Genetic vs greedy scheduling in multirobot systems for smart warehouses

A. Xhyra, P. Tropeano, M. Marino

September 2021

Contents

1 Introduction **3**

2 Problem formulation **4**
2.1 MRTA taxonomy 5
2.2 Environment 5
2.3 Agents 6
2.4 Metrics 6

3 Technical aspects **8**
3.1 Collisions 8
3.2 Simulation workflow 10

4 GA for scheduling **12**
4.1 Chromosome representation 12
4.2 Selection 13
4.3 Crossover 13
4.4 Mutation 13
4.5 Fitness function 14
4.6 Stop criterion 14

5 Other approaches **15**
5.1 Random 15
5.2 Greedy 15

6 Experimental Results **16**
6.1 Standard hypothesis 16
6.2 Alternative hypothesis 20
6.3 Number of agents 22

7 Conclusion **25**
7.1 Future works 25

1 Introduction

In the context of Smart Warehouses, where autonomous robots perform tasks of object transportation, it's essential to identify the optimal scheduling strategy to minimize the operating costs and avoid collisions that may occur on the path of two or more robots. The literature describes problems in this class as Multi-Robot Task Allocation problems (MRTA); such problems are NP-hard [1], which requires efficient heuristics.

In this report, we focus on the task allocation process and the collision avoidance technique, intending to improve the heuristic proposed by P. Yang et al. in their previous works [2, 3]. In [3], genetic algorithms and a sub-optimal A* algorithm resulted in the best techniques respectively for centralized scheduling and pathfinding. We reproduced these algorithms, using standard A* instead of a sub-optimal path planning version [4] to establish a new baseline. Various task allocation methods were compared: (1) Genetic Algorithm (GA), (2-3) two techniques based on randomness, (4) a greedy approach, and (5) customization of the greedy approach in which agents can negotiate a task. For the collision avoidance problem, we introduced a cooperative mechanism that we will discuss later.

We organized this report as follows: In Section 2 a better description of the problem is given; Section 3 provides an overview of technical aspects of the simulation, such as the collision avoidance technique; in Section 4 the genetic algorithm (GA) is described, while Section 5 describes approaches (2-3-4-5); then the results are presented and discussed in Section 6.

2 Problem formulation

In the warehouse work context, the main activity is moving packages from one point to point to load or unload a conveyor belt or a shelf. In the simulation, each pod contains a package that must be handled and moved to another location. The job that involves reaching, moving, and repositioning a pod on its shelf is a *task* and can be identified by the pod's position (i.e., its shelf coordinates).

The goal is to generate the best scheduling which involves all agents and has the lowest execution time.

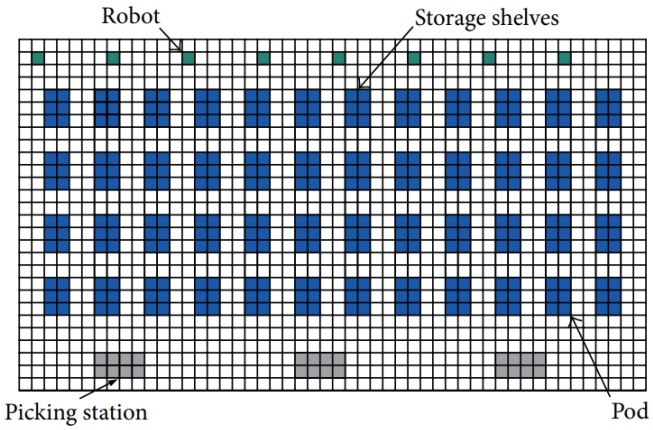


Figure 1: Map of the simulated warehouse.

Figure 1 shows the simulated warehouse. Storage shelves contain pods. Each robot collects one pod and takes it to a specific picking station appointed in the task.

It's necessary to adopt task allocation policies, alongside collision avoidance and path planning techniques, to perform tasks efficiently. In [3] A* resulted in the best path planning technique, which is why we used it as a standard for this work.

To build the simulation we made the following assumptions:

1. A grid-based map model is used for the warehouse environment
2. The environment of the system is unknown to the robots.
3. The robot is equipped with necessary sensors that detect the environment.
4. The robot can move in the Von Neumann neighborhood.
5. Robots can cooperate to perform a task.
6. The total cost and the whole time to complete the task are measured by the walking steps of the robots.

7. The elapsed time at the station and lifting the shelf is fixed and constant.

8. Robots move at a constant speed.

Robots must perform all the daily tasks to finish their job. Simulations address a working day producing a task list that the agents (i.e., robots) must perform in the environment (i.e., grid-based map 50x28). This task list is also called *schedule*.

2.1 MRTA taxonomy

The distributed approach is a *distributed solving of problem*.

Gerkey and Matarić [1] proposed a taxonomy for this kind of problems which describes *MRTA* problems along three axes:

- **Single-task robots (SR)** vs. **Multi-task robots (MR)**
- **Single-robot tasks (ST)** vs. **Multi-robot tasks (MT)**
- **Instantaneous Assignment (IA)** vs. **Time-extended Assignment (TA)**

This report will focus on Single-task robots with Single-robot tasks and Instantaneous/Time-extended Assignment simulations, hence simulations in which a robot can perform only one task at a time and don't require the simultaneous effort of more robots. IA doesn't allow planning for future allocations since the information about the environment is limited; on the contrary, TA permits planning over time.

2.2 Environment

The environment is responsible for agents' perception and mediated interaction, and for keeping track of the remaining tasks (task pool) allocated via a task handler. To shortly describe the environment we proceed to list its most important properties [5].

Accessible The environment is fully accessible: every agent can obtain complete, accurate, and up-to-date information about the environment's state.

Deterministic In the environment every action has a single deterministic effect.

