UNIVERSIDAD TORCUATO DI TELLA
TD V: Diseño de Algoritmos

Trabajo Práctico 3: Algorítmos greedy

Integrantes:

Gallo Tomas

Grinblat Ezequiel

Sneider Jazmin

19 de Junio 2024

Indice

0 Introducción

Uno de los problemas entre la comunidad científica en la actualidad es la búsqueda de resultados óptimos en tiempo polinomial de múltiples problemas de los que se desconoce sí tienen una forma de resolverse en este tiempo de forma óptima. Entre estos se encuentra el problema del viajante de comercio asimétrico que será en el cual se base el siguiente trabajo práctico e informe. Este problema hoy en día se encuentra considerado como un problema NP-hard por lo que buscaremos distintas heurísticas y búsquedas locales para intentar encontrar o acercarnos los más posible al óptimo en tiempo polinomial. En el siguiente informe podremos ver las distintas heurísticas y búsquedas locales que implementamos acompañadas de una exhaustiva experimentación con las mismas.

1 Ejercicio 1

Implementamos 4 heuristicas golosas.

1. **Ciudad más cercana:** En esta heuristica golosa decidimos pararnos sobre la última ciudad visitada del algoritmo, llamemosla **A**, y en base a esta buscar aquella ciudad a la que puedo llegar con menor distancia entre las no visitadas, llamemosla **B**. Luego tomo **B** y la agrego a la lista de ciudades visitadas y sigo buscando hasta tener todas las ciudades del grafo en la lista. Este algoritmo va buscando la menor distancia de **A -> B**. Con este algoritmo buscamos construir un circuito simple en base a minimos.

2. **Llegada más cercana:** Como el grafo es asimétrico, quisimos evaluar qué sucedía si buscabamos en las distancias de **A <- B** ya que las longitudes deberían de variar. En llegada más cercana, a diferencia de ciudad más cercana, nos paramos sobre la última ciudad visitada, llamemosla **A** y de ahí buscamos aquella ciudad que tiene menos costo en llegar hacia ella, llamemosla **B**. Es decir busca la menor distancia **A <- B**. Con este algoritmo buscamos construir un circuito simple en base a minimos de cada punto.

3. **Menor distancia promedio:** En este algoritmo aproximado decidimos pararnos sobre la última ciudad visitada **A** y de ahí visitar aquella ciudad cuya distancia promedio hacia todos los otros nodos no visitados sumado a la distancia desde **A** sea menor. Con este algoritmo intentamos construir un circuito simple con los nodos que tengan menor distancia hacia todos los otros nodos restantes.

4. **Minimo de distancias:** Con este algoritmo miramos 2 ciudades hacia adelante, buscamos visitar aquel destino cuya distancia desde la ciudad actual sumada a la de su ciudad más cercana no visitada sea menor.

2 Ejercicio 2

Implementamos 4 operadores de busqueda local.

1. **Relocate:** Como primera idea para la optimización de una solución inicial mediante un operador de búsqueda local, hemos decidido implementar el método denominado relocate. Este enfoque consiste en seleccionar nodos del camino pasado por parametro y reposicionarlos en otras ubicaciones, evaluando continuamente si estas nuevas posiciones mejoran la solución actual. Si se encuentra una mejora, el nodo permanece en su nueva ubicación; de lo contrario, se prueba con otra posición o nodo. Este proceso se repite para todos los nodos en todas las posibles ubicaciones, asegurando que no se pierdan posibles soluciones mejores.

