Búsqueda Local Básica para resolver el ATSP

Nicolás Robayo Pardo[1]

[1] Universidad de los Andes
Cra 1 Nº 18A - 12, 111711, Bogotá, Colombia
n.robayo10@uniandes.edu.co

Resumen

Esta tarea busca resolver el problema del ATSP (Problema del cartero viajante asimétrico) a través de métodos de búsqueda local. El objetivo del ATSP es recorrer un set de ciudades pasando una sola vez terminando en la ciudad inicial y reduciendo la distancia recorrida. Este problema tiene aplicaciones en amplios sectores como la navegación, el ruteo de vehículos en ciudades y la secuenciación de tareas. El uso de heurísticas se presenta como una buena alternativa de solución debido a que el problema es NP completo.

1 Introducción

El TSP es uno de los problemas mas conocidos en optimización combinatoria. Por su simplicidad y dificultad de solución ha sido estudiado sistemáticamente desde el siglo XIX. El objetivo es encontrar una ruta que pase por todas las ciudades del problema y termine en la ciudad inicial (creando un ciclo) visitando cada ciudad una sola vez y minimizando la distancia recorrida (o el costo). Este problema tiene dos variantes, el STST donde las distancias son simétricas y el ATSP donde la distancia $d(c, c')$ de ir de un punto a otro no es el mismo a la distancia inversa $d(c', c)$. Esta situación es más cercana a aplicaciones de la vida real como el desplazamiento por redes de transporte donde el costo de desplazamiento de una ciudad a otra puede ser distinto a devolverse por motivos de costos de tiquetes, infraestructura, dinámicas de mercado, etc. Otra aplicación es la secuenciación de tareas de maquinas en donde se debe decidir el orden en que un set de tareas se debe hacer y donde la distancia de transición entre tareas no es simétrica [1].

El TSP y sus vertientes ha llamado la atención de investigadores desde hace siglos debido a que es fácil de entender, pero su solución es muy difícil. De hecho, este problema es NP-duro [2] y su espacio de solución crece exponencialmente con el número de ciudades a considerar. Es por esto por lo que el uso de heurísticas es recomendable para obtener soluciones en un corto tiempo. Se podría argumentar que el ATSP es más difícil de resolver que el STSP por medio de optimización o aproximación y la diversidad de instancias es mucho mayor al STSP [3]. Por otro lado, el espacio en memoria necesario para almacenar las distancias es el doble lo cual hace complicado manejar instancias con un elevado número de ciudades.

Una búsqueda local puede ser clasificada como una heurística de mejora que parte de una solución factible y a través de mecanismos de transición en la vecindad pasa a una solución cercana mejor. Además, cuenta con un criterio de parada, sea este número de iteraciones o no tener otra mejor solución en el vecindario. Las búsquedas locales pueden partir de una solución construida de forma aleatoria o a través de una heurística constructiva. Algunas búsquedas locales básicas utilizadas en el TSP están basadas en los mecanismos de transición $k - opt$ y swap los cuales intercambian trozos de rutas. En este caso, se define el vecindario de una solución como todas las soluciones que resultan de hacer k-opt o swap con todas las combinaciones de nodos. Después de revisar el vecindario, la heurística escogería una solución que mejore la solución actual y continuaría explorando el vecindario de la nueva solución hasta llegar a un criterio de parada.

Esta tarea se divide en 4 partes, primero se hará una descripción matemática del ATSP. Posteriormente, se hará una revisión de literatura de las heurísticas de búsqueda local más utilizadas para resolver el

ATSP. Luego se describirá la metodología de solución basada en operadores de búsqueda local y por último se evaluarán frente al set de instancias propuestas por Gerhard Reinelt de la Universität Heidelberg. Finalmente se harán unas conclusiones sobre los dos algoritmos de búsqueda local utilizados.

2 Descripción

2.1 Definición del ATSP

El problema del cartero viajante asimétrico (ATSP) se define en un grafo dirigido $G = (N, A)$, donde $N = \{1, \dots, n\}$ es el set de vértices/ciudades, $A = \{(i,j): i,j \in N\}$ es el set de arcos, y una matriz no simétrica de costos/distancias (c_{ij}) se define en A [4]. Debido a que no se permite mantenerse en una ciudad se define que el costo de ir a la misma ciudad de origen $c_{ii} = \infty$. El ATSP consiste en determinar el tour Hamiltoniano de mínimo costo en G.

2.2 Modelamiento Matemático

Se expone a continuación la formulación clásica con restricciones de integralidad y eliminación de subtoures.

