Implementación de VNS con una heurística de inserción generalizada para el problema del viajero con ventanas de tiempo*

Luis Felipe Díaz
Escuela Colombiana de Ingeniería Julio Garavito
Maestría en informática
2020

***Abstract*— Acercamiento a la implementación de un algoritmo basado en heurística constructiva para resolver el problema del viajante con restricciones de tiempo**

palabras clase: TSP, TSPTW, problema del viajante, problema del viajante con ventanas de tiempo, VNS, GENIUS

I. INTRODUCCIÓN

Este artículo describe el proceso investigativo para establecer el estado del arte sobre el problema del viajante con ventanas de tiempo, este problema es conocido como un problema np-completo. Posterior a esto se expone el proceso de implementación de una heurística de inserción generalizada, algoritmo publicado en 1995 por Michael Gendraeu, la solución propuesta por $Gendraeu$ usa una heurística de inserción con el algorimto GENIUS[1], un algoritmo propuesto en 1990 por Gendraeu y su equipo de trabajo para solucionar el problema del viajante sin la restricción de ventanas de tiempo.

II. ESTADO DEL ARTE

A. Historia

Un poco de historia, el problema del viajante tuvo su primera aparición en 1832 en una publicación Alemana titulada "El viajande de comercio: cómo deber ser y qué debe hacer para conseguir comisiones y truinfar en su negocio", posterior a esto en 1931 se presento con el término de "Traveling Salesman Problem" en la comunidad matemática. Desde este momento y hasta la actualidad se ha abordado este problema de forma progresiva y se han propuesto algoritmos que pueden resolver instancias del problema con hasta 85900 nodos[2]

El Concorde, compuesto de 13000 líneas de código en C, contiene las mejores técnicas conocidas en la actualidad para resolver este problema.

B. Dificultad

El problema del viajante tiene una dificultad alta ya que el número posible de soluciones aumenta significativamente según el tamaño de los nodos. si se toma un caso de 33 nodos, la cantidad de casos posibles son 32!, un número de 33 cifras, si bien este es un número finito, los estudios e investigaciones alrededor de este problema tratan de lograr encontrar la solución más optima sin tener que ir sobre todos los posibles casos. Para los problemas como estos que no tienen un algoritmo que genera la solución más optima se les conoce como problemas NP, de tiempo polinómico no determinista.

III. ALGORITMOS ENCONTRADOS

A. Algoritmos Exactos

Algoritmos de raminifación y acotación, $Dantzig$, $Fulkerson$ y $Jonhon$ (1945-1959)[3] propusieron un algoritmo con el fin de romper el conjunto de soluciones y aplicar tecnicas de acotación. Partiendo de soluciones no tan buenas, se hacen ramificaciones o subproblema para lograr alcanzar una cota inferior o minimo local. El proceso continua hasta que se consigue llegar a una solución lo suficientemente buena según los parametros.

B. Algoritmos heurísticas de construcción

Los algoritmos de construcción construyen la solución de forma gradual usando reglas predeterminadas para tomar decisiones en el transcurso del programa, un ejemplo de estos son los algoritmos de construcción heurísticos como el algoritmo heurístico de inserción generalizado propuesto por $Gendreau(1995)$.

C. Algoritmos heurísticos de mejora

Los algoritmos heurísticos de mejora toman una solución que es factible y la tratan de mejorar usando una o varias heurísticas, $Salvesbergh$[4] menciona que incluso construir una solución factible para un problema del viajante con restricciones de ventanas de tiempo puede ser una tarea NP-dificil (Np-hard), es por esto que los algoritmos que se basan en soluciones factibles deben apoyarse en otros algoritmos para la generación de un estado inicial. Un ejemplo de este tipo de algoritmos es el $k - opt$ u optimización k veces, consiste en invertir k arcos del grafo para buscar una solución mejor.

D. GENIUS

El algoritmo de GENIUS es un algoritmo de inserción en caminos $v_0...v_{i+1}$ por medio de precalculos de los vecinos más cercanos. Para esto se define $N_p(i)$, como los p nodos más cercanos al nodo i que hacen parte del camino $v_0...v_{i+1}$,

con estos p nodos se evaluará la posibilidad de incluir el nodo i que se encuentra fuera de la solución parcial[Figura 1]. Es importante precalcular los $N_p(i)$ para todos los nodos i cuando se hace una modificación en la solución.

