AI Cup

Artificial Intelligence, USI

Xiana Carrera Alonso

December, 2022

Contents

1 Introduction

The AI Cup is a competition organized in the Artificial Intelligence bachelor course at USI by Luca Maria Gambardella and Umberto Junior Mele. Students enrolled in the course are asked to propose, implement and test a heuristic algorithm for the Traveling Salesman Problem (TSP).

The TSP models a situation where a salesperson needs to traverse a certain number of cities in the most efficient way possible. In a general way, given N cities and a distance function between them, the objective is to find a tour that goes through every city once and only once, coming back to the starting node (closed tour) and minimizing the total distance, i.e., the sum of the individual distances between consecutive cities in the tour.

1.1 AI Cup Description

In the AI Cup, students are given 10 TSP problems which are all symmetric (the distance between a city i and a city j is the same as the distance between city j and city i, for all pairs of nodes) and Euclidean, so distance is computed as the euclidean distance between points on a plane:

$$d_{ij} = \sqrt{(x_i - x_j)^2 + (y_i - y_j)^2} \tag{1}$$

For a single problem, the implemented algorithm must approximate the global optimum as closely as possible within a 3 minutes time limit.

The length of the best known tour for each problem is given, and so the error (gap) between a student's computed solution is obtained as follows:

$$gap = \frac{student\ tour\ length - best\ known\ result}{best\ known\ result} \tag{2}$$

Specifically, the 10 problems are:

Name	Dimension	Best known length
ch130	130	6110
d198	198	15780
eil76	76	538
fl1577	1577	22249
kroA100	100	21282
lin318	318	42029
pcb442	442	50778
pr439	439	107217
rat783	783	8806
u1060	1060	224094

Table 1: Problem set

The final result is computed as the average of the 10 individual errors. A single seed must be used for all problem instances.

2 Approach

To approach the problem, I decided to implement the Ant Colony Optimization (ACO) algorithm ([1]), being one of the most effective ones to date.

Specifically, I used ACO with local search (2-opt and 2.5-opt) to increase the quality of the overall solution, and a candidate list to increment efficiency, decreasing computational time and doing a greater number of iterations per problem. This was specially significative in those with a bigger number of nodes. The size of the candidate list is selected as the minimum between 40 and $\frac{n}{4}$, where n is the total number of nodes.

Additionally, I decided to add improvements coming from a version of ACO called ESACO (Effective Strategies + ACO). However, ESACO ([2]) generally performs better with even bigger problems, and so not all of its strategies were a good fit. Nonetheless, its candidate set updating algorithm was incorporated:

Algorithm 1 Candidate set updating

1: **while** $i \neq n$ **do**
2: $best \leftarrow I$ where $\forall J \neq n$ then $\tau[i][J] \leq \tau[i][I]$ ▷ node with more pheromone
3: Remove the last node of the candidate list of city i
4: Insert *best* in the first place of the list
5: Remove the last node of the candidate list of city i
6: Insert the city that follows i in the best global tour in the first place of the list
7: $i \leftarrow i + 1$
8: **end while**

However, this addition only improved the final quality in problems where following a less strict distance-based heuristic was preferable. Therefore, it was only included for ch130, pcb442, pr439 and u1060.

For those problems that don't run with the candidate list update, the closest city is avoided when doing exploration, as it was observed that the closest node had the predominant probability of being chosen in almost all cases.

Furthermore, the local search is not always performed in the same way. In 50% of the iterations, a 2.5-opt is applied to the best ant in the iteration. In another 25%, a 2-opt is applied to the globally best ant. The other 25% is destined to a 2.5-opt for the globally best ant. Additionally, the best ant tour does not begin to be optimized until n_0 iterations have passed, and 2.5-opt is not used for it until $2 * n_0$ iterations have passed, n_0 being a parameter selected beforehand.

With or without using ESACO, I began to notice that some of the larger problems (rat783 and u1060) didn't manage to run for enough iterations on the 3 minute limit to make effective use of the knowledge slowly accumulated in the pheromone matrix. After several code improvements that made the program quite faster, the error was also significantly reduced, to around 6%. To further decrease it, I decided to add a special clausule: for rat783 and u1060, instead of starting with a random solution, the initial solution is computed with the best nearest neighbour (best NN) algorithm and is optimized with 2-opt and 2.5-opt until no improvement can be achieved. To not waste excessive time, each individual local search function doesn't do more than 250 iterations. The results were positive and the initial time computation didn't exceed 5 seconds.

