Experimental Study of Search Algorithms in the Traveling Salesman Problen: A* & Genetic Algorithms

[1]Santiago Fernández Carballal (UO283523), Miguel González Navarro (UO282337), Mario Pérez Fernández (UO283720). Intelligent Systems. Degree in Computer Engineering. EII. University of Oviedo. Campus de los Catalanes.

Abstract. In this article, the aima-python project will be used to address the Traveling Salesman Problem (TSP) using heuristic search algorithms, specifically A* and genetic algorithms. A detailed experimental study will be done to evaluate the performance of these algorithms in solving the TSP. The main results reveal significant comparisons in terms of efficiency and precisions between the different methods, which provides valuable information for the optimization of route problems. The genetic algorithms proved their effectiveness when addressing large-scale TSP, while A* algorithms stood out when resolving smaller instances.

Keywords: TSP (Traveling Salesman Problem), A*, Genetic Algorithms, heuristic search, PEA*, state space search, experimental study, Hamiltonian cycle.

1 Introduction

Route optimization is a fundamental challenge in artificial intelligence and computational theory research. One of the most iconic problems in this field is the Traveling Salesman Problem (TSP), which consists of finding the shortest route that visits a set of locations and returns to the starting point. The combinatorial complexity of the TSP has made it a problem of continuos interest, which has led to the development of several resolution strategies. Among them, heuristic search algorithms stand out.

The main objective of this work is to carry out a detailed experimental study to evaluate the efficacy of two heuristic search approaches, A* and genetic algorithms, in solving the TSP. Through a series of experiments, the performance of these algorithms with different instances of the TSP will be analyses. The goal is to prove their advantages, limitations and applicability in real situations.

2 Traveling Salesman Problem (TSP)

The *Traveling Salesman Problem*, abbreviated to TSP, consists in finding the shortest route that connects a set of cities, with known distances between each pair of cities (see Fig. 1). This route must cover all cities exactly once and return to the city of origin. More simply, it can be described as: “If a traveler starts from one city and knows the

[1] Authors appear in the alphabetical order of the last name.

distances between the different cities he must visit, which is the optimal route needed to travel through all the cities once and return to the starting city?"

The TSP has applications in different areas, such as logistics, route planning, printed circuit board design, computational biology and more. In computational biology, it can model problems related to DNA sequencing. The TSP consists in determining a route of minimum cost. To clarify it, the figure below shows a graph with 8 vertices, highlighting a Hamiltonian cycle.

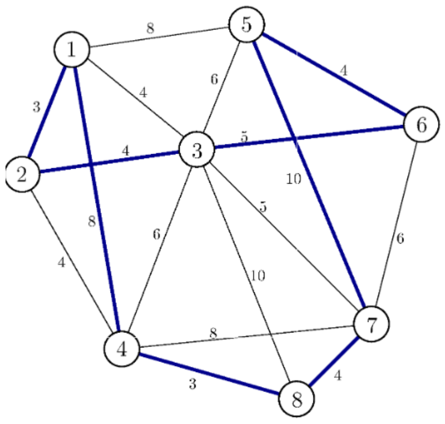

Fig. 1. Representation of an instance of the TSP [4].

The optimal solution is highlighted in dark blue, with a total cost of 41 (8+3+4+10+4+5+4+3).

3 Search Algorithms

This section provides a comprehensive description of the search algorithms used in this study, highlighting their most significant features and referring to key literature for a deeper understanding. Although we will not include the pseudocode of the algorithms, we will explain their function in the text.

3.1 A* Algorithm

The A* algorithm is a search algorithm renowned for its efficiency in solving problems in pathfinding. A* uses a combination of the Dijkstra algorithm's breadth-first search and the Greedy Best-First Search heuristic approach. Its properties are: admissibility, dominance and consistency. Its unique characteristic is its ability to provide an optimal solution by considering both the cost incurred to reach a particular state and an admissible heuristic estimate of the remaining cost [5].

A* presents a fundamental characteristic that involves the definition of the following concepts:

- $g^*(n)$: Indicates the cost of the shortest path from the initial node to node n.

- $h^*(n)$: Represents the cost of the shortest path from node n to the target node.
- $f^*(n)$: Represents the cost of the shortest path from the initial node to the goal nodes, ensuring that it passes through node n. Consequently, $f^*(n)$ equals the sum of $g^*(n)$ and $h^*(n)$.
- $C^* = f^*(\text{initial}) = h^*(\text{initial})$

In problems of a certain complexity, computing these values in a reasonable amount of time may be impractical. Therefore, we resort to approximations of $g^*(n)$ and $h^*(n)$:
A* = BF with $f(n) = g(n) + h(n)$
- $f(n)$ is an estimation of $f^*(n)$
- $g(n)$ = best cost of initial to n so far. A variable of A* (n.PATH_COST)
- $h(n)$ = positive estimate of $h^*(n)$, with $h(n) = 0$, if n is goal. It is the HEURISTIC and must be defined using information about the problem, in particular about the state n.