2.2.1 Conjuntos

$$N: Ciudades/Vertices$$

2.2.2 Parámetros

$$c_{ij}: Costo\ asociado\ al\ arco\ que\ conecta\ la\ ciudad\ i \in N\ a\ j \in N$$

2.2.3 Variables de decisión

$$x_{ij} = \begin{cases} 1 & si\ el\ arco\ que\ conecta\ las\ ciudades\ i \in N a\ j \in N\ es\ parte\ de\ la\ solución \\ 0 & dlc \end{cases}$$

2.2.4 Función Objetivo

Se busca minimizar la distancia total recorrida.

$$\min \sum_{i=1}^{n} \sum_{j=1}^{n} c_{ij} x_{ij}$$

2.2.5 Restricciones

(1) Restricción de asignación: Cada ciudad es incidente a un arco saliente.

$$\sum_{j=1}^{n} x_{ij} = 1 \forall i \in N$$

(1) Restricción de asignación: Cada ciudad es incidente a un arco entrante.

$$\sum_{i=1}^{n} x_{ij} = 1 \forall j \in N$$

(2) Restricciones de eliminación de subtures

$$\sum_{(i,j)\in S} x_{ij} \leq |S| - 1 \forall S \subset V$$

S es el conjunto de arcos que producen subtures.

(3) Naturaleza de las variables.

$$x_{ij} \in \{0,1\} \forall i,j \in N | i \neq j$$

3 Revisión de Literatura

El estudio experimental de heurísticas para resolver el ATSP está mucho menos avanzado que para el STSP. Antes del 2001 la mayoría de los trabajos se concentraba en las instancias del TSPLIB y en instancias generadas aleatoriamente. Las heurísticas constructivas más clásicas son el Vecino mas Cercano y el Goloso. En el vecino más cercano se parte de una ciudad inicial aleatoria y se va escogiendo la ciudad más cercana que no haya sido visitada devolviéndose al final a la ciudad inicial. En el goloso se sortean los arcos entre ciudades por costos. Se analizan los arcos "elegibles" que se pueden agregar a la solución sin crear un ciclo no Hamiltoniano o hacer que el grado de un nodo no sea uno en valor absoluto. Una implementación es escoger aleatoriamente de entre los dos arcos con menor distancia hasta que un tour sea construido. Estas heurísticas se pueden utilizar para construir un tour inicial que sirva de insumo en una búsqueda local.

Entre las heurísticas más conocidas de búsqueda local están las que usan el operador 2 opt. El operador 2-OPT realiza los siguientes pasos: primero se eliminan dos nodos no adyacentes de la solución inicial para formar dos toures desconectados. Luego, se añaden arcos que juntan los dos toures de forma inveresa. Este operador fuerza a que uno de los toures cambie de dirección y por consiguiente tiene un alto costo computacional al tener que recalcular los nuevos costos de la ruta cambiada. Ha habido sugerencias que reducen esta heurística a un tiempo de $O(N^2)$ a través de estructuras que precalculan la nueva distancia al realizar los movimientos 2-OPT [1].

Otra heurística de la misma familia es la 3-OPT. El vecindario en esta búsqueda local consiste en todos los tures que pueden ser obtenidos al eliminar tres arcos y permutar las rutas resultantes [3]. Dado que las distancias entre ciudades no son independientes del sentido, se podría solo considerar la permutación donde ninguna de las tres rutas cambie de sentido. Tener que cambiar el sentido de una ruta puede cambiar de forma importante el costo de esta, además del costo computacional de calcular el cambio. Una variación de esta heurística es la de Johnson-McGeochhacer 10N iteraciones donde N es el número de ciudades.

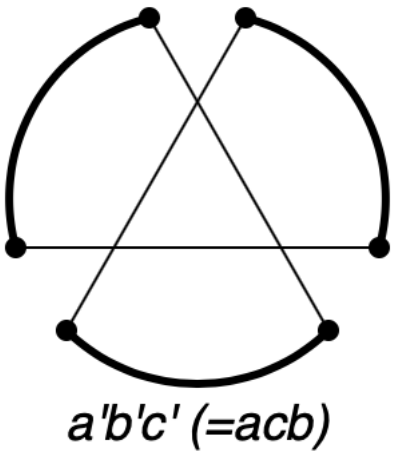


Ilustración 1 Búsqueda 3-OPT. Obtenido de [5]

Otra heurística muy simple es el Swap el cual consiste en intercambiar la posición en el tour de dos ciudades. Este movimiento es el caso especial de dos movimientos 2-opt: el primero incluyendo ambas ciudades y el segundo sin ellas. También puede ser considerado como un tipo especifico de movimiento 4-opt donde se eliminan 4 trozos de ruta y se añaden 4 trozos.