2.1 Parameters

Each problem has characteristics that make it unique. The total size is important, as bigger problems will take more time per iteration and are, in general, harder to optimize. But not all problems of similar size are equal, and other particularities such as clustering will greatly affect the final result. See, for example, figures 1 and 2.

Therefore, each problem benefits from different parameters:

- α, the coefficient for the global pheromone update. Most problems use $\alpha \approx 0.1$, but others benefit to values up to 0.4.
- ρ, the coefficient for the local pheromone update. Very problem-specific, varying from 0.09 to 0.6.
- β, which sets the importance of the distance heuristic in contrast to the pheromone when choosing a node.
- Q_0, the probability of choosing exploitation in contrast to exploration. It was approximated as the nearest integer to the value $1 - \frac{12}{n}$, with some deviation if it showed improvements.

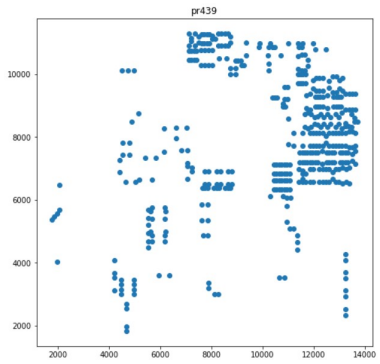

Figure 1: pr439 city distribution

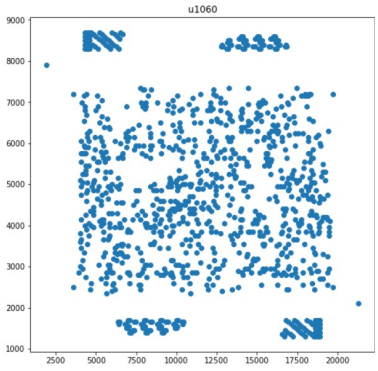

Figure 2: u1060 city distribution

- m, the number of ants. Almost all problems show the best results when $10 <= m <= 20$. The exact number was chosen to be 10, 15 or 20 depending on the experimental results, although there is a tendency towards higher m values when n increases.
- τ_0, the initial pheromone value. Usually computed as $\tau_0 = \frac{1}{n * NN_{length}}$, although in some instances a higher initial pheromone was preferable, and so was changed to $\tau_0 = \frac{n}{NN_{length}}$.
- n_0, the number of iterations without using 2.5-opt to improve the best global solution. This addition slowed the convergence process in small sized problems and led to better overall results, though the exact value was changed depending on the problem.

3 Implementation

The code was implemented in C++. Problems are read in *Problem.hpp*, while the rest of the header files contain the algorithms and auxiliary functions needed to run the program.

I used several abbreviations and type definitions common in competitive programming for faster writing. Note that they do not affect the execution, as they are resolved in compilation time.

A bash script is included to automatically compile the script and execute it for all problems with the same seed. Compilation is performed with the -O3 option, and vector extensions are activated for even better performance. Results will be both printed by terminal and written to a .csv file, *results.csv*. The script can be run through the command `./bash_script.sh`

4 Results

Using a fixed seed of 5053, the average gap was 0.95723098%. The detailed results are shown in table 2:

Name	Best known length	Student length	Gap
ch130	6110	6110	0%
d198	15780	15786	0.0380228%
eil76	538	538	0%
fl1577	22249	22491	1.08769%
kroA100	21282	21282	0%
lin318	42029	42550	1.23962%
pcb442	50778	51370	1.16586%
pr439	107217	108027	0.755477%
rat783	8806	9040	2.65728%
u1060	224094	229984	2.62836%
Total			0.95723098%

Table 2: Experimental results

5 Conclusion

Results show that ACO is a really effective strategy for solving TSP, and that it can even be improved through the use of local search, candidate lists, special strategies, etc. However, each TSP problem has unique characteristics that make parameter selection decisive to improve results.

References

[1] Dorigo M., G. Di Caro and L. M. Gambardella. Ant Algorithms for Discrete Optimization. Artificial Life, 5,2, pp. 137-172, 1999. `https://people.idsia.ch/~luca/ij_23-alife99.pdf`

[2] Ismkhan, H. Effective heuristics for ant colony optimization to handle large-scale problems. Swarm Evol. Comput. 2017, 32, 140–149. `https://doi.org/10.1016/j.swevo.2016.06.006`