VNS APLICADO A TSP

Héctor E. Núñez*, Ricardo D. Quiroga * y Nelson Efrain A. Cruz **

*Universidad Nacional de Salta
Salta, Argentina
e-mail: gimlihen@hotmail.com

*Universidad Nacional de Salta
Salta, Argentina
e-mail: l2radamanthys@gmail.com

**Universidad Nacional de Salta
Salta, Argentina
e-mail: neac03@gmail.com

1. ABSTRACT

El problema de TSP es uno de los mas complejos, mas utilizados y mas estudiados de los problemas de optimización combinatorial. Es una pequea e irónica paradoja en la que existe un algoritmo sencillo para hallar la solución óptima pero que es, en la practica imposible de aplicar por su altísimo costo computacional. En este informe atacamos al problema de TSP aplicando la metaheuristica de VNS basico con unas pequeas modificaciones que intentan y logran (!!!!!) mejorar los resultados obtenidos mediante una mejor elección de soluciones iniciales. Es importante buscar alternativas de resolución pues al ser un problema NP completo ...

2. INTRODUCCION

Una gran cantidad de problemas importantes de optimizacin no pueden ser resueltos usando mtodos exactos, es decir, no es posible encontrar su solucin ptima con esfuerzos computacionales aceptables aunque se pueda contar con computadores de alta velocidad operando en paralelo. Un gran problema de la optimizacin es el fenmeno llamado explosin combinatorial, que significa, que cuando crece el nmero de variables de decisin del problema, el nmero de decisiones factibles y el esfuerzo computacional crecen en forma exponencial.

El problema del viajante de comercio, mas ampliamente conocido por su nombre en ingles Travelling Salesman Problem (al cual llamaremos TSP de ahora en mas) es de un planteamiento bastante sencillo: "Dado un conjunto de n ciudades que poseen caminos que permiten ir de una ciudad a otra sin tener que pasar por ninguna intermedia, se desea conocer cual es la ruta mas corta para recorrer todas las ciudades si es que se conocen las distancias de todos los caminos y además cada ciudad se debe visitar solo una vez".

El problema planteado por TSP puede ser en esencia hallado en muchos ámbitos que van por ejemplo desde implementaciones en logística de transporte hasta encontrar la forma de realizar la menor cantidad de desplazamientos para realizar perforaciones en una plancha, de hay la importancia de hallar métodos mas rápidos (o eficaces) para resolver estos problemas pues como se ve no es un problema que no tenga aplicacion real. En este informe aplicamos La metaheuristica VNS como enfoque basico para intentar hallar soluciones, aunque despues de implementar VNS basico, nos dimos cuenta que no era suficiente, pues si bien se hallaban soluciones relativamente buenas, llegar a ellas requeria de bastantes iteraciones y por lo tanto de bastante tiempo, fue asi que introducimos una memoria de largo plazo, con el fin de lograr una mejor forma de explorar el espacio de las soluciones. (agregar!!!!)

En la sección siguiente proveemos información un poco mas detallada sobre el problema en cuestión y alguno s de los enfoques ya utilizados para atacar este problema, luego en la sección 3 pasamos a describir como encaramos fianlemente el problema y algunos comentarios sobre la implementacion del mismo, a continuacion de ello brindamos detalles de las pruebas realizadas con el algoritmo sobre las instancias de problemas publicadas en la conocida libreria de TSPLib, finalmente se podran hallar las conclusiones que obtuvimos luego de realizar la experiencia.

3. EL PROBLEMA

TSP puede ser representado gráficamente mediante un grafo completo, en el que los nodo representan los puntos a ser visitados y las aristas (al asignarles pesos) representan el coste de ir de un punto a otro, entonces el camino elegido para realizar el recorrido no seria mas que un ciclo hamiltoniano aunque (en la teoría a el camino se lo suele llamar tour). Visto de este modo se puede formular un modelo matematico de programación lineal entera tal como lo hicieron George Dantzig, Ray Fulkerson y SelmerJohnson [**?**] :

$$min \sum_{i=1}^{n} \sum_{j=1}^{n} c_{(i,j)} x_{(i,j)} \tag{1}$$

s.a:

