Implementación de un algoritmo de programación dinámica para resolver el Problema del Viajero con Ventanas de Tiempo (TSPTW)

Nicolás. Gómez[1]

[1] *Departamento de Ingeniería de Sistemas, Escuela Colombiana de Ingeniería "Julio Garavito", Bogotá, Colombia*

Resumen— Este documento presenta en detalle el Problema del Viajero con Ventanas de Tiempo (TSPTW por sus siglas en inglés), estrategias de solución consultadas en la revisión documental, la descripción detallada del algoritmo implementado y los resultados obtenidos contra casos de prueba bien conocidos en la literatura.

Palabras clave— tsptw, tsp, programación dinámica

Abstract— This document presents in detail the Traveling Salesman Problem with Time Windows (TSPTW), solution strategies consulted in the documentary review, the detailed description of the implemented algorithm and the results obtained against well-known test cases in the literature.

Keywords— tsptw, tsp, dynamic programming

INTRODUCCIÓN

El problema del Viajero con Ventanas de Tiempo (TSPTW por sus siglas en inglés) ha sido ampliamente estudiado debido a que es un problema NP-hard, y por tanto existen diferentes algoritmos para tratar de estimar la mejor solución para un escenario dado. En este documento se define el problema, se muestran algunas de los algoritmos, una implementación exacta con ciertas pruebas para disminuir el tamaño del problema y finalmente resultados contra los casos de prueba bien conocidos propuestos por (Potvin y Bengio, 1996).

ESTADO DEL ARTE

El Problema del Viajero (TSP por sus siglas en inglés) se podría enunciar de la siguiente manera: Considere un grafo $G = (N,A)$ donde N es un conjunto finito de elementos conocidos como nodos o vértices y A es un conjunto de pares (i, j) de vértices que llamaremos aristas o arcos, los cuales conectan a dichos nodos. Cada arco $(i, j) \in N$ puede o no tener un costo de viaje llamado c_{ij}, tiene un tiempo de viaje llamado t_{ij} que a veces puede contener el tiempo de servir al nodo i, denotado como s_i en caso que esté especificado. También definamos un ciclo como una sucesión de vértices $n_1, n_2, \ldots, n_p \in N$ tales que $n_1, n_2, \ldots, n_{p1}$ son distintos, $n_p = n_1$ y $(n_i, n_{i+1}) \in A$. Encuentre el ciclo con la longitud más corta en G. En terminos coloquiales, la respuesta se reduce a: dada una lista de ciudades y las distancias entre cada par de ellas, ¿cuál es la ruta más corta para visitar cada ciudad exactamente una vez y retornar a la ciudad de origen? Si a esta definición le sumáramos que cada nodo $i \in N$ está asociado a una ventana de tiempo $[a_i, b_i]$ en la cual dicha ciudad debe ser visitada, estaríamos hablando entonces del TSPTW.

El TSPTW es clasificado como un problema NP-complejo, y (Savelsbergh, 1985) demostró que incluso encontrar una solución factible del TSPTW es NP-completo. Este problema se puede encontrar en una gran variedad de aplicaciones de la vida real, como enrutamiento, planificación, problemas de fabricación de materias y entregas de las mismas entre otros; por esta razón, se ha estudiado ampliamente.

(Focacci et al., 2002) habla de que los primeros enfoques se deben a (Christofides et al., 1981) y (Baker, 1983) y consideran una variante del problema donde el tiempo total del programa debe minimizarse (MPTW). Ambos artículos presentan algoritmos de branch-and-bound, sin embargo el segundo busca formular una ruta crítica con limitaciones de tiempo. (Dumas et al., 1995) propuso un enfoque de programación dinámica para el TSPTW que explota ampliamente

Datos de contacto: Nicolás. Gómez, , Tel: , nicolas.gomez-s@mail.escuelaing.edu.co

los tests de factibilidad para reducir las posibilidades generadas.

Hoy en día existen numerosos enfoques y heurísticas para aproximar el TSPTW. Entre los consultados para este trabajo destacan:

- (Cheng y Mao, 2007) desarrolla un algoritmo de hormigas modificado, denominado ACS-TSPTW, basado en la técnica conocida originalmente como ACO para resolver el TSPTW.
- (Boland et al., 2017) explora un enfoque en el que se explota el potencial de una formulación de programación lineal entera (IP) expandida en el tiempo sin siquiera crear explícitamente la formulación completa. El enfoque funciona con redes parcialmente expandidas en el tiempo cuidadosamente diseñadas, que se utilizan para producir límites tanto superiores como inferiores, y que se refinan iterativamente hasta que se alcanza la optimización.
- (Tüű-Szabó et al., 2017) presenta un nuevo algoritmo que consisten en la combinación de un algoritmo evolutivo bacteriano con búsquedas locales de 2 y 3 opciones (2-opt y 3-opt).
- (Hungerländer y Truden, 2018) expone dos programas lineales de enteros mixtos (MILP) para el TSPTW especialmente sencillos de comprender y de implementar que permiten explotar computacionalmente la estructura de las ventanas de tiempo y también son aplicables para tiempos de viaje asimétricos, para los cuales las desigualdades triangulares no se cumplen.