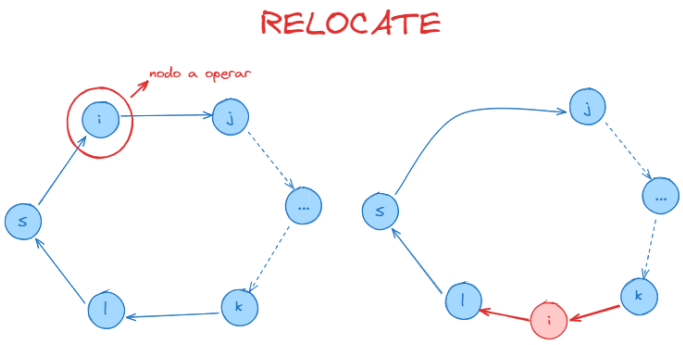

Figure 1: ilustracion del operador: relocate

2. **Swap:** Otro operador interesante que implementamos es swap. Al igual que relocate, este método trabaja con todos los nodos del camino proporcionado como parámetro, intercambiándolos con todos los demás nodos del mismo camino, siempre y cuando no sean el mismo nodo. Por cada intercambio (swap), evaluamos si se ha mejorado la solución y actualizamos la misma en consecuencia. Al finalizar el proceso, si no se encuentra una mejor solución, se devuelve la solución inicial; de lo contrario, se devuelve la nueva solución mejorada.

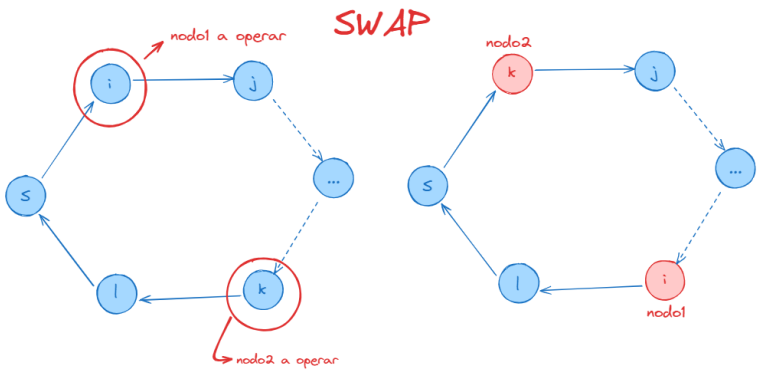

Figure 2: ilustración del operador: swap

3. **2-opt:** Además, decidimos implementar un operador más complejo que los anteriores con la esperanza de al complejizar el operador lograr una mayor optimización, denominado 2-opt. Este método consiste en elegir un par de nodos, donde el primero es menor que el segundo y no son nodos consecutivos. A continuación, se realiza la operación de 2-opt, que se detalla de la siguiente manera:

   - **Selección de nodos:** Elegimos un par de nodos, designando el primer nodo del par como i y su siguiente como j. El segundo nodo del par se llama k y su siguiente como l.
   - **Desconexión y reconexión:** Desconectamos la arista entre i y j, y conectamos la arista entre i y k. Luego, desconectamos la arista entre k y l.
   - **Inversión de aristas:** Invertimos todas las aristas desde k hasta j y conectamos la arista entre j y l. El resto del camino permanece igual.

   Siempre aplicamos el operador sobre la solución inicial pasada por parámetro, guardando el nuevo camino más óptimo si se encuentra una mejora. Este operador se aplica a todos los pares de nodos posibles para no perder potenciales soluciones mejores, garantizando una búsqueda exhaustiva de mejoras.

4. **2-opt mejorado:** El operador 2-opt mejorado funciona de manera similar al 2-opt estándar, con una única diferencia clave. En lugar de aplicar siempre el operador sobre la solución inicial, este método aplica el operador sobre la mejor solución encontrada hasta el momento. A medida que se encuentran soluciones más óptimas, estas se guardan y se actualizan. Luego, el operador se aplica sobre la solución más reciente y mejorada, asegurando así una mejora continua y eficiente en el proceso de optimización.