$$\sum_{j=1}^{n} x_{(i,j)} = 1 \,, \forall i \in n \tag{2}$$

$$\sum_{i=1}^{n} x_{(i,j)} = 1 \,, \forall j \in n \tag{3}$$

$$\sum_{i \in S} \sum_{j \in S} x_{(i,j)} \leq |S| - 1 \,, \forall \, S \subset N, S \neq \emptyset \tag{4}$$

$$x_{(i,j)} \in \{0, 1\} \,, i, j \in N \tag{5}$$

Donde $x_{(i,j)} = 1$ si se usa la arista (i, j) como parte del camino elegido y $x_{(i,j)} = 0$ en caso contrario. La función a minimizar (1) representa el costo total de el camino elegido y los $c_{(i,j)}$ representan el costo de ir desde el nodo i al nodo j, las funciones (2) y (3) son para que en el camino elegido cada arista se use una sola vez y finalmente la ecuación (4) esta para evitar que se formen sub-caminos, donde $|S|$ es la cantidad de nodos recorridos en el sub-camino S.

3.1. Encontrando soluciones a TSP

Como ya dijimos, muchos enfoques distintos se han realizado para resolver TSP entre ellas podemos encontar los siguientes :

1. Métodos de construcción de caminos [**?**][**?**][**?**] : Como por ejemplo la heuristica del vecino mas cercano o la heuristica voraz (Greedy) .

2. Métodos de mejoramiento : se trata de métodos que partiendo de una solución inicial realizan cambios en ella hasta obtener una mejor solución, podemos nombrar a métodos de búsqueda local descendente como 2-opt [**?**], 3-opt [**?**], una variacion de la anterior K-opt (heurísticas basada en Lin-Kernighan) [**?**][**?**], simulated annealing [**?**][**?**], algoritmos genéticos [**?**], Tabu search [**?**][**?**].

3. algoritmos Branch & Bound

4. Optimizacion mediante Colonia de Hormigas

Una forma común de medir la calidad (cuanto se acerca a el optimo) de una heurística, es compararla con la cota inferior de Held-Karp (HK), esta cota se obtiene como la solución relajada de la formulación de programación lineal de TSP. Obtener la cota de ese modo puede resultar imposible para instancias de TSP muy grandes siendo solo valido para instancias chicas.

4. SOLUCION PROPUESTA

La metaheuristica de Variable neighborhood search o Busqueda Local por Vecindades (de ahora en mas VNS) que traducido seria algo así como búsqueda de entorno variable en su versión básica dice así:

Pseudocodigo de VNS basico

```
1  Inicializar
2          seleccionar estructura de entornos N_k con k = 1...k_max
3          generar una solucion inicial x
4          elegir un criterio de parada
5  Repetir hasta que se alcance criterio de parada
6          k = 1
7          repetir hasta que k = k_max
8                  generar una solucion inicial x' tal que x' ∈ N_k(x)
9                  aplicar busqueda local sobre x' llamar a esta nueva solucion x''
10                 si x'' < x entonces x = x'' y k = 1
11                 sino k = k + 1
```

La Metaheurística VNS está basada en un principio simple cambiar la estructura de entornos (vecindad) cuando la búsqueda local se estanca en un óptimo local para poder escapar de este, pues una de los tres echos en los que se basa VNS es que el óptimo local en una estructura de entornos no lo es necesariamente en otra. (REFERENCIAR)Pero para obtener buenos resultados con VNS es tambien muy importante como se definen las estructuras de vecindarios, para poder definir los vecindarios usualmente se definen medidas de distancia para poder establecer una lejanía o cercanía de las soluciones.

