Experimental Analysis of Heuristic Solutions for the Moving Target Traveling Salesman Problem Applied to a Moving Targets Monitoring System

Rodrigo S. de Moraes[a,b,*], Edison P. de Freitas[b]

[a] *Department of Computer and Information Science (IDA) - Linköping University - Sweden*
[b] *Graduate Program on Electrical Engineering - Federal University of Rio-Grande do Sul, (UFRGS) - Av. Osvaldo Aranha, 103 - Bairro Bom Fim, 90035-190 - Porto Alegre - RS - Brazil*

Abstract

The Traveling Salesman Problem (TSP) is an important problem in computer science which consists in finding a path linking a set of cities so that each of then can be visited once, before the traveler comes back to the starting point. This is highly relevant because several real world problems can be mapped to it. A special case of TSP is the one in which the cities (the points to be visited) are not static as the cities, but mobile, changing their positions as the time passes. This variation is known as Moving Target TSP (MT-TSP). Emerging systems for crowd monitoring and control based on unmanned aerial vehicles (UAVs) can be mapped to this variation of the TSP problem, as a number of persons (targets) in the crowd can be assigned to be monitored by a given number of UAVs, which by their turn divide the targets among them. These target persons have to be visited from time to time, in a similar way to the cities in the traditional TSP. Aiming at finding a suitable solution for this type of crowd monitoring application, and considering the fact that exact solutions are too complex to perform in a reasonable time, this work explores and compares different heuristic methods for the intended solution. The performed experiments showed that the Genetic Algorithms present the best performance in finding acceptable solutions for the problem in restricted time and processing power situations, performing better compared to Ant Colony Optimization and Simulated Annealing Algorithms.

Keywords: Ant Colony Optimization, Genetic Algorithms, Simulated Annealing, Artificial Intelligence, Moving Target Traveling Salesman Problem, Moving Targets
INSPEC:,

*Corresponding author
Email addresses: rodrigo.moraes@liu.se,rodrigo.moraes@ufrgs.br (Rodrigo S. de Moraes), edison.pignaton@ufrgs.br (Edison P. de Freitas)

1. Introduction

The well-known Traveling Salesman Problem (TSP) consists in finding the shortest path to visit a set of cities exactly once and return to the initial point (first visited city). Its specializations and generalizations have been a common and important topic of study for both mathematicians and computer scientists along the years, as many problems can be reduced to the TSP (Rajesh Matai & Mittal, 2010). With its first mentions in literature dating back to the nineteenth century, this routing problem has evolved with the technology around it. Nowadays TSP's are found as sub-problems of several other more general applications such as transportation infrastructure planning, logistics, PCI (Printed Circuit Board) projects, DNA sequencing, communication routing and even military intervention planning (Rajesh Matai & Mittal, 2010). These are just a small enumeration of the problems that are mapped to and need to deal with TSP in their solutions.

A great number of different approaches for solving the TSP has been proposed during recent decades (Osaba et al., 2016; Saenphon et al., 2014; Xing et al., 2008) . Most of these methods can be divided into heuristic and exact methods. Exact methods search for the optimal solution and can only optimally solve small problems since their approaches result in exponential computational complexities for large problems. Heuristic methods, however, trade optimality for speed, being satisfied by local optimal solutions presenting good results in solving large problems.

Recently, the usage of new technological assets such as Unmanned Aerial Vehicles (UAVs) and autonomous agents such as robots or autonomous vehicles (including autonomous UAVs) has exponentially increased since cheaper and more complete autonomous platforms have appeared on the market (Keller, 2015). This trend, allied to other technological developments in both software and hardware fields has originated a great number of new applications, some of which need to deal with TSP-like problems. Autonomous cars, for example, need to calculate the best route to drive to their destination given traffic conditions, a common example of a TSP usage in this new scenario.

Among other usages, autonomous agents have become an important tool to law enforcement forces, allowing them to track and follow targets or survey areas or buildings quietly, safely and almost undetectably. Amidst these uses, crowd control and monitoring applications for law enforcement present a set of interesting and complex characteristics that represents a special case of the TSP, Moving Target Traveling Salesman Problem (MT-TSP) (Helvig et al., 2003) or Time Dependant Traveling Salesman Problem (TD-TSP).

The MT-TSP is a generalization of the Traveling Salesman Problem (TSP) in which the cost to travel from each city to the next depends on the time spent since the tour has started, or the position of these cities on the tour. An even more special case, and the one that appears constantly in law enforcement applications is the one in which not only the cost to travel between cities changes with the time, but the cities, or targets, in this case, move with a given speed as time passes. An example of such a scenario would be the interception of

multiple evading targets on a crowd. Imagining that an electrical autonomous law-enforcement UAV has to intercept each evading target to identify it and do that as to minimize battery consumption, it is easy to recognize that this is a hard task as it is to find the best route to accomplish this multi-target interception mission. This special case of both TSP and TD-TSP is especially challenging because the position of each city (in this case a target) must be recalculated each time a new target is reached on a tour considered as a potential solution for the problem. Moreover, it must be done to all the tours considered as potential solutions, considerably increasing the computational cost to find such solutions.

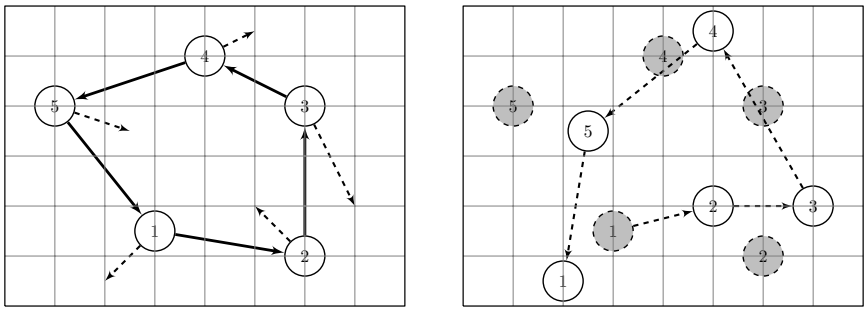
Considering the presented scenario, this work investigates different heuristic and artificial intelligence methods' performances on solving the MT-TSP. Moreover, a second contribution presented here is the development of a movement prediction module to be associated with the investigated heuristics, to compose a complete solution. The work reported here is part of a research on law-enforcement usage of autonomous vehicles for crowd control scenarios, and thus focuses its efforts on solving the MT-TSP for a small number of targets that move considerably slower than the traveler or interceptor.

The work is organized as follows: Section 2 presents a deeper discussion on Time Dependant and Moving Target Traveling Salesman Problems and the related works that served as inspiration for this research; Section 3 presents an overview of the meta-heuristic models implemented to solve the MT-TSP; Section 4 explains how the solution and the algorithms were implemented to solve the MT-TSP problem; in Section 5, the results of this proposed solution is presented, the different methods are compared and evaluated; Section 5 provides a detailed discussion on the results presented in the previous section and their possible application; Section 7, in turn, closes the paper presenting the conclusions and future work directions.

2. Moving-Target TSP

The Moving Target Traveling Salesman Problem (MT-TSP) is a generalization of the traditional Traveling Salesman Problem in which the targets can move with a given speed, including zero which represents the traditional or stationary case. The MT-TSP can be described as a problem in which, given a set $S = \{s_1, ..., s_n\}$ of n targets, each moving at a given speed v_i and a pursuer starting at the origin, and having maximum speed $v \gg v_i \forall i \in \{1, .., n\}$, find a tour which intercepts all targets (Helvig et al., 1998, 2003). The approximation complexity of MT-TSP was studied by Hammar & Nilsson (1999), where it was shown that it cannot be approximated better than by a factor of two by a polynomial time algorithm unless $P = NP$, even if there are only two moving points in the instance.

Figure 1 shows a graphical representation of an example of this problem. On the static case of the TSP the traveler, who is initially in city 1, calculates in instant t_0 the best path to solve the static problem (Figure 1a). The problem with MTTSP is, however, that in instant t_1, after the traveler leaves departing



(a) Intant t_0 (b) Intant t_1

Figure 1: An example of the Moving Target Traveling Salesman Problem

city (node 1), all nodes have moved and changed their positions. This fact requires a new tour to enable the traveler to intercept the nodes after these movements, where they have moved to (Figure 1b). As the nodes keep moving, each time the traveler leaves one node to next, this process must be repeated until it returns to node 1, which is in a new position (completely different from the departing one) at the end of the tour.

In order to solve this particularly complex problem, multiple heuristic-based approaches have been proposed in the literature. For instance, Choubey (2013) present a Genetic Algorithm approach to solve a set of TD-TSP problems. The results showed that the Genetic Algorithm implemented by the authors is more effective than solving the problem using a similarly modified greedy method, proving the advantages of the heuristic method to solve this kind of problem.

Abeledo et al. (2013) proposes a polyhedral basis Branch-Cut-and-Price (BCP) Algorithm that suggests polyhedral theory can play an important role in improving algorithms for the MT-TSP and TD-TSP. Performing studies on the TD-TSP polytopes their method proposes polyhedral cuts that can be used to modify the BCP approach allowing the solution of 22 TSP instances with up to 107 vertices.

A repetitive heuristic solution of the traveling salesman problem is used by Englot et al. (2013) to solve a TSP in which a large number of moving targets must be intercepted by a single agent as quickly as possible. The author's work compares the Lin-Kernighan heuristic (LKH) to a greedy heuristic over different problem parameterizations showing that the benefits of a non-greedy solution depend on the speed of the targets relative to the agent. The work proves that LKH is superior to greedy heuristics when the targets are moving at low speeds, and that its relative performance to greedy algorithms improves as sensor noise worsens.

3. Heuristic Methods to solve TSP

3.1. Genetic Algorithm Methods

Genetic Algorithms (GAs) (Holland, 1992) are a kind of evolutionary algorithms that try to simulate the concepts of natural Darwinian evolution to solve complex optimization problems. The concept behind these methods is to evolve an initial population of candidate solutions until one of the solutions on this pool or population is considered good enough to solve the desired problem.

The idea behind Genetic Algorithms is the application of a Survival of the Fittest principle on a population of individuals representing potential solutions to the problem being solved. By using this principle it is granted that only the fittest individuals are able to survive and mate. Such behavior leads the candidate solutions to evolve and become better, ultimately leading them towards the optimal solution to the problem, achieving very good results without performing an extensive brute force search in the solution space. As in real life, each individual on the algorithm has a unique genetic code that rules the characteristics of the specific candidate solution represented by each individual. Through a series of operations such as selection, crossover and mutation, the genetic code of the population is evolved for a certain number of generations, or iterations, until the algorithm decides to stop and selects the best solution from the pool of solutions it has created.