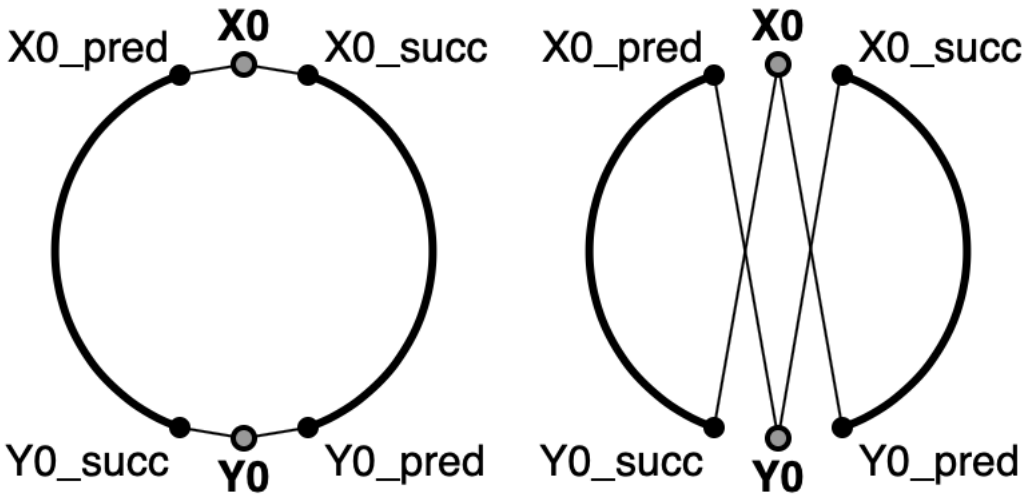


Ilustración 2 Búsqueda local SWAP. Obtenido de [6]

Otra heurística de búsqueda local es Kanellakis-Papadimitriou [7]. Esta consiste en dos procesos de búsqueda alternantes. El primero es una búsqueda restringida secuencial de profundidad variable que trata de encontrar un movimiento tipo k-opt ($k \geq 3$) que tenga alguna mejora en la solución. El segundo proceso es una búsqueda de una mejora tipo 4-opt que se no secuencial "con doble puente". En este movimiento, las rutas resultantes del empezar el movimiento 4-opt (a,b,c,d) quedarían (b,a,d,c) al final del movimiento sin invertir el sentido de la ruta.

Por último, estas búsquedas locales se pueden hacer de forma iterativa usando cualquier procedimiento para obtener una primera mejor solución y luego se repite el siguiente procedimiento un número determinado de veces: Aplicar un movimiento 4-opt aleatorio a la solución actual y luego usar la solución resultante como inicio del algoritmo inicial. Si la solución obtenida al final de la iteración es mejor que la actual, esta se convierte en la mejor solución [8]. Existen algunos procedimientos y reglas que aceleran el proceso de encontrar buenos movimientos como "don't look bits" el cual se basa en obviar movimientos realizados en las iteraciones pasadas.

Un aspecto muy importante de estas búsquedas locales es criterio de parada. Debido a la naturaleza combinatoria del problema, un operador k-opt o swap puede revisar todo el vecindario para encontrar la mejor solución o quedarse con la primera solución que mejore la función objetivo. A estos criterios de parada los llamaremos Best Improvement y First Improvement. La ventaja de usar un criterio como Best Improvement es que todo el vecindario es analizado en búsquedas de mejoras y la mejora por iteración aumenta, pero el desplazamiento por el espacio de solución es mucho más lento, así como el tiempo computacional. La ventaja de First Improvement es su bajo tiempo de computo y el rápido desplazamiento por el espacio de solución. Estos criterios pueden ser combinados en el mismo algoritmo donde primero se busque llegar rápidamente a buenas soluciones a través de First Improvement y luego se calcule el vecindario de las mejores soluciones encontradas a través de Best Improvement.

4 Metodología de solución

Para esta tarea implementaremos en Python dos heurísticas de búsqueda local. La primera usará el mecanismo de transición Swap y la segunda el mecanismo de transición 3-OPT cuya permutación mantiene el orden de la solución. Debido a que, para instancias con un número de ciudades, enumerar las posibles combinaciones de posibles swaps o cambios 3-opt es prohibitiva se utilizará un enfoque aleatorio que busque posibles combinaciones en el vecindario. En cada iteración se creará una lista de combinaciones de nodos para ejecutar el mecanismo de transición en vez de recorrer todo el vecindario. Las ventajas de este enfoque son claramente el tiempo computacional y la principal desventaja es la perdida de buenas combinaciones que lleven a una mejor solución.

Para crear las soluciones iniciales se utilizará el algoritmo del vecino más cercano que fue descrito en la sección pasada. Este, según la literatura, brinda mejores soluciones que el algoritmo goloso y su implementación es muy fácil creando soluciones en tiempos muy cortos. Por otro lado, crearemos el primer algoritmo con un criterio de parada de First Improvement y el segundo contará con ambos criterio dentro de su vecindario restringido aleatorizado.

Un aspecto importante de la implementación son las estructuras de datos usadas. Para poder calcular eficientemente el posible cambio en la función objetivo se crearon dos listas que tuvieran por cada posición la siguiente o anterior ciudad de la posición actual. Entonces si una ruta estaba compuesta como 4-76-3, en la posición 76 del vector de siguientes ciudades se encontraría el 3 y en la misma posición del vector de ciudades anteriores estaría el 4. Esta estructura permite conocer rápidamente el orden de la solución y calcular el cambio en la FO dada una perturbación en el orden de las ciudades.

