UNIVERSIDAD TORCUATO DI TELLA
Métodos Computacionales

TP 3: Heurísticas para el Problema del Viajante de Comercio Asimétrico (ATSP)

Integrantes:
Valentina Gayo
Caterina Villegas
Valentina Vitetta

16 de Julio del 2024

Contents

1 Introducción

Mediante el presente informe, se introduce y analiza el Problema del Viajante de Comercio Asimétrico (ATSP), una variante crucial del clásico Problema del Viajante de Comercio (TSP). En el ATSP, un vendedor debe visitar un conjunto de ciudades exactamente una vez y regresar a la ciudad de origen, minimizando la distancia total recorrida. La diferencia clave entre el TSP y el ATSP radica en la asimetría de las distancias: la distancia de la ciudad A a la ciudad B puede ser diferente de la distancia de la ciudad B a la ciudad A. Este problema refleja situaciones reales donde los costos de traslado no son simétricos, como en ciudades con calles de un solo sentido.

El objetivo principal de este trabajo es desarrollar e implementar diferentes heurísticas para resolver el ATSP, evaluándolas mediante instancias de benchmark de la reconocida biblioteca TSPLIB. La importancia de este trabajo radica en la necesidad de encontrar soluciones eficientes a problemas de optimización complejos que tienen aplicaciones prácticas en diversas áreas como la logística, el diseño de circuitos integrados y la optimización de códigos genéticos.

Las aplicaciones prácticas del ATSP son diversas y abarcan áreas como la logística y el transporte, donde se utiliza para optimizar rutas y reducir costos de flota, así como en la perforación de circuitos integrados, optimización de códigos genéticos y problemas de recolección en depósitos. A pesar de su complejidad y su clasificación como problema NP-hard, la investigación en ATSP ha dado lugar a algoritmos extremadamente eficientes capaces de resolver instancias de miles de clientes en tiempos razonables.

Este trabajo práctico se enfoca en la implementación y evaluación de diferentes heurísticas para resolver el ATSP utilizando instancias de benchmark de la TSPLIB. Se busca desarrollar heurísticas constructivas y operadores de búsqueda local, combinar estos métodos y realizar experimentaciones exhaustivas para comparar la calidad de las soluciones y el tiempo de ejecución. El objetivo final es proponer un método robusto que combine las mejores técnicas para resolver eficientemente el ATSP. En este informe, se presentará una descripción detallada de los algoritmos implementados, las decisiones de diseño tomadas, los resultados obtenidos en la experimentación y las conclusiones derivadas de estos resultados. Además, se incluirán observaciones sobre posibles mejoras y dificultades encontradas durante el desarrollo del trabajo.

2 Implementación

Cabe destacar que se optaron por utilizar los operadores 2-opt y relocate por lo diversos que son entre sí. 2-opt permite explorar nuevas configuraciones del tour sin agregar nodos adicionales. Funciona revirtiendo segmentos específicos del tour y reorganizando el orden de visita de los nodos. Al hacer esto, se pueden eliminar cruces y mejorar la disposición del recorrido, lo que potencialmente reduce significativamente el costo total del tour. Dado que el ATSP implica costos asimétricos entre los nodos, el 2-opt es valioso porque puede ajustar la secuencia de visitas para minimizar estos costos asimétricos. Por otro lado, relocate, es útil porque permite mover nodos dentro del tour sin cambiar la secuencia completa de los nodos. Esto es importante en el ATSP, donde la posición relativa de los nodos puede afectar el costo del recorrido. Al reubicar nodos, se pueden encontrar configuraciones que reduzcan el costo total al mejorar la disposición espacial de los nodos dentro del tour. Además, al igual que el 2-opt, el relocate no introduce nodos adicionales en el tour, lo que lo hace eficiente y adecuado para optimizar soluciones existentes.

Además, hace falta mencionar que utilizamos la biblioteca NetworkX para la representación y manipulación de grafos.