4.1. Distancias, vecindarios y búsqueda local

Teniendo en cuenta que en TSP todas las soluciones posibles no son mas que permutaciones de una cantidad n de nodos. Definimos distancia de las soluciones x_1 a x_2 como:

$$\rho(x_1, x_2) = \text{número de nodos en diferentes posiciones}$$

Por ejemplo dadas dos soluciones factibles distintas x_1 y x_2 que están descriptas por la siguiente sucesión de $n = 6$ nodos:

$$x_1 = [a, b, c, d, e, f]$$
$$x_2 = [c, a, b, d, e, f]$$
$$\text{es } \rho(x_1, x_2) = 3$$

Ya que tienen 3 elementos en posiciones distintas, es de notar que la distancia mínima que puede haber entre dos soluciones es 2, ya que si un elemento de la solución 1 esta en una posición distinta que en la solución 2 hay forzosamente otro elemento que esta en una posición distinta en

solución 1. La distancia máxima es igual a n, que seria el caso en que todos los nodos se hallen en posiciones distintas en ambas soluciones. Una ves definida la distancia entre soluciones podemos definir nuestra estructura de vecindarios como sigue:

$$x' \in N_k(x) \Leftrightarrow \rho(x', x) = k$$

Aun falta por esclarecer el modo en que vamos a recorrer los vecindarios y cuantos vecindarios vamos a usar, ya que podríamos usar los $n-1$ vecindarios posibles pero esta acarrearía una mayor cantidad de iteraciones. Nosotros optamos por usar una cantidad fija de entornos, que se puede pasar como parámetro, este parámetro también define de forma implicíta que vecindarios se visitan pues en realidad para k generamos un elemento del vecindario $N_{k'}(x)$ donde $k' = \frac{n}{k}$ o sea que para $k = 1$ generamos un elemento del vecindario $N_n(x)$ siendo n el tamao del problema, esto se podría describir como empezar a buscar en el vecindario mas lejano de x para luego empezar a buscar en las cercanías de x ya que a medida que crece k, k' decrece mucho mas rápido, llegando al final por problemas de redondeo a buscar en los mismos vecindarios (en la cercanía de x).
Para la heurística de busqueda local utilizamos movimientos 2-opt hasta encontrar el 2-opt-optimo (REFERENCIAR) que si bien no es muy bueno, pues en media llega a valores encima del 5 % de la cota de HK, precisa de poco tiempo para llegar a este óptimo local y es algo que buscábamos pues aplicar un 3-opt, por ejemplo, precisaba de mucho mas tiempo y esto ralentizaba demasiado el algoritmo por que se realizaban bastantes búsquedas locales.

4.2. Elección estadística del vecino

Una vez probado VNS básico notamos que si bien el algoritmo producía, relativamente, buenas soluciones este era un poco "tonto", pues en pos de la diversificación se generaban de forma aleatoria los elementos de las vecindades correspondientes y dado ello se conseguían demasiadas comparaciones, un poco acercandose a la fuerza bruta, que eran muy probablemente desechadas pues los óptimos locales conseguidos no eran mejores que el óptimo global conocido (no el óptimo global real). Es decir precisábamos una mejor manera de elegir los vecinos, tratando así de reducir las comparaciones realizadas o desde otro punto de vista que los vecinos elegidos sean "buenos vecinos.ª los que al aplicarse una búsqueda local generasen buenas soluciones.
Una aproximación a ello fue que a medida que se obtenían nuevos óptimos locales se iba tomando cuenta de las aristas usadas en una memoria de largo plazo. Entonces cada vez que se usa una arista esto es registrado en la memoria aumentando en uno el contador correspondiente a esa arista. La idea tras esto es que aquellas aristas que aparezcan de forma seguida en las (buenas) soluciones muy probablemente formen parte de la mejor solución o bien que estas aristas sean buenas candidatas para formar parte de una buena solución. Veamos que dada una arista podemos referirnos a ella como un par de nodos, que serian los nodos que conecta esa arista, así la arista (a, b) conecta los nodos a con b. Entonces si la arista (a, b) tiene un contador con un valor alto quiere decir que en las soluciones (buenas u óptimos locales) que fuimos encontrado la subsucesíon $[a, b]$ o $[b, a]$ apareció muy frecuentemente.
Definimos que si dado que una arista (y, x) tiene el valor de contador mas alto, este tendrá la probabilidad mas alta de **seguir** siendo usada en el vecino que se esta generando y las demás aristas tendran una probabilidad proporcional dependiendo del valor de su contador.