Episodic Agent's experience can be divided into atomic steps where the agent performs a single action based only on the current episode.

Static The environment is static: only agents' positions and the task pool change.

Discrete There is a fixed, finite number of actions available for the agents in the environment (i.e., movement actions, interactions with pods, interactions with the task handler).

2.3 Agents

We developed and used tropistic agents (robots) for all task scheduling algorithms.

The "see" action is mapped to the *detect collision*, "do" is the simulation stepping and can be one of the following "actions":

- interact with shelf (pick or leave pod);
- interact with other agents (collision avoidance);
- interact with picking station (deliver good);
- interact with environment (receive new task).

Robots "see" the environment through sensors and choose the respective action to perform (*simple reflex agents*).

Interactions between robots only happen locally when these agents are in their neighborhood, so the communication strategy is spatially founded.

2.4 Metrics

To evaluate a simulation we tracked three metrics. To discuss them it's essential to introduce a formal notation.

Let $T = \{t_1, ..., t_n\}$ be a set of n tasks to allocate and $R = \{r_1, ..., r_m\}$ the set of robots of the system. $c_{ij} = c_{ji}$ is the travel cost between the pair of tasks $(t_i, t_j) : t_i, t_j \in T$. d_{ij} is the travel cost between a robot r_i and a task t_j, and w_i is the cost of task t_i, that is the cost of bringing the pod from the its shelf to the picking station and then carrying it back. All of the costs depend only on the path lengths.

Consider a partitioning of T into m subsets $T' = \{T_1, ..., T_m\}$; the subset T_i is allocated to robot r_i. T' is a **schedule** of the system. Let $k = |T_i|$, $T_i = \{t_{i1}, ..., t_{ik}\}, \forall i = 1, ..., m$.

The total cost of the tasks allocated to robot r_i is defined as $W(r_i) = w_{i1} + ... + w_{ik}$. Now it's possible to introduce the three metrics tracked.

Individual Travel Cost (ITC) is the time required by a robot r_i to perform all of its tasks.

$$ITC(r_i, T_i) = D_{r_i t_{i1}} + \sum_{j=1}^{k-1} c_{t_{ij} t_{i(j+1)}} + W(r_i), \tag{1}$$

where $D_{r_i t_{i1}}$ is the time required by robot r_i to reach the task t_{i1}.

$\sum_{j=1}^{k-1} c_{t_{ij} t_{i(j+1)}}$ is the total time required to reach the other $(k-1)$ tasks.

Travel Time (TT) is the elapsed time to compute the simulation. Time is measured in steps.

$$TT = \max_i ITC(r_i, T_i) \tag{2}$$

Total Travel Cost (TTC) is the total number of steps made by the agents during the simulation.

$$TTC = \sum_{i=1}^{m} ITC(r_i, T_i) \tag{3}$$

Balancing Utilization (BU) is a real number between 0 and 1. The larger the number, the more balanced is the usage time of each robot. If $BU = 1$ then each robot has the same ITC.

$$BU = \frac{\min_i ITC(r_i, T_i)}{\max_i ITC(r_i, T_i)} \tag{4}$$

3 Technical aspects

We give in this section an overview of the technical details of simulations.

To reduce the computational cost of simulations, we preprocessed the best paths from each walkable cell to every pod, and from each pod to each picking station using the *A* algorithm*. These are all and only the trails needed to perform a simulation, as to solve a task is necessary to know (1) the path from each tile to a pod, (2) the path from each pod to each picking station, and then (3) the path to bring back the pod to its shelf (i.e., the reverse of (2)).

The simulation logic is synchronous. At every environment tick, each agent gets updated and moves one step.

Every simulation starts with a task pool defined as a finite set of tasks.

3.1 Collisions

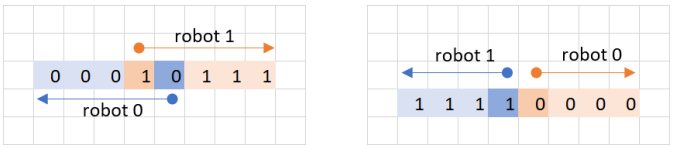
We addressed collision avoidance between robots by introducing a cooperation mechanism in which robots swap their tasks or give the right-of-way to one of the robots involved in the imminent collision.

Classification

Five classes of collisions may arise, however we solve only four of these for the sake of simplicity of the simulation and computational cost. The fifth type of collision leads to a deadlock, so in this case, we let the simulation die.

The classification we propose is described in the following paragraphs.

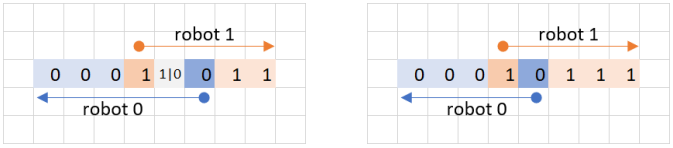
Type 0 This type of conflict is found whenever two robots are facing each other and their routes overlap (Fig. 2). We solve this conflict by letting the two robots swap all of their remaining tasks, including the pod they are currently handling. The cost of this action is assumed to be constant and equal to the time required by a movement step.



(a) Before (b) After

Figure 2: Collision of type 0.

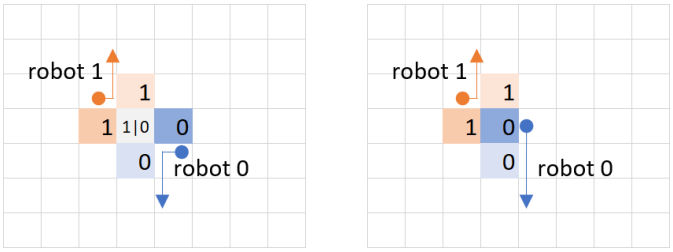
Type 1 The second type of conflict takes place when two robots are in line facing each other, separated by a cell and their routes intersect each other (Fig. 3). This case is converted into a type 0 collision, by letting the robot which detects the collision wait for a turn.