This process of selecting and evolving a solution starts by selecting a representation of the problem as to make sure that each candidate solution has its own representation. An initial population of candidate solutions is then created, usually at random even though heuristics can be used for that. Each candidate of the population is analyzed, it's fitness, or how well the candidate solves the problem, calculated and the generation sorted according to fitness values. The Survival of the Fitness principle is then applied, meaning that most fit solutions have more chance of breeding generating offspring to be part the next generation. According to this principle, two parents among the candidates are then chosen, it's genetic material combined into a new genetic code, and an offspring created. Next, mutation occurs. Some of these offspring are randomly selected to suffer mutations on its genetic code, creating new unique patterns on the population genetic pool. This process is repeated until a new generation is completely created. After that, the algorithm applies the Survival of the Fitness principle again on this new generation and all the crossover and mutation process repeats itself. The whole algorithm is iterated over and over until either a stop condition is found or the maximum number of generations exceeded. By the end of the whole process, the best solution to the problem is chosen amongst the pool of individuals created during the execution of the algorithm.

One of the most important characteristic of this kind of algorithm is its ability to avoid presenting poor local optima candidate solutions as definitive solutions until the algorithm is finished. It is very important that worst fit individuals still have a chance to mate, and not be discarded, being able to create useful genetic material that may contribute to the generation of better solutions than local optimal ones.

For comprehension purposes, a graphical representation of the steps and workflow of a genetic algorithm solution can be seen in Figure 2

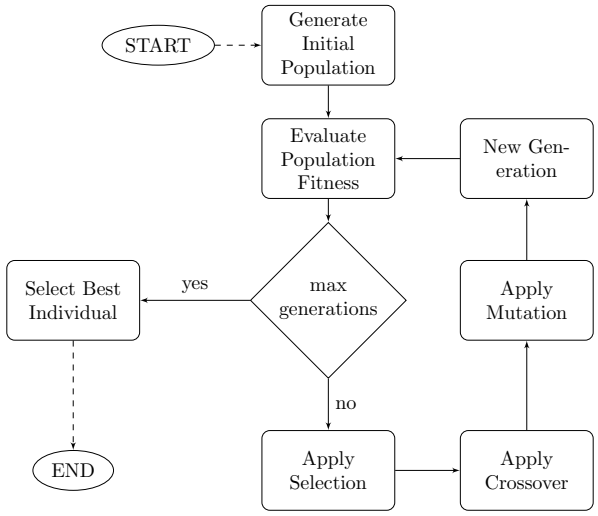


Figure 2: GA Fluxogram

3.1.1. Genetic Algorithms application in TSPs

Besides its simple concept, "pure" genetic algorithms were not designed to solve combinatorial problems. The main intent behind the development of this type of algorithm in the '60s and '70s was the optimization of numerical functions (Holland, 1992). Consequently, the original algorithm had to be modified to represent and handle combinatorial optimization problems like the TSP (Potvin, 1996).

The first problem in using GAs for any kind of optimization problem is choosing the correct representation method (Potvin, 1996) for the desired problem solution search space, ensuring that only valid and unique representations of each solution exist, and that a chosen representation method does not influence crossover, selection or mutation operations as to generate unsuitable results.

The second and third principal problems are related to the crossovers (Potvin, 1996; Üçoluk, 2002; Mendiratta & Goyal, 2014; Goldberg & Lingle, 1985) and mutation (Abdoun et al., 2012) phases of the algorithm. As previously illustrated on the text and in Figure 2, crossover, and mutations operators are the methods responsible for creating new solutions for the search space, ensuring that new and possibly better solutions are examined each time a new generation is processed. Their choice plays, therefore, an important role on the convergence of the algorithm and on the quality or optimality of the solutions that can be found.

The fourth problem deals with the selection problem (Shukla et al., 2015; Mohd Razali & Geraghty, 2011) meaning what strategy is used to choose individuals from a previous population to be parents and generate the offspring that will compose the next generation.

3.2. Simulated Annealing

Simulated Annealing (SA) is a heuristic optimization method based on the idea that the process of finding (near) optimal solutions for optimization problems is similar to the physical annealing process used to obtain low energy states on metals. The main concept behind a Simulated Annealing optimization algorithm is that the optimization search starts on a high energy state, accepting different candidate solutions whether they are near optimal, optimal or not good ones. As the process runs, however, the search algorithm starts to cool down, reducing its willingness to accept bad candidate solutions and focusing on the good ones.

Proposed by Kirkpatrick et al. (1983), Simulated Annealing is, thus, a stochastic method used to avoid local, non-global optimal results to be considered as optimal ones. This is done by accepting, based on a heuristic probability function, worst solutions that would otherwise be ignored and that may lead to optimal solutions once the local minimum point is transposed. However, during the process the chance of accepting these deteriorations is decreased, meaning that after a certain point the algorithm accepts that it has transposed local minimum and is getting closer to the global minimum, accepting only solutions that are better than the current on from this point on.

As for the process, the algorithm starts by generating a random or quasi-random candidate solution and performing local searches to optimize this initial solution iteratively. On initialization, after generating the first candidate solution, the algorithm links it to an initial temperature of T_0, which represents the degree of refinement of such solution, or its internal energy state. On each iteration, a given number n of local searches are performed on the candidate solution and accepted or not based on the acceptance probability for the current iteration, after that the current temperature T_i of the optimization process on iteration i is decreased. In case one of the local searches find a solution which is better than the current candidate, this improvement is automatically accepted and the improved solution becomes the current candidate. Otherwise, if the local search finds a solution x that is worse than the current candidate this solution is tested against a probability P_x of being accepted according to its evaluation $f(x)$ and the current temperature T_i as shown in Equation 1.

$$P_x = exp\left(-\frac{f(i) - f(x)}{T_i}\right) \tag{1}$$

The $f(i)$ on the term $f(i) - f(x)$ on Equation 1 stands for the evaluation of the current candidate, meaning that the probability of a worse solution x being accepted depends on how worse than the current solution it is.

The convergence of this type of algorithms to find near-optimal solutions for optimization problems has widely been proved in literature, some works, as

Dekkers & Aarts (1991), provide an extensive mathematical analysis of simulated annealing convergence. Such a convergent behavior is closely related to the decreasing exponential formulation of the probability P_x which decreases the chance of a poor solution being accepted exponentially as the temperature decreases, granting that at the end, the algorithm is just accepting the best solutions it finds. The initial temperature parameter is similarly important for the convergence once, if too low, can lead the algorithm to never accept a solution, and, on the other hand, if too high, leads the algorithm towards accepting lots of solutions, delaying convergence.

A graphical representation of the workings of a Simulated Annealing solution as described above can be seen in Figure 3

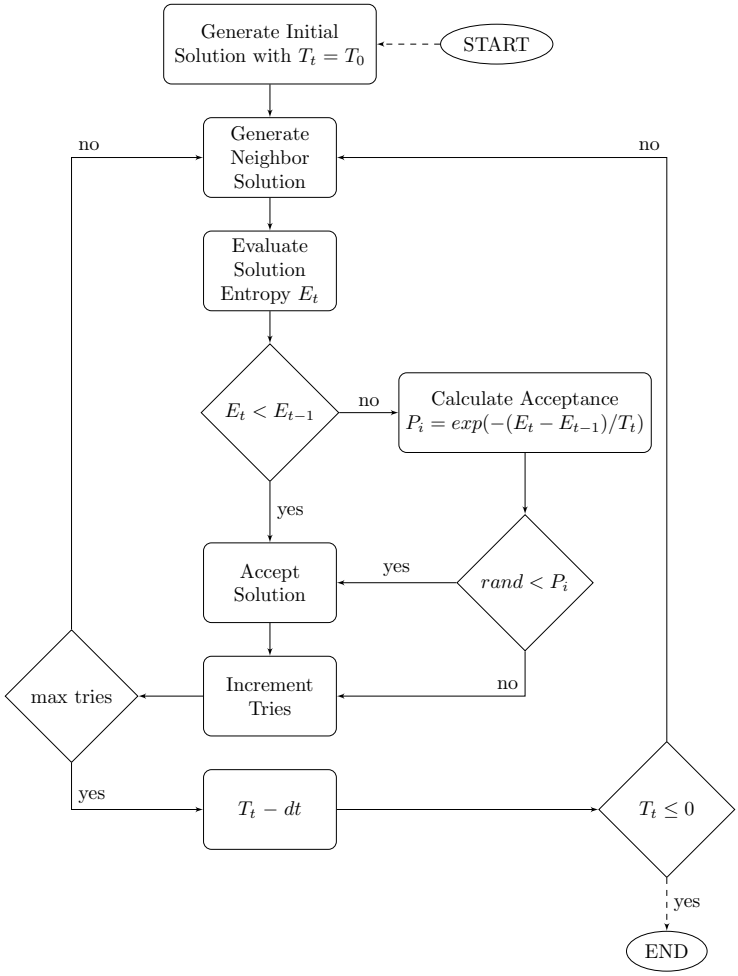


Figure 3: SA Fluxogram

3.2.1. Simulated Annealing application in TSPs

Similarly to genetic algorithms, simulated annealing (SA) approaches were first intended to solve generalized combinatorial problems, needing adaptations to correctly handle TSPs. The adaptations Simulated Annealing, require are, however, much simpler than the ones required by a Genetic Algorithm.

The process to use Simulated Annealing to solve the TSP problem does not differ much from the general case, the representation of the solution and the local search algorithm, however, must be correctly selected. As for the representation, a simple list of cities organized on the order they will be visited on the candidate solution tour is a simple, usual, and straight forward method representation method. The local search algorithm, however, presents a more complicated choice. Many local search algorithm may be used on a Simulated Annealing paradigm, between them, k-opt (2-opt and 3-opt) are among the most common, simple, and effective, not degrading the overall performance. Other local searches have, nonetheless, been reported in the literature, including even genetic algorithms (Xu, Mingji et al., 2017) being used as SA local search methods in more sophisticated approaches.

3.3. Ant Colony Optimization

Ant Colony Optimization (ACO) (Dorigo et al., 2006; Dorigo & Di Caro, 1999) is another biologically inspired approach to solve optimization problems. The paradigm behind this kind of solution is inspired by the communication between social insects, ants in this case, that cooperate with each other to reach their objective in nature. Because of their high cooperative societies, ants have inspired a great number of algorithms, most of them based on their communication mechanisms, based on the deposition of pheromones in the environment. In nature, ants communicate through the deposition of chemical markers on the ground (pheromones), the idea used by ACO, distributing virtual ants to perform searches on a set of candidate solutions and to indicate, through virtual pheromones, the way to find better solutions.

Following the natural inspiration, ACO is an iterative optimization algorithm in which a number of virtual ants build a set of candidate solutions to the problem, and communicate the quality of these solutions through the deposition of virtual pheromones. Interpreting the trails of pheromones, virtual ants of the next iteration have guidelines on how to build better candidate solutions and iteratively refine the search space until an optimal or quasi-optimal solution is found.

As a simple and very useful algorithm, ACO has been used in many fields and adapted to solve many different applications. Some of the most common variations of such algorithms are the Ant System (AS) (Dorigo et al., 1996) and the Max-Min AS (Stützle & Hoos, 2000), both widely used for optimization purposes.

