**Universidad Simón Bolívar**

**Departamento de Computación y Tecnología de la Información**

**CI5652 - Diseño de Algoritmos II**

**Trimestre Abril - Julio 2012**

**INFORME II**

**Resolviendo el Capacited Vehicle Routing Problem (CVRP)**

**Integrante:**

**Bishma Stornelli 08-11094**

**Vicente Santacoloma 08-11044**

**Sartenejas, 25 de junio de 2012**

**INTRODUCCIÓN**

1. **Motivación del proyecto**

El presente informe pretende realizar una descripción y análisis de la metaheurística **Tabu Search** para resolver el problema del **Capacited Vehicle Routing Problem (CVRP).**

Para realizar el análisis de la metaheurística se empleará diversos resultados experimentales para las corridas de catorce instancias bien conocidas del CVRP que son las propuesta por Christofides, Mingozzi y Toth [1]. Estos resultados permitirán discutir aspectos como: calidad de las soluciones obtenidas,  esfuerzo computacional, robustez y fiabilidad.

Además se prevé discutir otros aspectos acerca de la metaheurística en cuestión, que permitirá tener un criterio acerca de su utilización.

1. **Breve descripción del problema**

El **Vehicle Routing Problem (VRP)**, se define como un grafo no dirigido donde es el conjunto de todos los vértices y es el conjunto de los arcos. El vértice es el depósito de donde podrán salir m vehículos idénticos con capacidad Q, mientras que los vértices restantes representan clientes. La matriz definida sobre E representa los costos no negativos, distancia o tiempo de viaje. Cada cliente tiene una demanda no negativa y un tiempo de servicio no negativo . El **VRP** consiste en diseñar un conjunto de m rutas **i)** del menor costo total, **ii)** cada una empezando y terminando en el depósito, tal que **iii)** cada cliente sea visitado exactamente una vez por el vehículo, **iv)** la demanda total de cada ruta no exceda Q y **v)** que la duración total de cada ruta no exceda una cota D.

El **Capacited Vehicle Routing Problem (CVRP)** es la versión más básica del **VRP**. En el **CVRP** a todos los clientes se les entregarán bienes y sus demandas son determinísticas, conocidas de anticipado. Los vehículos son idénticos y partirán de un único depósito central y solo las restricciones de capacidad de los vehículos son impuestas. El objetivo es minimizar el costo total en despachar a todos los clientes.

1. **Descripción del contenido del informe**

El presente informe consta de:

* **Portada.**
* **Introducción.** Donde se describe brevemente el problema.
* **Algoritmo para el CVRP.** Se presenta una descripción de la metaheurística **Tabu Search**, donde se detalla su funcionamiento, estructuras empleadas y otra información adicional.
* **Detalles de software y hardware empleados.**
* I**nstrucciones de operación.** Descripción detallada de como compilar y correr el software, así como el estado actual de la misma.
* **Resultados experimentales y discusión.** Estudio experimental para caracterizar el rendimiento de la metaheurística propuesta en base a tres aspectos.
* **Conclusiones y recomendaciones.**
* **Referencias bibliográficas.**