(a) Before (b) After

Figure 3: Collision of type 1.

Type 2 This third type of conflict happens when two robots attempt to move into the same cell at the same time (Fig. 4). We solve this situation by letting the robot which detected the collision wait for a turn and letting the other one move into the common cell. At the next time step, the two robots will be able to keep on their routes.



(a) Before (b) After

Figure 4: Collision of type 2.

Type 3 The fourth type of conflict takes place when a robot approaches a pod to pick it up or to drop it back on its shelf, while a second robot is close to the first one (Fig. 5). At the next step, we let the second robot wait for a turn, otherwise it would bump because shelf operations require a time step. At the end of this waiting turn, either the first robot ends its schedule and then moves to its home position, or gets a new task and one of these two cases might occur:

1. the new task is located in the same position of the previous one, so the second robot must wait for another turn for the first robot's operation;
2. the new task is located in another position, so the first robot moves straightaway during the current time step.

Type 4 The fifth type of conflict takes place when a robot approaches a shelf to interact with it, while three other robots are nearby the first one, forming a square shape and a circular wait (Fig. 6).

The first robot has to wait for a turn because it is interacting with a shelf, so the second robot has to wait for a turn too because otherwise, it would crash into the first one; similarly, do the third and the fourth one.

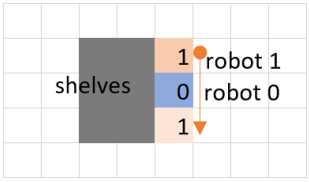


Figure 5: Collision of type 3.

When the first robot ends its shelf operation and has to move to the position occupied by the fourth robot, which is waiting for the third to move, and the third waits for the second which is waiting for the first one; the simulation results in a deadlock, then it dies with $TT = TTC = MAXINT$, $BU = 0$.

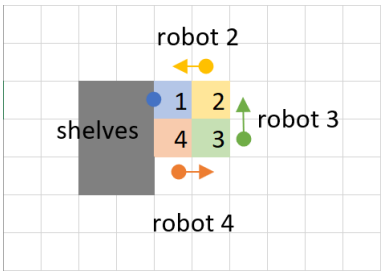


Figure 6: Collision of type 4.

3.2 Simulation workflow

A single simulation consists of two main parts: task assignment and task execution, which includes collision detection and avoidance (Fig. 7). In the initial part, tasks are assigned to agents using one of the task allocation algorithms described in Sections 4, 5. After the task assignment, the simulation continues until at least one task is completed.

In the task execution part, collision avoidance techniques shown above are used to allow agents to solve possible collisions.

When an agent completes a task, a new one is assigned to it. The simulation ends when the task pool is empty or a deadlock occurs.

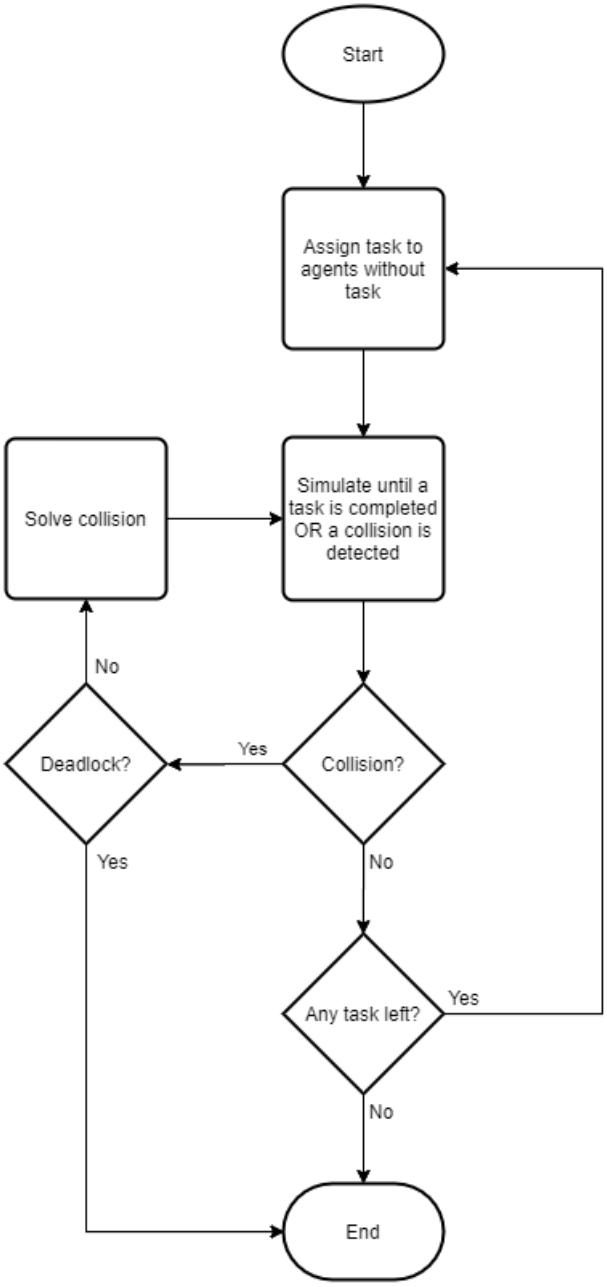


Figure 7: Simulation workflow.

4 GA for scheduling

The purpose of this section is to provide details about the Genetic Algorithm (GA) used (i.e., chromosome representation, genetic operators, fitness function).