Figure 4 presents a graphical representation of the algorithm described above.

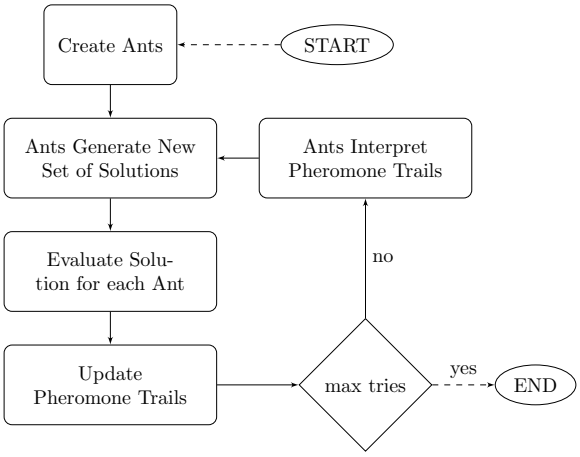


Figure 4: ACO Fluxogram

3.3.1. Ant Colony Optimization application in TSPs

As in the other methods, ACO also needed some modifications to handle the TSP. Besides GAs, ACO algorithms are amongst the most studied for TSP usage, therefore many applications and variations have been created to better approach the problem. The approach behind the usage of ACO in TSP is distributing the ants in the cities and let them walk around trying to complete a tour as if they were on the nature walking from their nest to food sources, and back. At the end of this exploration process, the ants deposit pheromones on the trails they took between the cities to perform their tours. The intensity of the pheromone an ant will deposit is directly related to the length of the tour they performed, the shorter the tour, the strongest the pheromone. On the next exploration round, the ants will attribute to the trails, or edges between cities, a probability of being followed that is directly proportional to the amount of pheromone on that edge. This system ensures that every edge has a chance to be followed, thus avoiding local minimal once, but creating a trend towards better solutions on every iteration. This common behavior to the solutions is graphically represented in the diagram of Figure 5.

Besides this common approach, the different algorithms (Stützle & Hoos, 2000; Tuba et al., 2013; Cheng & Mao, 2007) differ on how they attribute pheromones to the edges, on how they evaluate new solutions, and on how they create these solutions. Many algorithms also perform a mid-algorithm optimization, performing local search iterations on candidate solutions after an ACO iteration is concluded, optimizing the ants' paths, thus the pheromone trails, before a new generation of ants is let loose on the cities (Su et al., 20011).

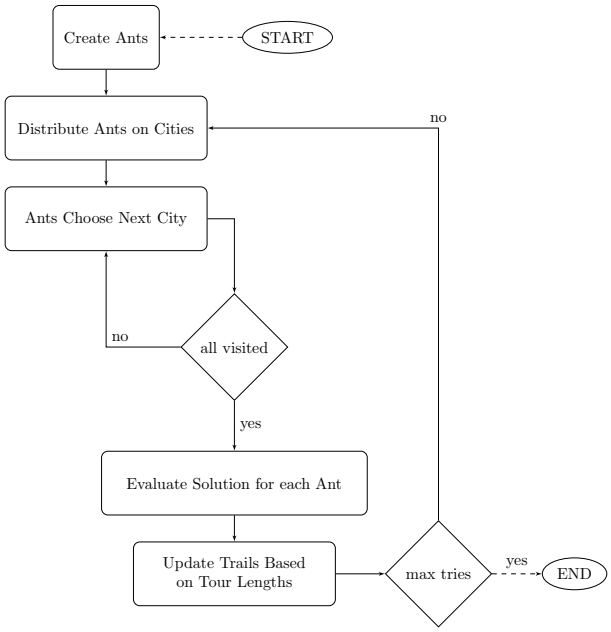


Figure 5: ACO for TSP Fluxogram

4. The Proposed Solution

Given a set $S = \{s_1, ..., S_n\}$ of targets, each s_i moving at constant velocity $\vec{v_i}$ from an initial position p_i, and given a pursuer starting at the origin and having maximum speed $v > |\vec{v_i}|$, this work tackles the problem of finding the shortest tour starting (and ending) at the origin (first target), that intercepts all targets. Mathematically, the cost function to be optimized is defined as follows:

$$\begin{aligned} \underset{D_t}{\textbf{minimize}} \quad & D_t = \sum_{i}^{n} \sum_{j}^{n} c_{ij} d_{ij} \\ \textbf{s.t.} \quad & \forall i \neq j \end{aligned} \tag{2}$$

$$\text{where} \quad c_{ij} = \begin{cases} 1; \text{ if salesman travels from target i to target, j} \\ 0; \text{ otherwise} \end{cases} \tag{3}$$

and d_{ij} is the distance between targets, i and j.

To solve the studied problem a hybrid method composed of a numerical integrator and movement predictor was implemented on the evaluation stage of

each of the heuristic methods presented in Section 3. This numerical approach allows the solutions and heuristic methods to correctly evaluate the length of tours by traveling them and calculating, after arriving at a new target or interception point, the next point it should go to in order to intercept the next target. This is done by making sure that all targets are intercepted and the tour is completed. The idea behind this algorithm is represented in Figure 6, in which an interceptor, or traveler, represented by the black circle on the figure, starts on time $t0$ by predicting the movement of two targets T1 and T2, and their estimated positions in time $t1$, in which it expects to intercept target T1, and after predicting the position of T2 in time $t2$, moment in which it predicts to intercept T2.

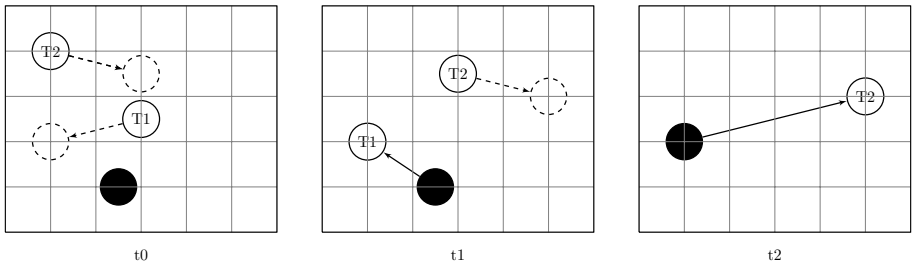


Figure 6: Behavior of an Interceptor Node predicting the movement pattern and intercepting two Targets T1 and T2

The internal workings, implementations, and design decisions of each heuristic approach, as well as the details on the movement prediction module, are presented in the following sections.

4.1. Numeric Movement Prediction Module

The movement prediction module works by taking as input a list of waypoints representing an initial position S_0 and the initial positions of the targets in time $t_0 = 0$ according to the tour order of the candidate solution under evaluation. Following from t_0 and S_0 the method applies numeric integration loop to find the interception position s_j of the next target. The process is repeated by each target on the waypoint list, being the parameters t_j and S_j updated at the end of each step, always calculating the interception point of the next target on the current tour time, and departing from the current tour position. In each of these iterative steps, the current length of the candidate solution is updated according to the distance between s_j and s_{j-1}, resulting, by the end, on the total tour length for the candidate solution. A diagram illustrating this approach is presented in Figure 7

The whole module is based on the idea that, for a given interceptor moving with speed $\vec{v_i}$, and a target moving from point $\vec{s_t}$ with speed $\vec{v_t}$, the interception point will be the one in which the interceptor, after moving for a period τ, finds

itself on the same point as the target. Mathematically, considering the position of the target as the origin, this situation can be written as:

$$\vec{v_i}\tau = \vec{v_t}\tau + \vec{s_t} \tag{4}$$

applying a scalar product operator to both sides of the equation in order to eliminate the vectorial properties of the terms, Equation (4) can be rewritten as:

$$(\vec{v_i}\tau) \cdot (\vec{v_i}\tau) = (\vec{v_t}\tau + \vec{s_t}) \cdot (\vec{v_t}\tau + \vec{s_t}) \tag{5}$$

which is the same as:

$$|\vec{v_i}|^2\tau^2 = |\vec{v_t}|^2\tau^2 + |\vec{s_t}|^2 + 2(\vec{s_t} \cdot \vec{v_t})\tau \tag{6}$$

or, reorganizing the terms:

$$0 = (|\vec{v_t}|^2 - |\vec{v_i}|^2)\tau^2 + |\vec{s_t}|^2 + 2(\vec{s_t} \cdot \vec{v_t})\tau \tag{7}$$

Solving equation 7 for τ and using this value in equation 4, the point of intersection between the interceptor and the target can be easily found. The algorithm presented on this work makes use of a simplified numerical evaluation loop to find the roots of Equation 7, a quicker solution than using square-root calculation methods, which are very resource consuming. Listing 1 shows the pseudo-code of the algorithm implemented in this module:

Listing 1: Numerical Interception Method

```
inputs: current_time, target_start_pos, interceptor_pos
        interceptor_vel, target_vel


dist ← ∞
dt ← ∞
ds ← ∞
next_target_pos ← target_start_pos + target_vel * current_time
while dist > 0
    ds = Distance(interceptor_pos,next_target_pos)
    dt ← ds/interceptor_vel
    prev_target_pos ← next_target_pos
    next_target_pos ← target_start_pos + target_vel * (current_time + dt)
    dist= Distance(prev_target_pos,next_target_pos)
return next_target_pos
```

4.2. The Solution Using Ant Colony Optimization

The implemented ACO method is based on the Min-Max Ant System (MMAS) approach (Stützle & Hoos, 2000). This method has the advantages of converging faster towards an optimal solution, especially when used in TCP problems, configuring a better choice than other ACO approaches for the problem tackled in

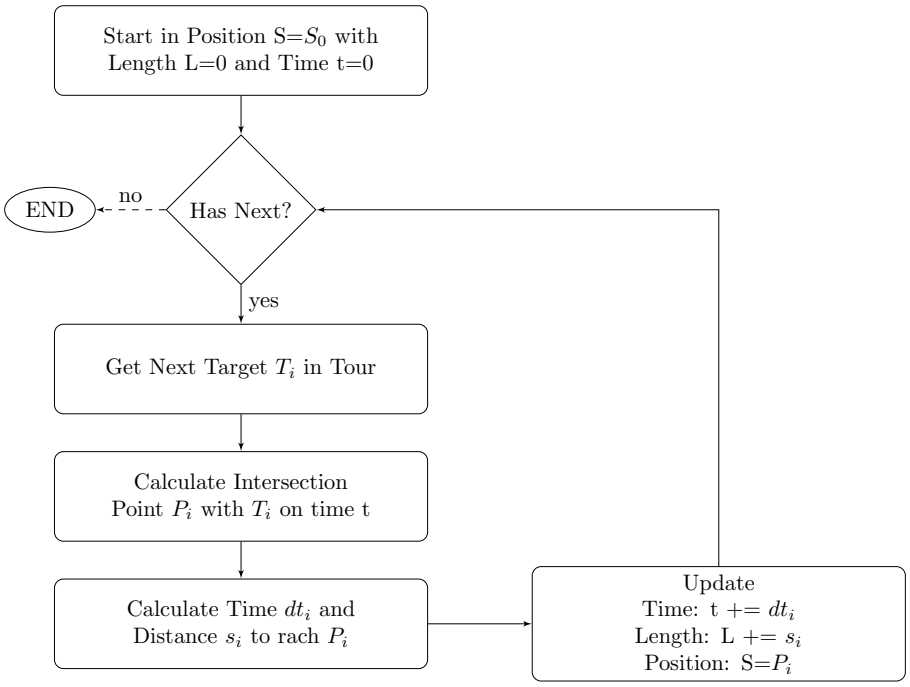


Figure 7: Evaluation Fluxogram

this work. The idea behind the algorithm is that at the end of each generation of the MMAS, only the best ant marks the environment with pheromones, a remarkable difference from other algorithms in which all ants mark their trails. Besides that difference, on this approach, the amount of pheromones deposited on each edge between targets is bound on the upper side by a max value, and on the lower side by a min value. The intensity M of the pheromone an ant deposits on edge e in time t is, thus, calculated in terms of its tour length L and the upper max and lower min bounds as described on Equation 8:

$$M_e(t) = \left[M_e(t-1)(1-\alpha)) + \frac{1}{L} \right]_{min}^{max} \tag{8}$$

where α represents a temporal decaying constant and the operator $[x]_{min}^{max}$ means that $M_e(t)$ will be bound assuming the min or max values in case it goes beyond these thresholds.

