Inteligencia Artificial

Estado del Arte: Vehicle Routing Problem *(VRP)*

Elian Vallejos

07 de Junio de 2014

Evaluación

Mejoras 1ra Entrega (10 %):	_____
Código Fuente (10 %):	_____
Representación (15 %):	_____
Descripción del algoritmo (20 %):	_____
Experimentos (10 %):	_____
Resultados (10 %):	_____
Conclusiones (20 %):	_____
Bibliografía (5 %):	_____
Nota Final (100):	_____

Resumen

Lo que se pretende presenta en el siguiente documento, es el problema *Vehicle Routing Problem*.
Este problema es una variante del conocido *Traveling Salesmen Problem (TSP)*, lo que aquí se quiere realizar, es satisfacer la demanda de los clientes, entregando se mercadería, minimizando los costos asociados a las rutas de entrega. Para esto se cuenta con una flota de vehículos, los que son todos homogéneos. Además de realizar la entrega, los vehículos deben volver al punto de origen, o sea a su depósito. Lo más importante en esta problemática es poder satisfacer la necesidad de los clientes, teniendo en cuenta los recursos con los cuales se cuenta, dado que no hay muchas variables en el problema, no es necesario ahondar más en esto. A continuación se procederá a definir el problema y buscar un método que logre obtener una solución mas cercana al óptimo. Para esto, se presentara un planteamiento formal, estado del arte del problema, posibles métodos de solución y descripción del modelo matemático elegido.

1. Introducción

Vehicle Routing Problem (VRP), es un problema actual que comprenden todas aquellas empresas de reparto, La tarea de organizar las flotas de vehículos. La tarea de minimizar los costos asociados a las entregas, sin desmerecer los tiempos y satisfacción de los clientes, es un desafío complejo para la logística y una problemática difícilmente de resolver.
Antes de ir más alla, cabe destacar de que este es un problema que nace del TSP (*Traveling Salesman Problem*), dado que este es el problema más simple de abordar lo que se presentara aquí es una variable de este, pero en sus principios y bases son iguales [6].
Actualmente el mejoramiento y la optimización en los procesos de distribución trae consigo grandes ahorros, dado que antiguamente no era una problemática tan compleja como es en los días de hoy. El problema ha ido evolucionando hasta encontrar las distintas variantes, inclusive la mezcla de cada una de ellas conlleva una aproximación a lo que es el mundo real de hoy en día.
Para comprender el significado de esto, es que a través de los distintos temas abarcados en este Paper, se conocerán las variables que existen, junto con su explicación y que diferencia tienen entre ellos. Se entregara una visión actual del problema, cuales son los algoritmos que se utilizan actualmente [5], no se explican técnicas completas, dado que estas no son utilizadas en plenitud en la actualidad debido al tiempo de computo que necesitan para resolver, es por esto que se usan técnicas incompletas de resolución, para así tener en un tiempo aceptable, soluciones de buena calidad. Como se trata de un problema matemático, se dará a conocer su modelo, además de las restricción que este implica, su función de evaluación, y las constantes o parámetros que se tienen que considerar.
Algo para tener en cuenta, es que la motivación que persigue esto, es la necesidad de comprender mas en profundidad y con problemas reales, la problemática que se enfrenta al implementar soluciones para esta clase de problemas y tener un contraste entre lo que se aprende en un aula, junto con lo que se aprende en la investigación.