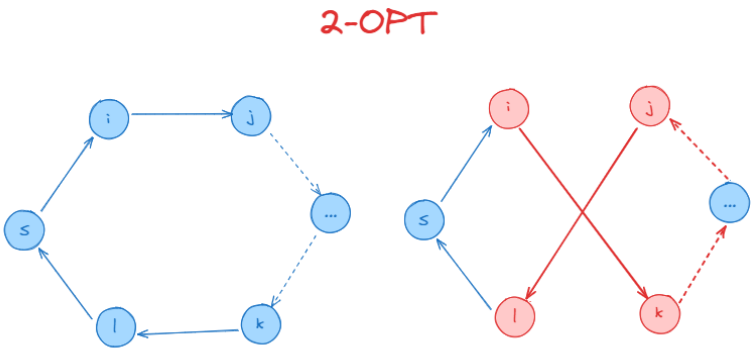

Figure 3: ilustración del operador: 2-opt

Por convención decidimos pasar por parametro, a cada operador, una respuesta generada por una de nuestras heuristicas para que, a la hora de realizar cualquier operador, se encuentre mas cerca del optimo global.

3 Ejercicio 3

En este ejercicio, implementamos la metaheurística ILS (Iterated Local Search). Esta técnica recibe la mejor solución generada por una de nuestras heurísticas y la perturba, es decir, modifica el orden de los nodos. En nuestro caso optamos primero por utilizar un operador de tipo swap para conseguirlo pero luego de experimentar con otros operadores (como se podra ver mas adelante) nos dimos cuenta que al usar relocate y 2opt mejorado obteniamos mejores óptimos pero 2opt mejorado tardaba mucho más en darnos una respuesta por lo que dejamos relocate finalmente. Posteriormente, se aplica nuevamente la misma heurística a la solución perturbada. Este proceso nos permite movernos entre diferentes vecindarios y valles de soluciones locales, con el objetivo de aproximarnos al óptimo global.

Para implementar ILS, desarrollamos dos enfoques. En el primer enfoque, se obtiene la solución inicial a partir de una de nuestras heurísticas y se guarda en una lista. Luego, se crea otra lista con 6 números aleatorios, que corresponden a las posiciones de los nodos que se intercambiarán. Estos números están en el rango de 0 al tamaño de la lista de solución menos 1. Con las posiciones seleccionadas aleatoriamente, se mezclan los nodos de la solución inicial, creando una solución perturbada. A continuación, se aplica nuevamente la heurística original a la solución perturbada. Se compara el costo de la nueva solución con el de la solución inicial y se guarda el menor costo. Este proceso se repite durante un número determinado de iteraciones.

En el segundo enfoque, se adapta la cantidad de posiciones a perturbar según el tamaño de la instancia. Para todas las muestras, se calcula el 6 porciento del tamaño de la instancia, y esta será la cantidad de nodos a intercambiar. Elegimos este porcentaje ya que luego de varias experimentaciones vimos que no se perturba tanto la solución inicial como para perder sus características originales, pero tampoco se perturba tan poco como para que el costo no varíe significativamente. Ambos enfoques nos permiten explorar diferentes vecindarios y mejorar la calidad de las soluciones, acercándonos así al óptimo global.

4 Experimentación:

Experimento 1: Efectividad de cada algoritmo

En este experimento quisimos evaluar la efectividad de los 4 algoritmos aproximados que realizamos y ver cuál era el que mejor se acercaba a las soluciones. Para esto calculamos la diferencia entre la mejor solución hallada hasta el momento para los archivos de *TSPLIB* y la encontrada por nuestras heuristicas golosas.