2.1 `vecino_mas_cercano()`

$O(n^2) \rightarrow$ (`while` $*$ `for`)

Recibe un grafo dirigido y ponderado G y un nodo inicial nodo_inicial. Devuelve un recorrido que visita cada nodo una vez y regresa al inicial, minimizando el costo total. Utiliza una heurística que selecciona, en cada paso, el vecino no visitado con el menor costo de arista, considerando tanto la distancia de ida como de vuelta debido a la asimetría de los costos en el ATSP. Finalmente, completa el recorrido regresando al nodo inicial.

2.2 `elegir_nodo()`

$O(n^2) \rightarrow$ (`for` $*$ `for`)

Recibe un conjunto de nodos visitados visitado, un grafo G, una lista solucion con el recorrido parcial y un booleano mas_cercano que indica si se busca el nodo más cercano o más lejano. Devuelve el siguiente nodo a visitar. El algoritmo evalúa todas las aristas desde cada nodo en solucion hacia nodos no visitados. Selecciona el nodo con el menor o mayor costo según mas_cercano. La asimetría se maneja considerando las direcciones de todas las aristas, permitiendo así elegir correctamente en contextos con costos de ida y vuelta diferentes.

2.3 `insertar_nodo()`

$O(n) \rightarrow$ (`for`)

Recibe un nodo v, un grafo G y una lista solucion que representa un recorrido parcial. Devuelve los nodos min_u y min_w entre los cuales se debe insertar v para minimizar el incremento en el costo del recorrido. El algoritmo inicialmente considera insertar v al final del recorrido, comparando el costo de esta inserción con otras posibles posiciones dentro de la lista solucion. Evalúa la diferencia de costos al insertar v entre cada par de nodos consecutivos en solucion. La posición que genere el menor incremento en el costo total es seleccionada como el lugar óptimo para insertar v, teniendo en cuenta las aristas dirigidas y sus costos, lo que permite manejar la asimetría en el ATSP.

2.4 `atsp_insercion()`

$O(n^3) \rightarrow$ (`while` $*$ `elegir_nodo()` / `insertar_nodo()` $*$ `.append` / `.insert`)

Recibe un grafo dirigido y ponderado G y un booleano mas_cercano para determinar si se buscan nodos más cercanos o lejanos. Devuelve el costo total del recorrido y la lista solucion que representa el tour completo. El algoritmo comienza con un ciclo inicial de tres nodos y marca estos nodos como visitados. Luego, en cada iteración, elige el siguiente nodo no visitado utilizando la función elegir_nodo y lo inserta en la posición óptima del recorrido parcial utilizando la función insertar_nodo. El costo total se actualiza con cada inserción, considerando las aristas dirigidas para manejar la asimetría del ATSP. Finalmente, se cierra el tour agregando el nodo inicial al final del recorrido. Este enfoque permite construir un tour eficiente incrementando gradualmente su longitud mientras minimiza el costo adicional en cada paso.

2.5 `costo_tour()`

$O(n) \rightarrow$ (`for`)

Recibe un grafo dirigido y ponderado G y una lista tour que representa un recorrido. Calcula y devuelve el costo total del recorrido sumando los pesos de las aristas entre nodos consecutivos en la lista tour.

2.6 `do_2opt()`

$O(n)$

Recibe una lista tour y dos índices i y j. Realiza un intercambio 2-opt en el recorrido, que

consiste en revertir el segmento de la lista comprendido entre los índices i+1 y j+1. Este enfoque se utiliza para mejorar la solución del ATSP al explorar una nueva configuración del tour, con la esperanza de reducir su costo total. Al revertir los nodos entre los índices especificados, se eliminan dos aristas y se reemplazan por dos nuevas, lo que puede potencialmente disminuir la longitud total del recorrido.