Para generar $N_p(i)$ se deben ordenar los nodos, para esto se define la funcion r que calcular la distancia proximidad entre dos nodos según sus ventanas de tiempo

$$r_{ij} = min\left\{b_j, b_i + cij\right\} - max\left\{a_j, a_i + cij\right\}.$$

Sin embargo también se propone una función de distancia entre nodos más concreta llamada $PseudoDistancia$ formada por una combinación convexa entre c_{ij} y r_{ij}, esta función es una perturbación en la distancia original teniendo en cuenta la diferencia de las ventanas de tiempo.

$$d_{ij} = \alpha c_{ij} + (1 - \alpha)\, r_{ij} \;\; donde \;\; \alpha \;\; [0, 1]$$

Fig. 1. Generación de caminos usando GENIUS

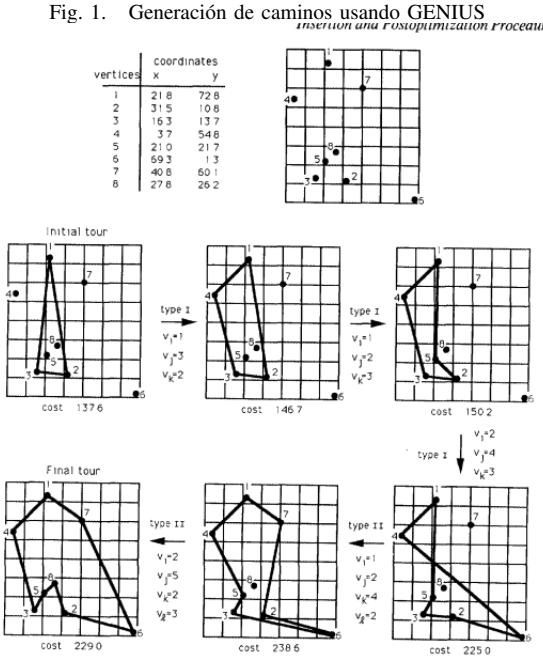


Figure 3. Illustration of the **GENI** algorithm on an 8-vertex problem.

Luego de esto se intentará usar los algoritmos de inserción de Tipo 1 o Tipo 2 [7] para aumentar la candidad de nodos que están en la solución parcial.

IV. Implementación

A. Estructura de datos

Creemos que es importante pensar también en la estructura de datos que se va a utilizar para resolver este problema, en este caso se ha construido una clase especial para representar el camino que actualmente se va construyendo, esta clase implementa una $LinkedList$ y un $HashMap$ que hace referencia a los nodos. Esto nos va a permitir hacer operaciones de inserción, eliminación y búsqueda de nodos de forma constante [Figura 1].

Fig. 2. Tipos de inserción con GENIUS

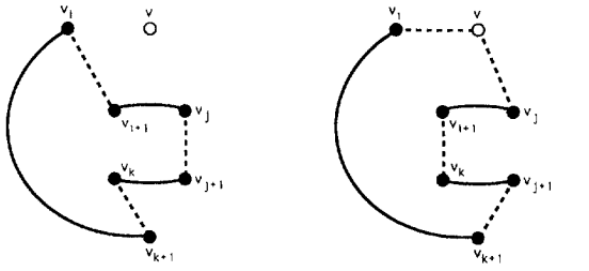


Figure 1. Type I insertion of vertex v between v_i and v_j.

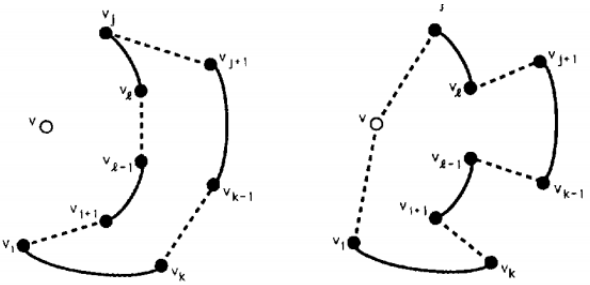


Figure 2. Type II insertion of vertex v between v_i and v_j.

