Ant Colony Optimization for the Dubins Traveling Salesman Problem

Ilaria Martelli, Lucia Pallottino

February 2023

***Abstract* - In the world of robotics we can find a wide variety of autonomous robots, of any shape or form and for any locomotion medium; terrestrial self-driving cars, flying drones, and submarine vehicles are some of the most widespread types. These kinds of motorized robots usually have a specified turning radius based on speed, wheel dimension, wingspan or general propulsion method, so the routes they can pursue are limited by the kinematics of the system. The Dubins Traveling Salesman Problem (DTSP) belongs to a branch of path planning problems focused on searching a continuous path that can touch a specified number of points of interest trying to achieve the shortest tour for non-holonomic constrained systems, useful in many applications like patrolling, mapping and delivery services. This work presents a new algorithm that combines the distributed intelligence approach of the Ant Colony Optimization, with a Dubins Shortest Path Problem (DSPP) solver, such as the Pulley Algorithm, to find a solution for the DTSP. Their behavior will be explained and compared with other state of the art algorithms. Finally, a statistical analysis of the results is presented to show the performance of the newly proposed algorithm.**

1 Introduction

Usually, when one thinks of a robot or any robotic task, the first thing that comes to mind is a single operator with an autonomous processing unit that allows it to perform the assigned job. There is however a less intuitive branch of robotics that offers a different approach: distributed robotic systems [26]. Within these systems, it's studied how multiple robots work and interact with each other, and how working together improves effectiveness and can lead to results that would not be possible for an isolated individual. Examples of decentralized, multi-agent, self-organized systems are common in nature, such as schools of fish, flocks of birds, colonies of insects like ants, termites, bees, etc.
This same reasoning can be applied to abstract concepts instead of physical robots, and when it's referred to algorithms it's called *distributed intelligence* [2]. This field concerns the study of heuristics to solve complex learning, planning, and decision making problems. From these concepts stem interesting heuristic methods to approach complex NP-hard problems for which a trivial exact solution can't be found. One of them is the *Dubins Traveling Salesman Problem* (DTSP).
The DTSP, which takes its name from Dubins' car, originally described by L. E. Dubins [10], is an important tool for demonstrating concepts in robotics and control theory including path planning subject to non-holonomic constraints, and as a way to model wheeled robots, airplanes and underwater vehicles. As the TSP from which it derives, the DTSP has the following typical formulation: given a set of cities and a cost to travel from one city to another, we want to identify the tour that will allow a salesman to visit each city only once, starting and ending in the same city, at the minimum cost.

Since many robots from various robotics application fields follow Dubins' constraints, or can generally be modeled with sufficient accuracy with a Dubins system, the DTSP is the natural next step to resolve the same routing problem while keeping track of the kinematic constraints of the system.
The most intuitive application is robot-operated surveillance of a certain area: the DTSP is initialized by

building a graph of all the possible points of interest that the robot has to pass by, and by solving the problem, a minimum cost tour is found. This allows to minimize the length of the path and in turn the fuel/battery usage, in addition to shortening the time it takes to travel the entire tour. In a similar way we can think about exploration or mapping tasks.
Another application scope is found in delivery services. While the online path searching program would have to keep in mind specific pathways and natural obstacles in a changing environment, an higher level tour search has to be done before departing on general neighborhoods and recipient addresses: solving the DTSP would provide the fastest "rough" path, that would later be refined using exact algorithms on pre-existing urban routes, and refined furthermore online with data from sensors on the robot itself.

The proposed architecture focuses on the *Ant Colony Optimization* (ACO) heuristic, initially devised by M. Dorigo [6] and later developed by Dorigo, Maniezzo, Colorni and Gambardella [4], [9], [11], [7]. The algorithm was designed as a tool to solve the TSP: when compared with classical heuristic algorithms, the ACO showed encouraging results or even dominance over the others [19].

The idea for this work thus was to implement the ACO heuristic paired with a supplementing DSPP-solver, to create a novel approach called *ACO-DTSP algorithm*: its goal will be to find a feasible solution for the DTSP that is both fast and as close to optimal as possible. Due to the iterative nature of the presented algorithm, the balance between cost and efficiency will also be addressed. Then, a statistical analysis will be applied to showcase the results.

2 The ACO-DTSP algorithm

2.1 The ACO algorithm

The Ant Colony Optimization algorithm, as its name suggests, draws inspiration from the emergent behavior of ants, more specifically their path seeking abilities as they move between the colony and a source of food. Like other systems studied in swarm robotics, even if a single ant has only simple capabilities, interesting aspects arise from the observation of a whole ant colony, which, as a result of coordinated interactions, is instead highly structured. This distributed problem solving environment is thus proposed, and it's used to search for a solution to the TSP.
For the formal description of the ACO, we define:

$\tau \rightarrow$ the "pheromone trail level", that simulates the stimuli ants experience with their peculiar communication method;
$\eta \rightarrow$ the "attractiveness" of a possible path (or equally, of the next possible edge to anoher node), usually set as the inverse of the distance among the nodes;
$M_k \rightarrow$ the "working memory" of each ant, a table of every node that was already visited by that particular ant, used to avoid the creation of non-hamiltonian tours.
Finally, the "desirability" is the trail level multiplied by the attractiveness (that is, $\tau \cdot \eta$) [7], and is the core of the algorithm, as those two are the main metrics which the optimization is based on [8].
The ACO algorithm for the TSP is composed by three fundamental rules.