El primer algoritmo se puede describir de la siguiente forma. Se inicia con una solución factible y se empieza iterar hasta un número máximo de iteraciones. En cada iteración se escogen al azar dos nodos de la solución y se calcula la mejora que se obtendría en la función objetivo si se intercambian. Si esta es negativa se hace el cambio y se actualiza la ruta (y las estructuras de datos). Esto se realiza un número determinado de veces y se devuelve la mejor solución. Nótese que cada vez que se determina que el par de nodos se puede swapear obteniendo un beneficio, esto se realiza y la solución cambia. Por esto se puede decir que el criterio es de First Improvement. De forma predeterminada, el número máximo de iteraciones será $\left\lfloor \sqrt{\binom{N}{2}} * 120 \right\rfloor$. Este número permite crear un número de combinaciones dependiendo del tamaño de la instancia que permita abarcar un tamaño razonable del vecindario sin volverse exponencial con el número de nodos.

El segundo algoritmo también empieza con una solución factible y empieza a iterar un número determinado de veces. Cada iteración tiene una ruta asociada y un vecindario común; en cada iteración se crea una lista de posibles combinaciones de tres nodos para ejecutar el mecanismo 3-opt. Para cada combinación se calcula el cambio de la función objetivo dado una operación 3 opt en los nodos de la combinación. Si el modo first_improvement está activado se actualizará la ruta con la primera combinación que disminuya la función objetivo, si no se evaluará todas las combinaciones y se escogerá la que más disminuya la función objetivo. Si no se encuentra ninguna combinación que disminuya la función objetivo el algoritmo se detendrá y devolverá la mejora solución encontrada.

Para determinar el número de combinaciones a revisar en cada iteración de la segunda búsqueda local se debe tener en cuenta el tamaño del problema. Estas combinaciones son creadas de forma aleatoria y en experimentos preliminares se encontró que el número $\left\lfloor \sqrt{|N|} * 350 \right\rfloor$ crea las suficientes combinaciones para obtener buenas soluciones en un tiempo corto en las instancias utilizadas. Ahora, este puede

ser un parámetro por determinar en una futura implementación de este algoritmo. De forma predeterminada el número máximo de iteraciones es 200, el cual en la mayoría de los casos nunca es alcanzado dado que en alguna iteración anterior se pudo haber no encontrado un cambio que mejore la función objetivo. En este caso se devolvería la mejor solución encontrada y el algoritmo terminaría.

4.1 Pseudocódigo del Primer Algoritmo de Búsqueda Local

Input: init: Solución Inicial, max_iter: Número máximo de combinaciones de nodos a revisar, d: matriz de distancias del problema.
Output: bestSol: La mejor Solución encontrada.
bestSol ← init
Crear estructuras nodo posterior y anterior
longitud_de_ruta ← |*bestSol*|
For i from i = 1 *to MAX_ITER*:
    Combinacion = *generarCombinaciónDosNodos*[*longitud_de_ruta*]
    For cada nodo in Combinación:
        mejora = $d(\text{anterior}_{nodo}, \text{otroNodo}) + d(\text{otroNodo}, \text{posterior}_{nodo}) - d(\text{nodo}, \text{posterior}_{nodo}) - d(\text{anterior}_{nodo}, \text{nodo})$
        If mejora < 0:
            bestSol ← *swap*(*combinación*)
            ActualizaEstructuras(*bestSol*)
Return bestSol

4.2 Pseudocódigo del Segundo Algoritmo de Busqueda Local

Input: init: Solución Inicial, max_iter: Número máximo de iteraciones, FIRST_IMPROVEMENT: booleano si se utiliza como criterio de parada la primera mejora, se utiliza Best Improvement dlc., d: matriz de distancias del problema, numCombinaciones: Numero de combinaciones a revisar por iteracion.
Output: bestSol: La mejor Solución encontrada.
bestSol ← *init*
longitud = |*bestSol*|
Crear estructuras nodo posterior y anterior
for i in i = 1 *to* max *_iter*:
    bestNodes ← { }
    bestChange ← ∞
    combinaciones = *CrearMCombinacionesde3NodosConsecutivos*(*numCombinacioens*, *longitud*)
    for n_1, n_2, n_3 *in combinaciones*:
        mejora = $-d(n_1, posterior(n_1)) - d(n_2, posterior(n_2)) - d(n_3, posterior(n_3)) + d(n_1, posterior_{n_2}) + d(n_2, posterior(n_3)) + d(n_3, posterior_{n_1})$
        if mejora < *bestChange*
            bestNodes = (n_1, n_2, n_3)
            bestChange = *mejora*
            if FIRST_IMPROVEMENT:
                break
    if bestNodes = ∅
        STOP, return bestSol
    bestSol ← *swap*(*bestNodes*)
Return bestSol

