Comparación de algoritmos A* y GA en la resolución del TSP

*Marco Quintana García[UO287872], Martín Cancio Barrera[UO287561], and Miguel Méndez Murias[UO287687]

Sistemas Inteligentes. Grado en Ingeniería Informática. EII. Universidad de Oviedo. Campus de los Catalanes. Oviedo

Abstract. El Problema del Viajante (TSP) es un problema clásico de optimización combinatoria que busca la ruta más corta para visitar un conjunto de ciudades. Este trabajo explora dos enfoques para resolver el TSP: el algoritmo de búsqueda A* y los algoritmos genéticos. Mientras que A* garantiza soluciones óptimas, su alto costo computacional lo limita en problemas grandes. Los algoritmos genéticos, por su parte, ofrecen soluciones subóptimas en menos tiempo, siendo más eficientes para instancias grandes. Se comparará ambos métodos en términos de precisión y tiempo de ejecución, destacando sus ventajas y limitaciones.

Keywords: A* · Algoritmos genéticos · Búsqueda heurística · Heurístico · Inteligencia Artificial · Optimización · TSP

1 Introducción

El Problema del Viajante de Comercio (TSP)[2] es uno de los problemas más estudiados en el ámbito de la optimización combinatoria, cuyo objetivo es determinar la ruta más corta para que un viajante visite todas las ciudades de un conjunto exactamente una vez, retornando al punto de origen.

Este trabajo presenta dos enfoques para abordar el TSP: el algoritmo de búsqueda A*[1] y los algoritmos genéticos[3]. El algoritmo de búsqueda A* garantiza la solución óptima en caso de que se utilicen heurísticos admisibles, pero ello puede conllevar un uso intensivo de la memoria y una gran cantidad de tiempo para instancias del problema de gran tamaño. Por otro lado, los algoritmos genéticos (GA), una técnica de optimización basada en la evolución biológica, permite encontrar soluciones aproximadas en menos tiempo, siendo particularmente eficiente en instancias del problema en las que no resulte práctico obtener una solución exacta por su elevado coste.

El resto del documento se organiza de la siguiente manera:

- **Sección 2:** se describe el TSP y su relevancia
- **Sección 3:** introducción a los algoritmos utilizados (A* y algoritmos genéticos)

* Los autores están ordenados por orden alfabético

- **Sección 4:** implementación de los algoritmos en el contexto del TSP
- **Sección 5:** estudio experimental y análisis de resultados
- **Sección 6:** conclusiones del trabajo

2 El problema del viajante de comercio

El Problema del Viajante de Comercio (TSP)[2] es un problema combinatorio con aplicaciones en diversos campos, desde la logística y la planificación de rutas, hasta la bioinformática y el diseño de circuitos.

Este problema consiste en, dado un conjunto finito de ciudades y las distancias entre cada par de ellas, determinar un camino hamiltoniano que conecte todas las ciudades, de manera que se recorra la menor distancia posible.

Dado que la complejidad del problema aumenta de manera exponencial con el número de ciudades (lo que lo clasifica como un problema *NP-hard*), encontrar la solución óptima se vuelve impracticable para instancias grandes utilizando métodos exactos, ya que el tiempo necesario para resolverlo crece rápidamente. Por ello, en estos casos, se suelen emplear aproximaciones o algoritmos que buscan soluciones lo suficientemente buenas en lugar de la mejor posible.

3 Los algoritmos de búsqueda

3.1 Algoritmo A*

A*[1] es un algoritmo de búsqueda informado que extiende el algoritmo Best-First Search utilizando la función de evaluación

$$f(n) = g(n) + h(n)$$

donde $g(n)$ es el coste del camino desde el nodo inicial hasta n, y $h(n)$ es una estimación heurística del coste restante desde n al nodo objetivo. Esta combinación permite que A* elija primero los nodos que parecen más prometedores, lo que significa que se enfoca en las rutas que, con mayor probabilidad, lo llevarán más rápido al objetivo.