B. HeuristicInsertion

Se decidió implementar el algoritmo VNS junto con la heurística de inserción generalizada.

Tomando como inspiración tres publicaciones, la publicación sobre VNS por $Rodrigo\ Ferrerira$[5], la implementación para busqueda de vecindarios de $Christos\ Papalitsas$ y la publicación sobre inserción generalizada por $Gendraeu$, se propuso un algoritmo que combina tecnicas presentes en los tres papers.

El algorimo se divide en dos sub-algoritmos, la primera es la busqueda de una solución factible por medio de la heurística de inserción generalizada. Esta heurística es una heurística de construcción donde se construye un camino inicial factible y se comienza a expandir con la busqueda de nodos vecinos que mantengan la fiabilidad de la solución, no es permitido en este algorimo partir de soluciones no factibles ya que la probabilidad de encotrar una solución factible disminuye considerablemente (Algoritmo 1).

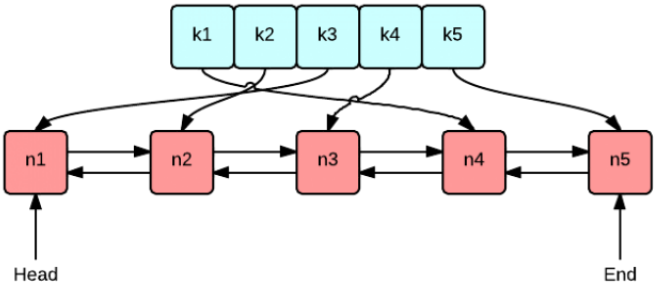
C. HeuristicInsertion - Fase 1

$RemoveUnfeasibleArcs$ removera cualquier arco que no pueda ser contemplado en una posible solución usando la siguiente regla

$$a_i + c_{ij} <= b_j$$

cualquier camino de un nodo i a un nodo j debe poder perminitirnos llegar antes del tiempo de cierre de la ventana de j, en caso que esto no se cumpla marcamos los arcos

Fig. 3. Figura LinkedHashSet



Algorithm 1 construcción de posible solución factible

```
1:  procedure HEURISTIC-INSERTION                          ▷
2:                                                         ▷ Fase 1
3:      RemoveUnfeasibleArcs()
4:      solution ← ConstructFirstPath()
5:      solution ← InsertRemainingNodes()
6:      solution ← InsertUsingGenius()
7:                                                         ▷ Fase 2
8:      solution ← UnstringAndString()
9:      solution ← PostOptimization()
10:
11:     return solution                  ▷ completa o incompleta
```

como **NotFeasible** y evitamos procesamiento adicional.

$ConstructFirstPath$ construye una ruta inicial factible, sin violar ningún tiempo en los nodos, para eso se ordenan los nodos por la magnitud de sus ventanas de tiempo $b_i - a_i$ de forma ascendente. Posterior a esto se arma un camino inicial, no necesariamente con todos los nodos.

$$X = v_0 ... v_{i+1} \ donde \ X \ es \ factible$$

$InsertRemaininNodes$ y $InsertUsingGenius$ tratará de insertar todos los nodos que no lograron ser incluidos en la construcción inicial, para esto se toman los p nodos más cercados $N_p(i)$. Primero hacemos un barrido para ver si es posible insertar algún nodo en una posición válida, si no es posbile esto se intenta insertar los nodos usando las tecnicas de inserción Tipo 1 y Tipo 2.

D. HeuristicInsertion - Fase 2

En esta fase se busca optimizar cualquier posible solución encontrada en la fase 1, para esto se usó el approach propuesto por $Gendraeu$ de hacer backtracking removiendo nodos de la solución actual, moverlos al final de una cola e intentar agregar nodos pendientes por agregar. A este proceso de desconstrucción y construcción lo bautizó $unstring$ y $string$

E. VNS

Para esta parte se ha usado la propuesta de $Rodrigo \ Ferrerira$ sobre la heurística VNS (Variable Neighborhood Search) $Algoritmo2$, se decidió usar esta heurística como plan B en dado caso que el algoritmo inicial no tuviera exito encontrando una solución completa y factible, para esto se coma el resultado de $HeuristicInsertion$ y es pasada como entrada a VNS, cabe resaltar que VNS no espera una solución factible y es por eso que se usa como plan B para intentar obtener una solución desde el mejor resultado que pudo obtener $HeuristicInsertion$.