4.3 Detalles de Implementación

Se implementaron los algoritmos en Python 3.8 usando una conceptualización de clases para priorizar

el ahorro de memoria. La implementación recibe como parámetros el archivo de la instancia a solucionar, el número del algoritmo a utilizar (0 para Swap, 1 para 3OPT), el parámetro max_iter y si es el segundo algoritmo 1 si se utiliza First Improvement, 0 dlc. y por ultimo el numero de combinaciones en caso de usar el 2do algoritmo.

```
// python HW2.py “nombre_de_la_instancia.txt” #heuristica max_iter First_Improvement combinations
// para correr la segunda heurística con 30 iteraciones, First_Improvement y 5000 combinaciones:
$ python HW1.py “ftv44.atsp.txt” 30 1 5000
```

También se genero un archivo ejecutable para equipos que no tengan Python instalado para equipos tipo Linux. Las instancias se pueden correr con un archivo .sh para correr todos los experimentos acá presentados. Se incluyo un README en los soportes de esta tarea que indican como ejecutar estos archivos.

5 Experimentación y Análisis de Resultados

Para experimentar con los algoritmos propuestos se usaron las instancias para el ATSP de la librería TSPLIB propuestos por G. Reinelt [9]. Esta librería contiene mas de 100 ejemplos con tamaños de ciudades desde 14 hasta 85900 para problemas de ATSP[1]. Estas fueron corridas en un procesador dual core “Broadwell” 2.7 GHz Intel “Core i5” (5275U) con 8 GB 1866 MHz LPDDR3 SDRAM corriendo macOS Catalina 10.15.6 con Python 3.8. Dada la aleatoriedad de los algoritmos, estos fueron ejecutados 10 veces por algoritmo para la experimentación y discusión de resultados.

5.1 Instancias Utilizadas

Las instancias utilizadas fueron recopiladas por G. Reinelt en su librería TSPLIB. Estas provienen de aplicaciones de la vida real y su descripción precisa puede ser encontrada en [8]. De entre las utilizadas se encuentran 4 de tipo rbg las cuales fueron creadas en un problema de aplicación de elevadores de mercancías en bodegas. Las 2 instancias ft vienen de un problema de optimización de secuenciación de tareas en una planta de resinas de colores. Las 17 instancias ftv vienen de un problema de domicilio de una farmacéutica en Bologna con pequeñas variaciones aleatorias. Las instancias ry48p y kro124p vienen de instancias con distancias simétricas que fueron perturbadas aleatoriamente para crear asimetrías. La instancia p43 viene de un problema de programación de tareas en ingeniería química. La instancia br17 viene de una fuente desconocida.

Se clasificarán las instancias para el análisis de resultados de la siguiente forma: de acuerdo con su tamaño y a su proveniencia (usando etiquetas referentes al problema original). A continuación, un análisis de datos con el óptimo del problema, el número de nodos, la categoría según tamaño (donde las instancias con menos de 50 nodos son pequeñas y con más de 150 nodos son grandes) y por proveniencia del problema. La columna de simetría es el calculo de la división entre las desviaciones estándar de las distancias en ambos sentidos entre ciudades, un valor de 1 indicaría que el problema es STSP. La columna triangulo cuantifica que tanto se viola la inequidad del triangulo al calcular el promedio de las desviaciones de cada pareja de ciudades de la inequidad del triangulo. Un valor de 1 indicaría que las distancias respetan la inequidad del triangulo. La inequidad del triangulo es importante dado que algunos algoritmos se basan en la geometría del problema para acelerar su búsqueda. Estos valores fueron sacados de [8].

Se adjunta adicionalmente la solución encontrara por el vecino más cercano, insumo para las heurísticas de búsqueda local implementadas.

[1] Estas instancias pueden encontrarse en la página web del Zuse Institute Berlin: http://elib.zib.de/pub/mp-testdata/tsp/tsplib/atsp/index.html.

Instancia	Óptimo	Nodos	Categoría	Etiqueta	Simetría	Triangulo	Solución Inicial NN
rbg443	2720	443	Grande	Elevadores	0.9507	0.5642	3444
rbg403	2465	403	Grande	Elevadores	0.9505	0.5511	47506
rbg358	1163	358	Grande	Elevadores	0.9517	0.5749	2420
rbg323	1326	323	Grande	Elevadores	0.957	0.6108	2639
ftv170	2755	170	Grande	Domicilio	0.989	1	3888
kro124	36230	124	Mediano	De Simetrico	0.9992	0.9724	5754
ft70	38673	70	Mediano	Secuenciacion	0.9573	1	3873
ftv70	1950	70	Mediano	Domicilio	0.9844	1	1791
ftv64	1839	64	Mediano	Domicilio	0.985	1	2023
ftv55	1608	55	Mediano	Domicilio	0.9853	1	1765
ft53	6905	53	Mediano	Secuenciacion	0.8739	1	92
ry48p	14422	48	Pequeño	De Simetrico	0.9994	0.9849	16757
ftv47	1776	47	Pequeño	Domicilio	0.9832	1	43186
ftv44	1613	44	Pequeño	Domicilio	0.985	1	1778
p43	5620	43	Pequeño	Planeación	0.7565	0.8965	2014
ftv38	1530	38	Pequeño	Domicilio	0.9832	1	9514
ftv35	1473	35	Pequeño	Domicilio	0.9823	1	1683
ftv33	1286	33	Pequeño	Domicilio	0.9838	1	1778
br17	39	17	Pequeño	Desconocido	0.999	0.8474	2571