Una de las propiedades más destacadas de A* es la **admisibilidad**, que se cumple si el heurístico $h(n)$ nunca sobreestima el costo real para alcanzar el objetivo, lo que garantiza que el algoritmo encontrará siempre la solución óptima si existe. Además, el heurístico h es **consistente** si, para cada par de nodos n y n', la estimación satisface la relación

$$h(n) \leq c(n, n') + h(n')$$

donde $c(n, n')$ es el costo de mover de n a n'. Si el heurístico cumple esta propiedad también será admisible, pero al contrario no ocurre lo mismo. Otra propiedad importante de los heurísticos es la **dominancia**, la cual indica que, dados dos heurísticos h_1 y h_2, h_2 está mejor informado que h_1 si se cumple la relación

$$h_1(n) < h_2(n) \leq h^*(n)$$

siendo h^* el heurístico óptimo. Además, esto también implica que todo nodo expandido por $A^*(h_2)$ también es expandido por $A^*(h_1)$.

3.2 Algoritmos ε-admisibles

Estos algoritmos encuentran, con seguridad, una solución cuyo coste no excede $(1 + \varepsilon)C^*$[5][6], donde C^* es el coste óptimo de la solución. En este caso, nos centraremos en el algoritmo de Ponderación Estática (PEA*), el cual utiliza la siguiente función de evaluación (En todos los casos, h debe ser un heurístico admisible):

$$f(n) = g(n) + (1 + \varepsilon)h(n), \quad \varepsilon > 0$$

Propiedades de los algoritmos ε-admisibles:

- **Garantía de calidad**: La solución obtenida por un algoritmo ε-admisible tiene un coste que no supera $(1 + \varepsilon)C^*$. Esto asegura que la solución es cercana a la óptima, manteniendo un límite en el error.
- **Velocidad de convergencia**: Dado que se da mayor peso al heurístico $h(n)$, el algoritmo explora primero las rutas que parecen más prometedoras en base a la estimación heurística, lo cual acelera la búsqueda.
- **Aplicación de heurísticos admisibles**: El heurístico h debe ser admisible, es decir, nunca debe sobreestimar el coste real hasta el objetivo. Esto garantiza que el algoritmo mantiene la condición de ε-admisibilidad y se aproxima a una solución de calidad controlada.
- **Ajuste de** ε: El parámetro ε permite ajustar la eficiencia de la búsqueda. Valores mayores de ε resultan en búsquedas más rápidas, aunque con un compromiso en cuanto a la cercanía al coste óptimo.

Los algoritmos ε-admisibles, como el PEA*, son ideales para escenarios donde el equilibrio entre velocidad y precisión es esencial, como en aplicaciones de planificación en tiempo real, en sistemas de navegación y en problemas de optimización aproximada.

3.3 Algoritmos genéticos (GA)

Los algoritmos genéticos[3] son algoritmos evolutivos que se basan en la evolución biológica, más concretamente en la selección natural y en la herencia genética, y son ampliamente utilizados para resolver problemas de optimización. Los componentes principales de estos algoritmos son:

- **Esquema de codificación**: es la forma en la que se representan las soluciones del problema (individuos) en el espacio de búsqueda
- **Población inicial**: es el conjunto inicial de posibles soluciones al problema y puede ser creada aleatoriamente o mediante el uso de heurísticos
- **Función de evaluación (*fitness*)**: permite evaluar la calidad de los cromosomas

- **Conjunto de operadores genéticos**: seleción, cruce, mutación o reemplazo
- **Parámetros configurables**: tamaño de la población, número de generaciones (iteracciones), tasa de mutación o cruce, etc.

3.4 Búsqueda local

Los algoritmos de búsqueda local[4] son técnicas de optimización que exploran soluciones en un espacio de búsqueda iterativamente, enfocándose en soluciones cercanas a una solución actual. Estos algoritmos generan "vecinos" de la solución actual mediante pequeñas modificaciones y seleccionan la mejor opción entre ellos. Su principal ventaja es la rapidez y simplicidad, aunque no garantizan la obtención de la solución óptima global y a menudo se quedan atrapados en óptimos locales.