Following the idea presented in the diagram of Figure 5, each ant is distributed on a given target, or city, and chooses their next city based on the pheromone level of the edges leading to the cities it has not visited yet, accord-

ing to the following probability function:

$$P_i^j(t) = \frac{M_{ij}(t)\frac{1}{d_{ij}}}{\sum_0^n M_{in}(t)\frac{1}{d_{in}}} \tag{9}$$

meaning that the probability of an ant choosing to go to city j from city i among n cities that where not yet visited, is directly proportional to the pheromone level M_{ij} on the edge that links i to j, and inversely proportional to the distance d_{ij} between these cities.

Observing the mathematical behaviour defined by equations 8 and 9, it is easy to see that the values of *max* and *min* can have a significant influence on the exploratory and exploitatory behaviors of the Ant-Colony Algorithm. Since the pheromone level on any trail between two cities, or targets, is limited by the value of *max*, no trail will be extremely attractive for the ants, a behavior that could easily undermine the exploration of new solutions in the search space, since ants would always follow the path with the highest pheromone level. Similarly, the value of *min* limits the pheromone levels on trails and helps to avoid situations in which a given trail, or path, between cities is not attractive for ants, which would prevent them from exploring new routes.

A similar case can also be made for the effect of *max* and *min* in the exploitation behavior, meaning the investigation of solutions or routes that are in the vicinities of routes already preferred by the ants. The value of *max*, again, limits the pheromone levels as to not make a given trail extremely more attractive to the ants than others, while the value of *min* limits lets every trail to be at least a little attractive for the ants.

The relation between *max* and *min* is, however, the factor that defines if a given instance of the MMAS is going to work more towards the exploration or the exploitation side. If *max* is significantly bigger than *min*, ants will be more attracted to stay in the current best routes, or at least on the ones with stronger pheromone levels. In this case, if ants take detours, which will be rare, they will quickly be attracted again towards the current best path due to the high concentration of pheromones on that path, configuring an exploitatory behavior. If, on the other hand, the relation between *max* and *min* is small, ants will have more liberty to choose different paths since no path will be significantly more attractive than the others, staying away from the current best path for longer periods, configuring a more exploratory behavior. Listing 2 presents a pseudo-code of the implemented Ant-Colony algorithm:

Listing 2: Ant-Colony Algorithm

```
1  inputs: population_size, max_genertions, max, min,
2          α, max_cities
3
4
5  pheromone_map ← new map(max_cities)
6  for each trail in pheromone_map
7      trail_level ← max
8  gen ← 0
```

```
while gen < max_generations
    population ← new AntPopulation(population_size)
    best_tour ← empty
    for each ant in population
        visited ← empty
        while  Length(visited) < max_cities
            city ← ChooseNextCity(pheromone_map)
            if city in visited
                do nothing
            else
                visited ← city
        tour ← MakeTour(visited)
        route_length ← Length(tour)
        if tour_length ≤ Length(best_route)
            best_tour ← tour
    pheromone_map ← UpdateMap(best_tour ,max, min ,α)
    gen ← gen +1
end
```

4.3. The Solution Using Simulated Annealing Optimization

The Simulated Annealing algorithm implemented for this TSP problem is quite straightforward. At the beginning a candidate tour is randomly generated, and, as the simulation goes, a local search algorithm is used to generate a new solution. This new solution may or may not be accepted according to the criteria previously presented in both Figure 3 and Equation 1.

The local search method chosen for this implementation was a simple 2-opt method, a method that takes two edges between targets and swaps their extremities, keeping the tour closed, but still generating a neighbor candidate solution similar to the initial one. The main idea behind a 2-opt, or any k-opt local search algorithm, is to take a route that crosses over itself and reorder it so that the crossing disappears, potentiality improving the solution.Listing 3 presents a pseudo-code of the implemented algorithm:

Listing 3: Simulated Annealing Method

```
inputs: max_retries , delta_temp , temp_threshold
        max_temp

temperature ← init_temp
solution ← new random solution
retry ← 0
while temperature > temp_threshold
    while retry < max_retries
        candidate ← 2_Opt(solution)
        if Fitness(candidate) ≤ Fitness(solution)
            solution ← candidate
        else
            entropy ← −(Fitness(candidate)−Fitness(solution))
```

```
            chance ← Exp(entropy/temperature)
            if Random() ≤ chance
            solution ← candidate
        retry ← retry + 1
    temperature ← temperature − delta_temp
    retry ← 0
end
```

4.4. The Solution Using Genetic Algorithm Optimization

The Genetic Algorithm implemented to solve the TSP problem on this work is based on a Path Representation (Potvin, 1996) of the tours. For each generation, a certain number of solutions are chosen according to a crossover rate cr, to act as parents for the new solutions that will compose the next generation. The crossover rate controls the the number of solutions that will be created as a crossover between the parents, defining also the amount of totally new solutions that will be generated at random to be part of the population of the next generation. The selection of the parents is made by a Rank Based Roulette Wheel Selection (Mohd Razali & Geraghty, 2011), while the crossover is done by a PMX Crossover Operator (Goldberg & Lingle, 1985). Finally, the new individuals, or new candidate solutions, on the population are mutated, using an RSM Mutation Operator (Abdoun et al., 2012). The mutation occurs according to a given mutation rate mr that defines the rate of occurrence of mutations, meaning how many solutions from the candidate population will suffer a mutation. Listing 4 presents a pseudo-code of the implemented algorithm:

Listing 4: Genetic Algorithm Method

```
inputs: population_size, max_generations,
        crossover_rate, mutation_rate

gen ← 0
population ← new random population(population_size)
Evaluate(population)
while gen ≠ max_generations
    pop ← new empty population
    for i in range(0,population_size)
        if Random() ≤ crossover_rate
            parents ← RBRW_Selection(population)
            children ← PMX_Crossover(parents)
            pop ← children
        else
            pop ← new random individual
    for each individual i in pop
        if Random() ≤ mutation_rate
            RSM_Mutation(individual i)
    population ← pop
    Evaluate(population)
    gen ← gen + 1
```

```
select best solution in population
end
```

The chosen Path Representation (Potvin, 1996) is considered to be the most classical and natural representation for solutions on Genetic Algorithms, it works by representing tours as a list of the cities ordered as they appear on the tour. The crossover operators based on this representation typically generate offspring that inherit the relative order of the cities. This representation, however, has to be checked for uniqueness, since each tour can be represented in $2n$ distinct ways because any city can be placed at position 1.

A Rank-Based Roulette Wheel Selection (Mohd Razali & Geraghty, 2011), in turn, is a strategy in which the probability of an individual in the roulette wheel is based on its fitness rank relative to the entire population. The rank-based selection method first sorts individuals according to their fitness and then calculates selection probabilities according to their ranks rather than fitness values. This method is considered one of the best methods for selection in GA, because it is able to maintain a constant and adjustable pressure in the evolutionary search, controlling convergence. The downside of this method is, however, that it requires a sorting algorithm to be put in place, which can become time-consuming as the number of cities increases. As the algorithm developed on this work is intended to be used in real-life-like problems with a limited number of cities, the used of a Rank Based solution is justified once the sorting will not be too time-consuming.

In the Reverse Sequence Mutation (Abdoun et al., 2012) operator, a random sequence S is taken from a tour and the genes in this sequence are copied and then inversely placed on the sequence again. Besides simple, the RSM mutation operator has shown promising results in terms of convergence when compared to other methods (Abdoun et al., 2012).

In turn, the PMX crossover (Goldberg & Lingle, 1985) is a crossover operator method created to perform one point crossovers on Path Representations. Taking two parents $p1$ and $p2$, PMX constructs offspring by changing the positions of the cities on $p1$ one by one, with the cities on $p2$ that occupy these same positions. However, in order to keep the solution valid, the cities in these positions are not just overwritten, they are swapped. This way, to set position i to city c, the city in position i and the city c swap positions, granting that all cities are on the tour and that no city is repeated.

5. Experimental Evaluation

5.1. Methodology

A set of experiments has been specifically designed to evaluate the performance of each optimization method to solve the problem. Once in real life situations the interception algorithm will mostly be used to intercept a limited number of targets, and not several hundreds of them, each method was tested against sets of 8, 12, 16, 20, 32, 64 and 100 randomly placed targets,

with speeds ranging from 0 m/s to 2 m/s in 0.5m/s steps (a reasonable range for human walking speed). The speed of the interceptor, or traveler, was set to 20m/s throughout all the simulations (a reasonable speed for a small surveillance UAV). Table 1 summarizes this general parametrization.

Table 1: Tour Length Performance Testing Parameters

Number of Targets	8,12,16,32,64,100
Interceptor Speed	20 m/s
Target Speeds	0 m/s, 0.5m/s 1m/s, 1.5 m/s, 2 m/s
Target Distribution Area	Square 600m x 600m

In order to guarantee the best possible and homogeneous results for posterior statistical analysis, each one of the designed experiments was run 100 times. Of these 100 executions, the first two on each set were discarded since a significant execution time deviation was observed in these runs. Further analysis showed that these anomalies were due to hardware and resource issues related to variable initializations and caching of target data. The same target distribution was used for all the executions of a same experiment for all three different algorithms, providing a uniform dataset that can be used to compare the algorithms against each other.