Tabla 1 Características de las Instancias Utilizadas

5.2 Resultados de las instancias

Todos los resultados acá presentados (Función objetivo y tiempo computacional) se basan en 10 realizaciones aleatorias por problema y algoritmo, y son promedios. El tiempo computacional tiene en cuenta solo el tiempo de ejecución de la heurística sin tener en cuenta el tiempo para procesar los datos y construir la solución por vecino más cercano.

5.2.1 Heurística Swap

Instancia	Función Objetivo	Tiempo Computacional (s)	GAP
rbg443	3226.8	0.981897667	19%
rbg403	2890.1	1.191301602	17%
rbg358	1567	0.792890617	35%
rbg323	1635.3	0.698384653	23%
ftv170	3800.3	0.367408791	38%
kro124	46387.9	0.317388693	28%
ft70	42423.6	0.148908095	10%
ftv70	2420.9	0.151619732	24%
ftv64	2522.5	0.162342411	37%
ftv55	1962.7	0.119296168	22%
ft53	8999.6	0.114648482	30%
ry48p	16600.7	0.102101166	15%

ftv47	2285	0.116309437	29%
ftv44	2014	0.095487439	25%
p43	5686.2	0.090758287	1%
ftv38	1761.9	0.085980422	15%
ftv35	1788	0.076724481	21%
ftv33	1659	0.075294115	29%
br17	41.3	0.035429386	6%

Tabla 2 Resultados Heurísticas Swap

5.2.2 Heurística 3-OPT con Best Improvement

Instancia	Función Objetivo	Tiempo Computacional (s)	GAP
rbg443	2834.8	1.590497363	4%
rbg403	2552.6	1.579681606	4%
rbg358	1357.4	0.608462734	17%
rbg323	1496.8	0.448823441	13%
ftv170	3806.8	0.017369989	38%
kro124	43468.2	0.057907367	20%
ft70	41801.3	0.036699335	8%
ftv70	2350.9	0.027957218	21%
ftv64	2257.5	0.04401623	23%
ftv55	1872.1	0.026788718	16%
ft53	8033.2	0.047482405	16%
ry48p	15693.5	0.019422775	9%
ftv47	1975.3	0.03298785	11%
ftv44	1835.3	0.017688388	14%
p43	5642.8	0.017572605	0%
ftv38	1623.3	0.020732923	6%
ftv35	1568.7	0.017464189	6%
ftv33	1401.2	0.02114841	9%
br17	39	0.006046235	0%

Tabla 3 Heurística 3-OPT de Best Improvement

5.2.3 Heurística 3-OPT con First Improvement

Instancia	Función Objetivo	Tiempo Computacional (s)	GAP
rbg443	3028.7	0.367793799	11%
rbg403	2655.6	0.447638769	8%
rbg358	1416.7	0.219496118	22%
rbg323	1525.3	0.176407451	15%
ftv170	3847.6	0.014355679	40%
kro124	43862.2	0.03916447	21%

ft70	42095.4	0.020310574	9%
ftv70	2361.1	0.016079869	21%
ftv64	2269.6	0.024086389	23%
ftv55	1895.3	0.013685541	18%
ft53	8091.9	0.024302636	17%
ry48p	15630.2	0.016182288	8%
ftv47	1989	0.020514533	12%
ftv44	1841.1	0.011261403	14%
p43	5638.9	0.011823339	0%
ftv38	1693.4	0.008400078	11%
ftv35	1580.9	0.010851298	7%
ftv33	1413.2	0.01176037	10%
br17	39	0.003175267	0%

Tabla 4 Resultados Heurística 3-OPT First Improvement

5.3 Comparación con el óptimo y análisis de resultados

Primero realizaremos una comparación por categorías:

Tamaño de Instancia	**GAP Swap**	**Tiempo Swap (s)**	**GAP 3OPT Best Improvement**	**Tiempo 3OPT Best Improvement (s)**	**GAP 3OPT First Improvement**	**Tiempo 3OPT First Improvement (s)**
Grande	26.40%	0.8064s	15.10%	0.8490s	19.10%	0.2451s
Mediano	25.20%	0.1690s	17.40%	0.0401s	18.20%	0.0229s
Pequeño	17.70%	0.0848s	7.00%	0.0191s	7.80%	0.0117s
Todas las instancias	22.30%	0.3013s	12.40%	0.2441s	14.10%	0.0767s

