Una implementación de la variante TSPPDL del problema del viajante

Estudiantes
David Emmanuel López, Javier Enrique Marsicano
Docente
Liliana Favre
Catedra
Análisis y diseño de algoritmos II
Universidad
Universidad Nacional del Centro de la Provincia de Buenos Aires

Resumen. Se describe en este trabajo una implementación de una variante del problema del viajante con operaciones de *pick-up* y *delivery* realizadas en orden LIFO denominada TSPPDL (*Traveling Salesman Problem with Pick-up and Delivery with LIFO loading*). La implementación está basada en una heurística particular denominada VNS-Tree (*Variable Neighborhood Search- Tree*) que representa a las soluciones factibles mediante árboles y las genera mediante operadores de búsqueda basados en la estructura del árbol. Se desarrolló un software en C++ para experimentar con la heurística VNS-Tree y analizar su efecto sobre las soluciones factibles construidas aplicando los diferentes operadores de búsqueda.

Palabras clave: Problemas NP, Algoritmos heurísticos, Problema del viajante, TSPPDL, VNS-Tree

1 Introducción

Este trabajo fue desarrollado como proyecto de la materia "Análisis y diseño de algoritmos II" dictada en el segundo año de la carrera Ingeniería de Sistemas de la Universidad Nacional del Centro de la Provincia de Buenos Aires.

En la mencionada materia se analizan diversas técnicas algorítmicas vinculadas a teoría de grafos, desarrollo de juegos, geometría computacional y string matching. El énfasis es puesto en el análisis de técnicas algorítmicas heurísticas y aproximadas para la resolución de problemas de dificultad NP [3]. En este marco, desarrollamos una implementación de una variante del problema del viajante con operaciones de *pick-up* y *delivery* realizadas en orden LIFO denominada TSPPDL (*Traveling Salesman Problem with Pick-up and Delivery with LIFO loading*). La solución implementada está basada en la técnica de búsqueda de entorno variable (*Variable Neighborhood Search, VNS*) que es una metaheurística basada en cambiar de estructuras de entornos dentro de la búsqueda [4]. La idea esencial en VNS es realizar búsqueda local usando una variedad de operadores para llegar a un óptimo local y luego, diversificar el proceso de búsqueda usando un operador de perturbación. En [1] se presenta una extensión de VNS, la heurística denominada *VNS-Tree* basada, a la vez, en la heuristica *VNS-List* descrita en [2]. Ambas heurísticas están basadas en VNS y difieren en la manera de representar soluciones factibles (recorridos) ya sea

mediante árboles o listas. VNS-Tree representa a las soluciones factibles mediante árboles y las genera mediante operadores de búsqueda basados en la estructura del árbol.

La técnica *VNS-Tree* en cuestión emplea operadores sobre la estructura de árbol que reubican nodos y subárboles, de distintas maneras y exploran combinaciones posibles para generar soluciones factibles.

1.1. Organización del trabajo

En una primera etapa analizamos las soluciones existentes para este problema, en particular la solución presentada en [1]. A partir de este análisis se obtuvo un diseño de la solución, se identificaron los TDA intervinientes y se analizaron sus posibles representaciones. Posteriormente se desarrolló un software en C++ para experimentar con la heurística VNS-Tree y analizar su efecto sobre las soluciones factibles construidas aplicando los diferentes operadores de búsqueda. Finalmente, se realizó un análisis experimental de la solución lograda, comparando los resultados obtenidos con los presentados en [1] y [2].

Se presenta un análisis detallado de estas etapas en las siguientes secciones. En la sección 2 se describe el problema y las bases de la solución presentada en [1]. La implementación de la heurística VNS-Tree se describe en la sección 3. La sección 4 presenta un análisis experimental de la propuesta comparándola con los resultados existentes. Finalmente, en la sección 5, se presentan las conclusiones de la experiencia.

2 TSPPDL y la representación de recorridos factibles

El problema del viajante con operaciones *pick-up y delivery* (TSPPD - *Traveling Salesman Problem with Pickup and Delivery*) consiste en una restricción al problema del viajante clásico, en la cual se exige que ciertos nodos sean visitados antes que otros en forma de pares (*pick-up, delivery*). Para visitar un nodo *delivery*, se debe haber visitado con anterioridad el *pick-up* asociado.

