Multi-agent based guarding and covering as a variation of the classical "Art Gallery Problem"

Group 20

François Chastel chastel@kth.se
Alexandros Ferles ferles@kth.se
Christos Matsoukas matsou@kth.se

Quentin Vecchio quentinv@kth.se
George Zervakis zervakis@kth.se

September 29, 2018

Abstract

The Multi-agent based guarding and covering problem is a variation of the classic "Art Gallery Problem". In this paper, we address the problem through the multiple Traveling Salesman Problem (mTSP) and examine a few different approaches on the subject. We analyze Tabu Search, Simulated Annealing and Genetic Algorithms and apply those techniques in order to solve the problem. Finally, we display our results and conclusions based on the experiments. The source code can be found at `https://gits-15.sys.kth.se/matsou/ai17_Group20`

1 Introduction and motivation

Nowadays, coverage and coordination problems have been a pole of interest in a broad application domain from military surveillance to automatic vacuum cleaners. Thus, we decided that it would be very interesting to carrying out a project on this topic. The description of our problem is practically a variation of the Art Gallery Problem (AGP). The original AGP corresponds to a real-world problem of guarding an art-gallery, using the minimum possible guards in optimal spaces who combined can observe the whole gallery. In our version, the room with obstacles, a set of points of interest, the number of guards and their sensor range is given and we want to find the paths which make the guarding optimal in the sense of all points been seen at least once in minimal time.

This problem is equivalent to the well-known Multi-Traveling Salesman Problem (mTSP), which is a generalization of the traveling salesman problem (TSP), where more than one salesman has to visit a set of n > m cities, with the constrain of each location to be visited only once [1][2]. Therefore, the algorithms that have been used over time to solve this problem can be adapted to our scenario. From a broad range of proposed algorithms who solve this problem we will focus on Simulated Annealing, Tabu Search and Genetic Algorithms. The reasons which lead us to choose those three algorithms where their appealing convergence towards optimality, their simplicity -in terms of implementation and comprehension of their naive approaches- and their broadness as they are three completely different approaches on the subject.

2 Methods

2.1 The multiple Traveling Salesman Problem (mTSP)

The multiple traveling salesman problem is just an extension of the classical traveling salesman problem (TSP). The original, is referring to a salesman that wants to visit some cities and return back to the initial city. The goal is to find the optimal trajectory for the salesman. Notice that he can only visit each city only once[3]. MTSP, more generally, accepts the fact of having more than one salesmen in the game[2][2].

The mathematical formulation of the mTSP is provided as detailed below[2]:

Suppose that we have m salesmen and a graph $G = (V,A)$, where V is a the set of n cities and A is the set of lines (arcs) that connect every two cities. Every salesman is placed at a single city (depot node). The remaining cities that are to be visited are called *intermediate nodes*. The mTSP is about finding tours for all m salesmen, who all start and end at the depot, so that the total cost of visiting all nodes is minimized. We define as $C = (c_{ij})$ the cost or distance associated with every line. C is said to be *symmetric* if it satisfies the equation $c_{ij} = c_{ji}$, $\forall (i,j) \in A$ and *assymetric* othewise. Furthermore, if $c_{ij} + c_{jk} \geq c_{ik} \forall i,j,k \in V$, then C is said to satisfy the *triangle inequality*. We now introduce a binary variable $x_{ij} \in \{0,1\}$ where $x_{ij} = 1$ if arc *(i,j)* is used on the tour and $x_{ij} = 0$ otherwise.

Hence the problem can be formulated as:

$$min \sum_{i=1}^{n} \sum_{j=1}^{n} c_{ij} x_{ij}$$

$$\sum_{j=2}^{n} x_{1j} = m(1)$$

$$\sum_{j=2}^{n} x_{j1} = m(2)$$

$$\sum_{i=1}^{n} x_{ij} = 1, j = 2, ..., n(3)$$

$$\sum_{j=1}^{n} x_{ij} = 1, i = 2, ..., n(4)$$

$$\sum_{i \in S} \sum_{j \in S} x_{ij} \leq |S| - 1, \forall S \subseteq V \backslash \{1\}, S \neq \oslash(5)$$

$$x_{ij} \in \{0, 1\}, \forall (i, j) \in A(6)$$

Constraints (3),(4) and (6) are the assignment constraints. Moreover, (1) and (2), make sure that exactly m salesmen depart from and return back to node 1. (5) belongs to the class of subtour elimination constraints (SECs). These constraints prevent subtours, which are tours made between intermediate nodes and not connected to the origin.