SOLUCIÓN IMPLEMENTADA

El método descrito a continuación es una aplicación parcial del algoritmo de programación dinámica expuesto por (Dumas et al., 1995). Fue el método escogido para tener un mejor entendimiento del problema, poder analizar los tests de factibilidad utilizados y además en teoría es un algoritmo capaz de solucionar el problema para una baja cantidad de nodos.

Definición recurrente

(Dumas et al., 1995) define $F(S,i,t)$ como el menor costo de una ruta que comienza en el nodo 1 y pasa por cada nodo de S exactamente una vez y termina en el nodo $i \in S$ y está lista para servir al nodo i en el momento siguiente. La función $F(S,i,t)$ se puede calcular mediante la recurrencia que se enuncia a continuación:

$$F(S,i,t) = \min_{(i,j)\in A} \left\{F(S-\{j\},i,t') + c_{ij}\right\},\ F(\{1\},1,0) = 0$$

donde

$$t' + s_i + t_{ij} \leq t$$
$$a_i \leq t' \leq b_i$$

Por tanto, si se busca la solución para el conjunto de N nodos para ir y volver desde el nodo 1, la solución se encontraría en la siguiente ejecución:

$$\min_{(j,1)\in A} \left\{F(N,j,t) + c_{j1}\right\} \qquad a_j \leq t \leq b_j$$

Definición de la ecuación dinámica

Esta solución tal cual como se plantea no es viable, ya que calcula todos los posibles conjuntos en el grafo, lo cual será alto a medida que $|N|$ crece. La estrategia de la definición del método por programación dinámica será empezar por el estado inicial, generar posibilidades para los estados (S,i,T) con $|S| = 1$, luego a partir de estos se generarán los siguientes con $|S| = 2$ hasta llegar a $|S| = N$, siempre y cuando se eliminen estados no factibles en el proceso. Para ello definimos δ_k como el conjunto de estados que contiene todos los estados factibles (S,i,t) tal que $|S| = k$. El primer estado factible es, entonces, $\delta_1 = \{(\{1\},1,0)\}$. El algoritmo de programación dinámica se enuncia a continuación:

```
1  δ_1 := {({1},1,0)}
2  F({1},1,0) := 0
3
4  for k := (2,3,...,N) do
5      for F(S,i,j) ∈ δ_{k-1} do
6          S' := S{j}, (i,j) ∈ A
7          t' := max(a_j, t + s_i, t_ij)
8          if (S',j,t') es factible do
9              δ_k := δ_k ∪ (S',j,t')
10             F(S',j,t') := F(S,i,t) + c_ij
11
12 return mín_{(j,1)∈A} {F(N,j,t) + c_j1},    a_j ≤ t ≤ b_j
```

Test de factibilidad

En el anterior algoritmo en la línea 8 se menciona de que el estado futuro (S',j,t') se agrega a δ_k si y solo si es factible. Diferentes autores han propuesto tests de factibilidad donde la idea es eliminar de manera temprana tantos estados como sea posible. En el documento de (Dumas et al., 1995) se mencionan tres tests de factibilidad, pero solo se indagará acerca del test #2, ya que es el que más arcos remueve en la práctica.

En resumen, este test indica que no se puede tomar un estado factible hacia j si no se han visitado todos los nodos que necesitan ser visitados para llegar al nodo j. Para ello, se define $BEFORE(j)$ como el conjunto de todos los nodos que se deben visitar antes de visitar el nodo j. Teniendo en cuenta esto, rechace (S,i,t), $a_i \leq t \leq b_i$, si $\exists j \notin S \wedge (i,j) \in A \wedge BEFORE(j) \not\subset S$.

Test de dominancia

Una optimización adicional contempla la generación de estados. Se debe rechazar un estado $(S,i,t') \in \delta_k$ si y solo si $\exists (S,i,t) \in \delta_k$ tal que $F(S,i,t) \leq F(S,i,t') \ \wedge\ t \leq t'$, es decir, que exista un estado que ya sea más óptimo para el mismo conjunto S hasta el nodo i.

Extras

Para optimizar en la práctica el algoritmo, se decide asegurar que en cada iteración δ_k solo contenga los estados $(S,i,t) \in \delta_k$ que minimicen $F(S,i,t)$ y t para cada (S,i). Esta implementación tiene un inconveniente y es que puede existir un estado (S,i,t), no necesariamente el mínimo en ambos criterios, que genere expansiones factibles hacia una solución

óptima.