4 Aplicación de los algoritmos al problema del viajante de comercio

Para aplicar A*[1] al TSP[2], se emplean heurísticos como el camino mínimo entre las ciudades restantes o la solución del TSP[2] relajado (h_NN y h_BI), que no siempre son admisibles pero ofrecen buenos resultados. En el caso del GA, se utilizan operadores de cruzamiento basados en subsecuencias comunes entre padres y una mutación basada en inversiones de segmentos.

4.1 Algoritmo A*

Para aplicar A*[1] al TSP[2], se emplean diferentes heurísticos que proporcionan estimaciones del coste restante. A continuación, se describen los heurísticos utilizados:

- **Heurístico Pobre:** Este heurístico cuenta el número de ciudades que faltan por visitar, calculando $N - k + 1$, donde N es el total de ciudades y k es el número de ciudades visitadas. Es un heurístico poco informado y no proporciona información sobre los costes de las conexiones.
- **Heurístico 1 (h1):** Considera los arcos mínimos de las ciudades que aún no han sido visitadas. Calcula el coste mínimo de viajar a cada ciudad no visitada desde la ciudad actual y suma estos valores. Aunque no está basado en una relajación formal, ofrece una mejor aproximación que el heurístico pobre.
- **Heurístico 2 (h2):** Este heurístico es una relajación que calcula la suma de los $N - k + 1$ arcos de menor coste del grafo residual, donde k es el número de ciudades visitadas. Proporciona una estimación más ajustada del coste total del camino.

- **Heurístico de Árbol de Expansión Mínimo (h_MST):** Calcula el coste de un árbol de expansión mínimo del grafo residual utilizando el algoritmo de Kruskal. Este enfoque proporciona una buena estimación del coste mínimo para visitar las ciudades restantes y retornar a la ciudad inicial.
- **Heurístico Húngaro (h_HUNGARO):** Se basa en la asignación mínima entre las ciudades no visitadas y la ciudad actual, utilizando el algoritmo húngaro. Este heurístico estima el coste mínimo de asignar ciudades no visitadas, lo que lo convierte en una buena aproximación del coste óptimo del problema relajado.
- **Heurístico del Vecino Más Cercano (h_NN):** Implementa la estrategia de "Nearest Neighbor", donde se elige la ciudad más cercana a la actual en cada paso. Este heurístico suma las distancias entre la ciudad actual y el vecino más cercano, y cierra el ciclo al regresar a la ciudad inicial.
- **Heurístico de Mejor Inserción (h_BI):** Utiliza la estrategia de "Best Insertion", que busca insertar las ciudades no visitadas en el camino actual en la posición que minimiza el coste total. Este método ajusta el recorrido dinámicamente para encontrar un coste heurístico eficiente.

Los heurísticos **h_NN** y **h_BI** son ejemplos de heurísticos no admisibles, lo que significa que pueden sobreestimar el costo real hasta el objetivo. Esto puede llevar a resultados más rápidos, aunque con menor precisión en la calidad de la solución óptima, ya que no garantizan la admisibilidad. Sin embargo, una estrategia para mejorar la precisión de estos heurísticos no admisibles es ponderarlos a la baja multiplicándolos por un factor $(1 - \varepsilon)$, donde $0 \leq \varepsilon \leq 1$. De esta forma, se reduce la sobreestimación del costo, acercándolos al comportamiento de un heurístico admisible y manteniendo cierto control sobre el error. La selección de ε permite ajustar el balance entre velocidad de búsqueda y precisión de la solución.