2.2 Simulated Annealing

The simulated annealing (SA) algorithm derives from the physical annealing process and from discrete minimization problems that seek for minimal solutions[4]. Annealing involves the actions of heating and cooling a material in order to find a low level energy state of the metal. As the metal cools its new structure becomes fixed, consequently causing the metal to retain its newly obtained properties. In simulated annealing temperature is a determinant factor that indicates the randomness of the energy phase[5].

The algorithm consists of two while loops one within the other. The inner loop stops when the solution is stable at a fixed value of temperature whereas the outer continues till the stopping criteria is met. The method starts by generating a random initial move. Then, we begin to look for a neighbor move which we select with a certain probability based on our accept function. This function, assigns a probability of accepting a move by measuring the cost difference ($\Delta\delta$) and the controlling temperature parameter (T) as mentioned above. So if the cost drops, we assign the probability of accepting this move with one, else we give it the value of $e^{-\Delta\delta/T}$. Therefore, by gradually slowing the temperature, the algorithm converges to the global minimum after a sufficient time of iterations[6][5].

2.3 Tabu Search

2.3.1 Nearest Neighbor Search (NNS)

Nearest Neighbor Search (NNS) is a searching method, frequently used for solving problems related to various topics such as pattern recognition, vector quantization and memory-based reasoning. This process, to its simplest form, is basically searching for the nearest neighbor of a point given an environment, a set of points and a measure of distance [7].

2.3.2 Tabu Search

Tabu Search is a strategy developed to encounter combinatorial optimization problems, within a wide range of applications; from space planning, scheduling and graph theory to even more classical problems such as the traveling salesman problem (TSP) [8].

More specifically, tabu search is numbered among a category of algorithms called Metaheuristics. The purpose of these strategies, is to improve the optimality of the current solution within a judicious time limit [9].

The foundation of tabu search, lies beyond three principle concepts[10]:

i) *Memory Structures* selection: Compared to other search methods such as A* or simulated annealing where the pick of the memory structures is either inflexible or restricted, in tabu search we go for more flexible attribute-based memory structures. This is occurring due to the need of having an evaluation criteria and in order to use previously visited data more exhaustively.

ii) *Control Mechanism* of the structures: We make use of what is called tabu restrictions and aspiration criteria. These are essentially the conditions upon which we count on, in order to decide the freelance or restriction of the search process.

iii) *Intensification - Diversification* search approach: The former term, refers to the ability of giving emphasis to move combinations and to useful characteristics of previous solutions whereas the latter is permitting the search to delve deep and explore new areas.

The algorithm begins from an initial solution and tries to find a better solution by looking at a neighborhood of our current estimate. The main idea of this technique, is to find all possible solutions given a sequence of moves. By applying the best next admissible move to our current solution, we obtain a new one which we use in the next iteration. Admissibility is measured in terms of two factors: tabu restrictions and aspiration criteria. To avoid falling into infinite loops and local optimals, we make use of what is called a tabu list, that stores forbidden past moves. When the last best solution is obtained, whether we stop the procedure or update our admissibility conditions and repeat the algorithm[11][10][8].

2.4 Genetic Algorithms

Genetic algorithms (GAs) [12][13] are biologically inspired meta-heuristic optimization techniques based on the process of evolution by natural selection [14][15]. Note that GA are not the only evolutionary inspired optimization techniques. Some other examples are the Ant Colony Optimization [16] and Particle Swarm Optimization [17] techniques. GAs, generate a population of candidate solutions, which are iteratively improved so that an optimal solution evolves over time [14] as the most suited individual survival of the population [18]. To do that, the algorithm simulate this evolutionary mechanism by using heredity and mutation [14][15]. Heredity is implemented in a two step process namely selection and cross over. Selection is the process of selecting the potential (individual) parents from the from the current generation in order for their genetic material to survive [18][15]. There are several ways to do that e.g. Tournament Selection, Fitness Proportional Selection, Rank Based Selection, but all of them involve the computation of the fitness/ranking function of each individual [18]. Sometimes the fittest individuals are cloned into the next generation in order to maintain their genetic material which is referred as elitism [14]. Cross over is probably the most interesting part of the algorithm in terms of robustness. The cross over is basically the mixture of genetic material of the parents in order for the offsprings to be generated [14][18] -as happens in the real leaving