Como armar un vecino de x correspondiente al vecindario k se logra cambiando de posición k nodos en el vector solución x, para poder aplicar la memoria de largo plazo al algoritmo que crea los vecinos de x basta con que cada vez que estemos por elegir un nodo a ser cambiado de posición

revisemos cual es la probabilidad de que este se mantenga en su posición (recordando que una arista se puede ver como un par de nodos) y en base a ello decidamos si mover o no este nodo o viendolo de otro modo si seguir usando o no la correspondiente arista teniendo en cuenta su probabilidad. Hay que tener en cuenta que mover un nodo de posición implica dejar de usar dos aristas por lo que habra que tener en cuenta dos probabilidades.

Algo importante que nos dimos cuenta despues de probar el algoritmo es que usar de este modo la memoria muy probablemente nos atrapaba en un óptimo local, pero se aceleraba la llegada a este óptimo y era en general una solución bastante buena (REVISAR), entonces se opto que inicialmente la memoria funcione de la manera descripta y si después de una cierta cantidad de iteraciones no se hallaba una mejora, se invertiría el uso de la memoria o sea que la arista con el valor de contador mas alto tendría la mayor probabilidad de **dejar** de usarse en el vecino que se esta generando. En el modo invertido de la memoria lo que se logra es una mayor diversidad pues se tienden a usar las aristas menos usadas, mientras que en el modo normal de la memoria se logra una menor diversificacion, de alli que podamos quedar estancados en un optimo local.

5. PRUEBAS

(RE-REVISAR) Los problemas que se utilizaron para probar los algoritmos fueron tomados de la TSPLib y las comparaciones en cuanto al valor del optimo local. Como mas arriba se menciona al algoritmo solo se le definen 4 parametros, el primero es el nombre y ruta del archivo que contiene el problema a resolver de la TCPLib, el segundo parametro es el numero de vecindades que se definiran, 3ro la cantidad de iteraciones que se realizaran por vecindad algo que tanbien se dejo constante en 50 durante todas las pruebas, y la probabilidad de permanencia de un elemento en la matriz de memoria, durante las pruebas nos dimos cuenta que era recomendable que este valor fuera bastante elevado osea superior a 0.9, el valor que se uso en las pruebas se mantubo constante y es de 0.98.

En total son 7 los problemas abordados el mas pequeo (COMO CARAJO ES LA ENIE) que se muestra es de 226 vertices, el resto varia entre 439 y 666 vertices, no se muestran mas problemas ya que o son muy pequeos (menos de 200 vertices) o demaciados grandes y por ello para lograr bajar la cota del 3 % de error se tendria que usar una cantidad mayor de vecindades por lo que el tiempo que tardaria en resolverse serian considerablemente alto.

La Siguiente tabla muestra los resultados finales obtenidos durante varias corridas de cada uno de los 7 problemas que evaluamos y como fueron mejorando (mediante el aumento de numeros de vecindades y cantidad de corridas del algoritmo) las soluciones hasta llegar a bajar la cota del 3 %; Por si se desea ver las salidas completas de cada corrida tanto las mejores soluciones '*.dat' como las salidas '.txt' de cada corrida fueron almacenados en la carpeta /TSP/Salidas que se adjunta con este informe.

Prob.	Nro Vec.	Corridas	Opt TSPLib	Opt Local	Error	% Error	Tiempo
p654	20	10	34643	35221	578	1.66844672805	0:00:00
p654	20	10	34643	35221	578	1.66844672805	0:00:00
pr439	20	10	107217	109529	2312	2.15637445554	0:00:00
rat575	20	10	6773	7226	453	6.68832127565	0:00:00
rat575	25	10	6773	7195	422	6.23062158571	0:00:00
u574	20	10	36905	39337	2432	6.58989296843	0:00:00
u574	30	10	36905	38856	1951	5.28654653841	0:00:00
u574	100	5	36905	37688	783	2.1216637312	0:00:00
gr431	25	10	171414	178637	7223	4.21377483753	0:00:00
gr666	25	10	294358	308649	14291	4.85497251646	0:00:00
d439	25	10	35002	36418	1416	4.04548311525	0:00:00
d439	100	5	35002	35422	420	1.19993143249	0:00:00

Nota: La columna tiempo es un valor promedio en horas que tarda el algoritmo, solo por cuestiones de comparacion y de paso fundamentar nuestra razon de usar un algoritmo 2-Opt modificado en ves de un 3-Opt, por ejemplo para el problema 'p654' el tiempo que tarda un el algoritmo 2-Opt modificado es de 1 minuto 30 segundos, mientras que el algoritmo 3-Opt tarda mas de 18 minutos y aunque la solucion que devuelve el algoritmo 3-Opt es mucho mejor que la proporcionada por el 2-Opt modificado, preferimos usar un metodo determinista que no es tan bueno como un 3-Opt o un K-Opt pero es muchisimo mas rapido.