State Transition Rule (Pseudo-Random Proportional Rule)

$$s = \begin{cases} \arg\max\limits_{u \notin M_k} \{[\tau(r,u)]^\alpha \, [\eta(r,u)]^\beta\}, & \text{if } q \leq q_0 \quad \text{(exploitation)} \\ S, & \text{otherwise} \quad \text{(biased exploration)} \end{cases} \tag{1}$$

Here, α is the pheromone power parameter, and controls how likely each ant will be to follow the same paths as the ants before it (too high and the algorithm will keep searching the same path, too low and it will search too many paths, virtually ignoring the pheromone on them); β is the distance power parameter, and controls the extent to which ants will prefer nearby points (too high and algorithm will essentially be a

greedy search, too low and the search will likely stagnate); q is a random number uniformly distributed in [0,1]; q_0 is the exploitation threshold parameter ($0 \leq q_0 \leq 1$); and S is a random node selected according to the following probability distribution:

$$p_k(r,s) = \begin{cases} \frac{[\tau(r,s)]^\alpha \, [\eta(r,s)]^\beta}{\sum_{u \notin M_k} [\tau(r,u)]^\alpha \, [\eta(r,u)]^\beta}, & \text{if } s \notin M_k \\ 0, & \text{otherwise} \end{cases} \tag{2}$$

We can see that $p_k(r,s)$ is the probability distribution function that shows the probability for which the (r,s) edge would be chosen, normalizing the desirability of each remaining candidate and assigning a probabilistic weight proportional to it.
This is done as a parallel to genetic algorithms [27], where the *mutation* term makes the solution research more flexible and less subject to stagnation in a local minimum.

Local Updating Rule

$$\tau(r_k, s_k) \leftarrow (1-\rho)\,\tau(r_k, s_k) + \rho\,\Delta\tau(r,s) \tag{3}$$

Here, ρ is the evaporation rate (or pheromone decay parameter), and is the proportion of pheromone that evaporates on each step; $\Delta\tau(r,s)$ can be set to τ_0, the initial pheromone level, because it's simple to implement and produces better or similar performance with respect to other tested options (Q-learning formula or 0) [8]; another given option is to use $\frac{1}{n\,L^-_{gb}}$, where L^-_{gb} is the best tour from the last iteration, to implement the correction rule for the Reinforcing Ant Colony System (RACS) [21] [20].
The variable τ_0 can in turn be set as an editable variable or $(n\,L_{nn})^{-1}$ [22], where n is the number of nodes and L_{nn} is the length of the naive tour found with the Nearest Neighbor algorithm; this last option is used as a very rough approximation of the optimal tour length.
This update is made every *step*, which means each time a new node is chosen for all m ants whose tours are still in-progress; this is done to simulate the pheromone deposited by an ant in its wake as it moves.

Global Updating Rule

$$\tau(r_k, s_k) \leftarrow (1-\rho)\,\tau(r_k, s_k) + Q\,\Delta\tau(r,s) \tag{4}$$

$$\Delta\tau(r,s) = \begin{cases} (L_{gb})^{-1}, & \text{if } (r,s) \in \text{global-best-tour} \\ 0, & \text{otherwise} \end{cases}$$

In this last control rule, Q can be chosen between an editable variable and the pheromone evaporation rate ρ. There is again the option to use the RACS, and in that case the Global Updating Rule also re-initializes trails that go over the upper bound $\tau_{max} = \frac{1}{(1-\rho)\,L_{gb}}$ [21] [25].
This update is made every *iteration*, which means each time a group of m ants has finished their search and a global best tour is being found; this is done to simulate a predilection for the shortest path by the colony, as with time more ants would walk on it and make the pheromone trail more intense and less prone to vanishing through evaporation.

2.2 The DTSP

Dubins car (or *unicycle*) is a kinematic two-dimensional model of a car that can only drive forward, or turn with a bounded turning radius. It's motion rules are:

$$\begin{cases} \dot{x} = V_0 \cos(\theta), \\ \dot{y} = V_0 \sin(\theta), \\ \dot{\theta} = \frac{V_0}{r}\,u, \quad \text{with } u \in [-1,1] \end{cases} \tag{5}$$

Theorem [Dubins] Being C a general circular segment of radius r (L/R if it's a left/right turn) and S a straight segment, the Dubins Path Theorem states that the shortest path between any two configurations of a Dubins vehicle in an environment without obstacles is of type CCC or CSC, or a subpath of either of these two types [10], [17]; this reduces the number of possible paths down to six:

$$\{LSL, RSR, RSL, LSR, RLR, LRL\}$$

which constitute the sufficient family of paths [1]. In other words, the shortest path is constructed by joining circular arcs of maximum curvature and straight lines.