organisms. After the new population has been generated, a gene mutation is applied on them with predefined probabilities in order to gain diversity and lost information of the solutions which leads to better effectiveness of the GAs [15]. Finally, the termination condition is when a solution has a better than the predefined minimum fitness, or a maximum number of generations have been reached [14]. GAs usually can escape from the local optimums if they are shallow enough which make them robust. Nevertheless, there is no guarantee that a GA has found the global optimum solution and in hard problems, in terms of complexity, the algorithm could be very sensitive to the input parameters i.e. the mapping function from phenotype to genotype and may fail if the inputs of the system are highly correlated [14].

2.5 Related Work

Up next we present a few approaches to address the mTSP[2]:

i) The *straight algorithm:* The idea behind this method, is to check whenever any of the subtour elimination constraints are breached, after obtaining an integer solution. At first, we loosen up our SECs and every time a violation is occurred, a new constraint is formulated to prevent such subtour.

i) The *reverse algorithm:* This algorithm works the same as the previous with the difference that the checking of the SECs is taking place before an integer solution is obtained.

iii) The *branch- and bound -based algorithm:* This method is referring to large-scale symmetric mTSPs addressing the Lagrangean Problem which is generated by relaxing the degree constraints. This problem is solved using a degree-constrained minimal spanning tree which spans over all the nodes. Lagrange multipliers are updated by a subgradient optimization procedure and fast sensitivity analysis techniques are applied to reduce the problem size.

iv) mTSP to standard TSP: The main concept here is to address the mTSP by transforming it into a standard TSP, where we could use any algorithm developed so far, in order to find a solution for the initial problem.

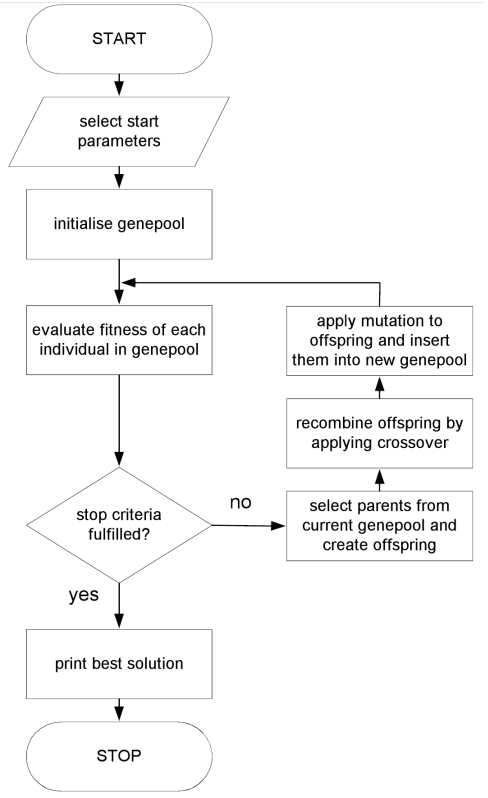


Figure 1: Flowchart of basic GA algorithm [14]

3 Implementation & Experiments

3.1 Modeling

The first main issue of this problem is based on the way of modeling the problem in a computable way. We have imagined few different way of realizing this model. We can consider the map as a graph, which "every places" that a guard can visit is a node, and after we allow certain movements for this purpose.

If we consider the problem as a graph we are facing the problem of the scheduling that couldn't be implemented in an easy way. In case of graph-modeling, there are walls (obstacles with a position X and Y) that can't be reached by the guards, guards (a finite number of guards at the beginning) that can be related to a position and reachable cases. Every case is reachable according to a square (up, down, left, right, left-down, left-up, right-down, right-up) after we can generate a path (list of reachable cases) for a given guard.