In the interest of analyzing different aspects of the algorithm, such as their behavior, how they compare to each other and how parameterization affects their performance, two different types of experiments were performed. The first category of experimentation aims to analyze how the different metaheuristic approaches compare, evaluating aspects such as tour length, execution time and solution convergence. The second category, on the other hand, is focused on studying the effects of parametrization on the behavior of the metaheuristics, inspecting how different choices in both the exploration and exploitation parameters of the algorithms can influence their performance.

Comparative Performance Experiment: The objective of this category of experiments is to provide a comparative assessment of the different heuristics, evaluating them against each other in terms of tour length performance, solution convergence, and execution time. To perform this comparison, each algorithm was exposed to 30 experiments, each one covering a different pairing of target speed and target count. In addition, in order to provide a common ground for comparison, each of the algorithms was set to limit its search space to a total maximum of 10,000 solutions. Limiting the exploration space allows for a direct quantitative comparison of the convergence behavior of the algorithms, highlighting the tipping points in which performance degradation occurs for each of them. The results of these experiments are prensented on subesection 5.2.1

Parameterization Performance Experiment: The parameterization experiments were designed to study the effects of parameterization on the behavior

of the metaheuristics. All the experiments in this category were run with 32 targets, which was chosen because it is neither too small nor too large as to hide the effects of the parameterization from the results. A smaller number of targets would have been a poor choice, since the search space is too small and the solution would converge too quickly, masking the results. A bigger number of targets would make the algorithms converge too slow, also preventing the results to be seen.

5.2. Experimental Results

5.2.1. Comparative Performance Evaluation

The first results of the comparative experiments are shown in Table 2, which presents the statistical analysis of the tour length performance of the algorithms gathered from 98 executions of each of the 30 experiments run for each metaheuristic. The results from this table is supported by the box-plot graphic in Figure 8, which presents a visualization of these trends for the 8,16, 32 and 64 target cases across 3 different speed values (0m/s,1 m/s and 2m/s). Similarly, statistics for the execution time of each of the experiments are presented in Table 3 and are supported by the graphic in Figure 9, which brings the visualization of these trends for the 8,16, 32 and 64 target cases across 3 different speed values (0m/s,1 m/s and 2m/s).

In addition to the graphics presented in Figures 8 and 9, Figure 10 presents a visualization of the results for the scenarios with 32 targets and 3 different speed values (0m/s,1 m/s and 2m/s). This figure was included since with 32 targets the trends of both tour length and temporal performances can be easily visualized, serving as a visual baseline for the comparisons. The figure also serves as a control for the experiments discussed in subsection 5.2.2, which were also run with 32 targets.

Tour Length Performance: Analyzing the results obtained from the experiments the tour length performance of the algorithms seems to follow a general trend: scaling with both the number of targets and their speeds. It can be seen that, for the stationary target case, meaning the case in which the speed of the targets is set to zero, the genetic and the ant-colony approaches, present the best results in terms of tour length for small numbers of targets. However, as the number of targets increase, ant-colony seems to have an edge on the genetic approach, providing better average results for most of the stationary target experiments. It is visible, however, that the variability of these results scales significantly for the ant-colony approach as the number of targets increases. On the other hand, the genetic algorithm variability is not observed to scale as much with the number of targets, indicating a more convent behavior, even if towards a local optimum, for this algorithm as the number of targets increases. The simulated annealing approach, contrastingly, performs poorly even for a small number of stationary targets, presenting results on average 30% to 50% worse

Table 2: Tour Length Performance of the Heuristic Algorithms (m)

	8 targets			12 targets			16 targets		
	Min	Avg	Std Dev	Min	Avg	Std Dev	Min	Avg	Std Dev
Genetic Algorihm									
0.0m/s	1425.9	1560.5	136.6	1612.7	1748.2	87.1	1721.0	1849.1	90.3
0.5m/s	1535.7	1614.3	91.9	1812.2	1937.6	73.5	1710.9	1881.7	117.5
1.0m/s	1339.4	1393.0	58.1	1677.9	1806.0	86.1	1863.4	2094.2	165.7
1.5m/s	1247.1	1311.7	57.8	1769.4	1936.7	118.8	2362.9	2604.9	160.1
2.0m/s	1297.2	1396.0	100.3	2205.7	2457.1	204.1	2330.9	2878.7	247.0
Simulated Annealing									
0.0m/s	1535.6	2286.1	379.6	1691.7	2434.2	363.6	1971.3	2882.8	495.2
0.5m/s	1594.3	2282.7	248.9	1944.3	2794.1	389.1	1807.8	2753.1	454.9
1.0m/s	1399.3	1993.0	292.5	1732.5	2664.8	500.6	2065.6	3159.6	619.7
1.5m/s	1285.4	1874.1	304.9	1936.2	2863.8	510.7	2602.2	3954.1	777.1
2.0m/s	1297.2	1998.7	333.7	2301.7	3730.1	828.9	2669.5	4964.6	1253.9
Ant Colony									
0.0m/s	1425.9	1433.0	17.1	1612.7	1678.6	58.0	1721.0	1856.5	78.9
0.5m/s	1535.7	1539.0	8.0	1812.2	1863.2	47.7	1710.9	1878.1	79.0
1.0m/s	1339.4	1349.2	12.1	1677.9	1769.7	50.3	1863.4	2086.0	130.2
1.5m/s	1247.1	1247.1	0.0	1769.4	1875.4	64.4	2367.0	2625.7	119.5
2.0m/s	1297.2	1309.7	25.2	2223.5	2410.3	121.8	2422.3	2949.6	247.2

	32 targets			64 targets			100 targets		
	Min	Avg	Std Dev	Min	Avg	Std Dev	Min	Avg	Std Dev
Genetic Algorihm									
0.0m/s	2511.9	2812.5	139.0	4883.4	5457.9	198.9	8212.3	9001.8	271.9
0.5m/s	2535.1	3037.4	185.8	6371.7	7390.2	388.0	14127.1	15916.7	757.3
1.0m/s	3055.9	3753.2	266.7	8066.4	9564.7	558.8	18337.8	22021.4	1532.8
1.5m/s	3819.0	4667.8	363.2	10986.6	14785.7	1249.5	31717.5	42673.1	4976.4
2.0m/s	3651.8	5374.0	705.0	14209.6	18154.2	1850.8	42854.2	63840.8	9045.3
Simulated Annealing									
0.0m/s	2887.5	4088.7	465.2	5412.7	7136.5	659.0	8810.1	10744.0	748.8
0.5m/s	3101.9	4519.0	627.6	6022.1	8395.8	950.5	10669.1	14141.2	1251.2
1.0m/s	3650.3	5178.2	929.1	6513.3	10368.1	2294.8	13582.1	19701.3	3353.2
1.5m/s	4254.6	6862.4	1450.2	9934.4	17446.7	4756.1	20300.8	42226.3	11040.0
2.0m/s	4897.0	8975.8	2727.4	12335.0	24112.8	7187.1	31352.6	78158.7	27226.6
Ant Colony									
0.0m/s	2514.2	2900.6	153.0	4599.8	5301.5	300.8	6883.0	7937.7	410.2
0.5m/s	2797.3	3236.9	186.7	6685.5	8584.1	647.8	15755.2	20361.1	1958.2
1.0m/s	3534.2	4486.5	330.5	11014.3	16774.7	2079.9	39991.2	62766.9	11714.4
1.5m/s	4569.1	6078.4	564.7	21841.6	39526.5	7699.6	201434.0	431994.5	114257.6
2.0m/s	6115.2	8893.8	1125.5	39162.4	97441.0	27238.0	520547.9	1512037.2	697691.5

than the other approaches. Simulated annealing also presents more variability in the results, indicating a poor convergence towards an optimal solution. This behavior can be visualized by observing the SA boxes in the box-plot graphs of Figure 8, which show a higher median and a way higher spread for the tours predicted with the simulated annealing, for all situations of the stationary case (first column) when compared with the other two methods.

As the speed of the targets starts to increase, interestingly, the tour length does not follow the same pattern, even decreasing in some situations. This curious behavior is due to the movement pattern of the targets, that, in some situations, may generate shorter tours, getting closer to each other, or moving as to align with each other in some manner, generating shorter tours, a tendency observed across all 3 algorithms. The speed parameter also affects significantly the performance of the ant-colony algorithm, which was the best performer for the stationary case. For the moving target case, however, a significant perfor-

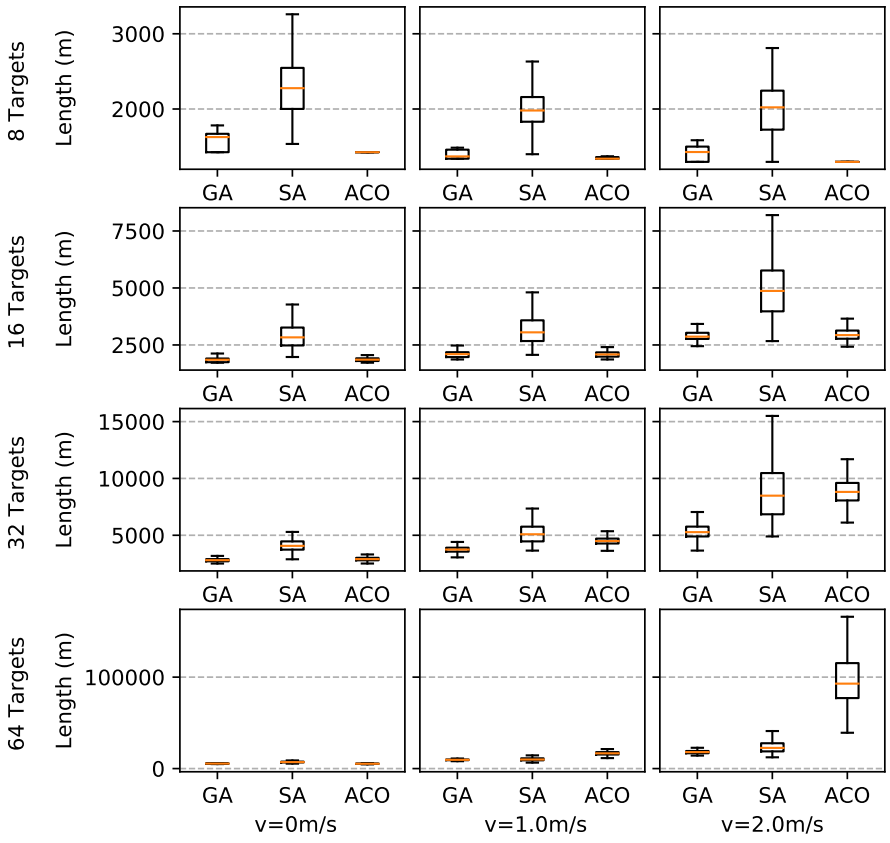


Figure 8: Tour Length for 8,16,32 and 64 targets across multiple target speeds

mance decrease is observed as targets start to move faster, especially for higher target counts. Simulated annealing is, as expected, as poor a performer as in the stationary case, however, interestingly enough, it manages to beat the ant-colony algorithm in higher speed and for high target counts, highlighting the extremely low performance of the latter in these situations. The genetic algorithm, on the other hand, seems not to be affected too much by the increments in target speed, still managing to be the best of the 3 algorithms in terms of its tour length performance. This last method also presents the least variable results among the algorithms across all experiments with moving targets.