The method used in this work to find optimal Dubins paths is based on the algebraic solutions by Shkel *et al.* [24]. However, rather than using angular symmetries to improve performance, the simpler approach of testing all candidate solutions is used [28], [16]; this can be done efficiently by any machine, precisely because the number of eligible candidates in the sufficient family of paths is negligible.

Once the Dubins optimal path is defined, all that remains is to find a solution for the DTSP. As previously mentioned, since it's an offshoot of the TSP, the DTSP is also NP-hard; besides, it adds a continue search space due to the headings optimization sub-problem, thus making the solution more complex and expensive, both in terms of time and resources.
For instance, for the general TSP the brute-force recursive method used in Heap's algorithm [13], that merely checks every possible path, has a time complexity of $\Theta(n!)$. That means that by the time we're considering 20 towns or nodes ($n = 20$), there's already more than 60 quadrillion solutions (60×10^{15}). Other exact approaches like the Dynamic programming [5] and the Held–Karp algorithm [15] are more efficient, as they yield a solution in $\Theta(n^2 2^n)$ time, which is considerably better but still too slow to use.
This emphasizes the need for a heuristic technique to quickly find a solution that, even if not guaranteed to be optimal, is nevertheless sufficient for reaching an immediate, adequate approximation. Many state of the art approaches have been developed to solve the DTSP, and were examined for the purposes of this work: the Nearest Neighbor Algorithm (or "greedy" approach), Randomized Headings Algorithm and the Heading Discretization Algorithm [17]; the Alternating Algoritm and the Recursive Bead-Tiling Algorithm [23]; the Look-Ahead Algorithm [18]; the Discretized Look-Ahead Algorithm [3]; the Genetic Algorithm [29]; the Pulley Algorithm [12].

2.3 From TSP to DTSP - the DSPP

DTSP solvers can be arranged in two different categories:

- ones that solve the TSP and modify the already best Hamiltonian tour;
- ones that directly solve the DTSP without solving the TSP.

The ACO algorithm already creates ordered sequences of waypoints for the TSP; therefore, to solve the DTSP, it needs an underlying function to get sensible node headings. Those headings then, together with the known node positions, would be converted in Dubins paths to get the tour length, and enabling the algorithm to evaluate the best Dubins tour for every iteration.
Finding a Dubins path through a pre-existing ordered set of targets is a simplified version of the DTSP, called the *Dubins Shortest Path Problem* (DSPP). For this reason, only the first category of DTSP solvers was considered as possible candidates.
What is needed is a fast algorithm, which can be used in conjunction with an iterative algorithm as is the ACO. Many possibilities were examined, and the available algorithms were skimmed to discard those that weren't relevant to the case in exam: any algorithm that inherently include choosing the order of nodes, which can't be separated from the choice of the headings (such as the Bead-Tiling); any algorithm that is iterative itself (such as the Genetic); or any algorithm that needs long computation times to perform (such as the Heading Discretization).
In the end, the Alternating Algorithm was implemented in the initial version, that later was improved with the Pulley Algorithm as a new, fast, performing DSPP algorithm.

2.3.1 Alternating Algorithm

The *Alternating Algorithm* (AA) was initially chosen as a way to easily transform a TSP tour in a DTSP tour [23], in order to redirect the objectives of the ACO algorithm to the optimization of a different version of the TSP, without modifying its core dynamics.
The fundamental idea of the AA is to alternate straight line edges, as you'd have for classic TSP tours, with minimum-length Dubins paths, preserving the point ordering.
However, this approach has a downside: if the curvature radius of the Dubins system is comparable or even bigger than the minimum distance between any two nodes, the resulting tour is bound to have "loops" that make the best tour obtainable less optimal. If a tour which has been optimized for the TSP is then converted into a Dubins path, it will almost inevitably become non-optimal, since the TSP as it's elaborated doesn't have headings taken into account. On the other hand, Fig. 1 displays the optimizing possibilities of running the AA online.
It can be proved that the AA performs well when the points to be visited by the tour are chosen in an adversarial manner [23]. However, it is not a constant-factor approximation algorithm in the general case. Moreover, the AA might not perform very well when dealing with a random distribution of the target points.