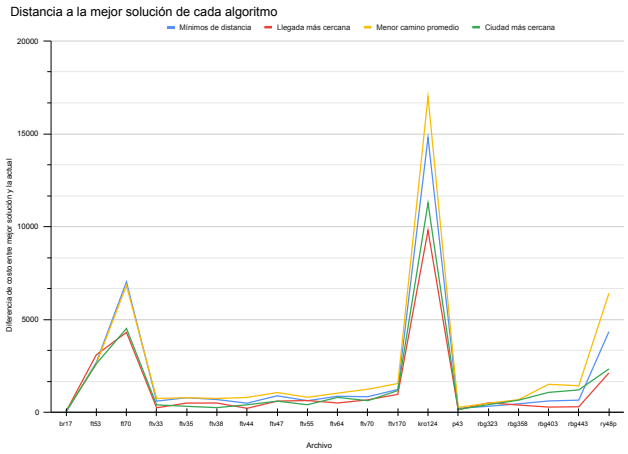

Figure 4: Aproximación de cada algoritmo al óptimo

Pudimos obsvervar que la cercanía de la solución al óptimo varía según la instancia. Es decir, que dependiendo el archivo va a funcionar mejor un algoritmo y para otras otro. Por ejemplo, en el archivo *br17* el algoritmo que mejor funciona es aquel que utiliza el *mínimo de distancias*, teniendo un costo de 42 mientras que el óptimo era de 39, una diferencia muy pequeña, con *ciudad más cercana* se obtiene 92, con *llegada más cercana* 94 de costo y *ciudad con menor distancia promedio* 101. Esto es curioso ya que el algoritmo de **mínimos de distancia** parecería ser aquel con mayor error junto con menor distancia promedio en el gráfico.

Sin embargo, por lo general, los algoritmos de ciudad más cercana y llegada más cercana parecerían ser aquellos con menor error según el gráfico. Para confirmar nuestra teoría y ver cual algoritmo era el que tenía menor error en general para todas las instancias de TSPLIB sumamos todas las diferencias entre el óptimo y la solución aproximada y luego realizamos un promedio para cada uno.

Algoritmo	**Promedio de diferencia**
Mínimos de distancia	2237.37
Menor camino promedio	2421.33
Ciudad más cercana	1719.95
Llegada más cercana	1533.37

Table 1: Lejanía del óptimo promedio por algoritmo para estas instancias

Pudimos observar que aquel algoritmo con menor error promedio es el de llegada más cercana. Sin embargo, llegamos a la conclusión de que si se quiere correr un archivo en especifico y no todas las instancias o muchas como en nuestro caso, deberíamos de utlizar aquel algoritmo goloso que funcione mejor para dicha instancia y no el que tiene menor error promedio.

Por otro lado, llegamos a la conclusión de que en el caso de las instancias de TSPLIB, la cantidad de ciudades

del archivo no afecta la distancia hacia el óptimo ya que, por ejemplo, en la instancia *kro124* que contiene 124 ciudades hay más error que en la *rbg443* que contiene 443 ciudades.

Experimento 2: Eficiencia de las búsquedas locales para cada algoritmo goloso

En este experimento quisimos ver cómo afectaban a cada respuesta de los algoritmos golosos propuestos en el ejercicio 1 nuestras búsquedas locales. Consideramos que, como la diferencia es muy pequeña, en algunos casos

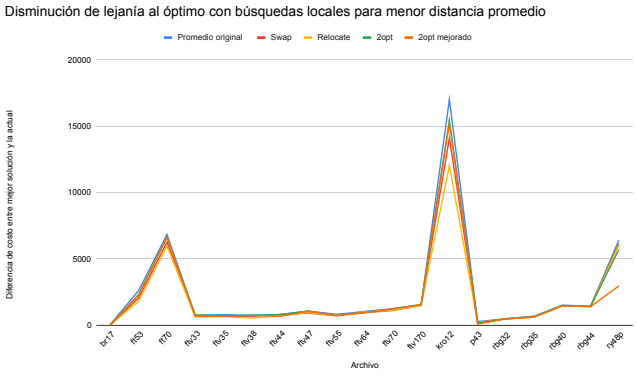

Figure 5: Búsqueda local + Menor promedio

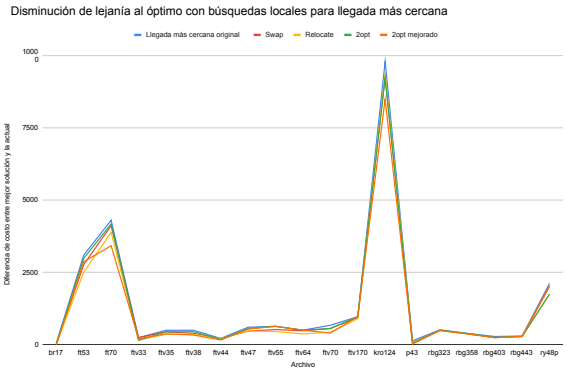

Figure 6: Búsquedas locales + Llegada más cercana

no se veía del todo bien en todos los gráficos, por eso pusimos solo 2 en el informe, pero experiementamos con los 4 algoritmos. Promediamos las diferencias de cada archivo testeado con su óptimo y pusimos los resultados en una tabla para sacar mejores conclusiones.