Algorithm 1 Genetic algorithm for scheduling.

```
Initialize population size Popsize, #generations Maxgen, crossover rate Pc,
mutation rate Pm ← 0.1;
Generate an initial population randomly;
Evaluate the initial population;
for gen=1 to Maxgen do
    if gen ≥ 1500 then
        Evaluate stop criterion;
    end if
    if gen ≥ 1000 then
        Pm ← 0.5;
    end if
    Select elite individuals;
    Select mating pool;
    if random(0,1) < Pc then
        Crossover pairs of individuals;
    end if
    for individual in population do
        if random(0,1) < Pm then
            Mutate individual by swap mutation;
        end if
    end for
    Evaluate the offspring;
end for
```

4.1 Chromosome representation

Let m, n be respectively the numbers of robots and tasks. A chromosome is an integer string (i.e. a tuple) of length $m + n - 1$, more precisely a permutation of integers (genes) from 1 to $m + n - 1$ representing a schedule.

Consider now the sequence of integers from 1 to n, then each of these represents a real task, while integers from $(n + 1)$ to $(m + n - 1)$ represent virtual tasks (VT), which serve as separators between the set of tasks of a robot and the set of tasks of the next one. Robots are considered in order of appearance of the VT, which means that if the first VT appears in position i then tasks from position 0 to $(i - 1)$ are assigned to the first robot, and so on. Each gene associated with a task is referred to as an element of the task pool.

E.g. Let $m = 3, n = 8$, R_1, R_2, R_3 the three robots and let the chromosome (6,2,1,8,4,9,3,7,10,5). Integers $9, 10$ act as separators, hence this chromosome associates the tasks in this manner: $R_1 \rightarrow (6, 2, 1, 8, 4)$, $R_2 \rightarrow (3, 7)$, $R_3 \rightarrow (5)$.

It follows that if two virtual tasks appear subsequently then the robot represented in between will have no tasks assigned.

4.2 Selection

A selection strategy, the GA adopts a combination of an elitist strategy with the roulette wheel method (the elitist strategy accelerates the algorithm convergence). This means that in a first moment the top 20% of the population is passed directly to the next generation, while the remaining individuals are selected proportionally with the roulette wheel method, forming the mating pool. Individuals in the mating pool are then crossed over and mutated.

4.3 Crossover

The crossover operator recombines two individuals in the mating pool producing two new offspring. The operator used determines two crossing points for each pair of individuals: the first point in the anterior half and the second one in the latter half [6].

For example let the two parents be

(7 5 9 1 2 4 3 6 10 8),
(8 2 5 10 1 3 9 6 4 7).

Suppose that the vertical bars represent the randomly selected crossing points:

(7 5 9 | 1 2 4 3 | 6 10 8),
(8 2 5 | 10 1 3 9 | 6 4 7).

Then the central subsections of the first and the second parent are copied at the beginning of the second and the first offsprings respectively:

Off1 (10 1 3 9 * * * * * *),
Off2 (1 2 4 3 * * * * * *).

Lastly, the genes of the first parent that aren't already present in the first offspring are copied into it in order. The same happens for the second parent and the second offspring:

Off1 (10 1 3 9 7 5 2 4 6 8),
Off2 (1 2 4 3 8 5 10 9 6 7).

4.4 Mutation

Mutations help to prevent premature convergence of the algorithm, introducing a certain variability in the population. The mutation strategy adopted is the swap mutation, which randomly selects two genes in the chromosome and swaps them [6].

E.g. Let (6,2,1,8,4,9,3,7,10,5) be the chromosome, the first and the sixth the genes to swap, then the new chromosome will be (9,2,1,8,4,6,3,7,10,5).

4.5 Fitness function

The fitness function design must reflect the objectives of the optimization problem, which in this specific case are (1) minimizing the overall time TT to complete tasks, (2) minimizing the total travel cost TTC of each robot, (3) maximizing the balancing utilization BU of the robots. Hence the fitness function $F(X)$ can be defined as

$$F(x) = \frac{MaxTest}{TT} + \frac{TotalTest}{TTC} + BU, \tag{5}$$

where $MaxTest$ and $TotalTest$ are respectively the maximum number of tasks allocated to a robot by the current schedule and the total number of tasks.

4.6 Stop criterion

Differently, from [3] we introduced a stop criterion to reduce the computational cost of the algorithm, due to the high number of fitness function evaluations and to the maximum number of generations (100000). The stop criterion is evaluated starting from the 1500th generation, stopping the algorithm if the fitness function improvement is less than $10^{-4}\%$ for 200 consecutive generations.

5 Other approaches

In this section, we provide details about the other scheduling algorithms used. The first two approaches are based on randomness, while the other two are greedy techniques. In particular, one should note that *Random*1, *Greedy*0 and *Greedy*1 consider our scheduling problem as an instance of an SR-ST-IA problem, allowing an online approach, while *Random*0 considers the problem as an instance of an SR-ST-TA, as the genetic algorithm scheduling does.

5.1 Random

Random 0 This algorithm creates a schedule by assigning a random amount of randomly chosen tasks picked from a given set to each agent, possibly leading to an unbalanced assignment.

Random 1 This algorithm is an *online* variant of the latter that assigns a random task to a robot each time it completes its previous one.

5.2 Greedy

Both approaches are distance-based. Different distances can be used to implement greedy approaches: in this work, an agent checks the shortest path to the task and chooses the closest one. Moreover, assuming that agents can also track the distance from each pod to the picking station, they can estimate the cost of the whole task and choose the most convenient one. Another possible distance that can be used is the euclidean distance between the robot and the pod; however, this metric doesn't consider the presence of shelves and other obstacles.

Greedy 0 This algorithm is an online variant of the greedy algorithm for solving a problem, which tries to optimize by always choosing the moves with a lower cost [7]. The closest available task is assigned to an agent each it completes its current one.