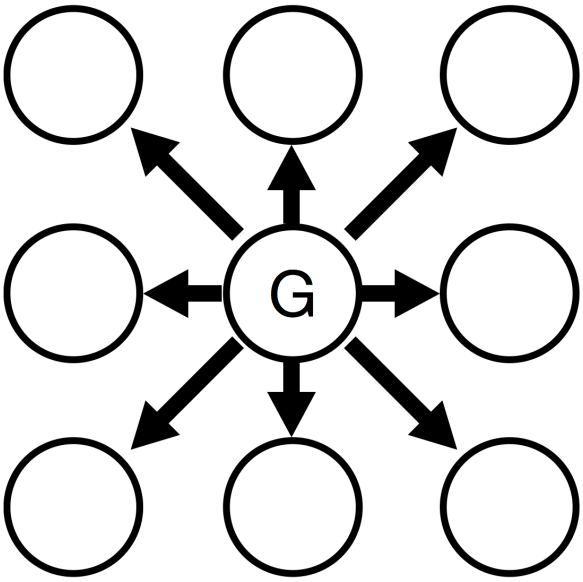


Figure 2: Example of modeling in a graph aspect

Another way of modeling this problem could be to consider the problem as a schedule for each case with the time when the guard gets in his eye-range. It will allow us to add more constraints to the problem like the fact we have to see a specific case all the time (the last one on the figure represent a case that needs to be seen all the time and the others need to be seen at least once at each six-turns).

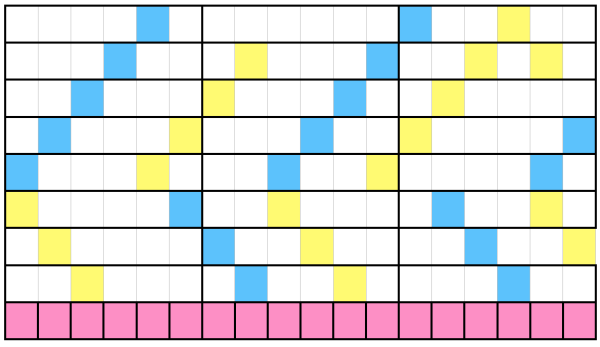
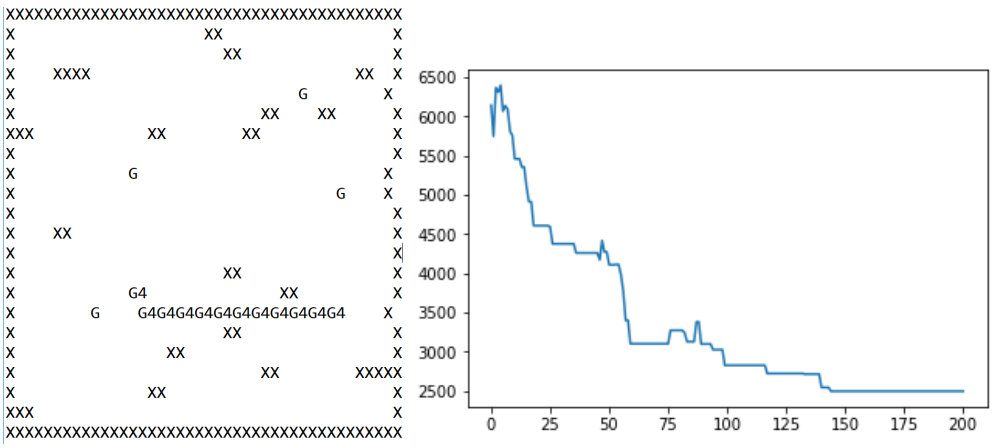


Figure 3: Example of modeling in a scheduling aspect

3.2 Simulated Annealing

The fitness function of a state, is used to evaluate the cost in the simulated annealing process. The algorithm tries to find a solution by looking at neighboring states, while smoothing the *temperature* variable. After a number of iterations, the algorithm converges to the global minimum.