Table 2: Resultados de los algoritmos con diferentes técnicas

Algoritmo	**Swap**	**Relocate**	**2opt**	**2opt mejorado**
Ciudad más cercana	1468	1404	1457	**1364**
Llegada más cercana	1256	1192	1264	**1172**
Menor distancia promedio	2141	**1983**	2252	2064
Mínimos de distancia	1797	**1572**	1942	1813
Total promediado	1666	**1538**	1729	1603

En el caso del algortimo de ciudad más cercana y llegada más cercana parecería ser que el algoritmo que mejor aproxima el resultado es 2opt mejorado, mientras que para Menos distancia promedio y mínimos de distancia es Relocate.

Sin embargo, al hacer un promedio de las diferencias de cada algoritmo, en general, el que menor error tiene es el relocate. Entonces, a la hora de elegir un algortimo que mejore las soluciones de todas las golosas vamos a elegir aquel que tenga más beneficios es decir, el que tenga menor error promedio y, si hay un empate o son muy cercanos, el que tenga menor tiempo de ejecución, si lo hacemos con 1 en particular enonces utilizaríamos la que mejor aproxime al algoritmo goloso en especifico.

Experimento 3: tiempos de ejecución de los algoritmos de búsqueda local

Para poder llegar a la conclusión de cual de nuestros algoritmos de búsqueda local es mejor, no nos alcanzaba unicamente con ver cuál tenía el menor error promedio, también quisimos evaluar el costo de ejecución de cada uno

Figure 7: Tiempos de ejecución de cada algortimo goloso

En base a este gráfico pudimos llegar a dos colnclusiones, primero, que la dimensión de cada instancia afecta los tiempos de ejecución. A más ciudades más tarda en ejecutarse la búsqueda local.

Además, pudimos ver que aquellas dos búsquedas locales con menor tiempo de ejecución fueron *Swap* y *Relocate*. Como en el experiemento anterior vimos que el que **mejor aproxima es relocate**, llegamos a la conclusión de que es de las mejores búsquedas locales para nuestros algoritmos golosos e instancias de TSPLIB.

Experimento 4: Variación metaheuristica según cantidad de iteraciones. Como contamos en el punto 3 del informe, en un principio hicimos swap con 6 nodos y luego decidimos que era más correcto swapear una cantidad de nodos dependiendo del tamaño de la instancia, se swapean el 6 porciento de las ciudades. En este experimento quisimos ver cómo afecta la cantidad de iteraciones dadas a ambas versiones de la metaheristica.

Como utilizamos nodos random para swapear, decidimos que para graficar los resultados vamos a utilizar un promedio de los resultados de varias ejecuciones y luego ver su lejanía con el óptimo.

Lo corrimos con la solución del algoritmo de *llegada más cercana* para 8 instancias, 4 con la mayor cercanía al óptimo y 4 que más error tenían con las heuristicas golosas originalmente (Esto lo concluimos en base a los experimentos anteriores). Estos archivos fueron, por un lado, br17, p43, ftv33 y rbg323 con menos error y kro124, ft70, ft53 y ry48p con mayor. Nuestro objetivo era ver qué tanto mejor se aproximaban aquellas con una peor estimación y ver si con las que estaban cerca se podía llegar al óptimo.

En base a estos gráficos pudimos observar que a más iteraciones, más cerca se está de la mejor solución hasta el momento para ambas formas del algoritmo. Sin embargo, al correr los archivos nos dimos cuenta de que si bien 10.000 iteraciones de la metaheuristica da una respuesta mucho mejor que la original, el tiempo de ejecución es muy alto y más que nada para instancias muy grandes como rbg323.

Experimento 5: variación en cambio de vecindario con operadores de busqueda local con metaheuristica.