Execution Time Performance: The results in Table 3 and Figure 9 present the simulated annealing method as the fastest method, usually running in half the time taken by the genetic algorithm, which is the second fastest

Table 3: Temporal Performance of the Heuristic Algorithms for 8-100 targets (time in ms)

	8 targets		12 targets		16 targets		32 targets		64 targets		100 targets	
	Avg	Std Dev	Avg	Std Dev	Avg	Std Dev	Avg	Std Dev	Avg	Std Dev	Avg	Std Dev
Genetic Algorithm												
0.0m/s	35.8	8.8	50.0	9.4	63.4	11.2	114.4	14.2	202.9	19.1	302.4	24.6
0.5m/s	46.9	3.6	66.5	6.2	82.9	4.9	127.1	8.2	182.7	6.4	251.1	2.2
1.0m/s	48.2	3.8	61.5	5.0	78.0	5.6	110.9	3.6	178.4	7.5	273.2	2.9
1.5m/s	47.0	4.8	64.3	5.8	77.5	5.2	110.1	2.9	190.9	2.5	297.9	3.3
2.0m/s	49.7	3.9	65.6	4.6	76.5	5.8	118.2	5.0	205.1	1.9	319.0	3.6
Simulated Annealing												
0.0m/s	16.2	1.7	22.9	2.2	28.9	3.1	49.7	5.5	90.7	8.1	121.7	14.2
0.5m/s	22.9	2.5	32.3	2.7	40.3	4.2	62.6	5.1	88.1	2.3	119.1	1.2
1.0m/s	23.2	2.1	30.4	2.8	36.4	3.7	54.8	2.7	83.6	1.5	128.8	1.6
1.5m/s	22.7	2.2	29.8	2.9	38.4	3.2	55.6	3.0	92.3	1.1	142.3	2.0
2.0m/s	24.4	2.4	32.6	3.0	38.2	2.5	58.7	2.6	99.3	1.3	152.3	2.0
Ant Colony												
0.0m/s	39.2	3.7	79.3	6.4	130.9	10.7	438.7	38.9	1452.8	249.0	3123.9	440.2
0.5m/s	45.8	5.0	86.9	6.9	126.4	9.1	365.7	23.4	1133.3	65.6	2766.6	13.8
1.0m/s	46.5	3.8	82.7	6.6	118.4	9.4	308.4	18.0	1076.1	6.3	2767.9	133.6
1.5m/s	43.3	3.9	77.7	7.1	112.7	10.3	300.2	6.7	1090.3	5.4	2798.0	24.0
2.0m/s	45.7	4.2	79.1	6.4	110.5	5.8	304.0	6.4	1099.8	6.3	2803.5	19.1

method. The ant-colony algorithm, on the other hand, is on average the slower of the three for almost every scenario, presenting an especially degraded execution time performance for the scenarios with higher target counts. In its turn, the genetic algorithm manifests a constant trend on its temporal behavior across the experiments, demonstrating a slight increase as the targets get faster, and a more accentuated, but still controlled, relation to the number of targets.

Remarkably, the ant-colony approach presents a curious behavior when it comes to its performance in the stationary case, showcasing an extremely high execution time for this kind of situation. Interestingly, the genetic algorithm also presents a different behavior for the stationary case, exhibiting a slightly increased variability in the execution times for this scenario, something not observed for the scenarios in which the targets are moving.

5.2.2. Evaluation in the Impact of Parameter Variation

This section presents the results of the study of parameterization effects on the behavior of the metaheuristics. Figures 11 to 14 present the parameterization effects on the genetic algorithm performance, while Figures 16 to 17 present similar results for the Ant-Colony approach. Figure 15, in turn, illustrates the effects of parameterization on the Simulated Annealing algorithm.

All experiments in this section were run for 32 targets and the target distribution is the same for all algorithms. Only the results for targets speeds of 0m/s, representing the stationary case, and 2m/s, representing the moving target case are presented, other speeds for the moving target case manifest the same trend of the 2m/s case and were, thus, omitted.

Genetic Algorithm Parameterization: Figure 11 presents the relation between the number of generations, the number of individuals per generation and the tour length performance of the genetic algorithm for the stationary case

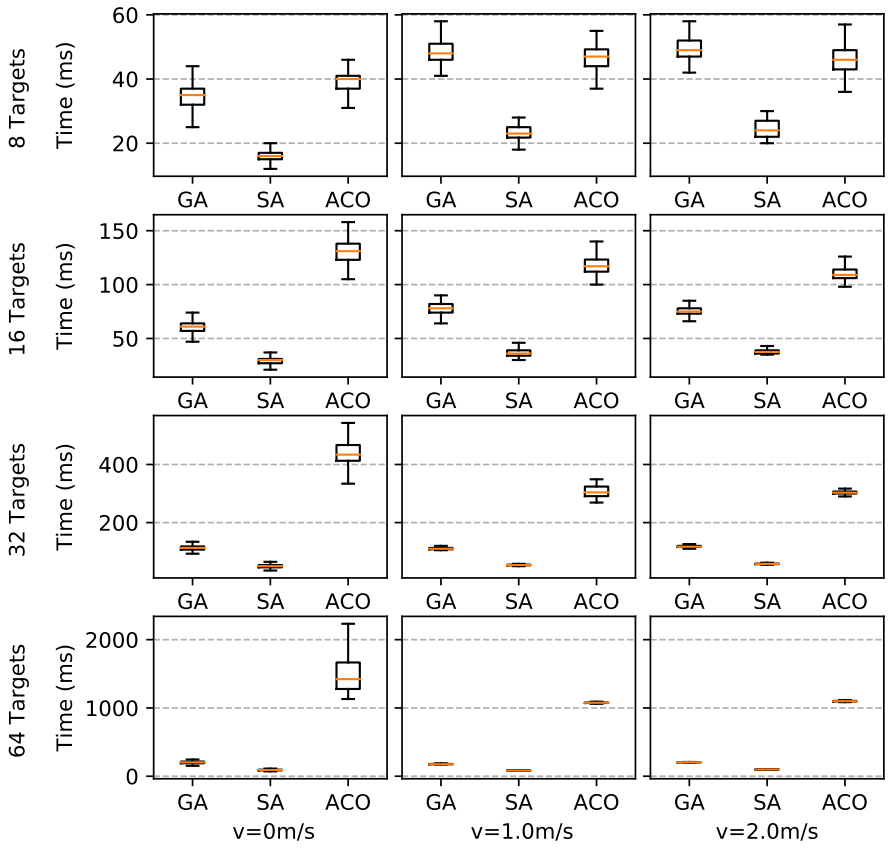
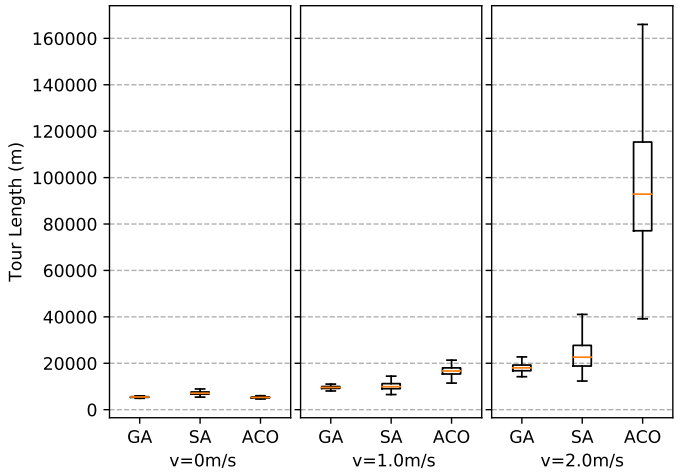


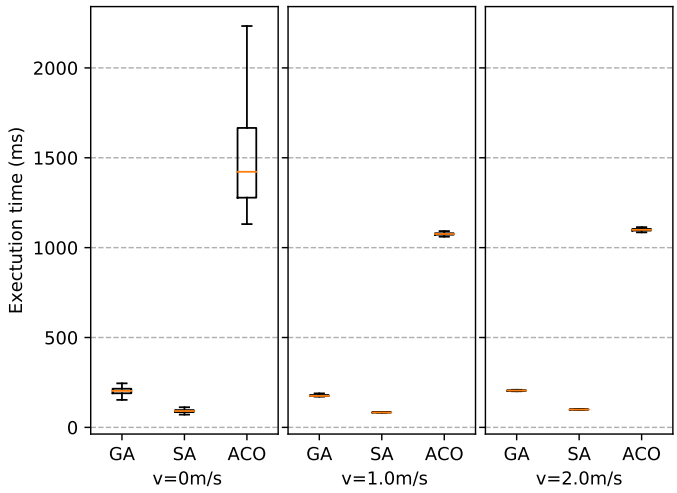
Figure 9: Execution Time for 8,16,32 and 64 targets across multiple target speeds

(Figure 11a), and for the moving target case (Figure 11b). As can be seen on these graphs, the tour lenght performance is very poor when both the number of generations and the population of each generation is small. This behaviour, however, shifts as both parameters increase. The number of generations rule the quality of the solution in the beginning when it is too small, the number of individuals on the population, however, starts to have a more significant effect on the results as the number of generations increase. The results also show that it is possible to obtain good results results just by having a high generation number and a small population size, to obtain the best possible results, however, the population size must be significantly augmented.

The execution time of the algorithm, as expected, increases both with the generation size and the population size. Both parameters govern the amount of exploration to be performed by the algorithm in the search space and seem to



(a) Tour Length

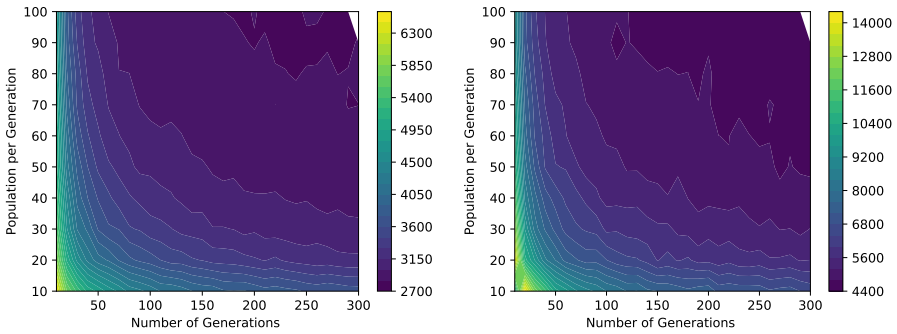


(b) Execution Time

Figure 10: Execution Time and Tour Length for 32 targets across multiple target speeds

have a similar influence on the execution time, as can be seen on Figure 12.