2. Definición del Problema

El problema *Vehicle Routing Problem* nace de la necesidad de tener el manejo efectivo de la provisión de bienes o servicios, ya sea en la o retiro de estos en distintos puntos geográficos, dada el inmenso crecimiento de las ciudades y de la población se hacia imposible generar rutas óptimas en tiempos lo suficientemente acotados, como para tener un mejor manejo de los recursos disponibles. Se ha demostrado que de acuerdo aplicaciones del mundo real que una buena planeación de los procesos de distribución genera ahorros del 5 % al 20 % en los costos de transporte global.
El problema *VRP* es un problema clásico de optimización combinatoria con múltiples aplicaciones. Este consiste en general en lo siguiente:

- Un depósito central
- Clientes que requieren que se les entregue productos, los cuales tienen cierta demanda.
- Una flota de vehículos disponibles para realizar las entregas.
- Se requiere la planeación para la entrega de productos a los consumidores.
- Se necesita mínimizar los costos de transporte para la entrega de los productos, sean los costos: distancia de cada circuito, número de vehículos utilizados, tiempo de transporte, etc. ítem Generar las rutas necesarias para la entrega de todos los productos a los clientes.

De acuerdo a lo anterior, se puede apreciar cuales son las variables, restricciones, etc. Primero se tienen las constantes, las que son la cantidad de vehículos y su capacidad. Las variables, son todos aquellos clientes que necesitan que se les entreguen productos, ademas del deposito central, si bien este no tiene demanda, pero si tiene una distancia asociada a cada ciudad. Las restricciones en este caso, es la capacidad que tiene que cada vehículo, dado que los vehículos no pueden entregar más productos que su máxima capacidad. La función de evaluación es tener la mínima distancia que cumpla la entrega para cada cliente, tomando en cuenta cada vehículo. Para describir los tour que se generan para cada vehículo, se describen mediante grafos, cada nodo es un cliente y los arcos son el camino que tiene asociado cada tramo. Además cada arco tiene asociado un costo.

2.1. Variantes del Problema

Existen múltiples variables a este problema, a continuación se describen, cabe destacar que este problema dada su complejidad es mas díficil que el TSP que es muy parecido

2.1.1. Traveling Salesman Problem (TSP)

El problema del vendedor viajero es uno de los problemas más estudiados, este problema responde a la pregunta, que dada una lista de ciudades junto con las distancias de esta, ¿ Cual es la ruta mas corta posible para visitar cada ciudad exacedemos con elogios solamente! es hora de dar el gran salttamente una vez, y regresar a la ciudad de origen?. Este es un problema NP-duro [6] además de ser netamente combinatorio. Porque mencionar este problema, dado que se puede decir que es el problema madre para *VRP*, prácticamente es el mismo salvo que para el *TSP* solo es un vehículo, por ende se puede encontrar mucha más literatura para la resolución de este problema, no se ha ahondara más dado que abarca demasiado y solo es útil su mención y en que consiste.
Del *TSP* se desprenden además variantes a este problema, que en este caso conviene mencionarlos igualmente dada su similitud con el *VRP* [6]

- Traveling Salesman Problem with Backhauls (TSPB)
- Traveling Salesman Problem with Time Windows (TSPTW)
- Multiple Traveling Salesman Problem (MTSP)

En general todas variaciones anteriores se aplican de una manera al *VRP*, dado que el *TSP* no es lo que se quiere explicar, no hay necesidad de detallar mas las variantes, de este.

2.1.2. Capacitated Vehicle Routing Problem (CVRP)

Esta variación al igual que el problema estudiado es una flota de vehículos de reparto con una capacidad uniforme, los cuales tienen que atender la demanda conocida de cliente, en distintos puntos geográficos, satisfaciendo la restricción de que las rutas descritas tienen que ser a un mínimo costo tal cual lo haría el *VRP*, con la salvedad de que la capacidad de cada vehículo es uniforme de una sola mercadería [3].
El objetivo de esta variación del problema es que se quiere reducir al mínimo posible la flota de los vehículos de reparto, además de la suma de los tiempos de viaje, considerando también de que la capacidad de cada vehículo no puede ser excedida por la ruta a tomar. Su formulación es la misma que para el *VRP* agregando la restricción de que la demanda total de los clientes en una misma ruta no debe sobrepasar la capacidad del vehículo asignado a la misma.