Una solución al problema consiste en un recorrido que visite todos los nodos del grafo una única vez, característica propia del TSP, respetando las restricciones de orden dadas en forma de pares (*pick-up, delivery*), característica propia de *pick-up y delivery*. Adicionalmente, se agrega al problema la restricción de carga LIFO (*Last In First Out*), es decir, en cualquier punto del recorrido la descarga siguiente estará asociada a la última carga efectuada. De esta manera queda conformado el TSPPDL.

La innovación principal de la técnica *VNS-Tree* consiste en la utilización de árboles para modelar las soluciones. Un nodo del árbol representa un par (*pick-up, delivery*). Un subárbol representa un sub-recorrido que comienza en el nodo de *pick-up* del nodo raíz del subárbol y termina en el nodo *delivery* del mismo, recorriendo los nodos internos en preorder. La representación de la solución con árboles simplifica su tratamiento y permite aplicar nuevos operadores respecto a la heurística *VNS-List* presentada en [2] que utiliza grafos para la representación de soluciones.

A continuación se muestra un ejemplo de representación de un recorrido con grafos (trivial) y con árboles:

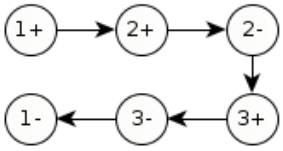

Fig 1. Representación del recorrido mediante un grafo.

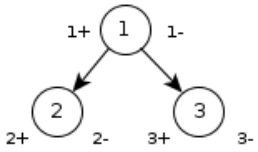

Fig 2. Representación del recorrido mediante un árbol.

Para facilitar el entendimiento, los nodos *pick-up* tienen añadido en su identificación el sufijo "+", mientras que los nodos de *delivery* el sufijo "-". Si coincide el número de identificación de dos nodos, ambos conforman un par (*pick-up, delivery*).

Nótese que al recorrer el árbol en preorder se reconstruye el recorrido del grafo. En [1] se presenta un análisis detallado respecto a la representación del recorrido con árboles.

La técnica *VNS-Tree* en cuestión emplea 8 operadores sobre la estructura de árbol para hallar soluciones: *node swap*, *subtree swap*, *node relocate*, *subtree relocate*, *perturbation*, *ATSP*, *crossover* y *multirelocate* [1]. Estos operadores reubican nodos y subárboles, de distintas maneras y exploran combinaciones posibles para generar soluciones factibles. La tabla 1presenta una descripción informal de los operadores.

Tabla 1. Operadores

node swap	Realiza todos los intercambios de nodos posibles, evalúa cada solución generada, y efectiviza el intercambio que genera la mejor solución entre las generadas.
subtree swap	Análogo a *node swap*. Intercambia subárboles en vez de nodos.
node relocate	Realiza todas las reubicaciones posibles para todos los nodos, y luego efectiviza la que aporta una mejor solución. Una reubicación consiste en desconectar un nodo del árbol e insertarlo en una nueva ubicación.
subtree relocate	Análogo a *node relocate*. Reubica subárboles en vez de nodos.
perturbation	Operación de diversificación que desconecta un subárbol aleatorio y reubica mediante *node relocate* cada nodo del subárbol en el árbol original.
ATSP	Realiza permutas sobre los nodos hijos de cada nodo del árbol, y efectiviza la permuta que genera la mejor solución.
crossover	Operación basada en algoritmos genéticos. Consiste en estructurar un árbol en base a información provista por otro árbol.

multirelocate	Optimización de *node relocate* en donde se efectúan varias reubicaciones sucesivas utilizando el costo computacional de una sola operación *node relocat*e.

Es importante destacar que es la misma estructura de árbol la que garantiza la validez de las soluciones generadas. Siempre y cuando se controle que todos los nodos estén insertos en el árbol al momento de evaluar la solución.

3 Implementación de la heurística VNS-Tree

El trabajo consistió en la implementación de la heurística *VNS-Tree* y una interfaz gráfica que permita la experimentación con diferentes casos de prueba y combinación de operadores. Utilizamos el lenguaje C++ y las librerías *Qt* [5] y *Igraph* [6] para la interfaz gráfica y representación de datos, respectivamente.