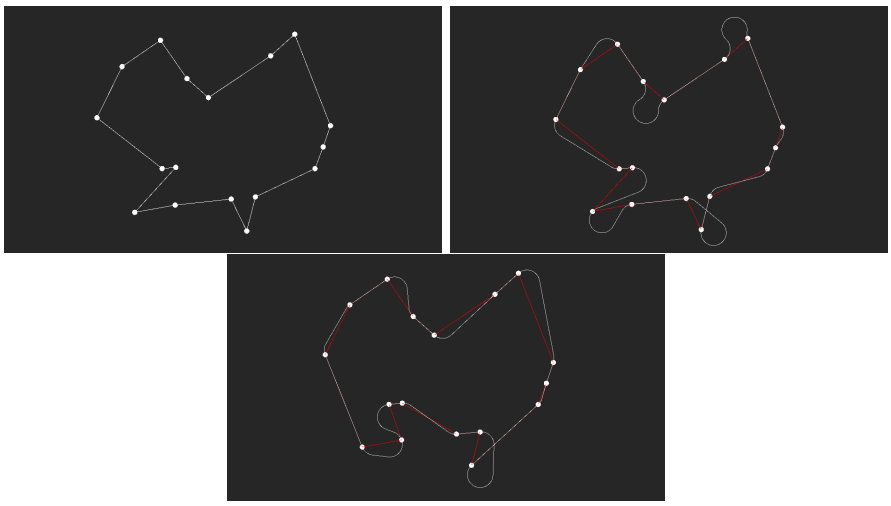

Figure 1: Example taken from the newly-proposed ACO-DTSP program using the AA. TSP tour length = 313.555; AA applied on the TSP tour length = 413.123 (same waypoint order as TSP tour visible in red); complete DTSP with AA tour length = 373.102 (different waypoint order, optimized for the DTSP).

2.3.2 Pulley Algorithm

Successively, it was chosen another more complex algorithm to convert the TSP optimal solution to a kinematically feasible optimal Dubins path, called the *Pulley Algorithm* (PA) [12].
This new method returns angles based on a natural physical system, namely the pulley, to get the tightest path given an ordered list of waypoints.
The algorithm can be described by the following two steps:

- place a circle of radius r_{min} at each waypoint, such that it intersects the arriving and departing straight paths in two equal chords, with the waypoint in the middle of the arc that starts with the first intersection and ends with the second one;
- connect every pair of consecutive circles with a line tangent to both.

The PA not only finds Dubins-constrained paths that better fit any arbitrarily ordered set of nodes, but unlocks faster search for better tours if used in addition to an optimization algorithm like the ACO-DTSP, since it makes tours that otherwise would be discarded feasible.

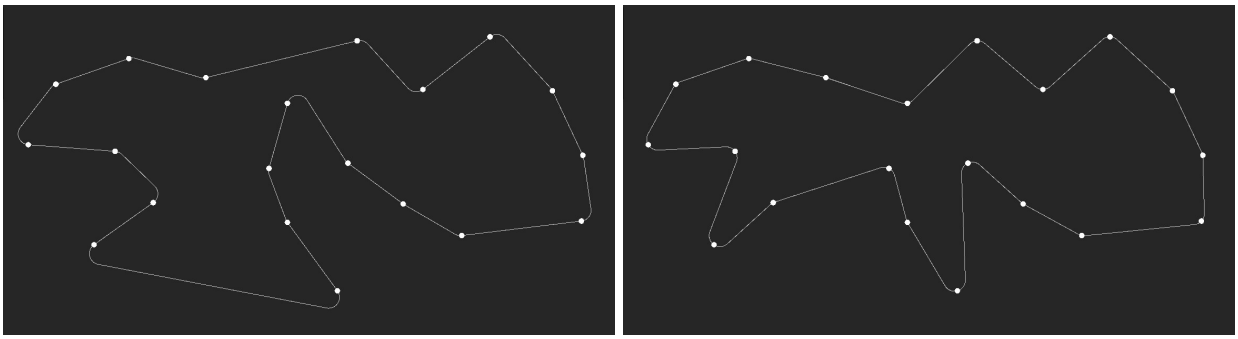

Figure 2: AA-PA comparison on the same set: AA = 686.853; PA = 630.314.

2.4 Integration

The last step for the implementation of the ACO-DTSP algorithm consists in the integration of the chosen DSPP method with the standard ACO algorithm, in a way that shifts the focus from the TSP, the main problem which the ACO was elaborated for, and directly addresses the DTSP. In the original formulation of the ACO algorithm, the length of any TSP tour found is used to compare them to each other and search for the shortest one, enabling for the optimization to take place. This behavior was slightly modified for the ACO-DTSP.
Each time an ant creates a new tour, through the repeated application of the State Transition Rule, the ordered waypoints are saved, and on every *iteration* they're converted into Dubins tours using the Pulley Algorithm; the resulting path length is then used to check which one is the shortest, so the best Dubins tour can be computed and visualized online as the algorithm goes on. In this way the loops issue indicated in Section 2.3 is more easily avoided, since the loops would make the paths longer and therefore are filtered out during the optimization process; however, they persist if the nodes that have a small distance between them with respect to the curvature radius are in a configuration that doesn't allow a smooth motion. Because of this, it's always advisable to choose a curvature radius that's smaller than the minimum distance between any two nodes of the system.