**ALGORITMO PARA EL CVRP**

1. **Estructuras de datos utilizados**

**PAQUETES:**

* **CVRP:** Algoritmo para la solución del CVRP mediante la metaheurística Tabu Search.
* **abstracts**
* **TerminationCriteria:** Clase abstracta del criterio de terminación.
* **classes**
* **ClarkeAndWrightAlgorithm:** Heurística Clarke and Wright [2] para obtener la solución inicial del CVRP.
* **CustomerTabu:** Tabu que prohíbe mover a un cliente.
* **CustomerRouteTabu:** Tabu que prohíbe mover a un cliente en una ruta específica.
* **CustomerRouteTabuPosition:** Tabu que prohíbe mover a un cliente a una ruta en una posición específica.
* **Instance:** Instancia del problema en cuestión que además contiene la configuración a realizar del problema.
* **MoveSingle:** Movimiento de un cliente a una ruta en una posición.
* **MoveSwap:** Movimiento de intercambiar dos clientes.
* **MoveTwoOpt:** Movimiento de 2-opt.
* **Neighbor:** Un vecino generado del CVRP para la solución en cuestión. Por cuestiones de almacenamiento en lugar de ser esta la solución vecina, se tendrá la solución en cuestión junto con el movimiento que producirá esa solución vecina.
* **NeighborSelectorBest:** Selecciona el mejor vecino de la vecindad.
* **NeighborSelectorFirst:** Selecciona el primer mejor vecino de la vecindad.
* **NeighborhoodStructureClassic:** Estructura para generar una vecindad a partir de una solución, mediante el movimiento de un cliente de una ruta a otra.
* **NeighborhoodStructureSwap:** Estructura para generar una vecindad a partir de una solución, mediante el intercambio de dos clientes.
* **NeighborhoodStructureTwoOpt:** Estructura para generar una vecindad a partir de una solución, mediante el 2-opt.
* **NeighborhoodStructureLambda:** Estructura para generar una vecindad a partir de una solución, mediante la combinación de cualquiera de las tres estructuras generadoras anteriores.
* **PrintableSolution:** Versión imprimible de una solución.
* **Route:** Representa una ruta de una solución.
* **Solution:** Representa una solución para el CVRP.
* **TerminationCriteriaImproving:** Criterio de terminación por un cierto número de iteraciones, basado en la cantidad de clientes, sin mejorar la solución actual encontrada.
* **TerminationCriteriaIteration:** Criterio de terminación por un número de iteraciones basado en la cantidad de clientes.
* **interfaces**
* **Move:** Interfaz de cada las estructuras de movimientos.
* **NeighborSelector:** Interfaz de las estructuras para la selección de un vecino de la vecindad.
* **NeighborhoodStructure:** Interfaz de las estructuras para la generación de una vecindad.
* **Tabu:** Interfaz de las estructuras del Tabu.
* **exceptions**
* **MaxCapacityExceededException:** Excepción por violación de la capacidad de carga permitida por un vehículo.
* **MaxDurationExceededException:** Excepción por violación de la duración permitida por una ruta.
* **TabuListFullException:** Excepción por haberse llenado la lista tabu en base a un valor dado. En dicho caso deberá ser parcialmente vaciada.
* **TerminationCriteriaNotStartedExcepcion:** Excepción por no inicializar el criterio de terminación antes de que este inicie.
* **UnexpectedAmountOfCustomersException:** Excepción por cantidad no permitida de clientes en la generación de un vecino.

1. **Descripción de los principales algoritmos para resolver el problema**

Para resolver el **CVRP**, como ya se mencionó anteriormente se empleará la metaheurística **Tabu Search**. Este metaheurística consiste básicamente en almacenar las diversas acciones tomadas para posteriormente tomar decisiones en base a ellas.

En el caso del **CVRP** se almacenarán los diversos movimientos realizados para prohibir que estos vuelvan a ser realizados por un periodo dado. Para este caso volverán a permitir movimientos prohibidos cada vez que se llene la lista Tabu, siendo estos movimientos los realizados con mayor anterioridad.

Para obtener la solución inicial del problema se utiliza el algoritmo de **Clarke & Wrights** [2], el cual consistes en recortar la duración de cada ruta mediante la mezcla de dos rutas para así reducir la distancia entre un extremo de cada ruta con el depósito.

1. **Parámetros que utiliza el algoritmo**

Este software requiere de dos archivos:

* **Instancia del problema:** Este archivo contiene en la primera línea el número de clientes, la capacidad de los vehículos, el tiempo máximo de las rutas y el tiempo de descarga. En la segunda línea contiene las coordenadas del depósito. En las siguientes líneas se encuentran las coordenadas de cada cliente seguido por su demanda.
* **settings:** archivo de configuración para especificar las modalidades con las que se resolverá el problema. Más detalles de este archivo en la sección de Instrucciones de Operación.

1. **Información adicional**

**DETALLES DE SOFTWARE Y HARDWARE EMPLEADO**

**Hardware Empleado**

**Procesador:** Intel(R) Core(TM) 2 Duo CPU    P8600    @ 2.40GHz

**Memoria RAM:** 4GiB

**Software Empleado**

**Sistema Operativo:** Debian GNU/Linux Wheezy.

**Kernel:** Linux version 3.1.0-1-amd64

**Lenguaje de Programación:** JAVA

JDK

JRE

**NetBeans:** 7.1.2

**Estado Actual**

**INSTRUCCIONES DE OPERACION**

Para instalar el software primero debemos descomprimir el archivo en formato .tar.gz mediante:

***$ tar -xvf STSP.tar.gz***

Para compilar el código en plataforma linux, deberemos posicionarnos en la carpeta STSP y luego ejecutar el comando:

***$ make***

Luego se producirá el ejecutable ***stsp***. Para ejecutarlo escribimos:

***$ ./stsp <tsplibfile>***

Donde ***<tsplibfile>*** es una instancia del STSP las cuales se encuentra en el subdirectorio ***ALL\_tsp***

Por ejemplo:

***$ ./stsp ALL\_tsp/eil51.tsp***

**RESULTADOS EXPERIMENTALES Y DISCUSIÓN**

**TABLA DE RESULTADOS 1**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **A** | **eil51.tsp** | **kroA100.tsp** | **d198.tsp** | **rat783.tsp** |
| **B** | 51 | 100 | 198 | 783 |
| **C** | 406 | 21282 | 15780.9 | 8966.2 |
| **D** | 0% | 0% | 0,005703422% | 1,819214% |
| **E** | 0 | 0 | 0,875595 | 10,36876 |
| **F** | 406 | 21282 | 15780 | 8948 |
| **G** | 10 | 10 | 4 | 0 |

**Reseña:**

1. Nombre de la instancia.
2. Número de ciudades.
3. Distancia promedio de 10 corridas de la heurística.
4. Porcentaje de desviación de la distancia promedio de la heurística, con respecto a la solución óptima.
5. Desviación estándar del valor promedio de la heurística.
6. Distancia de la mejor solución obtenida en las 10 corridas de la heurística.
7. Número de ocurrencias de la mejor solución en las 10 corridas de la heurística.

**TABLA DE RESULTADOS 2**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **A** | **eil51.tsp** | **kroA100.tsp** | **d198.tsp** | **rat783.tsp** |
| **B** | 406 | 21282 | 15780 | 8806 |
| **C** | 406 | 21282 | 15780 | 8948 |
| **D** | 0% | 0% | 0% | 1,614808% |
| **E** | 5100 | 10000 | 19800 | 78300 |
| **F** | 22,3 | 47,1 | 8109,2 | 36852,3 |
| **G** | 7 | 0 | 242 | 13786 |
| **H** | 0,137254% | 0,471% | 40,95556% | 52.93448% |
| **I** | 0,459228 | 2,960584 | 14,73412 | 303,2057 |
| **J** | 0,0068 | 0,0144005 | 6,026776 | 142,8785 |
| **K** | 0 | 0 | 0,184011 | 52,40328 |
| **L** | 1,480745% | 0,486407% | 40,90354% | 47,12263% |

**Reseña:**

1. Nombre de la instancia.
2. Distancia de la solución óptima.
3. Distancia de la mejor solución obtenida en las 10 corridas de la heurística.
4. Porcentaje de desviación de la mejor solución de la heurística con respecto a la solución óptima.
5. Número total de iteraciones.
6. Número promedio de iteraciones para encontrar la mejor solución.
7. Mejor número de iteraciones para encontrar la mejor solución.
8. Porcentaje de iteraciones promedio para encontrar la mejor solución con respecto al número total de iteraciones.
9. Tiempo promedio de las 10 corridas de la heurística, en segundos.
10. Tiempo promedio de las 10 corridas de la heurística para encontrar la mejor solución, en segundos.
11. Mejor tiempo para encontrar la mejor solución.
12. Porcentaje de tiempo promedio para encontrar la mejor solución con respecto al tiempo total promedio.

**ANÁLISIS DE RESULTADOS**

Tomando en cuenta los tres aspectos:

1. **Calidad de las soluciones que son obtenidas.**

Tal como se puede ver en las tablas de resultados, para 3 de las 4 instancias se logró la distancia mínima óptima. Además en las dos primeras se logró dicha distancia óptima en cada una de las corridas.

Para la última instancia sin bien no se logró la distancia óptima, podemos decir que esta fue bastante aceptable.

Ahora, para evaluar la calidad de las soluciones, es decir, ver que tan buenas son en comparación con las óptimas, deberemos considerar otros mecanismos puesto que para un problema no conocido, no podremos saber su solución óptima.

Uno de estos mecanismos es tomar en cuenta para un número máximo de iteraciones, a partir de ¿cuál esta logró obtener su mejor solución? Para la última instancia, se puede apreciar que en promedio a la iteración 36852 se logró la mejor solución, lo que representa aproximadamente el 53% de las iteraciones totales. Esto es, que en casi la mitad del número de iteraciones, la heurística no logró conseguir un mejor resultado. Por lo tanto podremos decir que la solución obtenida, puede ser buena, pudiendo en algunos casos llegar al óptimo tal como se aprecia en las demás instancias, en particular, en la primera y segunda, donde se necesitó menos del 1% para lograr la mejor solución.

Ahora bien, si por ejemplo nuestro algoritmo para un total de 1000 iteraciones encontró la mejor solución en la iteración 900, no podemos dar ninguna idea de la calidad de la solución, a menos que la cantidad de ciudades para la instancia del problema fuera muy pequeña.

Finalmente, cabe señalar que estos mecanismos para evaluar la calidad de una solución, solo deben ser usados para tener un indicio y nunca como un reflejo de la realidad, puesto que nuestra heurística no considera todo el espacio de soluciones; solo un pequeño subconjunto. Alternativamente en lugar de solo considerar el número de iteraciones, se podría hacer en base al tiempo total de ejecución, comparándolo con el tiempo en el que se encontró la mejor solución.