3.2 PEA* Algorithm

The PEA* (Partial Expansion A*) algorithm is a variant of the A* algorithm that optimizes the search by expanding only part of the search tree, which significantly reduces the computational complexity. It is used for large state space search problems, where A* may be computationally expensive. It’s objective is to approach $h^*(n)$ and it is not admissible.

The main difference between this variant and A* is in the evaluation function used: $f(n) = g(n) + (1+ \mathcal{E})\, h(n)$, $\mathcal{E} \geq 0$ O $f(n) = \mathcal{E}\, g(n) + (1- \mathcal{E})\, h(n)$, $0 < \mathcal{E} \leq 0.5$

4 Application of algorithms to the traveling salesman problem

4.1 Resolution with state space search

In the TSP, the search space is defined by all the possible permutations of cities not visited yet (see Fig. 2). A state consists of: initial city (A), set of visited cities and the actual city. States can be modeled by means of a tree or a graph, although the best option will be a graph. That is because using a tree means facing an exponential number of goal states (whereas in this case there is only one).

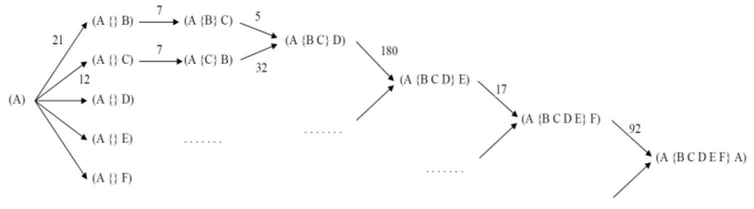

Fig. 2. Search space of the TSP using a graph.

For the TSP, a common heuristic is the computation of a minimum spanning tree, or MST. It conncets not visited cities and adds the distances to these cities to the current state. The heuristic must be addmisible, which means that it never overestimates the real cost of reaching the goal state. In the TSP this means that the MST must be a subset of the optimal route.

The following heuristics will be used to solve the TSP [7]:

H1: considers the minimum arcs of the cities that are still to be abandoned. It consists in calculating a lower bound of the problem. The cities that were not abandoned yet will be the set of not visited cities and the current one.
H2: Relaxation of R2, R3, R4. Sum of the N-k+1 least cost arcs of the residual graph.
H3: Relaxation of R3, R4. Sum of the N-k+1 least cost arcs of the residual graph such that for each city in the residual graph there is an arc touching it.
H_MST: Relaxation of R3. Cost of an MST of the residual graph. The MST can be calculated with Kruskal's or Prim's algorithm.

The problem's restrictions are the following: **R1:** That it has N-k+1 arcs. **R2:** That the arcs touch the cities A, X and the unvisited ones. **R3:** That the set of arcs has degree 1 for A y X, and degree 2 for the unvisited. **R4:** That the arcs connect every city A, X the unvisited with each other.

4.2 Resolution with genetic algorithms

For coding a solution is represented as a sequence of cities to be visited (a chromosome). Each gene in the chromosome represents a city and its position in the sequence. That is, we can code A as 1, B as 2, and so on. One of the most used crossovers is the order crossover (OX). This crossover selects a random subsequence from a parent and places it in the same position and order in the child. The rest of the positions are occupied by the remaining cities of the second parent in the relative order in which they appear (see Fig. 3).

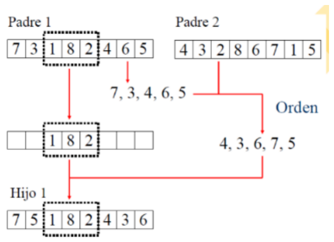

Fig. 3. Order Crossover (OX).

Selection chooses parents for breeding, crossover combines genes from the parents, mutation introduces random changes between cities, and replacement determines which solutions survive to the next generation. The fitness function evaluates the quality of a chromosome. In TSP, this translates into calculating the inverse of the total path length (1/cost). The goal is always to minimize this length.

Fitness scaling, which is a technique that adjusts the variables of an optimization problem so that they are on a similar scale, has also been introduced. It improves the convergence of the algorithm and facilitates the search for optimal solutions. Elitism has also been introduced, which is a strategy used to retain the best solutions over generations. It allows optimal solutions to be maintained in the population.