2.1.3. Distance Constrained Vehicle Routing Problem (DCVRP)

Esta variante del problema radica en que a diferencia del *Capacitated Vehicle Routing Problem (CVRP)* aquí la restricción de la capacidad del vehículo es cambiada por la distancia [6] en que se tiene que hacer el recorrido, dado que existe un limite máximo el cual tiene que tener la ruta. Sigue teniendo múltiples vehículos, punto de salida y final el mismo, etc. Aquí usualmente el costo se relaciona directamente con el viaje realizado o con el tiempo de este.

2.1.4. Vehicle Routing Problem with Backhauls (VRPB)

La variante *VRPB* también conocida como linehault-backhault problem, tiene la particularidad que un cliente, puede estar satisfecho o no con el producto que se le entrega, dada esta premisa, este podrá devolver o no el producto al vehículo repartidor [5], dada esta problemática es que se incorporan como restricciones el contenedor del vehículo como algo dimensional, ya que es necesario saber la capacidad de este además de si es necesario reordenar la carga, para así poder llevar todo nuevamente al depósito [5], sin que el vehículo se quede sin capacidad en algún momento.

2.1.5. Vehicle Routing Problem with Time Windows (VRPTW)

El *VRPTW* se diferencia con los antecesores en que aquí existe un periodo de tiempo en el cual el vehículo debe iniciar el recorrido para cada cliente, para así llegar sin problemas en el tiempo esperado por el cliente [1]. Esta variante además toma en consideración, de que si el vehículo llega antes del tiempo estimado, debe esperar un tiempo t para que empiece nuevamente el servicio, de manera que los plazos se cumplan cabalmente. Unas de sus aplicaciones son para el encaminamiento del autobús escolar y la programación de lineas aéreas.

2.1.6. Vehicle Routing Problem with Pickup and Delivery (VRPPD)

Aquí lo que se modela es que cada cliente pide una cierta cantidad de productos, los cuales tienen que ser recogidos en el almacén, para luego ser llevados por el vehículo hasta el cliente. Cada vehículo tiene una capacidad de carga, por lo que los productos deben caber dentro del vehículo para así poder ser entregados al cliente, aquí el problema es mucho mas realista en algunos puntos, al igual que sus antecesores se debe considerar la ruta, los clientes, y además poder minimizar los costos, ya sea ruta mas corta, cantidad de vehículos empleados, para así poder satisfacer la demanda de los clientes. [9].

3. Estado del Arte

El problema *VRP* en un problema NP-difícil , esto quiere decir que es un problema que intuitivamente es al menos tan difícil como cualquier problema NP. El *VRP*, es una variante mas compleja del TSP.
Actualmente la literatura se encuentra mucha información de este problema, siendo las variantes expuestas en la *Definición del Problema* , las mas ocurrentes, dado que se centran en la problemática de hoy en día, en cuanto a la entrega de productos mediante la vía de medios de transporte. El *VRP* es uno de los problemas más comunes y más estudiados, dada su derivación del *TSP*, su evolución histórica data del año 1956 hasta los trabajos mas recientes. El *VRP*, se conoce como tal desde el año 1959 [10], el cual se modela la entrega de combustible a través de una flota de camiones.
En un principio el problema se intento solucionar aplicando técnicas completas de búsqueda, en las cuales destacan Programación Lineal y Entera, su principal problema fue el tiempo de computa que tardaban en entregar una solución[10]. Luego se utilizaron Heurísticas, en las cuales su principal características es que era métodos constructivos de búsqueda de solución, la problemática que surgió a partir de estos, es que generaban buenas rutas en un inicio, pero el final no era el mejor[10]. Por último lo ultimo en usar fueron las Metaheurísticas. Estas se han utilizado en el último tiempo, junto con la inclusión de nuevos cambios como que los parámetros solo son conocidos durante el tiempo de ejecución[10].
En cualquier técnica que se utilice, la representación que mas frecuentemente se utilizo, es la de arboles, en la cual cada nodo (ciudad) se conecta con otra mediante una arista (arco o camino recorrido), para la conexión entre todas las ciudad se utiliza un grafo, en la cual matriz de adyacencia es la representación mas utilizada, dada su facilidad de manejo y simplicidad de implementar. A continuación se presentan algoritmos mas actuales los cuales tratan de encontrar solución a estas problemáticas:

3.1. Algoritmo Genético

Los Algoritmos genéticos, son métodos adaptativos que se basan en el proceso genéntico de los organismos vivos, que se usan para resolver problemas de búsqueda y optimización. Estos tratan de emular la evolución natural, actuando con el principio de sobrevivencia del mas apto, mejorando así el desempeño de individuos que sean mejores que sus antepasados. La aptitud de los individuos se mide a través de una función de evaluación o función de aptitud, para que ver que tan aptos son. Se genera una población inicial, los cuales compiten por la oportunidad de reproducirse, pasando sus aptitudes a la siguiente generación. Para la reproducción el algoritmo utiliza ciertos operadores, los cuales entregan una mejor opción para evolucionar en una buena solución.[4]

- Selección: Se busca dentro de la población o se seleccionan X cantidad de individuos, y se ve cuales son los mas aptos para ir al siguiente paso del algoritmo, el valor que dice si son aptos o no, es la función de aptitud el cual es la métrica para saber si son seleccionados o no.[4]
- Crossover: Aquí mediante una probabilidad, se ve si los individuos previamente seleccionados son cruzados entre si, para dar origen a sus, hijos existen varios operadores de cruce, los cuales pueden ser, cruzamiento en un punto, dos puntos, etc, esto dependerá del tipo de implementación utilizada. Dando así mejores individuos, que heredaran las características de sus padres.[4]
- Mutación: Este operador introduce un cambio aleatorio en los individuos, la gracia de este operador, es que entrega nueva información a la población, por ende permite mejorar y dar saltos de un lugar de búsqueda a otro. Para la utilización de este operador, se usa un probabilidad la cual dirá si el operador se usara en el individuo o no.[4]

3.2. Hill Climbing

El algoritmo Hill Climbing se basa en la búsqueda de profundidad, lo que se hace es que en cada paso se mejora la solución encontrada, de manera de incrementar la posibilidad de encontrar la cima. El algoritmo se basa los siguientes pasos[7]:

- Construir una solución inicial, que cumpla con las restricciones del problema
- Tomar la solución inicial o actual y construir un vecindario a partir de un movimiento.
- Evaluar el vecindario generado a partir de la solución actual y el movimiento, para luego tomar el mejor de estos
- Repetir el proceso hasta no tener mejores soluciones.

Para evaluar la calidad de las soluciones generadas se utiliza una función de evaluación. Existen varias variaciones del algoritmo, pero en general esta es la mas sencilla y es la base para los otros.

3.3. Tabu Search

El Tabu Search es muy parecido al método anterior, con la salvedad de que continue una lista tabú, la cual impide que el movimiento que genero el vecino de mejor calidad sea utilizado en la siguiente iteración, asi impide que se generen ciclos, los cuales dejarían estancado al algoritmo. El largo de la lista permite la intensificación y la diversificación.

3.4. Simulated Annealing

Al igual que el algoritmo Genético, este método mejora las soluciones con el tiempo. Su nombre proviene del proceso por el cual se enfrían los metales, lentamente, para obtener una estructura molecular cohesionada y resistente, dado que mediante imitando este proceso se desea encontrar soluciones.
Se determina una solución inicial y se permite un grado alto de movimiento por el espacio de las soluciones, buscando por todo el espacio de búsqueda. A medida que avanza el proceso, se van permitiendo movimientos cada vez más pequeños, hasta que se converge a una solución final. Esto es, que los movimientos iniciales son grandes y dirigidos a zonas donde la función objetivo es alta, y a medida que se converge, dichos movimientos son más acotados. Para esto se necesita [4]:

- Una descripción de las soluciones posibles.
- Un generador de cambios aleatorios entre las soluciones.
- Una función objetivo para las soluciones.
- Un parámetro de control T y una asignación de enfriamiento que describe como varían los parámetros con el tiempo.