Figure 13, in its turn, presents the influence of the crossover and mutation rate parameters of the algorithm. Crossover rate mainly defines the number of new solutions that will be created as a crossover of the existing ones in a new generation, with the other individuals of the population being created randomly. This parameter contributes, thus, to both the exploration and exploitation rates



(a) Target Speed:0m/s (b) Target Speed:2m/s

Figure 11: Impact of the Number of Generations and Population Size Parameters in the Tour Length Performance of the Genetic Algorithm (m)

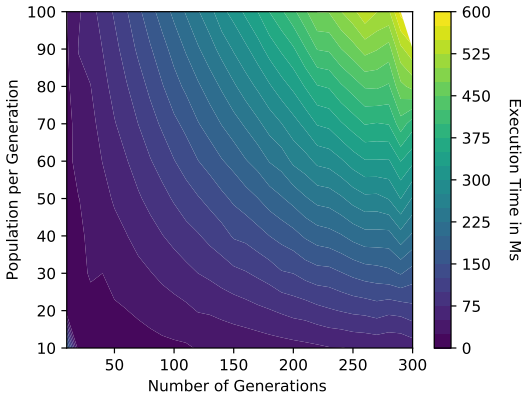
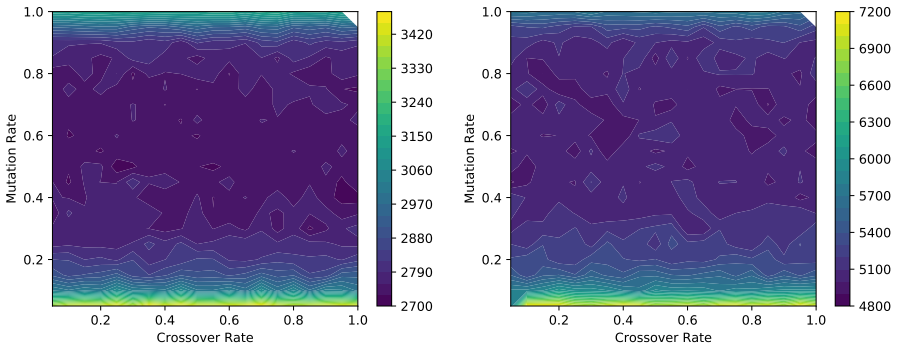


Figure 12: Impact of the Number of Generations and Population Size Parameters in the Execution Time Performance of the Genetic Algorithm (ms)

of the algorithm, meaning the rate in which new solutions are explored and the rate in which solutions in the neighborhood of previously visited points are searched. Mutation rate, on the other hand, contributes more to the exploitation rate of the algorithm, defining how t solutions that are in the neighborhood of already explored solutions are visited.

The results gathered from the experiments show that, apart from the case in which a really small crossover rate is used, the mutation rate is the parameter that defines more significantly the performance of the algorithm in terms of the tour length performance, exhibiting the best results when its value is around 0.6 or 60%. However, the trend observed in Figure 14 shows that the crossover



(a) Target Speed:0m/s (b) Target Speed:2m/s

Figure 13: Impact of the Crossover and Mutation Rate Parameters in the Tour Length Performance of the Genetic Algorithm (m)

rate has a significant impact on the execution time of the genetic algorithm. The mutation rate, on the other hand, does not have a considerable impact on the execution time, still slowing the execution sown, but not by such expressive amounts as the crossover rate.

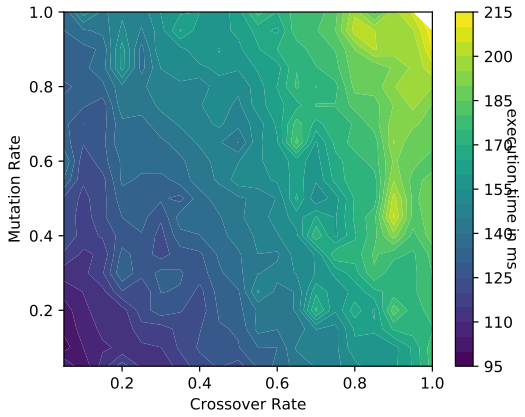
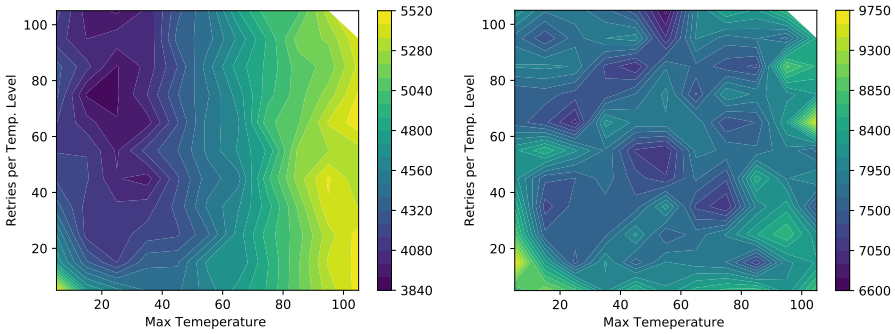


Figure 14: Impact of the Crossover and Mutation Rate Parameters in the Execution Time Performance of the Genetic Algorithm (ms)

Simulated Annealing Parameterization: Figure 15 shows the result of varying the initial or maximum temperature and the number of tries in the Simulated Annealing algorithm. Apparently, for the stationary case, in which the speed of the targets is 0m/s (Figure 15a), increasing the initial temperature

parameter seems to degrade the tour length performance of the algorithm. For the non-stationary case (Figure 15b), however, this effect, besides still noticeable, is less meaningful. The number of retries, on the other hand, seems to benefit the tour length performance as it increases, especially for the stationary case.

As both parameters influence the size of the search space, the execution time of simulated annealing algorithm scales with both of them, being smaller when both parameters are small, and increasing as the parameters are increased.



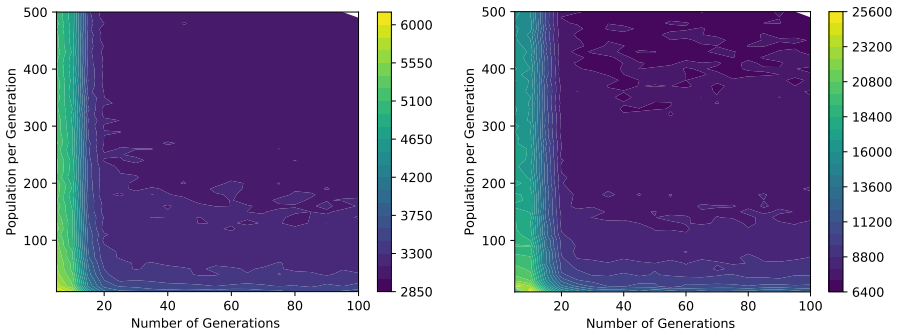
(a) Target Speed:0m/s (b) Target Speed:2m/s

Figure 15: Impact of the Maximum Temperature and Retry Count in the Tour Lenght Performance of the Simulated Annealing Algorithm (m)

Ant Colony Parameterization: The effects of population size and generation number in the Ant-Colony algorithm are represented in Figure 16 for the stationary (Figure 16a) and non-stationary (Figure 16b) cases. As expected, the quality of the tour length performance scales with both parameters. The generation number, however, has a considerable impact on the quality of the solution when it is smaller than 20 (meaning the algorithm is run for 20 generations). This trend shifts when the number of generations is larger than 20, meaning that, from this point on, the population size of each generation is the parameter ruling the tour length performance of the solution. It is, however, noticeable that good results can be obtained for both cases, stationary and non-stationary, with relatively reduced populations, while optimal, or the best possible results, require a considerably bigger population.

The execution time of the ant-colony algorithm scales both with the population size and the generation number. With neither of them presenting a particularly significant influence on the execution time. A graphic representation of this behavior was omitted since visually, the curves are very similar to those of the Genetic Algorithm, presented in Figure 12.

The Maximum and Minimum parameters of the *min max* Ant-Colony approach define the minimum and maximum bounds for the amount of pheromone



(a) Target Speed:0m/s (b) Target Speed:2m/s

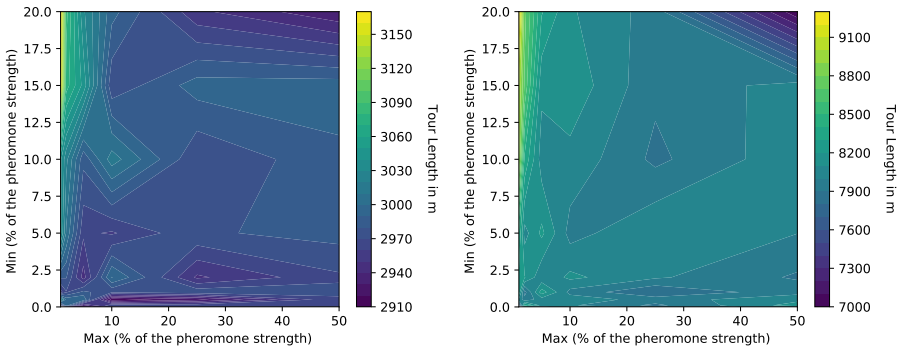
Figure 16: Impact of the Number of Generations and Population Size Parameters in the Tour Lenght Performance of the Ant Colony Algorithm (m)

in any trail between two targets in the search space. Changes on these parameters do not influence the execution time of the algorithm since no extra operations are added or removed when these parameters are modified. The values of Min and Max influence, however, the tour length performance of the algorithm. Figure 17 shows a graphical representation of the effects of these parameters on the tour length, Min and Max are represented on the figure as percentages of the maximum pheromone values an ant can deposit each time it goes through a path. For moving targets (Figure 17b), the best results are achieved when both Min and MAX are high. For the stationary case(Figure 17a), on the other hand, the best scenario is found when the Min value is equal to about 0.5% of the pheromone strength and the max value rounds 10%-15%. A zoom in graph highlighting this behavior is seen in Figure 18.

6. Discussion

The results presented in Section 5 show that each of the metaheuristics exhibits good performance in one or more criteria. The Genetic Algorithm is the better performer when it comes to the tour length performance across both the stationary and non-stationary scenarios, achieving near-optimal results for most of the stationary cases, and the best results among the three algorithms in the non-stationary case. The Genetic Algorithm also presented itself as a good choice in terms of execution time, being slower than the Simulated Annealing algorithm, which is the fastest one, but providing better results in terms of tour length.