1. **Esfuerzo computacional**

Para la última instancia el tiempo computacional requerido para el total de iteraciones fue relativamente pequeño, tomando para cada caso, su número de ciudades. Además si se observa el tiempo con el cual se logró obtener la mejor solución, podemos decir que este es mucho menor, que el que se necesitó a lo largo de todas las iteraciones.

Cabe destacar, que a medida que aumenta la cantidad de ciudades, mayor es el tiempo que se va a necesitar, para aumentar la posibilidad de conseguir mejores soluciones. Sin embargo para un rango de instancia que tenga una cantidad de pequeña a mediana de ciudades, el tiempo requerido para obtener buenos resultados es aceptable. Gracias a la simplicidad del algoritmo, este es bastante rápido en términos computacionales.

1. **Robustez y fiabilidad**

La heurística **iterated-hill-climber** permite obtener buenos resultados para problemas que no tengan una cantidad excesiva de ciudades, sin tener que modificar la cantidad de iteraciones a realizar. Tal como se mencionó antes, la cantidad de iteraciones, se asignó en función de la cantidad de ciudades multiplicadas por 100. Sin embargo si el problema es grande, muy probablemente necesitaremos más iteraciones, para quizás obtener mejor resultados. Claro está que no podemos simplemente colocarle una cantidad excesiva de iteraciones, porque nuestro algoritmo nunca terminaría.

Un aspecto importante que vale la pena señalar es que el obtener buenas soluciones está determinado por la escogencia aleatoria de un tour. En la tabla 2, se puede apreciar que en la iteración 0 para la segunda instancia, se logró obtener el mejor tour o uno de los tour vecino al mejor, que resultó ser o permitió conseguir el óptimo. Esto digamos, pudo ser un golpe de suerte. Para otras corridas podríamos no lograr nunca esto, y estar seleccionando siempre los peores tour. Lo que si podemos concluir es que a medida que crece el tamaño del problema, menor es la probabilidad de ir conseguir aleatoriamente mejores tours. Esto sería equivalente a ganarse la lotería. Para lo cual si queremos tener más chance de ganarla deberemos comprar la mayoría o todos los boletos, que se traduce en ejecutar la heurística por un tiempo muy largo, que en la práctica es inviable, pese a la ventaja en rapidez comparado con una búsqueda exhaustiva.

**CONCLUSIONES Y RECOMENDACIONES**

En base al análisis y discusión de resultados, realizado a lo largo del informe, podemos en primer lugar concluir que esta heurística representa una buena opción para resolver el problema CVRP y que tiene muchas variables ajustables para obtener los resultados.

Podemos decir que las ventajas de usar la metaheurística de tabu-search para resolver el problema CVRP, es que dado la complejidad del mismo, nos permite jugar bastante con la configuración que podemos usar para resolverlo y obtener así buenos resultados en un tiempo razonable.

Entre las desventajas podemos decir, que este algoritmo no garantiza obtener soluciones óptimas; la solución que se obtenga depende de varios factores que se pueden considerar aleatorios y por lo tanto no es determinístico; y finalmente, no es posible proporcionar una cota superior del tiempo computacional, especialmente para problemas muy grandes.

Evaluando las ventajas y desventajas se puede señalar que esta metaheurística es una de las mejores para resolver el problema del CVRP. Sin embargo, es necesario implementar mejores formas de generar las vecindades, agregando por ejemplo, criterios de aspiración, diversificadores e intensificadores que permitan alejarnos un poco más del espacio de búsqueda que se explora y optimizar ciertas partes de la solución para lograr conseguir mejores óptimos locales que puedan llegar a ser óptimos globales.

**REFERENCIAS BIBLIOGRÁFICAS**

1. Barr, R., Golden, B., Kelly, J., Resende, M., and Stewart, W. Designing and reporting on computational experiments with heuristic methods. Journal of Heuristics 1, 1 (1995).
2. Gutin, G., and Punnen, A. The traveling salesman problem and its variations, vol. 12. Kluwer Academic Pub, 2002.
3. Hoos, H., and Stutzle, T. Stochastic local search: Foundations and applications. Morgan Kaufmann, 2005.
4. Michalewicz, Z., and Fogel, D. How to solve it: modern heuristics. Springer-Verlag New York Inc, 2000.
5. Reinelt, G. Tsplib. http://comopt.ifi.uniheidelberg.de/software/TSPLIB95/, 2012.
6. Thomas Stuetzle. TSP-TEST, Version 0.9. Available from http://www.sls-book.net, 2004.