Tabla 5 Resultados por tamaño de instancia

Tipo de Instancia	**GAP Swap**	**Tiempo Swap (s)**	**GAP 3OPT BI**	**Tiempo 3OPT BI (s)**	**GAP 3OPT FI**	**Tiempo 3OPT FI (s)**
De Simétrico	21.60%	0.2097s	14.40%	0.0387s	14.70%	0.0277s
Desconocido	5.90%	0.0354s	0.00%	0.0060s	0.00%	0.0032s
Domicilio	26.70%	0.1389s	16.10%	0.0251s	17.30%	0.0146s
Elevadores	23.50%	0.9161s	9.30%	1.0569s	14.00%	0.3028s
Planeación	1.20%	0.0908s	0.40%	0.0176s	0.30%	0.0118s
Secuenciación	20.00%	0.1318s	12.20%	0.0421s	13.00%	0.0223s
Todas las instancias	22.30%	0.3013s	12.40%	0.2441s	14.10%	0.0767s

Tabla 6 Resultados por tipo de instancia

Al analizar los resultados, observamos que la heurística 3-OPT es completamente superior a la Swap en tiempo computacional y en función objetivo. Esto se debe a que el vecindario de la heurística 3-OPT es más grande que el de Swap y permite transiciones más sofisticadas. En promedio la heurística Swap obtuvo un GAP de 22.3% comparado con el GAP del 3OPT BI de 12.40%, una diferencia de casi 10 puntos. La heurística Swap además se demora considerablemente más que la heurística 3-OPT (0.3s vs 0.24s) y entrega peores resultados, tanto, que no logra encontrar el óptimo en ninguna instancia. El número de nodos a revisar en la heurística Swap fue un numero suficiente para cada instancia y aun así no pudo superar al 3-OPT.

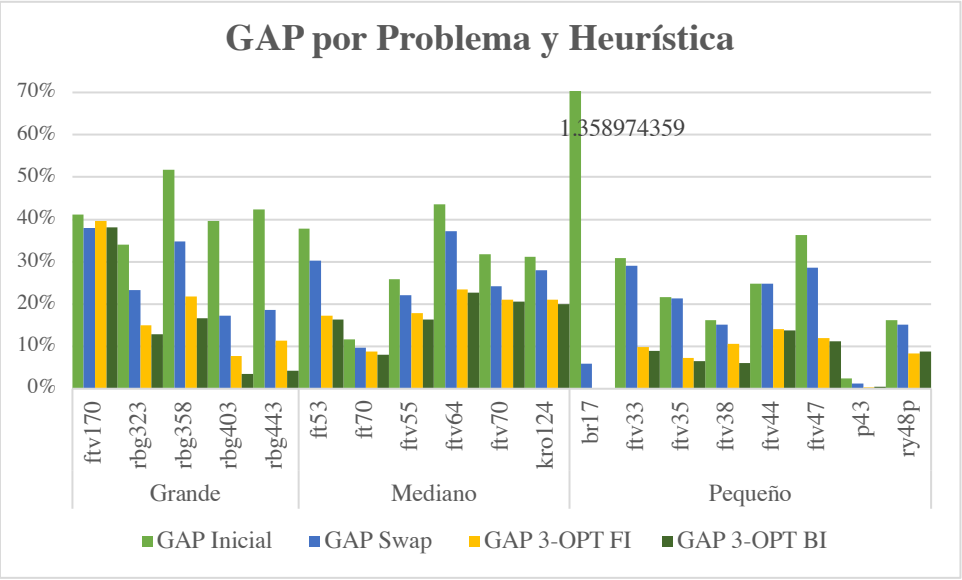
Con relación a la solución inicial, la mejora en GAP generada por la heurística Swap no fue muy significativa en algunas instancias incluso no pudiendo encontrar una mejor solución en ocasiones donde la heurística 3OPT si pudo hacerlo de forma importante.

Por otro lado, la heurística 3-OPT se desempeño de muy buena forma encontrando buenas soluciones

con GAP's inferiores al 15% hasta en instancias grandes. Para la heurística con criterio Best Improvement se obtuvo un tiempo promedio de 0.24s y un GAP promedio de 12.4% y obtuvo los mejores resultados para casi cada tipo de instancia (en tipo y tamaño). Al utilizar el criterio First Improvement el Gap promedio fue de 14.1% (1.7pts más que BI) y tuvo un tiempo computacional promedio de 0.0767s (0.1674s menos que BI). Podemos concluir que, aunque el criterio de Best Improvement brinda las mejores soluciones, utilizar First Improvement reduce el tiempo de solución en 70% sin aumentar de forma desbordada grande la función objetivo. Hay que notar que la diferencia en GAPs en problemas grandes entre los dos criterios fue del 4% lo cual puede llegar a ser significativo en algunos casos.