To roughly check the performance of the ACO-DTSP algorithm, given the lack of available datasets for the Dubins TSP, a simple test was carried out using geometry axioms: it was programmed for the nodes to be positioned in a certain way, in order for the resulting set to assume the shape of a circumference. Since for such a set the shortest tour will always be the one that draws the implied circumference, for a single execution it's trivial to visually check the optimization results. As shown in Fig. 3, the program successfully recognises the circular pattern as the best tour very quickly, usually merely within 5 iterations or less.

3 Software overview

For this work, two programs where used: "Ant Colony Optimization DTSP" and "ACO-DTSP Statistical Analysis". Both programs were written in C# programming language, and run on the Unity engine.
Unity is a real-time development platform that provides a graphical-driven workspace. The simulation takes place in the Scene window, where the nodes and edges of the dynamic graph are drawn and updated in real-time once the algorithm has started. In the Game window, another camera shows a text object with useful data on the current simulation: the number of nodes for the current execution; the duration of the current algorithm session, in seconds; the best TSP tour length found; the best DTSP tour length found; the turning curvature ("None" if the search is set on TSP). Moreover, in the same window the interactive Start and Stop button are located. Finally, the Inspector tab contains all the parameters for the execution.

To assess the quality of the model and compare the algorithm variations used, a statistical study was implemented. The main program "Ant Colony Optimization DTSP" was wrapped in a higher-level interface:

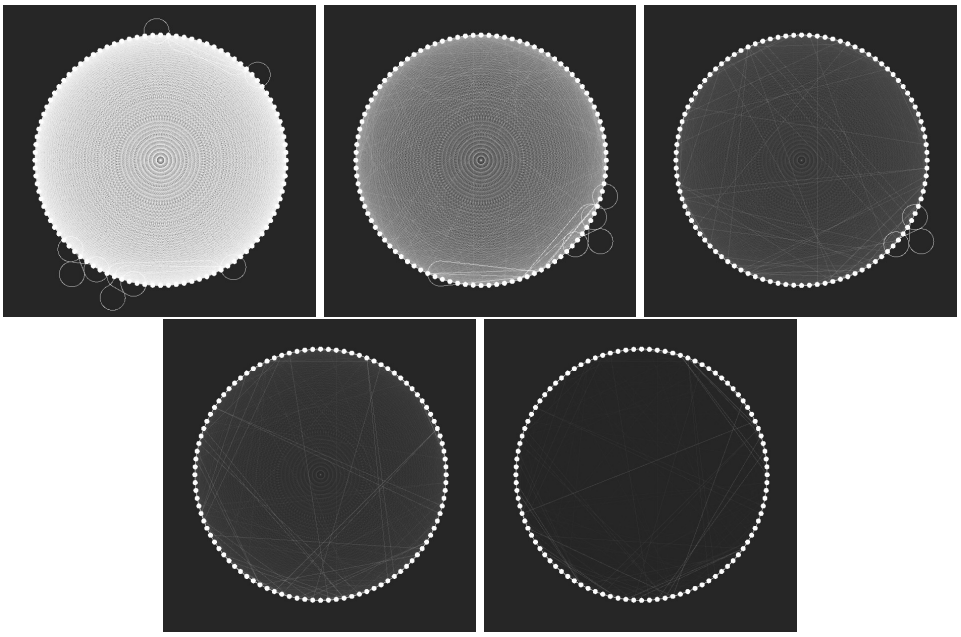

Figure 3: Circumference test on 100 nodes. The circumference pattern is correctly found in 4 iterations. A visual representation of pheromone trails can be seen as white graph lines, getting paler on every new iteration as the pheromone "evaporates" from bad paths and converges to the optimum.

the resulting program is the "ACO-DTSP Statistical Analysis". For this reason, the scripts used are the same as before, with the addition of the Wrapper script, that, as mentioned, wraps the fist program to allow for multiple subsequent executions.
The wrapper program lets the user choose all the parameters previously listed, plus some newer ones: the number of iterations and the number of repetitions. The former specifies how many iteration loops the ACO-DTSP algorithm keeps running before stopping, with each iteration ending when the Global Updating Rule occurs; the latter specifies how many times the whole ACO-DTSP algorithm is being initialized from scratch, started and consequently stopped.
The program was specifically created to automatize the report of statistically significant data: to get the most out of a single run and minimize the time spent modifying parameters, the program runs on *batches*, or arrays of successive repetitions with the same number of maximum iterations. After each batch, the best tour length average is computed and then displayed in the Console log.

4 Statistical analysis

The ACO algorithm by construction heavily relies on random chance to decide new paths to explore: even with the same initial parameters and the same number of iterations, two executions could return different final values. To prevent the return values to be skewed by random bias, the wrapper stores the results from every ACO-DTSP complete repetition, and then, when all the required repetitions are finished, it displays the arithmetic average of the results.
For every statistics applied the number of repetitions was set to 500, as a compromise between statistical accuracy and feasible computation times. Also, in the following Tables the classical TSP will be denoted as ETSP, which stands for *Euclidean Traveling Salesman Problem*, since the cost of each edge is given by the euclidean distance between the nodes (a straight line in 2D space).
The ACO parameters used for every statistics implemented are:
$m = 15$, $\alpha = 1$, $\beta = 2$, $\rho = 0.7$, $q_0 = 0.7$.