Se hizo hincapié en dos cuestiones principales: permitir la experimentación combinando manualmente diferentes operadores, y constatar los resultados experimentales de las pruebas originales de *VNS-Tree* descriptas en [1].

El hecho de utilizar operadores aislados o combinados manualmente, permite verificar experimentalmente cómo repercuten éstas en la constitución de una mejor solución al problema. Por otro lado, el hecho de constatar los resultados obtenidos por nuestra aplicación verifica la calidad del trabajo y confirma las pruebas mostradas en [1]. Se puede descargar el software y probarlo desde un sitio web dispuesto para tal fin [7].

Conforme a estos objetivos, el software utiliza instancias la de librería *TSPLIB* [8] con extensión para soportar *pick-up* y *delivery*, utilizado en el desarrollo de la técnica *VNS-Tree.*

3.1 Estructuras de datos

Para la implementación de las estructuras de datos (grafos y árboles) se utilizó la librería *Igraph* [4], potente para la representación eficiente en memoria primaria y almacenamiento en memoria secundaria de grafos. Fue creada para ser utilizada en la investigación. Permite implementar grafos de diversas características, provee mucha funcionalidad así como algoritmos para operar sobre los mismos, además de otras estructuras básicas como vectores, matrices, pilas y colas de prioridad. La librería brinda un soporte básico de grafos, pero no implementa árboles, esto nos llevó a implementar manualmente nuestro árbol utilizando la funcionalidad que *Igraph* ya proveía.

3.2 Consideraciones de implementación

La descripción de los operadores de VNS-Tree son muy generales, ya que no detallan particularidades que deben tenerse en cuenta a la hora de implementarlos. Muchos de éstos no disponen de pseudocódigo en [1], por tal motivo se hizo necesario interpretarlos y detallarlos para lograr una implementación eficaz y lo más eficiente posible.

Un operador conflictivo en particular fue *subtree swap*, que consiste en intercambiar pares de subárboles. El operador es descrito coloquialmente en [1] y no da un indicio de implementación. En particular, notamos que debe tenerse cuidado de no intercambiar dos pares de subárboles en donde uno esté contenido en el otro.

Ante este problema, se puede optar por generar todos los pares posibles de subárboles, verificar la factibilidad de la operación, y finalmente realizarla. Sin embargo este método resulta poco eficiente. Ideamos entonces, un algoritmo encargado de generar todos los pares de subárboles factibles de intercambiarse, evitando de este modo la acción de verificación de factibilidad. El algoritmo propuesto toma como principio el hecho de que los intercambios factibles son aquellos en donde la intersección entre el conjunto de nodos de cada subárbol implicado es vacía.

El mismo funciona realizando, por cada nodo del árbol, todas las combinaciones de pares de hijos del nodo, donde cada hijo es raíz de un subárbol y, genera conjuntos denominados componentes, en donde cada conjunto contiene todos los nodos de su subárbol asociado. A dicho par de componentes se le aplica la operación producto cartesiano obteniendo un conjunto de pares de nodos que representan intercambios factibles.

A continuación se muestra el pseudocódigo del mismo:

```
pila.insertar(raiz);
mientras(!pila.vacia())
              nodo = pila.extraer();
                ∀ a,b : (a,b ∈ hijos(nodo)) ∧ (a≠b)
                A = componente(a);
                B = componente(b);
                ∀ x,y : (x ∈ A)  ∧ (y ∈ B)
                    intercambiarsubarbol(x,y);
                    comprobarcosto();
                    intercambiarsubarbol(x,y);
                ∀ h : h ∈ hijos(nodo)
                  pila.insertar(h);
```

4 Análisis experimental del desarrollo

Dado un árbol que representa un recorrido, y por lo tanto una solución factible, es imprescindible calcular el costo asociado, para poder comparar diferentes soluciones. En adelante, al mencionar el costo asociado a un subárbol haremos referencia al costo del recorrido que representa ese subárbol.