En primer lugar, se debe contar con una solución inicial, sobre la cual se aplicarán ciertos movimientos que generarán un vecindario de soluciones, las cuales serán comparadas con la solución inicial, si alguna presenta de estas tiene alguna mejora reemplazara a la solución inicial dependiendo de una probabilidad de aceptación, el proceso se repite tantas veces sea necesario en un tiempo determinado.

4. Modelo Matemático

La formulación aquí presentada se centra en programación binaria. [2]

4.1. Parámetros

(1) G = (V,A), grafo completo no dirigido.

(2) $V = \{v_0, v_1, v_2, ..., v_n\}$ Conjunto de nodos, v_0 es el depósito

(3) $A = \{(i,j) : i,j \in V,\ i \neq j\}$ Conjunto de Aristas.

(4) d_i = demanda del nodo i.

(5) k Número de vehículos disponibles.

(6) Q capacidad de los vehículos

(7) $C_{i,j}$ Costo en ir desde el nodo i al nodo j

- Los parámetros (1),(2),(3) y (4) dado que estos valores son constantes se tiene un grafo el cual contenta un conjunto de Nodos V los que son las ciudades(clientes) mas el deposito el cual sera el primer nodo. El conjunto de aristas A es el cual cada ciudad esta conectada entre si, sin haber una arista que se conecte con el mismo nodo, o sea no hay un camino que interconecte a un nodo consigo mismo.
- El parámetro (4) indica cual es la demanda de cada nodo (cliente) que debe ser satisfecha.
- El parámetro (5) indica la cantidad de vehículos que se tienen en la flota, para satisfacer a los clientes.
- La capacidad que tiene cada vehículo es la misma para todos, la que se refleja en el parámetro (6).
- El camino que tiene que recorrer un vehículo desde una ciudad a otra tiene un costo de distancia, el que se ve reflejado en el parámetro (7)

4.2. Variables

Se usara una variable binaria para las decisiones

$$X_{i,j} = \begin{cases} 1, & \text{si la solución utiliza el arco}(i,j) \\ 0, & \text{si no} \end{cases} \tag{1}$$

4.3. Restricciones

$$\sum_{i \in V} X_{i,j} = 1, \forall j \in V_0 \tag{2}$$

$$\sum_{j \in V} X_{i,j} = 1, \forall i \in V_0 \tag{3}$$

$$\sum_{i \in V} X_{i,0} = k \tag{4}$$

$$\sum_{i \in V} X_{0,j} = k \tag{5}$$

$$\sum_{i \notin S} \sum_{j \in S} X_{x,i} \geq r(S), \forall S \subset V/\{0\}, S \neq \Phi \tag{6}$$

$$X_{i,j} \in 0, 1, \forall i, j \in V \tag{7}$$

1. Las restricciones (1) y (2) son para que no se repitan los arcos en las soluciones. Dado que solo se debe visitar una sola vez cada ciudad, por ende no pueden haber soluciones con un camino repetido.

2. Las restricciones (3) y (4) son para que entren y salgan la misma cantidad de vehículos al deposito. El nodo 0, se toma como el deposito por esta razón cualquier camino hacia el nodo i debe terminar en 0, y cualquier camino que termine en el nodo j debe llegar a 0

3. La restricción (5) no permite la existencia de substours.

4. r(S) es el número mínimo de vehículos necesarios para satisfacer la demanda de S. en Donde S es un subconjunto de nodos V menos el nodo deposito dado que este no tiene demanda. La cantidad S es distinta a Φ, que es la demanda total del circuito[8].