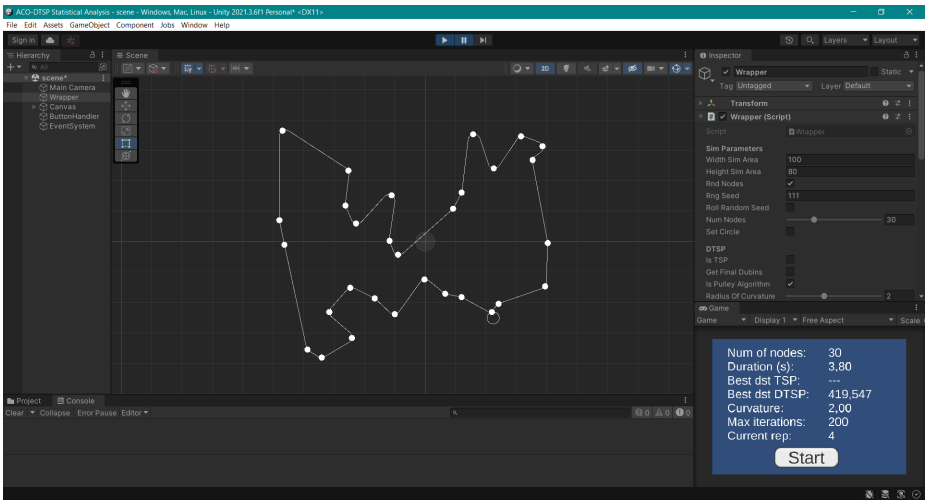

Figure 4: "ACO-DTSP Statistical Analysis" project in execution.

Set	Rad	Iterations	Best ETSP average	Best TSPtoD average	Best DTSP (AA) average	Best DTSP (PA) average
20	2	50	*357.433*	364.368	371.440	**362.702**
20	2	100	*357.057*	363.246	370.574	**361.490**
20	2	150	*356.433*	361.773	369.737	**360.761**
20	2	200	*356.595*	361.880	369.457	**360.419**
20	4	50	*357.387*	408.878	392.771	**373.658**
20	4	100	*356.877*	410.103	390.903	**372.517**
20	4	150	*356.692*	411.376	391.053	**372.045**
20	4	200	*356.311*	411.280	390.496	**371.362**
30	2	50	*397.536*	423.114	426.348	**420.360**
30	2	100	*397.417*	423.571	425.889	**419.793**
30	2	150	*397.242*	423.264	425.691	**419.936**
30	2	200	*397.262*	423.577	425.538	**419.441**
30	4	50	*397.466*	536.125	523.811	**505.774**
30	4	100	*397.363*	537.231	519.954	**502.286**
30	4	150	*397.290*	536.327	518.932	**500.031**
30	4	200	*396.995*	535.251	517.364	**500.033**
50	2	50	*516.176*	573.813	603.901	**569.828**
50	2	100	*511.354*	568.503	596.627	**564.098**
50	2	150	*511.135*	568.435	593.546	**561.668**
50	2	200	*509.472*	567.072	591.481	**560.052**
50	4	50	*516.060*	974.457	856.339	**846.952**
50	4	100	*511.225*	972.684	843.202	**829.531**
50	4	150	*509.999*	973.945	835.735	**817.679**
50	4	200	*508.878*	973.783	832.842	**812.145**

First the program was made to run an optimization for the ETSP tour, and as it found the best ETSP path for every repetition, the final tour was converted in a Dubins path before ending (this procedure used the PA and is designated as "TSPtoD"); this was done to later compare it with DTSP optimization. Also, the ETSP lengths given are the baseline both for the ACO algorithm operations and for the node set itself, as the ETSP is made of straight lines and as such always returns the shortest possible tour compared to any

other DTSP-variation. For ETSP tours it was chosen to document new lengths when the turning radius was changed, even if the change only affects Dubins paths. For comparisons, the best Dubins tour lengths are also reported in bold to identify them more easily.

For any applied mode, the tour lengths tend to get smaller as the number of iterations gets bigger (except for some random outliers), which is consistent with an optimization convergence to the best results. It's interesting to note though that for the TSPtoD the inverse proportion of number of iterations and tour length is less noticeable or sometimes downright absent: this is due to the fact that since the optimization is done specifically on ETSP tours, it's entirely possible that a worse ETSP tour leads to a better DTSP tour found. In fact, the problem of solving first the ETSP is that in a dense area of waypoints (where the minimum turn-radius is too big compared to the smallest distance between two nodes) the modifications done to the best path is more cost-effective, producing a bad DTSP tour in most cases. This also explains why this difference is even more noticeable for a bigger turning radius.
For each set is then reported the smallest distance between nodes, to allow a comparison with the used radius, and the average distance between nodes.