2.7 `atsp_2opt()`

$O(n^4) \rightarrow$ (`while` $*$ `for` $*$ `for` $*$ `do_2opt()`/`costo_tour()`)

Mejora un recorrido inicial en el ATSP utilizando el operador 2-opt en un grafo G. Comienza con un recorrido inicial tour_inicial, y busca iterativamente intercambios que disminuyan el costo total del recorrido. Devuelve el recorrido mejorado tour_actual y su costo costo_actual, actualizando continuamente hasta que no se encuentren más mejoras significativas.

2.8 `relocate()`

$O(n^4) \rightarrow$ (`while` $*$ `for` $*$ `for` $*$ `costo_tour()`)

Busca mejorar un tour en el ATSP utilizando la operación de reubicación. Recibe un grafo G y un tour inicial, y devuelve el tour mejorado junto con su costo. Aplica la reubicación de nodos dentro del tour para intentar reducir el costo total del recorrido, evaluando diversas combinaciones de reubicación en busca de mejoras.

2.9 `goloso_con_busqueda_local()`

$O(n^4) \rightarrow$ (en el peor caso: `atsp_2opt()`/`relocate()`)

Aclaración: tenemos dos funciones, `goloso_con_busqueda_local_2opt()` y

`goloso_con_busqueda_local_relocate()`, que reciben un grafo G, un nodo inicial *nodo_inicial*, y una heurística constructiva (`vecino_mas_cercano` o `atsp_insercion`). Estas funciones utilizan la heurística para generar un tour inicial y luego aplican el método de búsqueda local correspondiente (2-opt o relocate) para mejorar dicho tour en términos de costo.

2.10 `VND()`

$O(n^4)$

Implementa la metaheurística Variable Neighborhood Descent (Descenso por Vecindarios Variables) para mejorar un tour en el Problema del Viajante de Comercio Asimétrico (ATSP). Utiliza dos vecindades diferentes: relocate y 2-opt. Itera entre estas vecindades buscando mejorar el tour actual. Si encuentra una mejora, reinicia el proceso. Devuelve el tour mejorado y su costo final.

2.11 Aclaraciones de uso

1. Elección de instancia: descomentar la instancia especificada por filename a utilizar
2. Carga de datos: se crea el grafo para representar la información correcta
3. El script muestra los costos mínimos y tiempos de ejecución para cada método aplicado a la instancia seleccionada.

Lo que se imprime por terminal es:

- ATSP inserción y vecino más cercano : se calcula el costo mínimo y tiempo de ejecución utilizando el método correspondiente.
- VND: se usan dos variantes de VND sobre la solución inicial generada por ATSP Inserción y vecino más cercano, midiendo el costo y tiempo de ejecución para cada caso.
- Búsqueda local: aplica algoritmos golosos con búsqueda local (2-opt y Relocate) sobre las soluciones iniciales de ATSP Inserción y Vecino Más Cercano, mostrando el costo y tiempo de ejecución para cada método.

3 Experimentación

Para analizar el desempeño de las diversas heurísticas implementadas para el Problema del Viajante de Comercio Asimétrico (ATSP), realizamos una experimentación utilizando distintas instancias de benchmark de la librería TSPLIB. Decidimos ejecutar los algoritmos en las siguientes instancias:

- br17.atsp
- ry48p.atsp
- kro124p.atsp

- ftv170.atsp

De este modo, buscamos comprender cómo varía el comportamiento de las heurísticas a medida que aumenta la cantidad de ciudades en cada instancia. Además, comparamos los resultados obtenidos por cada heurística para evaluar su eficiencia y efectividad en diferentes escenarios.

3.1 Instancia: br17.atsp

	Goloso	Búsqueda Local: Relocate	Búsqueda Local: 2-opt	VND - combinación de operadores
Vecino más cercano	92	78	39	42
Inserción	39	39	39	39

Table 1: Mejor tour encontrado para ATSP: br17.atsp

	Goloso	Búsqueda Local: Relocate	Búsqueda Local: 2-opt	VND - combinación de operadores
Vecino más cercano	0.0	0.008	0.0	0.016
Inserción	0.0	0.007	0.0	0.008