Para nuestra metaheuristica decidimos utilizar el operador relocate para mejorar la solución y luego un swap de la misma para cambiar de vecindario. En este experimento vamos a cambiar el operador que hace variar el vecindario de la solución, es decir el que la "rompe". Corrimos la metaheuristica con la solución de ciudad más cercana y con 100 iteraciones para 4 archivos, 2 eran aquellos con menor distancia al óptimo, p43 y rbg323 y los otros 2 aquellos con mayor error, ft70 y ry48p.

Testeamos la cercanía a la mejor solución de la respuesta generada por el swap original y por un relocate, corrimos el algoritmo varias veces y realizamos un promedio de la respuesta, luego miramos la diferencia entre el óptimo y la solución generada por la metaheuristica estos fueron los resultados: Llegamos a la conclusión de que el swap y

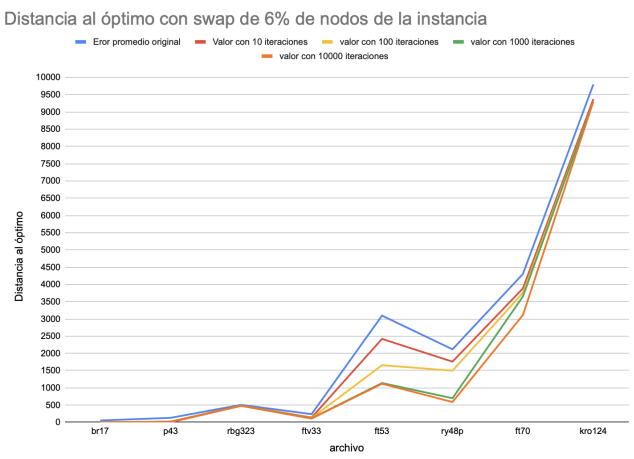

Figure 8: Variación en solución según iteraciones 1

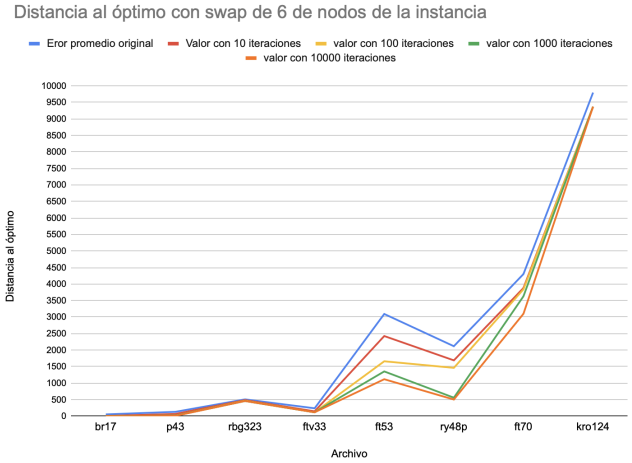

Figure 9: Variación en solución según iteraciones 2

Archivo	**Perturbador Swap**	**Perturbador Relocate**
p43	75	80
rbg323	408	396
ry48p	2107	2107
ft70	3945	3950

Table 3: Distancia a la mejor respuesta con cada perturbación

el relocate son casi igual de buenos para la perturbación de la solución. Según nuestros datos algunas veces anda mejor swap y otras relocate, pero debido al random podría tratarse de la aleatoriedad en la que se mueven los nodos o porque realmente en algunas instancias anda mejor un operador y en otras el otro.

Experimento 6: Cantidad de nodos intercambiados con swap y cercanía al optimo con metaheuristica:

En este experimento quisimos evaluar la eficiencia de ambas versiones de la metaheuristica. Por un lado, tenemos una que cambia de vecindario swapeando 6 nodos de la solución y por otro, otra que swapea el 6% de las ciudades o en el caso de que ese 6% sea menor a 2, intercambia 2 ciudades. Utilizamos la respuesta de la heuristica llegada más cercana ya que era la que mejor funcionaba en general para los archivos de TSPLIB.