Greedy 1 This algorithm is a customization of the *Greedy 0* algorithm, where each agent gets assigned the closest task which satisfies one of the following:

- the task has not been assigned yet;
- the task has been pre-assigned to another robot which is farther and has not picked up the pod yet.

If the second condition is met the other robot gets notified and attempts to get a new task.

6 Experimental Results

In this section, the results are discussed, showing each score as average value and standard deviation over twenty runs of the respective algorithms.

6.1 Standard hypothesis

Total Time Greedy techniques achieved an overall better TT score.

GA could lead to better results by using a different parameter tuning, e.g., changing the mutation rate or the early stop criterion.

Random approaches show different behavior: *Random 0* scored the worst TT while *Random 1* scored better than GA. This difference results because *Random 0* assigns randomly to each agent random subsets of tasks in advance, which might lead to an unbalanced schedule, both in the task number and in the task cost; hence it might lead to a high TT score. *Random 1* instead assigns a new task to an agent every time it ends its current one, leading to a more balanced schedule with a lower probability of getting a high TT score.

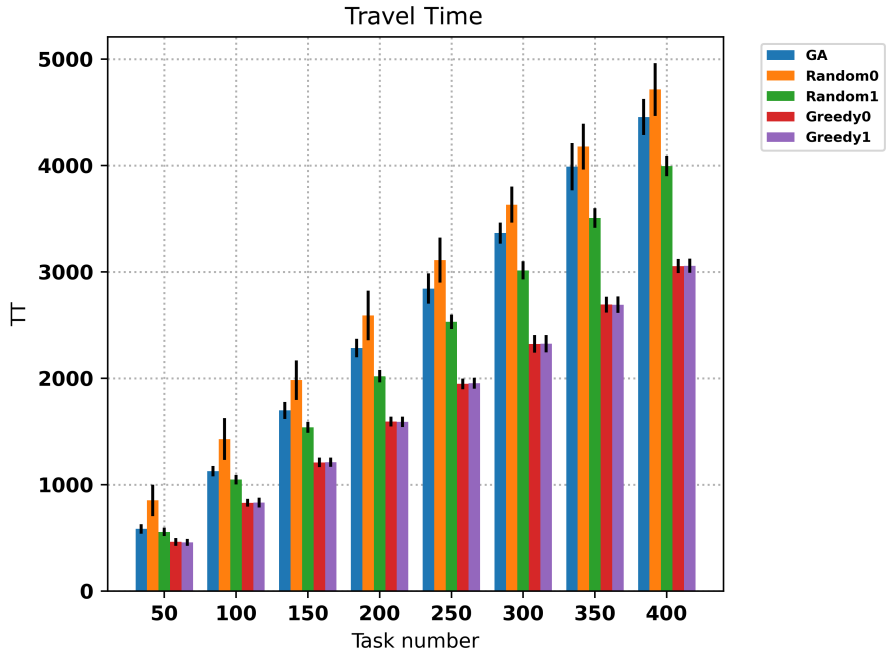


Figure 8: Total Time using different approaches. Standard deviation of the simulations is represented by the black stripes.

Balancing Utilization Since Instantaneous Assignment techniques (i.e., *Random 1*, *Greedy 0*, *Greedy 1*) intrinsically fairly prioritize the access to the task pool, letting it depend only on task cost, they scored a better BU.

Random 0 and GA (during the initial generations) tend to generate unbalanced task assignments. Moreover, GA's slow convergence speed reveals the complexity of the fitness function and stops after a relatively small amount of iterations due to the early stop criterion.

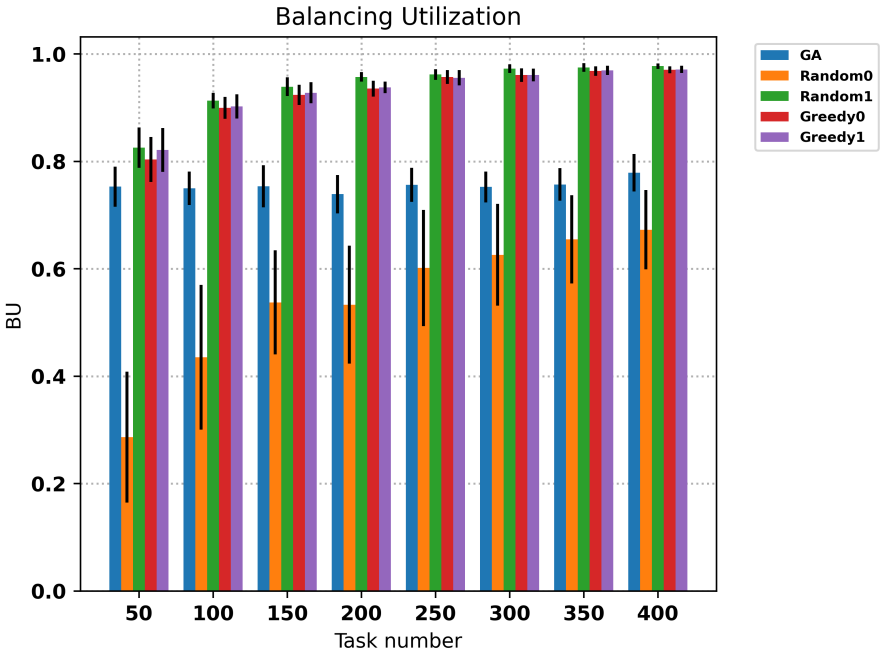


Figure 9: Balancing Utilization using different approaches. Standard deviation of the simulations is represented by the black stripes.

.

Given the number of tasks n, the Eq. 4 describing BU can be written as

$$BU_n = X_n/(X_n + Y_n), \tag{6}$$

where X_n is the minimum ITC, common to every agent, and Y_n is the difference between X_n and the maximum ITC achieved by agents.

