al valor de k para así aumentar el rango de salto o diversificaciónrespecto de la solución inicial.

El shaking consiste en, dada una solución y un número k, se retorna una nueva solución a base de aplicar k movimientos únicos y aleatorios de reinserción entre rutas. Esto a nivel conceptual, equivaldría a realizar un salta aleatorio en el espacio de soluciones a una distancia máxima k.

4.3. Intensificación por VND

Se declara un bucle que para cuando l sea igual al tamaño de la lista de estructuras de entorno. Dentro, se realiza una búsqueda local con la estructura de entorno *l-ésima* y se obtiene una nueva solución; esta se compara con la solución dada inicialmente y si mejora, se actualiza el valor de esa solución al de la nueva y se establece l a 0, haciendo que el proceso del VND comience de nuevo pero partiendo de la nueva solución generada. En caso contrario, se aumenta en 1 el valor de l, haciendo que el bucle VND comience de nuevo empleando la solución original pero con otra estructura de entorno.

4.4. Actualización

De las fases anteriores se obtiene una solución que se compara con la mejor solución global almacenada hasta el momento. Si la nueva solución es mejor (posee menor coste), se actualiza la mejor solución global con la nueva solución.

5. Resultados

Los resultados mostrados en estas tablas para cada instancia del problema son el valor medio de 6 ejecuciones para cada instancia con cada algoritmo (menos para Greedy puesto que siempre da los mismos resultados en el mismo tiempo). Adicionalmente, el tamaño de la lista restringida de candidatos fue fijada a 2 en el caso del GRASP y a 3 en el caso de GVNS pues aportaba los mejores resultados y en pos de simplificar lo máximo posible las tablas.

Cuadro 1: Greedy

N vehículos	Coste Total	Tiempo en ms
2	227	1
4	281	0
6	352	0
8	484	0

Cuadro 2: GRASP

N vehículos	Estructura de entorno	Coste Total	Tiempo en ms
2	Reinserción Entre	133	283
2	Reinserción Intra	137	433
2	Intercambio Entre	151	240
2	Intercambio Intra	152	242
4	Reinserción Entre	154	442
4	Reinserción Intra	200	273
4	Intercambio Entre	190	355
4	Intercambio Intra	215	184
6	Reinserción Entre	215	565
6	Reinserción Intra	277	216
6	Intercambio Entre	235	450
6	Intercambio Intra	280	165
8	Reinserción Entre	257	681
8	Reinserción Intra	335	190
8	Intercambio Entre	293	494
8	Intercambio Intra	336	150

Cuadro 3: GRASP

N vehículos	K max value	Coste Total	Tiempo en ms
2	4	114	496
4	4	137	347
6	4	171	478
8	4	203	456
2	3	124	290
4	3	143	258
6	3	182	256
8	3	221	298