VNS funciona con niveles de perturbación y búsqueda de soluciones vecinas a partir de las perturbaciones, si se encuentra una mejora en este proceso se sobreescribe la mejor solución por la entrada y se sigue iterando hasta encontrar una solución totalmente factible, es por esto que si encontramos una solución factible en con $HeuristicInsertion$ ya no es necesario llamar VNS.

Algorithm 2 VNS

```
1:  procedure VNS(a, b)                  ▷ The g.c.d. of a and b
2:      X ← HeuristicInsertion()
3:      X ← LocalSearch()
4:      while X ≠ factible and level <= A do
5:          X' ← Perturbacin()
6:          X'' ← LocalSearch()          ▷ try to include more
    nodes
7:          if X' es mejor que X then
8:              X ← X'
9:              level ← 1
10:         else
11:             level ← level + 1
12:         return b                              ▷ The gcd is b
13:
```

V. Resultados

Los siguientes resultados se ejecutaron en una MacOS 1.4 GHz Quad-Core Intel Core i5, 8 GB 2133 MHz

Los resultados nos muestran que el enfoque de esta implementación estuvo más enfocada a buscar soluciones que no violen las restricciones

VI. Conclusiones y trabajo a futuro

Los algoritmos heurísticos tienen un enfoque muy particular ya que dependen de las reglas que se establezcan para considerar que puede o no puede ser mejor. Esto en algunos escenarios puede ser muy útil pero hay casos en los que occurre todo lo contrario. La generación de soiluciones factibles es el punto crítico para algunas heurísticas qde mejora. A futuro es bueno investigar cual es el mejor algoritmo conocido para la generación de soluciones factibles no necesariamente optimas.

Casos	Solución	Tiempo	Violaciones	Makespan
rc_201.1	Si	0.02	0	793.2431
rc_201.2	Si	0.18	0	843.095
rc_201.3	Si	0.14	0	962.070
rc_201.4	Si	0.17	0	1033.0
rc_202.1	Si	0.27	0	1217.4
rc_202.2	Si	0.007	0	427.182
rc_202.3	Si	0.33	0	921.53
rc_202.4	No	3.3	1	829.2
rc_203.1	Si	0.17	0	597.45
rc_203.2	Si	0.11	0	1021.44
rc_203.3	Si	1.53	0	1383.2
rc_203.4	Si	0.008	0	470.5
rc_204.1	No	72	1	1434.1
rc_204.2	Si	0.11	0	957.5
rc_204.3	Si	0.35	0	1028.5
rc_205.1	Si	0.008	0	449.1
rc_205.2	Si	0.24	0	951.49
rc_205.3	Si	0.33	0	112.6
rc_205.4	No	3,43	0	1152.8
rc_206.1	Si	0.000	0	851.782
rc_206.2	No	18	1	1205.62
rc_206.3	Si	0.32	0	877.2
rc_206.4	No	19	1	1020.3
rc_207.1	Si	0.22	0	1364.23
rc_207.2	Si	2.96	0	970.64
rc_207.3	Si	0.13	0	893.94
rc_207.4	Si	0.001	0	160.03
rc_208.1	Si	7.62	0	1347.76
rc_208.2	Si	0.07	0	834.7
rc_207.3	Si	0.20	0	1127.449

REFERENCES

[1] "https://press.princeton.edu/books/hardcover/9780691129938/the-traveling-salesman-problem

[2] "https://www.researchgate.net/publication/225493761$_Implementing_the_Dantzig-Fulkerson-Johnson_algorithm_for_large_traveling_salesman_problems$

[3] "M.W. Salvesbergh, "Local search in routing problems with time windows, "Annals of Operations Research, 4 (1985), pp. 285-305

[4] "https://www.sciencedirect.com/science/article/pii/S1572528610000289

[5] "https://ieeexplore.ieee.org/document/7388106

[6] "https://pubsonline.informs.org/doi/10.1287/opre.46.3.330