Using IA approaches, the moment when an agent can't get a new task corresponds to the moment when the task pool is empty, hence this agent's ITC corresponds to min ITC. Since there's a finite number of possible tasks, the longest task currently handled by the remaining agents determines an upper boundary for ITC values, thus max ITC is limited and doesn't grow with n.

So Eq. 6 can be written as

$$BU_n = X_n/(X_n + Y). \tag{7}$$

Hence the following holds:

$$\lim_{n\to+\infty} BU_n = \lim_{n\to+\infty} \frac{X_n}{X_n + Y} = 1$$

As the number of tasks increases, *Random 0* BU tends to 1, since this approach assigns each task to a random agent. Random assignments follow a uniform distribution, but they are subject to Gaussian noise. According to the Law of large numbers [8], by increasing the number of tasks (or random choices) the average number of tasks assigned to each agent should be close to the expected value; the distribution of the assigned tasks should converge to a uniform distribution. In figure 10 it is possible to observe an overall increasing monotonic trend; however, the convergence to 1 is not entirely appreciable since the graph has been truncated to improve readability.

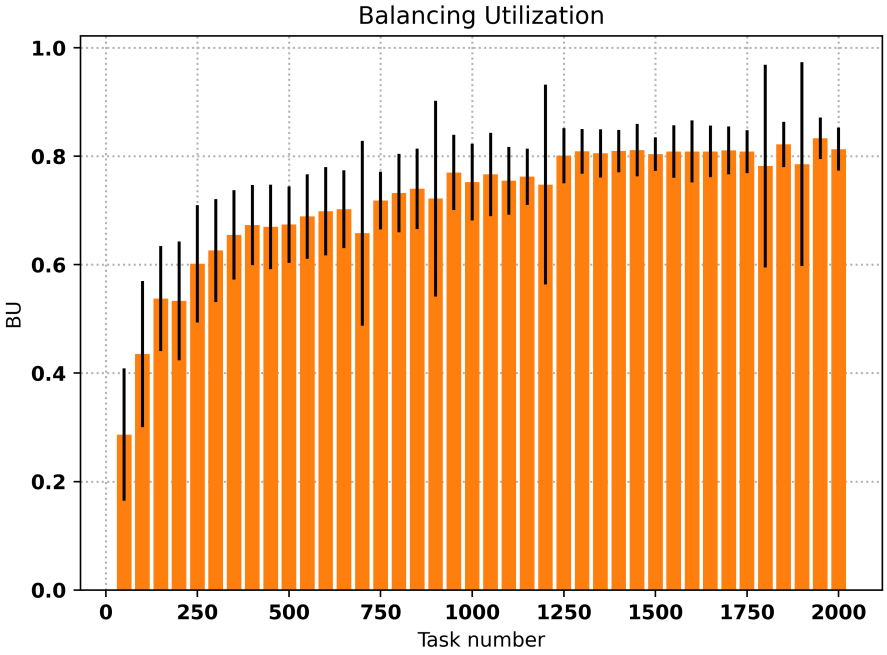


Figure 10: Balancing Utilization over an increasing number of tasks (50 to 2000), using the *Random 0* approach. Black stripes represent standard deviation. As the number of tasks increases, the BU score increases logarithmically.

Total Travel Cost Greedy techniques achieved the best TTC results, similar to TT.

It is important to note that greedy approaches exploit the knowledge of the initial environment to perform a better task allocation (i.e., robots choose tasks depending on the distance from the shelves), leading to better results than approaches that do not use this information.

Without collisions, the number of steps between a pod and the nearest picking station is constant. By assigning the closest task to the agents, greedy approaches reduce the number of steps required by each agent to reach the next pod, suggesting that the improvement of the results derives from the decrease in the number of steps to get to the next pod when a task is completed.

Given better parameter tuning and resources, GA could outperform greedy approaches; however, the limited amount of resources required and the computation times make this scenario not practical in a working reality.

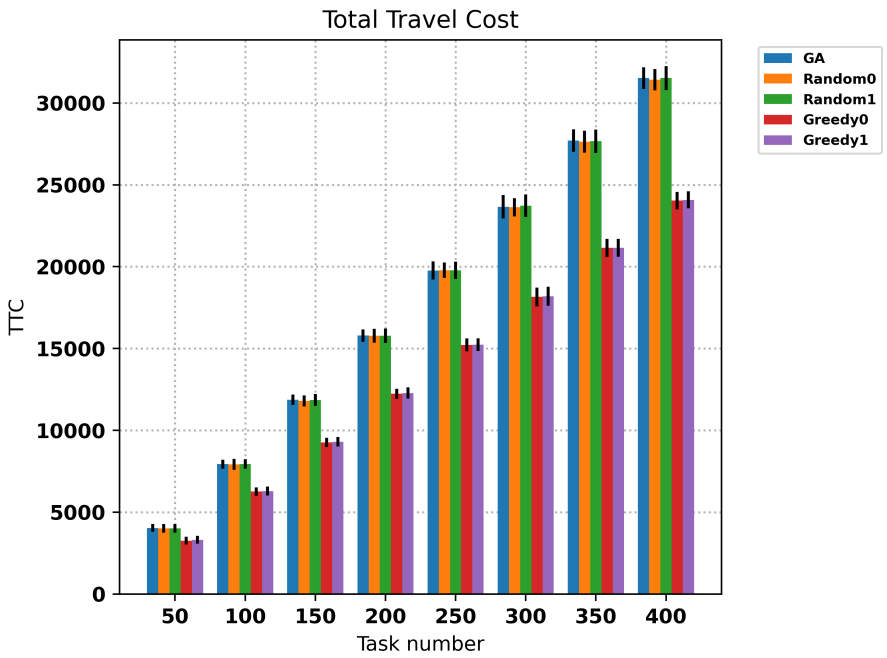


Figure 11: Total Travel Cost using different approaches. Standard deviation of the simulations is represented by the black stripes.