6. CONCLUSIONES

(REVISAR!!) El algoritmo con el cual se realizaron las pruebas es un hibrido de TSP basico pero cuando queda atrapado en un optimo local (algo a lo que los algoritmos TSP tienden) utiliza una Memoria a largo plazo TS (Busqueda Tabu) para intentar escapar de dichos optimo locales.
durante las pruebas realizadas se dectecto que el numero de vecindades a utilizar depende del tamao del problema por ejemplo para problemas de menos de 100 vertices o ciudades (los datos no aparecen en la tabla) el algoritmo lograba alcanzar el optimo global con solo 10 vecindades y una sola iteracion, en cambio en problemas mas grandes con 30 vecindades y 10 corridas del algoritmo no alcanzaba, a lo sumo se lograba estar a un 6 % del optimo global, aunque usar 100 vecindades en las pruebas fue algo excecivo nos aseguro estar por debajo del 3 % con tan solo correr 3 o 5 veses el algoritmo, y en algunos casos se logro estar a un 1 % del optimo.

REFERENCIAS

[1] G. Dantzig, R. Fulkerson, S. Johnson, Solution of a Large Scale Traveling Salesman Problem, Paper P-510, The RAND Corporation, Santa Monica, California, [12 April] 1954 [published in journal of the Operations Research Society of America 2 (1954) 393[410].

[2] J.L. Bentley. Experiments on traveling salesman heuristics. Proceedings of the First Annual ACM-SIAM Symposium on Discrete Algorithms, 1990, 91 - 99.

[3] N. Christofides. Worst-case analysis of a new heuristic for the traveling salesman problem. Tech. Report 388, Carnegie Mellon University, Pittsburgh, USA, 1976.

[4] D.E. Rosenkrantz, R.E. Stearns, P.M. Lewis. An analysis of several heuristics for the traveling salesman problem. SIAM Journal on Computing, 1977, Vol.6, 563581.

[5] G.A. Croes. A method for solving traveling-salesman problems. Operations Research, 1958, Vol.6, 791 - 812.

[6] S. Lin. Computer solutions of the traveling salesman problem. Bell System Tech. Journal, 1965, Vol.44, 2245 - 2269.

[7] S. Lin, B.W. Kernighan. An effective heuristic algorithm for the traveling-salesman problem. Opera-tions Research, 1973, Vol.21, 498 516

[8] K. Mak, A. Morton. A modified Lin-Kernighan traveling salesman heuristic. ORSA Journal on Com-puting, 1992, Vol.13, 127 - 132.

[9] S. Kirkpatrick, C.D. Gelatt, Jr., M.P. Vecchi. Optimization by simulated annealing. Science, 1983, Vol.220, 671 - 680.

[10] J. Pepper, B. Golden, E. Wasil. Solving the traveling salesman problem with annealing-based heuristics: a computational study. IEEE Transactions on Systems, Man, and Cybernetics, Part A, 2002, Vol. 32, 72 - 77.

[11] C.-N. Fiechter. A parallel tabu search algorithm for large traveling salesman problems. Discrete Applied Mathematics, 1994, Vol.51, 243 267

[12] J. Knox. Tabu search performance on the symmetric traveling salesman problem. Computers & Operations Research, 1994, Vol.21, 867 - 876.

[13] B. Freisleben, P. Merz. A genetic local search algorithm for solving symmetric and a symmetric traveling salesman problems. Proceedings of the 1996 IEEE International Conference on Evolutionary Computation, Nagoya, Japan, 1996, 616 - 621.

[14] Heuristics for the Traveling Salesman Problem, Christian Nilsson Linkping University, pp1-6