Table 2: Tiempo de ejecución de los métodos para ATSP: br17.atsp

3.2 Instancia: ry48p.atsp

	Goloso	Búsqueda Local: Relocate	Búsqueda Local: 2-opt	VND - combinación de operadores
Vecino más cercano	17039	16077	15163	15725
Inserción	15964	15960	15887	15883

Table 3: Mejor tour encontrado para ATSP: ry48p.atsp

	Goloso	Búsqueda Local: Relocate	Búsqueda Local: 2-opt	VND - combinación de operadores
Vecino más cercano	0.0	0.188	0.130	0.509
Inserción	0.0	0.105	0.099	0.294

Table 4: Tiempo de ejecución de los métodos para ATSP: ry48p.atsp

3.3 Instancia: kro124p.atsp

	Goloso	Búsqueda Local: Relocate	Búsqueda Local: 2-opt	VND - combinación de operadores
Vecino más cercano	45285	40623	42312	40475
Inserción	43349	40943	42474	40943

Table 5: Mejor tour encontrado para ATSP: kro124p.atsp

	Goloso	Búsqueda Local: Relocate	Búsqueda Local: 2-opt	VND - combinación de operadores
Vecino más cercano	0.008	3.451	3.227	5.516
Inserción	0.185	3.145	1.841	3.219

Table 6: Tiempo de ejecución de los métodos para ATSP: kro124p.atsp

3.4 Instancia: ftv170.atsp

	Goloso	Búsqueda Local: Relocate	Búsqueda Local: 2-opt	VND - combinación de operadores
Vecino más cercano	7105	4736	5628	4692
Inserción	3492	3423	3391	3331

Table 7: Mejor tour encontrado para ATSP: ftv170.atsp

	Goloso	Búsqueda Local: Relocate	Búsqueda Local: 2-opt	VND - combinación de operadores
Vecino más cercano	0.016	19.927	55.337	17.714
Inserción	0.417	4.378	5.528	18.284

Table 8: Tiempo de ejecución de los métodos para ATSP: ftv170.atsp

3.5 Análisis de Resultados

Para analizar los resultados obtenidos de la experimentación con los algoritmos golosos, de búsqueda local y VND para resolver instancias del problema ATSP, es fundamental destacar las diferencias en la calidad de las soluciones y los tiempos de ejecución obtenidos.

Los algoritmos golosos, como el vecino más cercano y la inserción, están diseñados para construir una solución inicial rápida. Por ejemplo para instancias como rbg443.atsp, estos algoritmos pudieron generar una solución rápidamente. Sin embargo, tienden a producir soluciones de calidad subóptima ya que se enfocan en hacer la mejor elección en cada paso sin considerar el impacto global en la solución final. Por ejemplo, para la instancia br17.atsp, la heurística constructiva de vecino más cercano encontró una solución con un costo de 92, mientras que la inserción logró una solución con un costo de 39, que es el óptimo.

Por otro lado, los algoritmos de búsqueda local, como 2-OPT y Relocate, comienzan con una solución inicial y buscan mejorarla mediante pequeñas modificaciones. Aunque estos métodos son más lentos que los golosos, tienden a encontrar soluciones de mejor calidad. Para la instancia. ftv170.atsp, 2-OPT mejoró la solución inicial obtenida por vecino más cercano de 7105 a 5628, mostrando una mejora significativa en la calidad de la solución, aunque sigue sin ser óptima. Similarmente, para kro124p.atsp, 2-OPT mejoró la solución inicial de vecino más cercano de 45285 a 42312, destacando su capacidad para refinar la calidad de las soluciones iniciales.

El método de Variable Neighborhood Descent (VND) combina varias estrategias de búsqueda local, lo que le permite explorar múltiples vecindarios y encontrar soluciones de alta calidad. En la instancia ftv170.atsp, VND con inserción como heurística constructiva logró un costo de 3331, mejor que los costos obtenidos por los métodos golosos y los de busqueda local solos. En un caso similar en la instancia kro124p.atsp, VND con vecino más cercano logró la mejor solución con un costo de 40475, mostrando su efectividad en mejorar significativamente las soluciones iniciales y de búsqueda local.