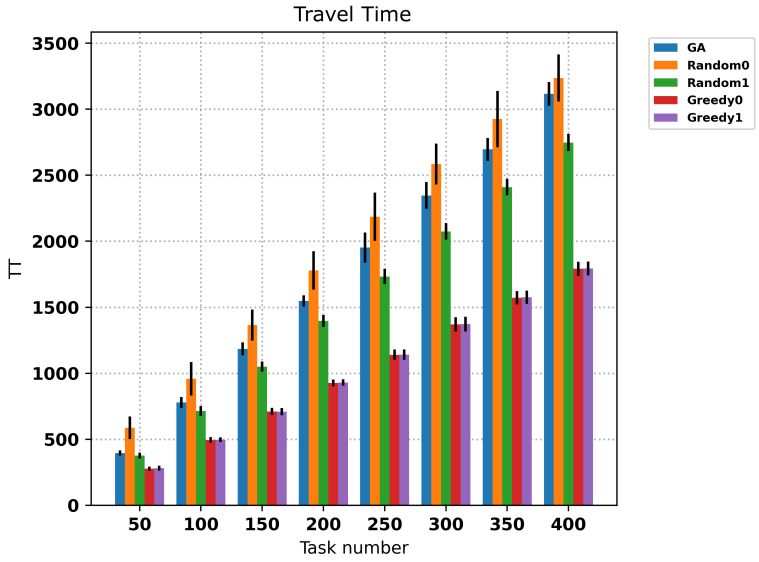


Figure 12: Alternative hypothesis. Total Time using different approaches. Standard deviation of the simulations is represented by the black stripes.

6.2 Alternative hypothesis

We propose a new view of the problem in which a pod can be delivered to the nearest picking station instead of a specific one; this averagely results in a shorter path from the shelf to the picking station. This hypothesis is applicable in a different warehouse context, in which parcels left in picking stations are transported by conveyor belts to their actual destination. It is not possible to directly compare the results since the proposed hypothesis refers to a hypothetically different warehouse, however it could be of interest to consider this alternative scenario.

With the standard hypothesis, each task is defined as a tuple $< A, B >$ with A the pod location and B the picking station of delivery, while with the alternative hypothesis a task is defined just by the pod location A.

The alternative hypothesis obtained a lower standard deviation because of the different task structure, leading to fewer possible paths; moreover, the alternative hypothesis got better TT and TTC scores (Fig. 12,14).

Compared to the alternative hypothesis, the standard one has a higher probability of collisions and deadlock, due to paths moving across the warehouse which intersect each other; instead, according to the alternative hypothesis, after an agent has picked up a pod, it has to move to the nearest picking station, which is most of the time reached by moving downwards.

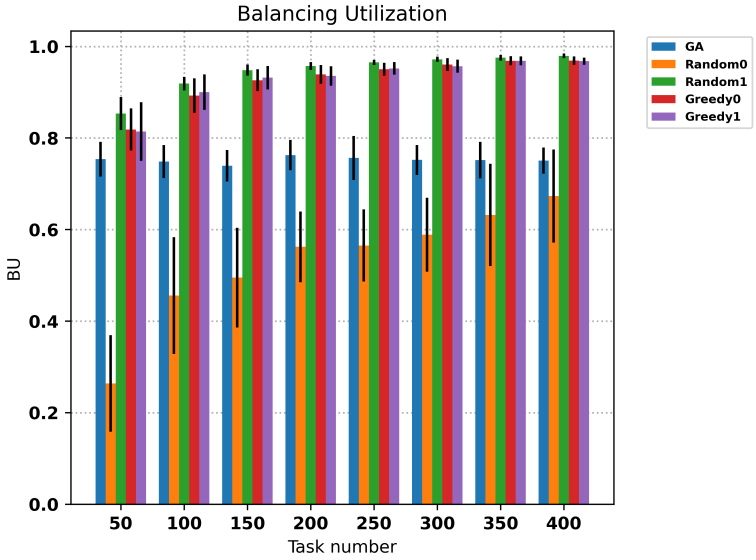


Figure 13: Alternative hypothesis. Balancing Utilization using different approaches. Standard deviation of the simulations is represented by the black stripes.

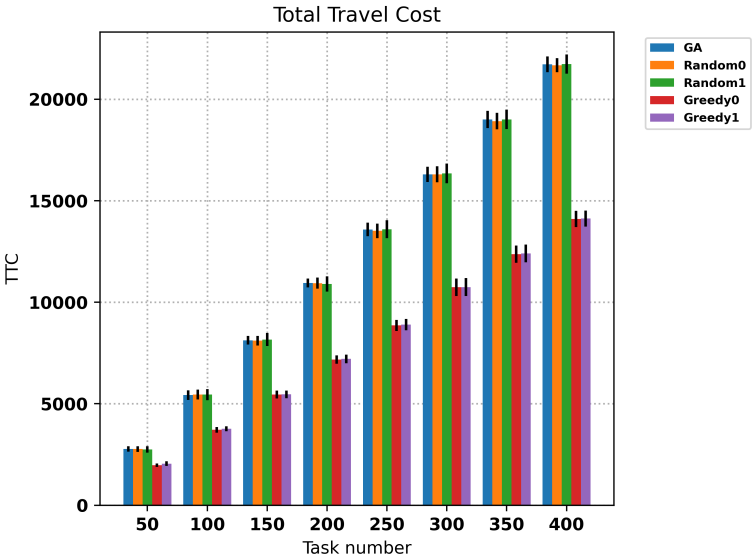


Figure 14: Alternative hypothesis. Total Travel Cost using different approaches. Standard deviation of the simulations is represented by the black stripes.