Num of points	Min dst	Avg dst	Min dst > Rad 2	Min dst > Rad 4
20	5.324	47.489	✓	✓
30	3.481	45.910	✓	✗
50	0.661	47.510	✗	✗

It can be seen that the 20-points set has a minimum distance that is higher than both the used radii, for the 30-points set it's higher than the second radius, and for the 50-points set is lower than both. This explains why more drastic variations in tour length are observed when the radius is increased for larger sets.

The DTSP(AA) is shown to perform significantly better than the TSPtoD when the turning radius is bigger, but it's worse otherwise. In general, it's trivial to assume that as the turning radius tends to 0, the ideal best DTSP tour tends to the ideal best ETSP: the simple AA forces the path to add curves haphazardly every other edge, and in the long run this approach still loses to the TSPtoD when the latter uses the PA.
Then the complete ACO-DTSP is analysed (therefore with the Pulley Algorithm as its DSPP-solving component), and compared with the previous results of the TSPtoD. As it's shown, all the resulting best DTSP tours are shorter than their TSPtoD counterpart, especially when running with a bigger turning radius: this highlights the optimizing power of the program, with up to a maximum of 16.6% decrease in tour length with respect to the TSPtoD. The difference is so evident that even considering the highest max iterations for the TSPtoD and the lowest max iterations for the DTSP, the performance of the latter is still similar or better. It's also worth noting how the DTSP manages to better exploit the tour when the turning radius is bigger than the minimum inter-node distance, avoiding loops or other headings configurations that may stretch the total path length.
Lastly, we can compare DTSP tours found using either the Alternating Algorithm or the Pulley Algorithm. The PA outperforms the AA in every category: this is due to the nature of the PA, that organically adapts to the considered path, as the Dubins transformation is practically form-fitting to the node net. The PA seems able to consistently shrink the total length, and the difference of best tour lengths is rather uniform across the board and doesn't considerably change for larger sets. However, since the tour lengths get bigger, this means that the rate of improvement actually decreases: this could be attributed to the denseness of the studied sets, since as it was recorded the minimum distance between points gets smaller and smaller as the number of points grows.
Generally, the DTSP using the PA is again proven to be the best method out of all those analysed, from computing speed to overall optimizing performance.

5 Conclusions

This paper explores the potential of the ACO algorithm combined with a DSPP solver, and presents the empirical outcome of the resulting DTSP optimization; the ACO-DTSP algorithm shows positive results as a cost-effective approach to the problem.
The modular nature of the program with respect to the DSPP sub-algorithm allows for further research in this scope; future works may thus implement new methods for heading search. Besides, the ACO algorithm itself offers many settings that can be finely tuned to get the best results. One way to explore them to the fullest could be through the use of artificial intelligence iterative algorithms, such as the generalized genetic algorithm: each setting could be seen as a gene, each ACO-DTSP algorithm running with those specific settings could be an individual in the population, and with the final best tour length set up as the fitness function.
A limit for this research was the absence of a library of sets for the DTSP, complete with the best tour length recorded for every set, similar to the concept of TSPLIB [14] for TSP tours; a resource like this would make it easier to measure the performance of the proposed algorithm, as well as any other DTSP algorithm. Usually these kinds of libraries rely on supercomputers that use exact algorithms which check for every possible tour (or a reasonably pruned subset) to get the best result with absolute certainty, at the cost of an expensive use of resources and high computation times, and then collect them as a list of sets on a public website that can be freely consulted. The only problem for this concept is that while for the TSP this can be done in a discrete searching space, since the number of possible TSP tours for a set of points is factorial with n, but it's nonetheless *finite*, the DTSP pairs it with the search for the optimal headings, making it a mix type continue search space. One option could be to use a finely discretized heading choice to get as close of a result to the global best as possible, added to the waypoint order, to return to a unified discrete space.

References

[1] D. Anisi. "Optimal Motion Control of a Ground Vehicle". In: (Jan. 2003).

[2] A. Bond and L. Gasser. "Readings in Distributed Artificial Intelligence". In: (Jan. 1988).

[3] I. Cohen et al. "Discretization-Based and Look-Ahead Algorithms for the Dubins Traveling Salesperson Problem". In: *IEEE Transactions on Automation Science and Engineering* PP (Sept. 2016), pp. 1–8. DOI: 10.1109/TASE.2016.2602385.

[4] A. Colorni, M. Dorigo, and V. Maniezzo. "Distributed Optimization by Ant Colonies". In: Jan. 1991.

[5] Subham Datta. *Traveling Salesman Problem – Dynamic Programming Approach*. 2022. URL: https://www.baeldung.com/cs/tsp-dynamic-programming.

[6] M. Dorigo. "Optimization, Learning and Natural Algorithms". PhD thesis. Politecnico di Milano, Italy, Jan. 1992.

[7] M. Dorigo and L.M. Gambardella. "Ant Colonies for the Traveling Salesman Problem". In: *Bio Systems* 43 (Feb. 1997), pp. 73–81. DOI: 10.1016/S0303-2647(97)01708-5.