RESULTADOS

El algoritmo fue implementado en Go 1.16 y ejecutado en un computador con Ubuntu 20.04 a través de la interfaz Windows Subsystem for Linux v2, CPU AMD Ryzen 5 1400 a 3.2GHz y 8GB de RAM. A continuación se encuentra la tabla de resultados.

TABLA 1: RESULTADOS OBTENIDOS CONTRA LOS CASOS DE PRUEBA PROPUESTOS POR (POTVIN Y BENGIO, 1996).

Test	$\|N\|$	Solución	Mejor solución conocida	Tiempo (ms)
rc_201.1.txt	20	567.84	444.54	6
rc_201.2.txt	26	764.30	711.54	6
rc_201.3.txt	32	847.48	790.61	14
rc_201.4.txt	26	*	793.64	1
rc_202.1.txt	33	*	771.78	2283
rc_202.2.txt	14	315.90	304.14	8
rc_202.3.txt	29	878.05	837.72	9
rc_202.4.txt	28	849.02	793.03	677
rc_203.1.txt	19	486.03	453.48	104
rc_203.2.txt	33	828.36	784.16	12420
rc_203.3.txt	37	889.03	817.53	319644
rc_203.4.txt	15	334.40	314.29	105
rc_204.1.txt	46	*	878.64	*
rc_204.2.txt	33	*	662.16	*
rc_204.3.txt	24	*	455.03	*
rc_205.1.txt	14	384.73	343.21	4
rc_205.2.txt	27	771.48	755.93	42
rc_205.3.txt	35	915.87	755.93	11779
rc_205.4.txt	28	786.997	755.93	26
rc_206.1.txt	4	117.85	117.85	0
rc_206.2.txt	37	889.76	828.06	3746
rc_206.3.txt	25	640.69	574.42	751
rc_206.4.txt	38	879.62	831.67	2855
rc_207.1.txt	34	860.30	732.68	33380
rc_207.2.txt	31	732.37	701.25	32989
rc_207.3.txt	33	734.29	682.40	172599
rc_207.4.txt	6	119.64	119.64	0
rc_208.1.txt	38	*	789.25	*
rc_208.2.txt	29	*	533.78	*
rc_208.3.txt	36	*	634.44	*

CONCLUSIONES

La implementación parcial del algoritmo de programación dinámica inicialmente propuesto por (Dumas et al., 1995) y modificado para reducir el costo espacial y temporal encuentra soluciones relativamente cercanas a las óptimas para casi todos los casos en los que los tests de factibilidad pueden aplicar. Para los demás casos se requeriría más recursos computacionales para llegar a una solución óptima.

TRABAJOS FUTUROS

Como se mencionó en la implementación, creemos que la optimización del conjunto δ_k es mejorable sacrificando (probablemente) costo espacial, ya que se podrían manejar dos estados mínimos en lugar de uno por cada criterio utilizado en la minimización para cada pareja $(S,i) \in \delta_k$, pero encontrando resultados aún más óptimos.

REFERENCIAS

[1] Baker, E. K. (1983). "Technical note—an exact algorithm for the time-constrained traveling salesman problem". *Operations Research*, 31(5):938–945.

[2] Boland, N., Hewitt, M., Vu, D. M., y Savelsbergh, M. (2017). "Solving the traveling salesman problem with time windows through dynamically generated time-expanded networks". En: *Integration of AI and OR Techniques in Constraint Programming*, pp. 254–262. Springer International Publishing.

[3] Cheng, C.-B. y Mao, C.-P. (2007). "A modified ant colony system for solving the travelling salesman problem with time windows". *Mathematical and Computer Modelling*, 46(9-10):1225–1235.

[4] Christofides, N., Mingozzi, A., y Toth, P. (1981). "State-space relaxation procedures for the computation of bounds to routing problems". *Networks*, 11(2):145–164.

[5] Dumas, Y., Desrosiers, J., Gelinas, E., y Solomon, M. M. (1995). "An optimal algorithm for the traveling salesman problem with time windows". *Operations Research*, 43(2):367–371.

[6] Focacci, F., Lodi, A., y Milano, M. (2002). "A hybrid exact algorithm for the TSPTW". *INFORMS Journal on Computing*, 14(4):403–417.

[7] Hungerländer, P. y Truden, C. (2018). "Efficient and easy-to-implement mixed-integer linear programs for the traveling salesperson problem with time windows". *Transportation Research Procedia*, 30:157–166.

[8] Potvin, J.-Y. y Bengio, S. (1996). *INFORMS Journal on Computing*, 8(2):165–172.

[9] Savelsbergh, M. W. P. (1985). "Local search in routing problems with time windows". *Annals of Operations Research*, 4(1):285–305.

[10] Tüű-Szabó, B., Földesi, P., y Kóczy, L. T. (2017). "An efficient new memetic method for the traveling salesman problem with time windows". En: *Lecture Notes in Computer Science*, pp. 426–436. Springer International Publishing.