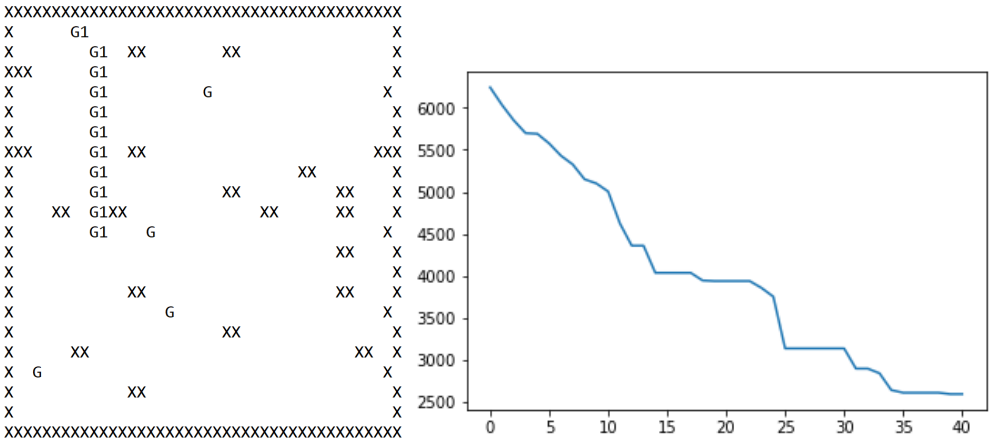


(a) (b)

Figure 4: Example of solution for one of the Guards using Simulated Annealing (a) and evolution of energy (b)

3.3 Tabu Search

Tabu search focuses on constantly checking the neighboring moves of an agent. Like in the classic problem of mTSP, a guard initially decides to make a move based on the fitness of each possible move available to it. This move becomes a *tabu* move; the guard won't choose it in the future, thus avoiding to fall in a loophole. This process is repeated until a threshold of iterations is met or all states have become *tabu* ones, meaning that the best possible solution has been found.



(a) (b)

Figure 5: Example of solution for one of the Guards using Tabu Search (a) and evolution of fitness (b)

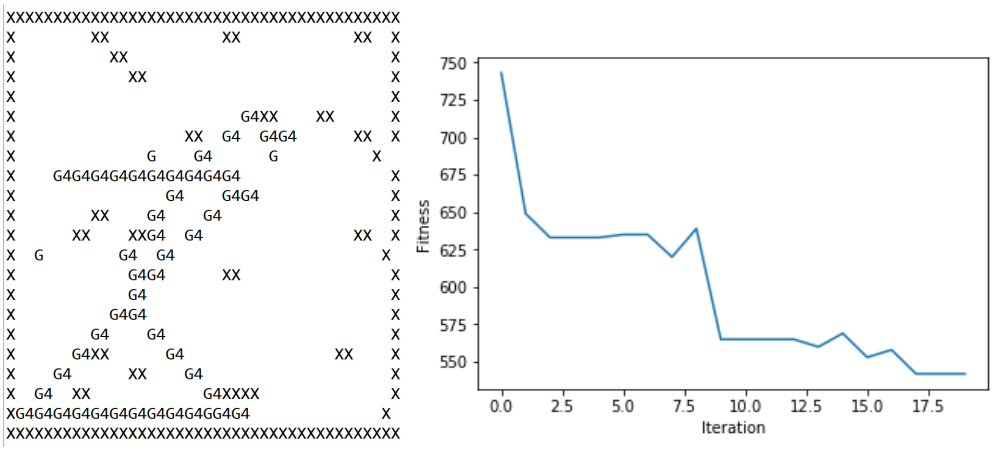
3.4 Genetic Algorithms

The implementation of the genetic algorithm consists basically of implementing the crossover and mutation function. The new solutions are provided from the same functions which used for creating the initial random solutions on TS and SA (see section above).

In a multiple-agent based architecture, there is the need to combine the solutions of distinct agents (in this case, combining solutions of each guard) before assessing the

performance of the whole system (via the fitness function). Each solution contains a guard path for each guard assigned to the problem. As a result, during the crossover the paths of each unique guard are isolated and then combined to a new path. In the end, a path from each guard is picked and merged with the paths of the other guards, to formulate a new solution, until all produced paths are exhausted.

The mutation procedure consists of randomly swapping elements from a solution with a probability around 10%. When a mutation is decided, a random element is picked from a solution's guard path and swapped with the neighboring element to it.



(a) (b)

Figure 6: Example of solution for one of the Guards using Tabu Search (a) and evolution of fitness (b)

4 Discussion and Conclusion

Given the limited time that we had on experimentation we cannot infer much. Nevertheless, it seems that the simulated annealing is the fastest algorithm in terms of computational time as it required the less iterations and running time compared to the other ones. On the opposite side, there is the Genetic Algorithm which required a noticeable extra run time in order to converge to a sensible solution. Somewhere in between, there is the Tabu Search, which provided similar results as with the SA, but it required more running time. We should also infer that due to its non monotonically behavior, the GA needs to save the best last solutions and finally return the best one among them. Otherwise, we might get a solution with much worse fitness than the best we already have found.