Para evaluar la proximidad de nuestras mejores soluciones a las mejores soluciones conocidas para cada instancia en TSPLIB, decidimos comparar nuestros resultados con estas referencias. Esta comparación nos proporcionó un punto de referencia crítico para evaluar el rendimiento de nuestras heurísticas. En el caso de la instancia br17.atsp, encontramos que nuestra mejor solución (39) coincidía con la mejor solución conocida según TSPLIB. Sin embargo, para las demás instancias, aunque no alcanzamos la mejor solución conocida, nuestros resultados fueron cercanos. Esta variabilidad podría deberse al tamaño de las instancias; es común que en instancias pequeñas como br17.atsp nos acerquemos más a la mejor solución conocida.

El siguiente gráfico busca visualizar la diferencia entre nuestras soluciones encontradas y las

mejores soluciones conocidas en TSPLIB:

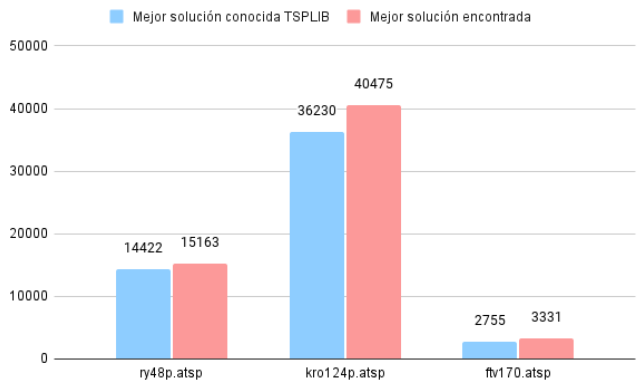

Figure 1: Comparación entre la mejor solución conocida y la mejor solución encontrada

Aunque nuestras mejores soluciones no alcanzaron las mejores soluciones conocidas en TSPLIB, mostraron una notable aproximación, especialmente en instancias como ry48p.atsp y ftv170.atsp. La diferencia observada, refuerza la importancia de continuar refinando nuestros enfoques y explorar nuevas estrategias para mejorar aún más la calidad de nuestras soluciones en futuras investigaciones.

Por otro lado, en términos de tiempo de ejecución, los métodos golosos fueron los más rápidos, con tiempos de ejecución menores en instancias como ry48p.atsp, donde el vecino más cercano y la inserción completaron en tiempos significativamente menores que 2-OPT y VND. Esto se debe a que los métodos golosos realizan una construcción inicial rápida del tour sin realizar mejoras significativas, mientras que los métodos de búsqueda local y VND invierten más tiempo en mejorar la calidad de las soluciones.

Nos llamó la atención que el tiempo de ejecución era mayor en inserción que en vecino más cercano en la mayoría de los resultados. Una posible explicación para la discrepancia observada entre la complejidad teórica y el tiempo de ejecución real del algoritmo de inserción en comparación con el vecino más cercano puede ser la implementación del algoritmo de inserción, que quizás está más optimizado para ciertos tipos de datos o instancias del problema, lo que reduce el tiempo de ejecución en la práctica. Además, para instancias pequeñas del problema, la diferencia en la complejidad teórica puede no ser significativa en términos de tiempo de ejecución real.

En conclusión, para instancias grandes de ATSP, los métodos golosos proporcionan soluciones iniciales rápidas pero de menor calidad. La búsqueda local mejora significativamente estas soluciones, mientras que VND ofrece las mejores soluciones finales en la mayoría de los casos, a pesar de requerir tiempos de cómputo más largos. Este análisis de las tablas correspondientes sugiere que los métodos de búsqueda local y VND tienden a mejorar las soluciones iniciales generadas

por los métodos golosos, logrando una mayor reducción en el costo total del tour. Esto destaca la importancia de combinar métodos golosos con técnicas de búsqueda local y VND para obtener soluciones eficientes y de alta calidad en problemas complejos como el ATSP.