Como siempre, debido al random, corrimos todo varias veces e hicimos un promedio de los resultados, luego calculamos su distancia con la mejor respuesta Corrimos los mismos archivos del experimento 4, menos kro124 e hicimos una tabla con nuestros resultados.

Table 4: Promedio porcentajes

Archivo	Error promedio original	Valor con 10 iteraciones	Valor con 100 iteraciones	Valor con 1000 iteraciones	Valor con 10000 iteraciones
br17	55	5	2	0	0
p43	132	22	11	11	5
rbg323	509	487	487	480	480
ftv33	239	140	125	118	110
ft53	3094	2420	1658	1141	1119
ry48p	2118	1760	1498	700	590
ft70	4302	3883	3754	3650	3111

Table 5: Swap 6 nodos

Archivo	Error promedio original	Valor con 10 iteraciones	Valor con 100 iteraciones	Valor con 1000 iteraciones	Valor con 10000 iteraciones
br17	55	7	4	2	1
p43	132	68	21	17	11
rbg323	509	487	470	460	460
ftv33	239	147	118	118	118
ft53	3094	2426	1662	1355	1119
ry48p	2118	1690	1463	560	504
ft70	4302	3880	3840	3630	3100

Pudimos observar que en algunos casos funcionaba mejor una versión y en otros la otra, entonces investgamos a qué se podía deber.

Pequeñas Perturbaciones: Si la cantidad de nodos intercambiados es pequeña, la perturbación será leve, y la solución perturbada estará cerca de la solución actual. Esto puede ser útil para una exploración fina en el vecindario de una buena solución.

Grandes Perturbaciones: Si la cantidad de nodos intercambiados es grande, la perturbación será significativa, y la solución perturbada estará más lejos de la solución actual. Esto puede ayudar a escapar de óptimos locales, pero también puede llevar a soluciones menos óptimas en general.

Tamaño del Problema: Para problemas con pocos nodos, una perturbación pequeña puede ser suficiente. Para problemas con muchos nodos, puede ser necesario aumentar la cantidad de nodos intercambiados para explorar adecuadamente el espacio de soluciones.

En este caso pudimos ver que el algoritmo de porcentaje anda mejor para las instancias de p43, br17, ftv33 y ftv55 (en algunos casos) funcionaban mejor con sus respectivos promedios, en todos los casos se swapean solo 2 nodos, es decir que como son instancias chiquitas, pequeñas perturbaciones les bastan para funcionar mejor. En los otros casos ambos algoritmos funcional casi igual y en rbg323 funciona mejor el swap de 6 nodos, esto se puede deber a que el promedio de nodos intercambiados es muy alto y me aleja de la mejor solución.

5 Conclusión

Este trabajo práctico nos permitió comprender la utilidad de diversas heurísticas aplicadas a distintos conjuntos de datos. Descubrimos que, a menudo, el óptimo deseado no se alcanzaba con un solo algoritmo goloso, pero sí con otros. Además, comprendimos mejor el funcionamiento e implementación de varias búsquedas locales, en particular 2-opt, que nos presentó desafíos debido a las numerosas restricciones necesarias para su correcto funcionamiento. Nos llamó la atención cómo, en ciertas situaciones, la búsqueda local más compleja no siempre devolvía la mejor solución. Este proceso nos brindó una visión más profunda de la complejidad de implementar un algoritmo capaz de resolver cualquier caso de manera óptima en tiempo polinomial. Por último, cabe mencionar que, como sabemos, el modelo planteado es un problema NP-Hard, por lo que decidimos implementar heurísticas que lo resuelvan en un tiempo eficiente y con buenas soluciones. Ahora bien, ¿esto quiere decir que siempre son buenas? Gracias a nuestras experimentaciones, logramos entender que en algunas instancias nuestras heurísticas encuentran óptimos buenos, mientras que en otras no funcionan tan bien, a pesar de ser polinomiales en tiempo de ejecución. Podemos concluir que la implementación de heurísticas es un buen método para resolver problemas muy complejos, pero es importante tener presente que no siempre es la mejor solución para todas las instancias del problema.