5 Experimental study

5.1 Experimental study design

First, the results obtained from solving the TSP with the A* algorithm by applying the heuristics mentioned before will be annotated. These results will be studied and compared. The algorithm was executed ten times with each heuristic, and the data collected were: time (seconds), expanded nodes, states reached and the cost. This process has been done for three instances of the problem: gr17 from the TSPLIB repository [8], and two more generated from it. We called them gr15 and gr12, created by removing the two and five last lines of gr17, respectively.
Next, the same process was repeated, but this time using the PEA* algorithm. The ε value will be varied for a specific heuristic. Finally genetic algorithms will be used, studying cost and fitness values through generations.
The machine that was used was a MacBook Pro with an Apple Silicon M2 Max processor (12CPU, 30GPU) and 32GB of memory. As for the implementation features, "aima-python" has been used with some modifications.

5.2 Experimental results

5.2.1. A* algorithm

Below is the table obtained from the executions of A* with the different heuristics for the gr17 instance:

	h1	**h2**	**h3**	**hmst**
Solution cost	2085	2085	2085	2085
Expanded nodes	94232	227308	227318	1899
Reached states	257193	411968	411970	7897
Mean time (s)	7.0965	18.5105	19.5749	0.1097
¿Optimal?	Yes	Yes	Yes	Yes

Table 1. Data obtained from the execution of A* with the heuristics for gr17

Thanks to the table it can be observed that hmst is the one with the least expanded nodes and takes the shortest time to reach the optimal solution; followed by h1, h2 and h3 being the slowest. It can be said that hmst is more informed than h1, which is more informed than h2, which is more informed than h3. In summary, the best one is hmst (see Table 1). All the heuristics are optimal as they find the optimal solution, so they are admissible (also because they are problem relaxations in the case of h2, h3 and hmst). They are well defined because h(n)>=0 for all n that is not goal, and h(n) = 0 when n is goal.

5.2.2. PEA* algorithm

To study PEA* the hmst was used because it is the best informed one. The route found (and if it is optimal or not), the expanded nodes, rectified for the same problem but with different Ɛ for the hmst. Instance gr17 will be used:

	Value of Ɛ						
	0.1	**0.5**	**1**	**2**	**3**	**4**	**5**
Solution cost	2085	2085	2085	2351	2502	2666	2674
Expanded nodes	521204	104660	520	29	20	18	18
Mean time (s)	24.272	5.306	0.025	0.00165	0.00136	0.00105	0.00104
¿Optimal?	Yes	Yes	Yes	No	No	No	No

Table 2. Data obtained from the execution of A* with hmst varying Ɛ for gr17

It can be observed that the higher Ɛ is, less nodes are expanded and the computation is faster. Once Ɛ is reached the algorithm stops (see Table 2). Therefore, even though it is faster than A*(h1), the heuristic is not admissible using PEA*. It is optimal for some Ɛ (while Ɛ <= 1.3). Heuristics are well defined because h(n)>=0 for all n that is not goal, and h(n) = 0 when n is goal.
When comparing A*(hmst) with PEA*(hmst), it can be said that in terms of time and expanded nodes, the latter is better while Ɛ <= 1.3 (gets optimal solution) and Ɛ >= 1. Otherwise, it consumes much more time, making A*(hmst) better. This is because PEA*(hmst) is not admissible, as it cannot reach the optimal solution.

5.2.3. Genetic Algorithms

As the Fig. 4 shows, the uniform crossover (UX) reaches sooner the maximum fitness value. This means it reaches the optimal solution first, and this holds true for both the best solution obtained by both crossovers, and for both fitness means obtained by executing the algorithm ten times with each crossover.
The fitness value has been defined as the inverse of the cost. The optimal solution's cost for gr17 is 2085 so the corresponding fitness is 0.000479616.