4.2 Algoritmos genéticos (GA)

El algoritmo genético[3] para resolver el TSP[2] comienza con un esquema de codificación basado en permutaciones de las ciudades, donde cada individuo representa un recorrido único que pasa por todas las ciudades una sola vez. Se implementan tres operadores de cruce:

- **Random crossover:** Este operador genera un hijo al azar tomando una permutación aleatoria de las ciudades del primer padre.
- **Uniform crossover:** Este operador copia aleatoriamente una fracción de las ciudades del primer padre en las mismas posiciones del hijo, completando el resto con las ciudades del segundo padre en el orden en que aparecen.
- **Order crossover:** Este operador selecciona un segmento de ciudades del primer padre en un rango específico de posiciones y las inserta en el hijo, mientras que las posiciones restantes se llenan con las ciudades del segundo padre en su orden relativo.

– **Extreme crossover:** Este operador intenta recrear el camino desde el principio del primer padre y desde el final del segundo padre, alternando las inserciones hasta que no se pueden añadir más ciudades únicas. Una vez que se han insertado todas las ciudades posibles de esta manera, los espacios vacíos en el hijo se rellenan con las ciudades restantes, que no hayan sido usadas todavía.

A este algoritmo se han añadido dos mejoras para incrementar su rendimiento: por un lado, se ha optimizado la población inicial mediante la inclusión de varios cromosomas con "buen material genético", es decir, individuos generados utilizando heurísticos específicas o soluciones aproximadas que proporcionen rutas de bajo coste desde el principio. En particular, se generan N cromosomas, uno por cada ciudad, aplicando la estrategia de *Nearest Neighbor* (NN), en la cual cada recorrido comienza desde una ciudad diferente y selecciona iterativamente la ciudad más cercana como siguiente destino. Esto permite que la población inicial esté compuesta por soluciones razonablemente buenas que guían la evolución hacia soluciones más óptimas desde las primeras generaciones.

Por otra parte, se propone la inclusión de un elemento nuevo: la búsqueda local[4]. Para ello, se considera una estrategia de vecindad en la que, para generar los vecinos de un cromosoma, se selecciona una posición aleatoria en el cromosoma y se generan $2x$ vecinos (con $x \geq 1$) intercambiando la ciudad en dicha posición con las ciudades en las posiciones cercanas, considerando el cromosoma como una estructura circular. El criterio de selección será el mejor vecino, siempre que mejore al cromosoma actual. Esta búsqueda local se aplicará después del operador de mutación con una probabilidad $pLSA$, lo que añade una capa adicional de optimización local para refinar las soluciones generadas por el algoritmo genético[3]. Para la búsqueda local de este estudio experimental se ha usado pLSA = 0.1 y x = N/2, siendo N la longitud de la solución.

Con estas dos mejoras se espera que el algoritmo genético converja más rápidamente hacia soluciones de mejor calidad, aprovechando tanto la diversidad global de soluciones en la población como el refinamiento de las mejores soluciones a través de la búsqueda local.

5 Estudio experimental

5.1 Diseño del estudio experimental

Para comparar el rendimiento de ambos algoritmos, se resolvieron varias instancias del TSP[2] de diferentes tamaños (17, 21 ciudades) utilizando implementaciones de A*[1] y GA[3] en Python (aima-python con modificaciones). Se midieron los tiempos de ejecución y la calidad de las soluciones obtenidas.

Los resultados obtenidos se han medido en un equipo con las siguientes especificaciones

– **Procesador** Intel(R) Core(TM) i5-10400F CPU @ 2.90GHz 2.90 GHz (6 núcleos / 12 hilos)
– **Memoria RAM** 16,0 GB

5.2 Resultados experimentales

Los resultados experimentales muestran que A*[1] es capaz de encontrar la solución óptima para instancias pequeñas (17 y 21 ciudades), pero su tiempo de ejecución crece exponencialmente con el tamaño del problema. Por otro lado, los algoritmos genéticos[3] encuentran soluciones cercanas a la óptima en tiempos significativamente menores, siendo más adecuados para instancias grandes.