Simulated Annealing, in turn, produces the best results in terms of execution time. It's tour length performance, however, is very poor in scenarios in which a reduced number of targets as presented in the experimental results. In the



(a) Target Speed:0m/s (b) Target Speed:2m/s

Figure 17: Impact of the Number of the Max and Min Parameters in the Tour Lenght Performance of the Ant Colony Algorithm (m)

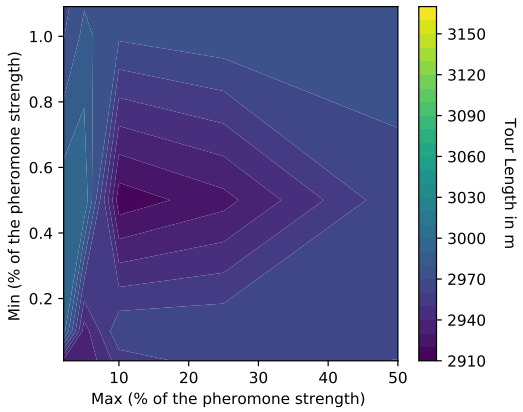


Figure 18: Impact of the Number of the Max and Min Parameters in the Tour Lenght Performance of the Ant Colony Algorithm for the Stationary Case - Zoomed (m)

scenarios with a bigger number of targets, like 64 or 100 targets, the simulated annealing managed to beat the Ant-Colony Optimization algorithm, while still being the fastest algorithm by a large margin.

As discussed in Section 5, the Ant-Colony algorithm is the best algorithm in terms of tour length performance for the stationary case, in which the targets do not move. As targets start to move and increase in number, however, the Ant-Colony algorithm performance degrades significantly for both metrics, the algorithm becomes, then, the worst algorithm among the three.

Analyzing the results concerning the parametrization of the genetic algorithm, it could be argued that, from the trends seen in Figures 11 and 12 the

genetic algorithm could be accelerated, depending on the application, the number of targets, and their speeds, by tweaking the generation number and population size parameters. It could be possible, in certain situations, to achieve a considerable speed up on the execution times of the algorithm by running it for many generations with a small population, gaining speed without compromising the tour length performance, especially when the number of targets is small and the search space reduced. Modifying the crossover rate and the mutation rate parameters, on the other hand, would not bring any benefits for either execution time or tour length, these are best left at their optimal values.

Such an approach would make sense in practical situations, such as when this algorithm is deployed on a UAV or mobile robot and used to calculate the best routes to intercept a series of mobile targets. An example of such an application would be a surveillance drone that needs to intercept a group of criminals on the run and photographs them. In such applications, it is better to have an algorithm that runs fast enough and outputs an acceptable solution, than to have a slow-paced algorithm that outputs the best solution but takes too much time to do it, giving the targets time to scape.

A similar case could be made for the Ant-Colony Optimization algorithm. Observing Figure 16, it could be argued that letting the algorithm run for a small number of generations, each of them with a big population, could speed up the algorithm and improve the poor tour length results for high-speed targets and/or a high target count. The performed experiments suggest, however, that the speed-up provided by this strategy is significant, but the improvement in the tour length performance is not, making this strategy not very useful in real applications.

The experimental results also show that no significant speed-up or improvement can be achieved for the implemented Simulated Annealing algorithm. According to the experimental simulations conducted during the development of this work, increasing the retry number parameter only helps to a certain point, not providing a significant improvement in the tour length performance, while hurting the execution time performance of the algorithm.

Overall, the genetic algorithm seems to be the best metaheuristic approach to solve the TDTSP problem. The experiments showed that this algorithm is the one that presents the most consistent and homogeneous tour length results for all numbers of targets, moving at all speeds, maintaining an acceptable trade-off between performance and execution time. The results also show that some parameter tweaking is still possible, making this algorithm especially suitable to be deployed in embedded, or even high performance, systems and adjusted as necessary to fulfill the requirements of the application it is used on.

7. Conclusion

This work presented a quantitative and qualitative comparison between three different heuristic algorithms to solve a Moving-Target-TSP. The results from this comparison have shown that the implemented genetic algorithm has achieved the best performance among the methods in both the quality of the

solutions (tour length performance) and temporal behavior. This result points towards a useful method that can be used in a variety of different emerging applications that require targets to be intercepted by an agent. UAV surveillance is a good example of these fields, as UAVs have become cheaper, they can be used as urban or crowd monitoring agents, being able to monitor moving targets in urban environments in real time by making use of such algorithm.

The three presented methods are, however, not the only methods that exist to deal with TSP problems that may be applied to this kind of problem and associated with the numeric interception module. Many other methods such as LKH heuristics, swarm optimization, branching, k-opt and memetic algorithms can also be experimented using a similar approach to verify their usability on an MT-TSP. Even the methods presented on this work have many variations that are worth investigating, ACO, for example, has suffered many modifications along the years that improved its performance on solving the stationary TSP. GA, in turn, has different selection methods, crossover operators, such as OX, ER, and CSP, and mutation operators, as Throas, Thrors, and PSM, that can be used and evaluated. Thus, future investigations can explore other approaches to solve the studied MT-TSP problem.

The work presented in this article could also be extended to consider the 1-D MT-TSP, which according to Helvig et al. (Helvig et al., 2003), can have its exact optima computed and compared to the results found by the metaheuristic algorithm presented in this work. Such a comparison could bring valuable insight to how well the algorithms perform against a method that is able to find the optimal solution to the 1-D MT-TSP problem, and to how significant the speed-up provided by the metaheuristics is.

Acknowledgements

The authors would like to thank the Brazilian Research Support Agency - CNPq and Sweden's Innovation Agency - Vinnova, by means of the NFFP7 project, for the support provided to develop this work.

References

Abdoun, O., Abouchabaka, J., & Tajani, C. (2012). Analyzing the performance of mutation operators to solve the travelling salesman problem. *CoRR*, *abs/1203.3099*.

Abeledo, H., Fukasawa, R., Pessoa, A., & Uchoa, E. (2013). The time dependent traveling salesman problem: polyhedra and algorithm. *Mathematical Programming Computation*, *5*, 27–55.

Cheng, C.-B., & Mao, C.-P. (2007). A modified ant colony system for solving the travelling salesman problem with time windows. *Mathematical and Computer Modelling*, *46*, 1225–1235.

Choubey, N. S. (2013). Moving target travelling salesman problem using genetic algorithm. *International Journal of Computer Applications*, *70*.

Dekkers, A., & Aarts, E. (1991). Global optimization and simulated annealing. *Mathematical Programming*, *50*, 367–393.

Dorigo, M., Birattari, M., & Stutzle, T. (2006). Ant colony optimization. *IEEE Computational Intelligence Magazine*, *1*, 28–39.

Dorigo, M., & Di Caro, G. (1999). Ant colony optimization: a new meta-heuristic. In *Proceedings of the 1999 congress on evolutionary computation-CEC99* (pp. 1470–1477). IEEE volume 2.

Dorigo, M., Maniezzo, V., & Colorni, A. (1996). Ant system: Optimization by a colony of cooperating agents. *Trans. Sys. Man Cyber. Part B*, *26*, 29–41.

Englot, B., Sahai, T., & Cohen, I. (2013). Efficient tracking and pursuit of moving targets by heuristic solution of the traveling salesman problem. In *52nd IEEE Conference on Decision and Control* (pp. 3433–3438).

Goldberg, D. E., & Lingle, R., Jr. (1985). Alleleslociand the traveling salesman problem. In *Proceedings of the 1st International Conference on Genetic Algorithms* (pp. 154–159). Hillsdale, NJ, USA: L. Erlbaum Associates Inc.

Hammar, M., & Nilsson, B. J. (1999). Approximation results for kinetic variants of tsp. In *International Colloquium on Automata, Languages, and Programming* (pp. 392–401). Springer.

Helvig, C., Robins, G., & Zelikovsky, A. (2003). The moving-target traveling salesman problem. *Journal of Algorithms*, *49*, 153–174. 1998 European Symposium on Algorithms.

Helvig, C. S., Robins, G., & Zelikovsky, A. (1998). Moving-target tsp and related problems. In *European Symposium on Algorithms* (pp. 453–464). Springer.

Holland, J. H. (1992). *Adaptation in Natural and Artificial Systems*. (2nd ed.). Cambridge, MA: MIT Press.

Keller, J. (2015). Wrong again: Uav market heading nowhere but up over the next decade. URL: `http://www.militaryaerospace.com/articles/2015/08/uav-market-blog.html`.

Kirkpatrick, S., Gelatt, C. D., & Vecchi, M. P. (1983). Optimization by simulated annealing. *Science*, *220*, 671–680. doi:`10.1126/science.220.4598.671`.

Mendiratta, K., & Goyal, A. (2014). Implementation of csp cross over in solving travelling salesman problem using genetic algorithms. *International Journal of Engineering Trends and Technology*, *11*, 455–459.

Mohd Razali, N., & Geraghty, J. (2011). Genetic algorithm performance with different selection strategies in solving tsp. In *International Conference of Computational Intelligence and Intelligent Systems* (pp. 1–6). volume 2.

Osaba, E., Yang, X.-S., Diaz, F., Lopez-Garcia, P., & Carballedo, R. (2016). An improved discrete bat algorithm for symmetric and asymmetric traveling salesman problems. *Engineering Applications of Artificial Intelligence*, *48*, 59–71.

Potvin, J.-Y. (1996). Genetic algorithms for the traveling salesman problem. *Annals of Operations Research*, *63*, 337–370.

Rajesh Matai, S. S., & Mittal, M. L. (2010). Traveling salesman problem: an overview of applications, formulations, and solution approaches. In *Traveling Salesman Problem, Theory and Applications* chapter 1. InTech.

Saenphon, T., Phimoltares, S., & Lursinsap, C. (2014). Combining new fast opposite gradient search with ant colony optimization for solving travelling salesman problem. *Engineering Applications of Artificial Intelligence*, *35*, 324–334.

Shukla, A., Pandey, H., & Mehrotra, D. (2015). Comparative review of selection techniques in genetic algorithm. In *2015 1st International Conference on Futuristic Trends in Computational Analysis and Knowledge Management, ABLAZE 2015* (pp. 515–519).

Stützle, T., & Hoos, H. H. (2000). Max–min ant system. *Future Generation Computer Systems*, *16*, 889–914.

Su, Z. C., Hlaing, S. H., & Khine, M. A. (20011). An ant colony optimization algorithm for solving traveling salesman problem. In *Procedures of International Conference f Compututer Scince and Information Technology 2011* (pp. 54–59). volume 16.

Tuba, M., Jovanovic, R., & Jovanovic, R. (2013). Improved aco algorithm with pheromone correction strategy for the traveling salesman problem. *International Journal of Computers, Communications and Control (IJCCC)*, *8*, 477–485.

Xing, L.-N., Chen, Y.-W., Yang, K.-W., Hou, F., Shen, X.-S., & Cai, H.-P. (2008). A hybrid approach combining an improved genetic algorithm and optimization strategies for the asymmetric traveling salesman problem. *Engineering Applications of Artificial Intelligence*, *21*, 1370–1380.

Xu, Mingji, Li, Sheng, & Guo, Jian (2017). Optimization of multiple traveling salesman problem based on simulated annealing genetic algorithm. *MATEC Web Conf.*, *100*, 2–25.

Üçoluk, G. (2002). Genetic algorithm solution of the tsp avoiding special crossover and mutation. *Intelligent Automation & Soft Computing*, *8*, 265–272.