In order to be sure about the optimal method, more experiments should be tested, so that we have enough results and hence robust statistics. Thus, additional experimentation should include:

- More iterations on each procedure and comparison of the results
- Comparison of the results given the same running time
- Fine tunning of initial conditions and coefficients of the algorithms
- Coarse search to find the optimality of each algorithm, when the area of the map and the obstacles are varying, given the same guards and afterwards with different number of guards

Nevertheless, in this project we were able to get involved with three of the leading algorithms that are proposed to solve this problem and we managed to get some results from all of them. The proposed experimentation was not feasible due to time constraints and due to the fact that we decided to broaden our horizons in terms of variation of algorithmic implementation, rather than focusing on just the Tabu Search and specialize on that.

References

[1] A. E. Carter and C. T. Ragsdale, "A new approach to solving the multiple traveling salesperson problem using genetic algorithms," *European journal of operational research*, vol. 175, no. 1, pp. 246–257, 2006.

[2] T. Bektas, "The multiple traveling salesman problem: an overview of formulations and solution procedures," *Omega*, vol. 34, no. 3, pp. 209–219, 2006.

[3] K. L. Hoffman, M. Padberg, and G. Rinaldi, "Traveling salesman problem," in *Encyclopedia of operations research and management science*, pp. 1573–1578, Springer, 2013.

[4] A. Dekkers and E. Aarts, "Global optimization and simulated annealing," *Mathematical programming*, vol. 50, no. 1, pp. 367–393, 1991.

[5] M. Malek, M. Guruswamy, M. Pandya, and H. Owens, "Serial and parallel simulated annealing and tabu search algorithms for the traveling salesman problem," *Annals of Operations Research*, vol. 21, no. 1, pp. 59–84, 1989.

[6] L. Ingber, "Simulated annealing: Practice versus theory," *Mathematical and computer modelling*, vol. 18, no. 11, pp. 29–57, 1993.

[7] P. N. Yianilos, "Data structures and algorithms for nearest neighbor search in general metric spaces," in *SODA*, vol. 93, pp. 311–321, 1993.

[8] F. Glover, "Tabu search- part i," *ORSA Journal on computing*, vol. 1, no. 3, pp. 190–206, 1989.

[9] S. Basu, "Tabu search implementation on traveling salesman problem and its variations: a literature survey," *American Journal of Operations Research*, vol. 2, no. 02, p. 163, 2012.

[10] F. Glover, "Tabu search: A tutorial," *Interfaces*, vol. 20.

[11] I. Alkallak and R. ShaâĂŹban, "Tabu search method for solving traveling salesman problem," *Al-Rafiden J Comput Sci Math*, vol. 5, no. 2, pp. 141–153, 2008.

[12] J. H. Holland, "Adaptation in natural and artificial systems. an introductory analysis with application to biology, control, and artificial intelligence," *Ann Arbor, MI: University of Michigan Press*, 1975.

[13] D. E. D. E. Goldberg, "Genetic algorithms in search, optimization, and machine learning," tech. rep., 1989.

[14] M. Köppen, G. Schaefer, and A. Abraham, *Intelligent Computational Optimization in Engineering: Techniques & Applications*, vol. 366. Springer Science & Business Media, 2011.

[15] M. Sedighpour, M. Yousefikhoshbakht, and N. Mahmoodi Darani, "An effective genetic algorithm for solving the multiple traveling salesman problem," *Journal of Optimization in Industrial Engineering*, no. 8, pp. 73–79, 2012.

[16] M. Dorigo, M. Birattari, and T. Stutzle, "Ant colony optimization," *IEEE computational intelligence magazine*, vol. 1, no. 4, pp. 28–39, 2006.

[17] J. Kennedy, "Particle swarm optimization," in *Encyclopedia of machine learning*, pp. 760–766, Springer, 2011.

[18] T. Zhang, W. Gruver, and M. H. Smith, "Team scheduling by genetic search," in *Intelligent Processing and Manufacturing of Materials, 1999. IPMM'99. Proceedings of the Second International Conference on*, vol. 2, pp. 839–844, IEEE, 1999.