Table 1. Comparativa de resultados para diferentes heurísticos de A* en distintas instancias del TSP.

Instancia	Heurístico	Nodos expandidos	Nodos reexpandidos	Tiempo (s)	Coste del camino
gr17	h_pobre	524290	0	48.612	2085
	h_1	321362	0	39.389	2085
	h_2	248137	0	82.757	2085
	h_MST	30413	0	16.560	2085
	h_Húngaro	14378	428	431.099	2085
	PEA*	2709	276	2.693	2191
	h_NN	18	0	0.003	2149
	h_BI	18	0	0.068	2090
gr21	h_pobre	-	-	-	-
	h_1	591821	0	131.879	2707
	h_2	561594	0	460.048	2707
	h_MST	7202	0	13.361	2707
	h_Húngaro	2870	10	259.930	2707
	PEA*	438	0	0.246	2949
	h_NN	22	0	0.005	2872
	h_BI	22	0	0.177	2943
gr48	PEA*	10303	1211	52.824	5514
	h_NN	49	0	0.07	5581
	h_BI	124	0	3.896	5317

La Tabla 1 compara diferentes heurísticos aplicados en el algoritmo A*[1] para resolver el problema del viajante (*TSP*) en distintas instancias de ciudades (17, 21, y 48 ciudades). Los resultados se miden en términos de:

- **Nodos expandidos y reexpandidos**: Indica el número de nodos que A* ha explorado y vuelto a explorar durante la búsqueda, reflejando la eficiencia del heurístico en reducir el espacio de búsqueda.
- **Tiempo (s)**: El tiempo total que tomó cada heurístico en segundos para encontrar la solución. Esto muestra cómo varía el rendimiento del algoritmo A* según la complejidad del heurístico.
- **Coste del camino**: Representa la longitud total del recorrido encontrado, donde valores más bajos indican rutas más cortas, cercanas a la solución óptima.

Los heurísticos básicos (`h_pobre`, `h_1`, `h_2`) tienen tiempos de ejecución altos y expanden muchos nodos, indicando menor eficiencia y precisión, mientras

que los heurísticos avanzados (`h_MST`, `h_Húngaro`) reducen considerablemente el número de nodos expandidos y, en el caso del `h_MST` también se reduce en gran medida el tiempo necesario para obtener la solución.

Los heurísticos anteriores no pueden usarse con grandes instancias del problema porque el uso de memoria y de tiempo es muy grande. Para poder obtener una solución en estos casos utilizamos la ponderación estática (`PEA*`) aplicacada a uno de los heurísticos anteriores (en la tabla se muestran los resultados de aplicarlo con el heurístico `h_MST` con un peso de 1.4).

Los heurísticos no admisibles(`h_NN`, `h_BI`) son los más rápidos y expanden muy pocos nodos, pero no nos garantizan encontrar la solución óptima.

Table 2. Comparativa de resultados para los heurísticos h_NN y h_BI de A*[1] ponderados a la baja con distintos valores de épsilon y distintas instancias del TSP.

Instancia	**Heurístico**	ϵ	**Nodos expandidos**	**Nodos reexpandidos**	**Tiempo (s)**	**Coste del camino**
gr17	h_NN	0.1	80	0	0.01	2085
		0.15	192	2	0.036	2085
		0.2	900	91	0.114	2085
	h_BI	0.1	270	0	0.224	2085
		0.15	3402	65	2.492	2085
		0.2	12979	397	7.904	2085
gr21	h_NN	0.1	55	22	0.011	2851
		0.15	354	36	0.108	2835
		0.2	1451	221	0.370	2760
	h_BI	0.1	149	1	0.485	2760
		0.15	605	21	1.581	2758
		0.2	2869	76	5.924	2707

La Tabla 2 muestra el efecto de ajustar los heurísticos `h_NN` y `h_BI` de A*[1] con distintos valores de ϵ para el problema del viajante (TSP)[2]. Multiplicar el heurístico por $(1 - \epsilon)$ reduce su valor, es decir, lo "pondera a la baja". Esto significa que el heurístico calcula un coste menor, lo que lleva a A* a explorar más nodos y caminos para asegurar la mejor solución. A medida que ϵ aumenta, el coste estimado es aún más bajo, haciendo que A* explore más nodos. Esto incrementa el tiempo de ejecución, pero también reduce el coste del camino final, mejorando la calidad de la solución.