Una aproximación trivial para lograrlo consiste en recorrer el árbol en preorder, obteniendo de esta forma el recorrido en nodos del grafo. Una vez obtenido el grafo, el cálculo del costo asociado es directo. Denominamos a tal algoritmo *Alg I.*

Sin embargo este algoritmo tiene la característica de requerir una computación lineal O(n), tanto para reconstruir el recorrido a partir del árbol, como para obtener el costo

a partir de un recorrido especificado en el grafo. Si bien la complejidad computacional no es grande, la penalización en tiempo de ejecución es alta ya que el cálculo se realiza cada vez que una operación genera un nuevo recorrido. Téngase en cuenta que las operaciones generan una cantidad de recorridos del orden de n^3, siendo *n* la cantidad de nodos del árbol.

4.1 Alternativa propuesta para el cálculo del recorrido

Con el objetivo de optimizar el cálculo del recorrido, ideamos un nuevo algoritmo de cálculo, basándonos en el hecho de que al efectuar operaciones de inserción o eliminación de nodos sobre el árbol, solamente se altera el costo asociado a los subárboles involucrados, aquellos que contienen a los nodos sobre los que se realizó el cambio.

Sin embargo, la propiedad mencionada en sí misma no implica ninguna mejora, debido a que todos los nodos están incluidos en el árbol completo, por lo tanto habría que aplicar el *Alg I* de cálculo sobre el árbol completo.

La mejora en cuestión se da implementando un método de almacenamiento parcial de costos de los subárboles. Cada subárbol tendrá almacenado su costo asociado. El costo de un subárbol se altera cuando se aplica un operador sobre algún nodo contenido en él. Por consiguiente, una operación alterará un conjunto de subárboles, integrado por todos los subárboles que incluyen al nodo afectado.

Al realizar una operación, el subárbol inmediatamente afectado en su costo, tiene como raíz al padre del nodo objetivo de la operación. Se actualiza su costo, afectando a su vez al subárbol que tiene como raíz al padre del nodo actualizado. El proceso se repite hasta actualizar el costo del árbol completo. Es importante aclarar que todos costos de los subárboles involucrados varían en la misma magnitud.

4.1.1 Ejemplo

Para describir el algoritmo, se muestra un ejemplo concreto en donde se realiza una operación de eliminación en un árbol. Cada nodo se encuentra numerado para identificarlo, y en el mismo se denota el costo asociado al subárbol del cual es raíz.

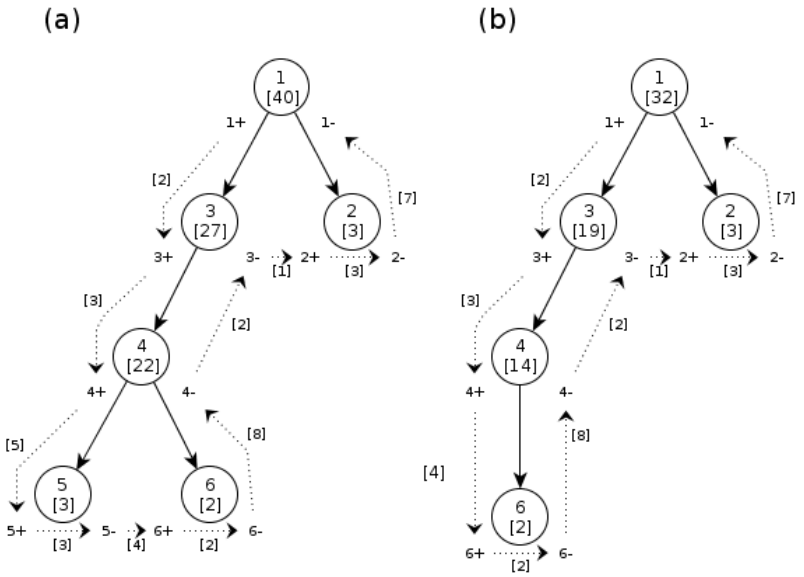

Fig. 3 El árbol *(b)* es producto de eliminar el nodo 5 del árbol *(a)*. Los costos asociados se muestran entre corchetes.