4 Ejercicio Extra

Cuando investigamos para hacer el ejercicio extra, descubrimos que se puede reducir este problema a TSP y decidimos explorar esta opción porque el TSP cuenta con varios algoritmos avanzados y técnicas optimizadas disponibles. Nos pareció interesante ya que esto haría que al convertir ATSP a TSP, se podrían utilizar estas soluciones existentes para simplificar la implementación y la resolución.

Para reducir el problema hicimos una función, `atsp_to_tsp()`, donde creamos una transformación del grafo original de ATSP a uno equivalente de TSP. En este proceso, cada ciudad en ATSP se representa con dos nodos en TSP: uno para la ida ("_in") y otro para la vuelta ("_out"). Las aristas que conectan ciudades en ATSP se convierten en dos aristas en TSP, una para cada dirección entre los nodos "_in" y "_out", manteniendo los mismos costos originales para reflejar la asimetría de los costos en ATSP. Además, añadimos aristas con peso cero desde cada nodo "_in" a su correspondiente nodo "_out", excepto para el propio nodo, asegurándonos de que todas las conexiones entre nodos "_in" y "_out" estén correctamente representadas.

Hicimos un ejemplo extra de una instancia chica de ATSP con 4 nodos. Esta se puede ver en el archivo `ejerextra.txt`. Luego, usamos la función `atsp_to_tsp` para transformar el grafo y lo graficamos usando la función `visualizar_grafo`, lo que nos permitió ver la representación gráfica tanto del grafo ATSP original como del grafo TSP transformado. Este ejemplo se puede ver en la figure 2

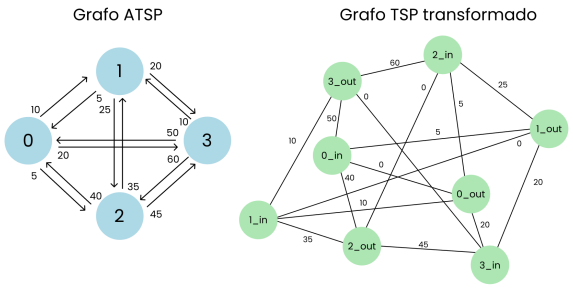

Figure 2: Visualización del grafo ATSP original y del grafo TSP transformado

5 Conclusión

El análisis y la implementación de heurísticas para el Problema del Viajante de Comercio Asimétrico (ATSP) han demostrado la importancia de utilizar combinaciones de algoritmos golosos y técnicas de búsqueda local para obtener soluciones de alta calidad en problemas de optimización complejos. Los algoritmos golosos, como el vecino más cercano y la inserción, proporcionan soluciones iniciales rápidas pero suelen ser subóptimas debido a su enfoque en decisiones locales. En cambio, las técnicas de búsqueda local, como 2-opt y relocate, mejoran estas soluciones iniciales al explorar modificaciones que pueden reducir el costo total del recorrido.

La metaheurística Variable Neighborhood Descent (VND) ha mostrado ser particularmente efectiva, combinando múltiples estrategias de búsqueda local para explorar diferentes vecindarios y encontrar soluciones de mejor calidad. Los resultados de la experimentación con instancias de benchmark de TSPLIB indican que VND, aunque más costoso en términos de tiempo de cómputo, logra las mejores soluciones finales, destacando su robustez y eficiencia en la optimización del ATSP.

La discrepancia observada en los tiempos de ejecución entre las heurísticas constructivas y los métodos de búsqueda local también resalta la importancia de la implementación y la optimización específica de los algoritmos para diferentes tipos de problemas y datos. En resumen, la combinación de heurísticas constructivas rápidas con técnicas de búsqueda local y metaheurísticas como VND es esencial para abordar de manera efectiva y eficiente el ATSP, proporcionando soluciones prácticas y de alta calidad para aplicaciones en logística, diseño de circuitos y otros campos donde los costos de traslado no son simétricos.