4.4. Función objetivo

$$Min \sum_{i \in V} \sum_{j \in V} C_{i,j} * X_{i,j}$$

- Lo que se quiere es minimizar los costos asociados a los caminos hechos. Por esta razón a cada variable X, se le multiplica su costo, y la sumatoria de todo da la función objetivo

5. Representación

Dado el modelo propuesto anteriormente, la representación mas intuitiva que se encontró para las conexiones entre las ciudades y el deposito es una matriz de *NxN*, en donde N es la cantidad de ciudades en cada instancia. En *C++* esto se logra mediante un arreglo bidimensional. En cada celda *i,j* de la matriz se almacenara la distancia de la ciudad *i* a la ciudad *j*. La celda *(0,0)* es la que contiene el deposito, si bien es cierto no necesariamente el deposito esta en el principio de las ciudades, para simplificar se usa de esta manera, dado que las distancias se calculan a partir de la posición de estos. Otro dato a considerar, es que esta es una matriz simétrica en donde su diagonal son *0*, dado que el ir desde una ciudad a la misma no tiene costo. A continuación se muestre un ejemplo:

	Deposito	Ciudad 1	Ciudad 2	Ciudad 3
Deposito	0	2	5	1
Ciudad 1	2	0	4	6
Ciudad 2	5	4	0	3
Ciudad 3	1	6	3	0

Ejemplo Matriz de adyacencia.

Como se ve, el ir del deposito a la ciudad 1 tiene un costo de 2 (matriz[0][2] = 2), y de la ciudad 1 a la ciudad 3 tiene un costo de 6 (matriz[1][3] = 6).
Una vez que se tiene la matriz de adyacencia es necesario ir almacenando las rutas que sigue cada vehículo, es por esta razón que la estructura que se utilizo es el vector, dada su rapidez a la hora de hacer operación sobre este.

Deposito	Ciudad 4	Ciudad 2	Ciudad 3	Deposito

Ejemplo Vector de rutas.

Para almacenar los datos correspondientes, como demanda de la ciudad, distancia, etc. se almaceno dentro de nodo del vector un objeto con los datos de cada ciudad, de esta manera se sabe el orden con cual se visitaron las ciudades.

Ciudad
Datos
+Set() +Get()

Ejemplo Clase Ciudad.

Dado que los parámetros son conocidos antes del tiempo de ejecución se tiene que para estos solo se usaron variables globales. Las cuales son.

- *DIM*: Dimensión de la matriz.
- *k*: Número de vehículos disponibles.
- *Q*: Capacidad de los vehículos.

Ejemplo constantes.

6. Descripción del algoritmo

El algoritmo que se uso para resolver este problema fue *Minimal Forward Checking (MFC)*. La implementación de MFC fue realizada en *C++*.

Algorithm 1 Minimal Forward Checking

```
1: DIM ← Inicializar
2: k ← Inicializar
3: Q ← Inicializar
4: matriz ← Inicializar
5: LlenarMatriz(matriz)
6: «vector»Ruta_k ← Inicializar
7: «vector»Visitadas ← Inicializar
8: Ruta←Deposito
9: repeat
10:   for i < Q do
11:     Ruta_i←Deposito
11:     function MFC(matriz,Ruta_i,Visitadas)
12:     repeat
13:       Encontrada ←BuscarCiudadCercana(matriz,última ciudad en la ruta)
14:       if ((Vehiculo_i-DemandaEncontrada>0)&&(Encontrada no este en visitadas)) then
15:         Ruta_i ← Encontrada
16:         Visitadas←Encontrada
17:         CapacidadVehiculo_i ←(CapacidadVehiculo_i-DemandaEncontrada)
18:       end if
19:     until Vehiculo en la Ruta_i tenga capacidad
20:     return  Ruta_i,Visitadas
20:     end function
21:     i++
22:   end for
23: until CriterioParada =0
```

A continuación se detalle el funcionamiento de cada parte, para lograr entender de mejor manera su funcionamiento.