La eliminación del *nodo 5* del árbol, produce una variación del costo asociado al subárbol con raíz en el *nodo 4 (*padre del *nodo 5)*. La variación se calcula como sigue:

$$V_a = c_a(4+,5-) + c_s(5) + c_a(5-,6+) \qquad V_b = c_a(4+,6+)$$
$$V_a = 5+3+4 \qquad V_b = 4$$
$$V_a = 12$$

$$\Delta = V_b - V_a$$
$$\Delta = -8$$

Donde $c_a(i,j)$ es el costo asociado a la arista que conecta al nodo i con el nodo j del grafo y $c_s(\mathrm{x})$ es el costo asociado al subárbol con raíz en el nodo x del árbol.

Nótese que todos los subárboles que contenían al nodo 5 (nodos 4,3,1) variaron en la misma magnitud Δ.

Pseudocódigo del algoritmo

El algoritmo propuesto es simple, luego de realizar cualquier operación básica sobre el árbol se deben realizar las siguientes tareas:

1. calcular variación de la operación.
2. actualizar el costo del subárbol inmediatamente afectado.
3. proceder actualizando costos hasta el nodo raíz.

Denominamos al algoritmo propuesto *Alg II*.

4.1.2 Análisis y experimentación

El algoritmo *Alg II* tiene una complejidad O(n), al igual que el algoritmo *Alg I,* sin embargo su método de aplicación difiere. *Alg I* se ejecuta cada vez que se sea necesario obtener el costo asociado al árbol calculándolo de forma inmediata, mientras que *Alg II* se ejecuta cada vez que se realiza una operación básica sobre el árbol, realizando cálculos parciales.

Esta diferencia sustancial requiere un criterio de comparación especial, no es suficiente la categorización de la complejidad.

Para analizar el impacto de *Alg II* se realizaron varios experimentos sobre diferentes instancias y operaciones, para ambas alternativas de cálculo. El experimento consistió en ejecutar las operaciones aisladas sobre varias instancias utilizando un algoritmo de cálculo particular, para luego comparar los tiempos de ejecución empleados.

Se evidenció que *Alg II* mejora el desempeño de las operaciones, en mayor o menor medida dependiendo de la operación en particular. A continuación se mostrarán los resultados obtenidos sobre las operaciones *node relocate* y *subtree relocate* en las instancias *Brd14051* y *Pr1002* respectivamente, ambas pertenecientes a *TSPLIB*.

Para medir la mejora que implica *Alg II* en relación al *Alg I* calculamos la mejora neta de tiempo de ejecución **T(**A*lg II)* **-T**(*Alg I*) en relación con **T**(*Alg I*), es decir que proporción de **T**(*Alg I)* significa la mejora. Dada la proporción de mejora es trivial la obtención del porcentaje asociado.

$$Mejora = 100 * (T(Alg\ I) - T(Alg\ II)) / T(Alg\ I)$$

La Tabla 2 muestra los valores experimentales obtenidos aplicando los algoritmos de cálculo a la operación *node relocate* para la misma instancia junto con su mejora relativa. La unidad de tiempo utilizada es el milisegundo. La Tabla 3 muestra valores experimentales obtenidos aplicando los algoritmos de cálculo a la operación s*ubtree relocate* para la misma instancia junto con su mejora relativa.

Table 2. Valores experimentales de *Node Relocate*

Instancia	Tamaño	T(Alg I)	T(Alg II)	Mejora

BRD14051	25	13	4	69.0 %
	51	46	23	50.0 %
	75	128	64	53.1 %
	101	320	150	57.3 %
	251	4693	2001	63.5%
	501	40591	14800	63.5 %
	751	142517	49564	65.2 %
	1001	35881	113800	67,9 %

Table 3. Valores experimentales de SubTree Relocate

Instancia	Tamaño	T(Alg I)	T(Alg II)	Mejora
PR1002	25	16	1	93.7 %
	51	77	6	92.2 %
	75	148	17	88.5 %
	101	306	30	90.1 %
	251	2386	253	89.3%
	501	13483	1760	86.9 %
	751	42476	5489	87.0 %
	1001	98061	12828	86.9 %

En la Fig. 4 se puede observar como varía el tiempo de ejecución en función del tamaño de la entrada para la operación *subtree relocate* en la instancia *Pr1002* de TSPLIB para *Alg I* y *Alg II*. Se evidencia que el hecho de variar el algoritmo de cálculo no altera la complejidad de la operación en cuestión, pero mejora el rendimiento.