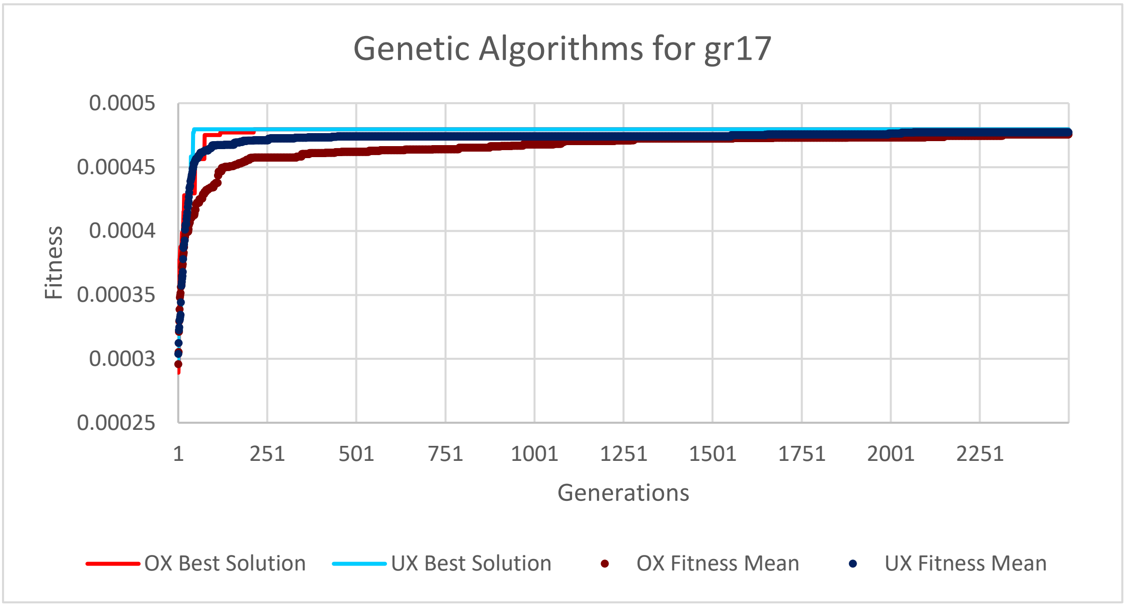

Fig. 4. Genetic Algorithms results obtained from the order crossover (OX) and uniform crossover (UX) executed with the instance gr17.

5.2.4. A* and AG comparison

The relative efficiency of A* and a genetic algorithm depends largely on the problem we are trying to solve. Depending on the different priorities at the time of resolution, we can say that using the A* algorithm is recommended when we need to know the most optimal solution. On the contrary, if what we want is a solution that is feasible in time and decent, we could use the genetic algorithm, since with few generations it can obtain a relatively low-cost solution (see in Fig. 5).

Note that, even with 10,000 generations, the genetic algorithm is not capable of locating the most optimal solution (staying very close to it) and, therefore, it is not indicated for these purposes.

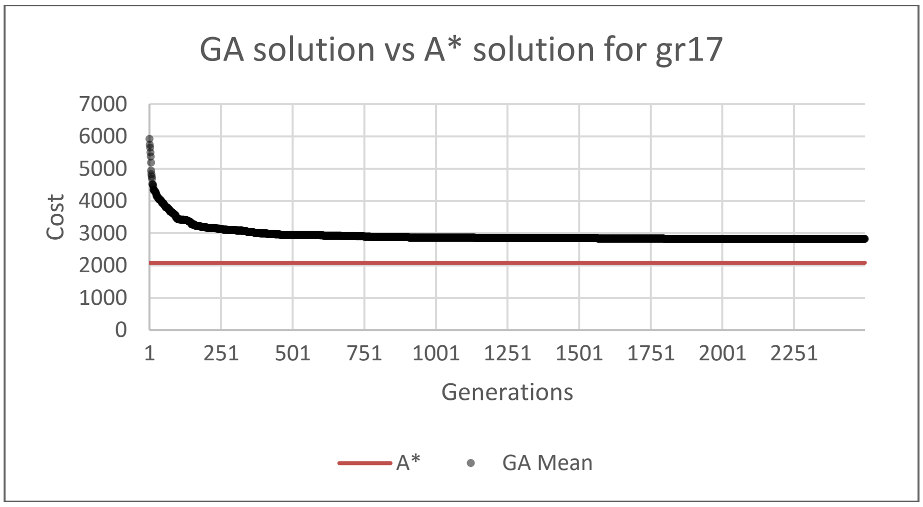

Fig. 5. Genetic Algorithms results vs A* results with the instance gr17.

6 Conclusions

To conclude the article, we will bring together several points to be taken into account. For A* we observed that although some heuristics are better than others (especially hmst), as the size of the problem increases the time also increases significantly. Certainly, it will always find the optimal solution, but it will take too long. Therefore, a shorter time would be more interesting, even if the solution is not optimal. This is when the use of PEA* becomes a great option. The value of Ɛ is important for keeping balance between execution time and the solution's quality. For genetic algorithms, when doing a large-scale study it can be observed that the solution cost is worse than the A* one, but they proved they are effective with very large-scale problems. A* has proved to be better for lower-scale instances.

7 References

1. Intelligent Systems course materials (Search Block). University of Oviedo
2. Traveling Salesman Problem. Wikipedia (https://es.wikipedia.org/wiki/Problema_del_viajante)
3. Traveling Salesman Problem (TSP) solving. (https://core.ac.uk/reader/211096990)
4. The Traveling Salesman. (https://knuth.uca.es/moodle/mod/page/view.php?id=3416)
5. A* algorithm. (https://es.wikipedia.org/wiki/Algoritmo_de_b%C3%BAsqueda_A*)
6. The TSP, heuristics based on genetic algorithms. (https://docta.ucm.es/entities/publication/6255a465-7d52-43f8-aed0-44acf997afdc)
7. TSP heuristics summary document. Intelligent Systems practice material. University of Oviedo
8. TSPLIB. (http://comopt.ifi.uni-heidelberg.de/software/TSPLIB95/tsp/)