Observamos que no hay una diferencia muy significativa entre los resultados de las instancias medianas y grandes por lo que se estas pueden ser vistas como una sola categoría que empieza desde los 50 nodos. Entre las instancias grandes vemos que el número de nodos no es un impedimento para llegar a buenos resultados (con GAPSs cercanos a 5% en instancias de más de 400 nodos).

Las instancias ftv (Domicilios en Bologna) fueron las más difíciles de resolver con un GAP de 16.1% en promedio (el menor) obtenido con la heurística 3OPT BI. Las instancias con origen simétrico también tienen problemas al ser resueltas con estas heurísticas ya que el mínimo GAP alcanzado en promedio fue de 14.4%. Una característica en común entre estos dos tipos de instancia es que ambas respetan la inequidad del triangulo para cada par de nodos a diferencia de las otras instancias que obtienen mejores resultados. Esto podría ser indicios que la estructura de la instancia (su geometría) tiene implicaciones para el alcance de los algoritmos implementados. Esto se podría resolver aplicando nuevas heurísticas que exploten esta propiedad (que es deseada en algunos algoritmos).



Gráfica 1 Resultados de todas las instancias por Heurística

6 Conclusiones

- Las soluciones obtenidas con la heurística Swap varían considerablemente al brindar soluciones aceptables en algunos casos y no mejorarla en otros. Su tiempo computacional es intensivo en comparación con la otra heurística y tiene un vecindario reducido que no le permite hacer movimientos complejos que mejoren de forma importante la solución. Sin embargo, se podria utilizar en conjunto con otro tipo de movimientos o con una lista restringida de combinaciones

para brindar mejores soluciones.

- La heurística 3OPT brinda muy buenas soluciones en casi todos los casos. Usando el criterio de Best Improvement se consiguen los mejores resultados (12.4% en promedio para todas las instancias probadas) con la desventaja de un tiempo computacional elevado (0.224s en promedio). Al usar el criterio de First Improvement se logra reducir este tiempo en 70% con el compromiso de solo 1.7pts en la función objetivo en promedio. Esta heurística se comporta muy bien incluso para instancias grandes donde, para la más grande logra llevar el Gap a 4.2% en 1.6s.
- Algunos tipos de instancia se comportaron mejor que otras en términos de la solución alcanzada. Sobretodo, las instancias cuyas distancias respetaban la desigualdad triangular tuvieron un peor desempeño que aquellas instancias que no lo hacían para algunos pares de ciudades. Podemos concluir que las heurísticas implementadas dependen de la estructura del problema a un nivel geométrico y necesitarían complementarse de alguna forma para poder resolver cualquier tipo de instancia independientemente de su geometría.
- Recomendamos usar el algoritmo 3OPT con Best Improvement para los mejores resultados y 3OPT con First Improvement para los mejores tiempos de ejecución.

7 Referencias

[1] L. Hong, L. Si-Yang, H. Yan-Wei, C. Yong-Quan and F. Zhang-Hua, "An Efficient 2-opt Operator for the Robotic Task Sequencing Problem," in *Proceedings of The 2019 IEEE International Conference on Real-time Computing and Robotics*, Irkutsk, 2019.

[2] M. Garey and D. Johnson, Computers and Intractability: A Guide to the Theory of NP-Completeness, San Francisco: Freeman, 1979.

[3] D. S. Johnson, G. Gutin, L. A. McGeoch, A. Yeo, W. Zhang and A. Zverovitch, "EXPERIMENTAL ANALYSIS OF HEURISTICS FOR THE A TSP," in *THE TRAVELING SALESMAN PROBLEM AND ITS VARIATIONS*, New York, Springer, 2007, pp. 445-487.

[4] T. Öncan, I. K. Altınel and G. Laporte, "A comparative analysis of several asymmetric traveling salesman problem formulations," *Computers & Operations Research,* vol. 36, pp. 637-654, 2009.

[5] TSP Basics, "3-opt move," Basics of Local Optimization in the Symmetric Traveling Salesman Problem (STSP), 6 Marxo 2017. [Online]. Available: http://tsp-basics.blogspot.com/2017/03/3-opt-move.html. [Accessed 10 Octubre 2020].

[6] TSP Basics, "Node Swap," Basics of Local Optimization in the Symmetric Traveling Salesman Problem (STSP), 11 Marzo 2017. [Online]. Available: http://tsp-basics.blogspot.com/2017/03/node-swap.html. [Accessed 10 Octubre 2020].

[7] P. Kanellakis and C. Papadimitriou, "Local Search for the Asymmetric Travelling Salesman Problem," *Operations Research,* vol. 28, no. 5, pp. 1086-1099, 1960.

[8] J. J. D. S. M. L. A. &. Z. W. Cirasella, "The Asymmetric TSP: Algorithms, Instance Generators, and Tests," *Notes in Computer Science,* pp. 32-59, 2001.

[9] G. Reinelt, "A Traveling Salesman Problem Library," *ORSA Journal on Computing,* vol. 3, pp. 376-386, 1991.