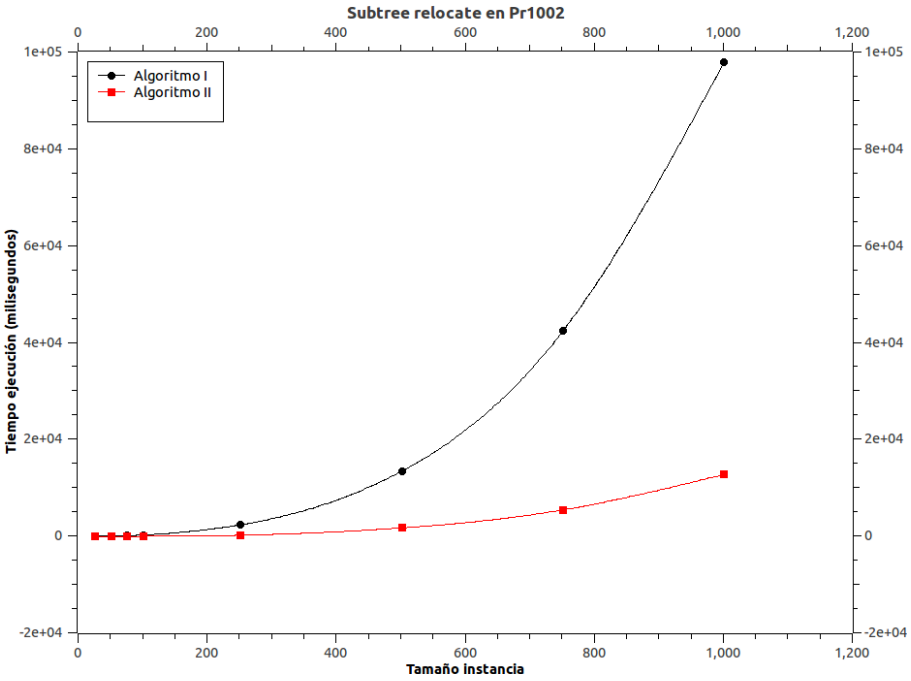

Fig. 4. Subtree Relocate: *Alg I* vs *Alg II*.

4 Conclusiones

Se describió en este trabajo una implementación del TSPPDL, un complejo problema de optimización combinatoria. La implementación se basó en la propuesta presentada en [1], la heurística VNS-Tree. Uno de los objetivos que nos planteamos fue implementarla intentando mejorar los tiempos de cómputo. En tal sentido aportamos un algoritmo nuevo para el cálculo de recorrido.

El software que desarrollamos permite experimentar combinando manualmente diferentes operadores y constatar los resultados experimentales presentados en [1].

Se prevé analizar variantes de TSPPDL que consideren restricciones de capacidad, varios vehículos y ventanas de tiempo. El software fue desarrollado de forma tal que puede ser fácilmente adaptado a estas variantes.

References

1. Li, Y., Lim, A., Oon, W.-C., Qin, H., Tu, D. The tree representation for the pickup and delivery traveling salesman problem with LIFO loading. European Journal of Operational Research 212 (2011) 482-496.
2. Carrabs, F., Cordeau, J.-F., Laporte, G. Variable neighborhood search for the pickup and delivery traveling salesman problem with LIFO loading. INFORMS Journal on Computing 19 (2007), 618–632.
3. Cormen, T., Leiserson, C., Rivest, R. Stein, C. Introduction to Algorithms, Third edition. The MIT Press. (2009)
4. Hansen, P., Mladenovic, N. Moreno Pérez, J. A. http://sci2s.ugr.es/docencia/algoritmica/Busqueda-Entorno-Variable.pdf (2003)
5. http://qt.nokia.com/products/
6. http://igraph.sourceforge.net/
7. http://tsppdl.wordpress.com/
8. http://comopt.ifi.uni-heidelberg.de/software/TSPLIB95/