[8] M. Dorigo and L.M. Gambardella. "Ant Colony System: A cooperative learning approach to the Traveling Salesman Problem". In: *IEEE Transactions on Evolutionary Computation* 1 (May 1997), pp. 53–66. DOI: 10.1109/4235.585892.

[9] M. Dorigo, V. Maniezzo, and A. Colorni. "Ant system: Optimization by a colony of cooperating agents". In: *IEEE Trans. Syst., Man, and Bybern, Part B* 26 (Jan. 2002), pp. 29–41.

[10] L.E. Dubins. "On Curves of Minimal Length with a Constraint on Average Curvature, and with Prescribed Initial and Terminal Positions and Tangents". In: *American Journal of Mathematics* 79.3 (July 1957), pp. 497–516. ISSN: 00029327, 10806377. DOI: 10.2307/2372560.

[11] L.M. Gambardella and M. Dorigo. "Solving symmetric and asymmetric TSP by ant colonies". In: June 1996, pp. 622–627. ISBN: 0-7803-2902-3. DOI: 10.1109/ICEC.1996.542672.

[12] W. Ghadiry et al. "Optimal Path Tracking With Dubins' Vehicles". In: *IEEE Systems Journal* PP (July 2020), pp. 1–12. DOI: 10.1109/JSYST.2020.3006990.

[13] B.R. Heap. "Permutations by Interchanges". In: *The Computer Journal. Section A / Section B* 6 (Nov. 1963). DOI: 10.1093/comjnl/6.3.293.

[14] Ruprecht-Karls-Universität Heidelberg. *TSPLIB - A Traveling Salesman Problem Library*. 1991. URL: http://comopt.ifi.uni-heidelberg.de/software/TSPLIB95/tsp/.

[15] M. Held and R. Karp. "A Dynamic Programming Approach to Sequencing Problems". In: *Journal of the Society for Industrial and Applied Mathematics* 10 (Mar. 1962), pp. 196–210. DOI: 10.2307/2098806.

[16] Patrick Lancaster. *cse490r_18wi*. 2018. URL: https://gitlab.cs.washington.edu/cse490r_18wi/lab4.git.

[17] J. Le Ny, E. Feron, and E. Frazzoli. "On the Dubins Traveling Salesman Problem". In: *IEEE Trans. Automat. Contr.* 57 (Jan. 2012), pp. 265–270. DOI: 10.1109/TAC.2011.2166311.

[18] X. Ma and D. Castanon. "Receding Horizon Planning for Dubins Traveling Salesman Problems". In: Jan. 2007, pp. 5453–5458. DOI: 10.1109/CDC.2006.376928.

[19] V. Ojha, A. Abraham, and V. Snasel. "ACO for Continuous Function Optimization: A Performance Analysis". In: Nov. 2014, pp. 145–150. DOI: 10.1109/ISDA.2014.7066253.

[20] C.M. Pintea and D. Dumitrescu. "Improving ant systems using a local updating rule". In: Oct. 2005, 4 pp. ISBN: 0-7695-2453-2. DOI: 10.1109/SYNASC.2005.38.

[21] C.M. Pintea, P. Pop, and C. Chira. "The Generalized Traveling Salesman Problem solved with Ant Algorithms". In: *J Univers Comput Sci* 13(7)rem (Aug. 2017). DOI: 10.1186/s40294-017-0048-9.

[22] D.J. Rosenkrantz, R.E. Stearns, and P.M. Lewis II. "An Analysis of Several Heuristics for the Traveling Salesman Problem". In: *SIAM Journal on Computing* 6 (Sept. 1977), pp. 563–581. DOI: 10.1137/0206041.

[23] K. Savla, E. Frazzoli, and F. Bullo. "Traveling Salesperson Problems for the Dubins Vehicle". In: *IEEE Transactions on Automatic Control* 53 (Aug. 2008), pp. 1378–1391. DOI: 10.1109/TAC.2008.925814.

[24] A.M. Shkel and V. Lumelsky. "Classification of the Dubins set". In: *Robotics and Autonomous Systems* 34 (Mar. 2001), pp. 179–202. DOI: 10.1016/S0921-8890(00)00127-5.

[25] T. Stützle and H.H. Hoos. "Improvements on the Ant-System: Introducing the MAX-MIN Ant System". In: (Jan. 1996). DOI: 10.1007/978-3-7091-6492-1_54.

[26] A. Tanenbaum and M. van Steen. *Distributed Systems: Principles and Paradigms*. Vol. 3. Jan. 2002.

[27] Alan Turing. "Computing Machinery and Intelligence". In: vol. 59. Oct. 1950. DOI: 10.1093/mind/LIX.236.433.

[28] A.B. Walker. "Hard real-time motion planning for autonomous vehicles". PhD thesis. Swinburne University, 2011.

[29] X. Yu and J. Hung. "A genetic algorithm for the Dubins Traveling Salesman Problem". In: *IEEE International Symposium on Industrial Electronics* (May 2012). DOI: 10.1109/ISIE.2012.6237270.