La Figura 1 muestra la evolución del mejor coste que se obtiene en cada generación con cada operador de cruce. Se puede observar que el coste mejora según aumenta el número de generaciones desde la población inicial, aunque en función del operador escogido las diferencias son notables. El operador `random crossover` se queda muy lejos del resto en la búsqueda de la solución óptima, mientras que el `order crossover` es el que mejor rendimiento muestra. También se puede observar que el `uniform crossover` presenta una convergencia prematura, ya que llega rápido a un punto estable y apenas mejora con el paso de las generaciones.

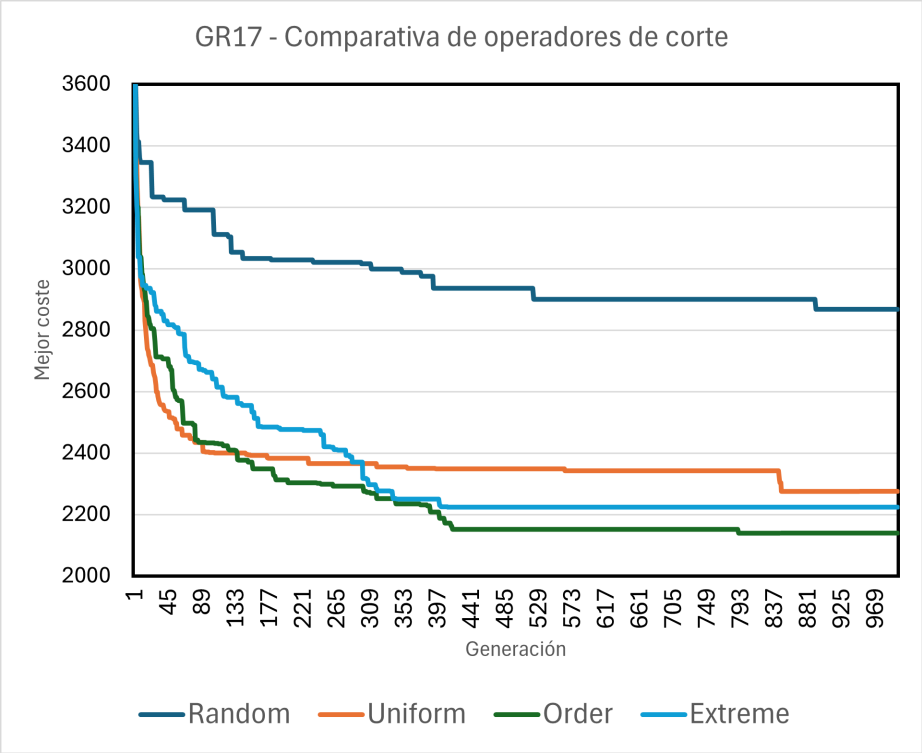

Fig. 1. Comparativa de diferentes operadores de cruce con la instancia del problema gr17