6.1. Inicialización de la Variables

Al inicio del algoritmo se deben ejecutar todas aquellos parámetros que se deben saber para el correcto funcionamiento, dado que esto se sabe con anterioridad al tiempo de generación de soluciones. Por esta razón es que la Dimensión de la matriz, el numero de vehículos y la capacidad de estos, es lo primero en inicializar. El problema que se ve tiene como requerimiento que estos parámetros sean establecidos antes, no durante el transcurso de búsqueda de soluciones. Luego se inicializa la matriz, la cual tendrá como dimensiones el numero de ciudades más el deposito que ocupara el lugar (0,0). La matriz se llena a continuación con las distancias entre cada ciudad. Como se dijo anteriormente, se inicializan el vector de las ciudades visitadas, mas 1 vector por cada vehículo, para luego en el vector de rutas, ingresar el deposito en cada uno, dado que como condición se debe partir de este. No se ingresa en el vector de visitadas el deposito puesto que es el único punto que se debe visitar 2 veces, tanto como al inicio como al fin de cada ruta.

Algorithm 2 Minimal Forward Checking-Inicialización

1: $DIM \leftarrow Inicializar$
2: $k \leftarrow Inicializar$
3: $Q \leftarrow Inicializar$
4: matriz $\leftarrow Inicializar$
5: LlenarMatriz(matriz)
6: «vector»$Ruta_k \leftarrow Inicializar$
7: «vector»Visitadas $\leftarrow Inicializar$
8: Ruta←Deposito =0

6.2. Filtrado

Algorithm 3 Minimal Forward Checking - Filtrado

1: **repeat**
2: **for** $i < Q$ **do**
3: Ruta$_i$←Deposito
3: **function** MFC(matriz,Ruta$_i$,Visitadas)
4: **repeat**
5: Encontrada←BuscarCiudadCercana(matriz,última ciudad en la ruta)
6: **if** ((Vehiculo$_i$-DemandaEncontrada>0)&&(Encontrada no este en visitadas)) **then**
7: Ruta$_i$ ← Encontrada
8: Visitadas←Encontrada
9: CapacidadVehiculo$_i$ ←(CapacidadVehiculo$_i$-DemandaEncontrada)
10: **end if**
11: **until** Vehiculo en la Ruta$_i$ tenga capacidad
12: **return** Ruta$_i$,Visitadas
12: **end function**
13: i++
14: **end for**
15: **until** CriterioParada =0

Como es un algoritmo de búsqueda completa una de los principales problemas es ver el resultado en un tiempo acotado, suponiendo instancias demasiado grande, en el cual no hay un tiempo polinomial asociado al computo se estable un criterio de parada, ya sea un tiempo especificado, cantidad de soluciones encontradas, etc.
Luego para cada vehículo Q se hace la búsqueda de soluciones, para continuar e ingresar en

su ruta el deposito. La función MFC toma como parámetros la matriz de adyacencia de las ciudades, la Ruta para el vehículo i, y el listado de ciudades visitadas. Dado que en la propuesta del problema la gran restricción es que se debe satisfacer la demanda, y no hay problema entre las ciudades (todas interconectadas) salvo que no se pueden visitar mas de una vez, lo que se busca es que el vehículo entrege la mayor cantidad de productos (satisfaga la mayor cantidad de clientes posible), por este motivo se tiene que el procedimiento se repita hasta que no el camión se quede sin capacidad.
Luego dentro de la función como se tiene que el primero en la ruta es el deposito se busca la ciudad mas cercana y se almacena en una variable, esta variable contiene toda la información de la ciudad encontrada. No se hace mayor incapie en la función BuscarCiudadCercana dado que el lector puede implementarla a gusto, dado que solo entrega esto, y la manera de implementación no afecta en mayor cantidad el grueso del *MFC*.
La condicion *if* lo que busca es chequear que la ciudad encontrada respete las restricciones, las cuales son que la demanda que tiene dicha ciudad pueda ser satisfecha por la capacidad que le queda al vehículo, y además que la ciudad no este en la lista de visitadas, así de esta manera no se caigan en un loop ya sea porque el vehículo tiene capacidad infinita o siempre se va a la misma ciudad.
Una vez dentro del *if* se procede a hacer las asignaciones, se ingresa dentro de la ruta la ciudad encontrada, y además en el vector de visitadas para así asegurar el correcto funcionamiento. Como ultima asignación a la capacidad del vehículo se le resta la demanda de la ciudad encontrada.
La función finalmente entrega ruta obtenida y las ciudades visitadas. Se entrega las visitadas, dado que este es un valor que se tiene que guardar en el tiempo para todos los vehículos. Luego se incrementa i que es lo que pasa de un vehículo a otro.
Para no incurrir en errores, se debe tener en cuenta que cada asignación de ciudad que se hace o guarde es un objeto, esto se aplica para todos los vectores y comparación entre ciudad cercana viable y ciudades en ruta o en el vector de visitadas.