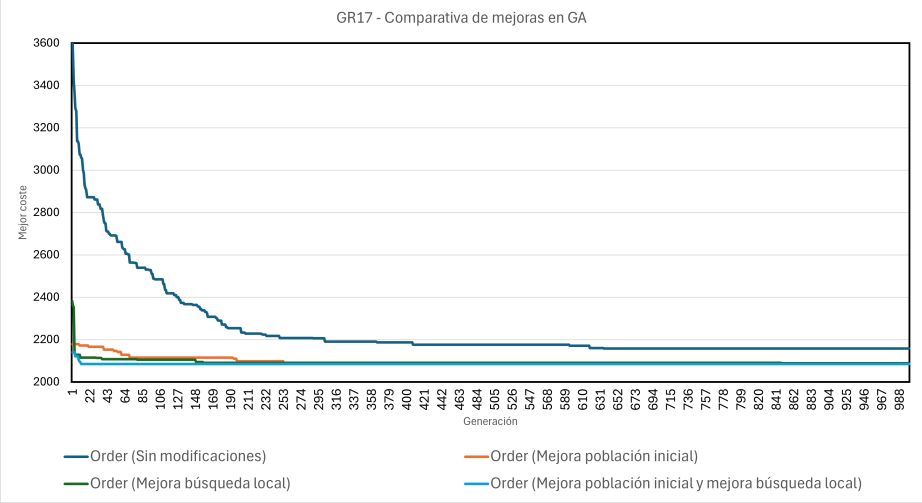

Fig. 2. Comparativa de mejoras en el algoritmo genético

La Figura 2 muestra la evolución del mejor que se obtiene en cada generación utilizando el operador de cruce `order crossover` con y sin mejoras. Se puede observar que si no aplicamos ninguna mejora el operador tarda mucho más en converger y en última instancia, la solución que encuentra es peor. Por otra parte, si se aplica la mejora con búsqueda local[4] el algoritmo genético converge muy rápidamente a una buena solución. La aplicación de la mejora de la población inicial también permite obtener una mejor solución, aunque el algoritmo necesita un mayor número de generaciones para converger al no utilizar máximos locales. En la figura también se puede ver que si combinamos ambas técnicas se consigue una solución buena y con un número muy bajo de generaciones para obtenerla.

6 Conclusiones

Este trabajo presenta una comparación entre el algoritmo de búsqueda A*[1] y los algoritmos genéticos para la resolución del TSP[2]. Mientras que A* ofrece soluciones óptimas, su ineficiencia en problemas grandes lo hace impracticable para aplicaciones reales. Los algoritmos genéticos[3], aunque no garantizan la optimalidad, proporcionan soluciones razonablemente buenas en tiempos mucho más cortos, lo que los hace más adecuados para problemas de gran tamaño.

Cabe mencionar que los algoritmos `h_NN` y `h_BI`, a pesar de no ser admisibles, presentan un rendimiento significativamente superior al de los demás heurísticos, y expanden un número mucho menor de nodos para las mismas instancias del problema.

Por otra parte, la implementación de las optimizaciones tanto de mejora de población inicial superior en comparación con la versión básica del algoritmo genético. La estrategia de inicialización basada en el vecino más cercano (`h_NN`) y la búsqueda local[4] con selección del mejor vecino permiten explorar soluciones de manera más eficiente, incrementando la probabilidad de obtener rutas de menor costo..

References

1. Hart, P. E., Nilsson, N. J., Raphael, B.: A formal basis for the heuristic determination of minimum cost paths. IEEE transactions on Systems Science and Cybernetics, 4(2), 100-107 (1968).
2. Gerhard R.: The Traveling Salesman: Computational Solutions for TSP Applications. Springer (1994)
3. Burke EK, Hyde MR, Kendall G, Ochoa G, O¨ zcan E, Woodward JR (2019) A classification of hyper-heuristic approaches: revisited. In: Gendreau M, Potvin JY (eds) Handbook of metaheuristics. International series in operations research & management science vol 272, pp 453–477
4. Aarts, E. H. L., Lenstra, J. K. (Eds.). (1997). Local search in combinatorial optimization (Wiley Series in Discrete Mathematics & Optimization). Wiley.
5. J. Pearl. Heuristics: Intelligent Search Strategies for Computer Problem Solving. Addison-Wesley, Reading, MA, 1984.
6. Pearl J, Kim JH. Studies in semi-admissible heuristics. IEEE Trans Pattern Anal Mach Intell. 1982 Apr;4(4):392-9