7. Experimentos

Para los experimentación se utilizo un Computador AMD con 4 Core de 3.6 GHz, 8 GB de RAM y Ubuntu Linux 13.10 de 64bit. line

8. Resultados

Qué fue lo que se logró con la experimentación, incluir tablas y gráficos (lo más explicativo posible). Los resultados deben ser comentados en esta sección.

9. Conclusiones

- El problema de *Vehicle Routing Problem*, es una problemática actual de la industria, por lo que encontrar soluciones acorde a las necesidades de los clientes y proveedores, no es una tarea fácil, inclusive dada la complejidad que este tipo de problemas puede tener,
- Dada su similitud con el *TSP*, el modelo matemático que se presenta es muy parecido, con lo que no hay problema entre la relación establecida con anterioridad, es por esta razón que el modelo descrito es acorde al problema y relaciona ambos problemas.
- Dada la complejidad del problema, se puede encontrar amplia literatura acerca de este, sobre todo algoritmos relativamente nuevos, dado que es un problema muy complejo de resolver, no se utilizan técnicas completas en la actualidad, dado que se necesitan buenas soluciones en tiempos acotados.

10. Bibliografía

Referencias

[1] Nabila Azi, Michel Gendreau, and Jean-Yves Potvin. An exact algorithm for a single-vehicle routing problem with time windows and multiple routes. *European Journal of Operational Research*, 178(3):755–766, 2007.

[2] Irma Delia Garcia Calvillo. El problema de ruteo de vehiculos. `http://lya.fciencias.unam.mx/computocientifico/archivos/RuteoVehiculos.pdf`, fecha de consulta 20/06/2014.

[3] Bala Chandran and S. Raghavan. Modeling and solving the capacitated vehicle routing problem on trees. *Springer*, 43:239–261, 2008.

[4] INC. ILOG. Optimization technology white paper. a comparative study of optimization techniques, 1997.

[5] C. Jacobs-Blecha and M. Goetschalckx. The vehicle routing problem with backhauls: properties and solution algorithms. *Georgia Institute of Technology*, 1992.

[6] Massimo Paolucci. Vehicle routing problems. `http://www.discovery.dist.unige.it/didattica/LS/VRP.pdf`, fecha de consulta 01/07/2014.

[7] porapi 3705912. Local search algorithms and optimization problems. hill climbing. `http://es.scribd.com/doc/7290255/Hill-Climbing-Methods`, fecha de consulta 20/06/2014.

[8] Paolo Toth and Daniele Vigo. Models, relaxations and exact approaches for the capacitated vehicle routing problem. *Discrete Applied Mathematics*, 123(1 - 3):487 – 512, 2002.

[9] Tali Eilam Tzoreff, Daniel Granot, Frieda Granot, and Greys Sosic. The vehicle routing problem with pickups and deliveries on some special graphs. *Discrete Applied Mathematics*, 116(3):193 – 229, 2002.

[10] Linda Rocha Medina y Elsa Gonzalez La Rotta y Javier Orjuela Castro. Una revision al estado del arte del problema de ruteo de vehiculos: Evolucion historica y metodos de solucion. *Ingenieria*, 16(2), 2011.