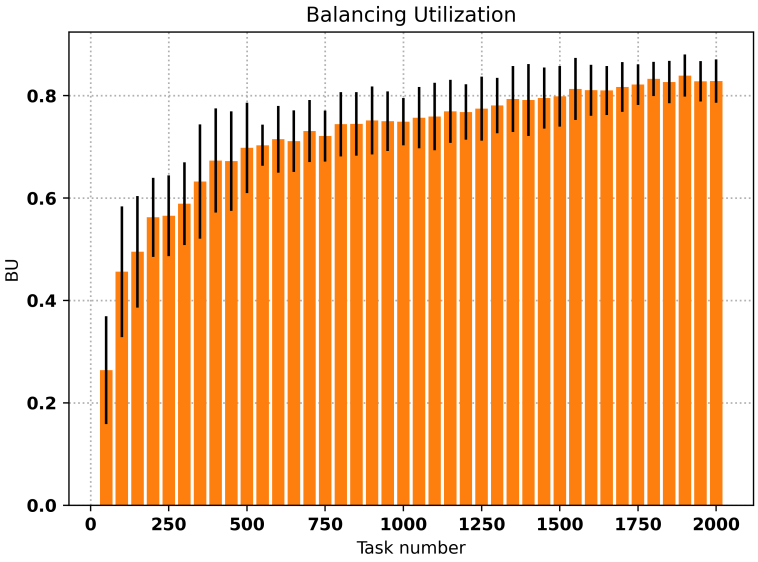


Figure 15: Alternative hypothesis. Balancing Utilization over an increasing number of tasks (50 to 2000), using the *Random 0* approach. Standard deviation of the simulations is represented by the black stripes.

6.3 Number of agents

Moreover different numbers of agents have been simulated to understand how metrics are affected. For each configuration, we considered 20 different agents initializations and mediated the results, using *Greedy 0* as allocation technique.

We limited the number of agents to 11 for computation purposes; moreover, more than 11 agents in such a limited space as the simulated warehouse leads to congestion and a high chance of deadlocks.

Travel Time decreases exponentially as the number of agents increases since tasks are more distributed among the agents (Fig. 16).

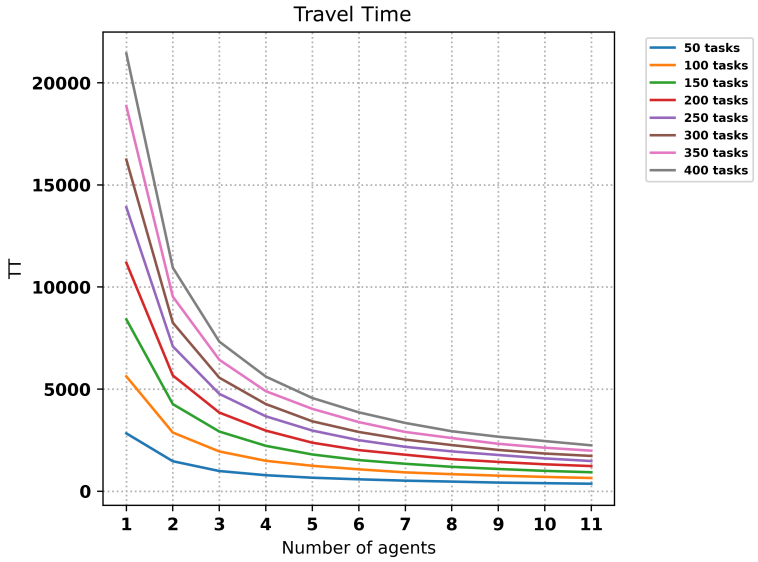


Figure 16: Travel Time for *Greedy 0*, varying the number of agents.

As the number of agents increases, Total Travel Cost also increases linearly (Fig. 17); we suppose this is due to the increasing number of interactions between agents to avoid collisions.

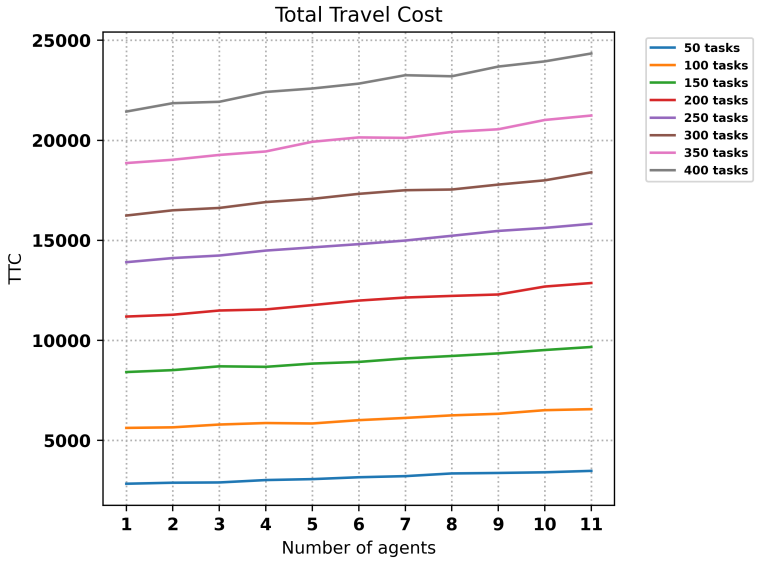


Figure 17: Total Travel Cost for *Greedy 0*, varying the number of agents.

Given the number of agents, the Balancing Utilization increases along with the number of tasks (Fig. 18), since the difference between min ITC and max ITC is always between a certain range, while their ratio increases.

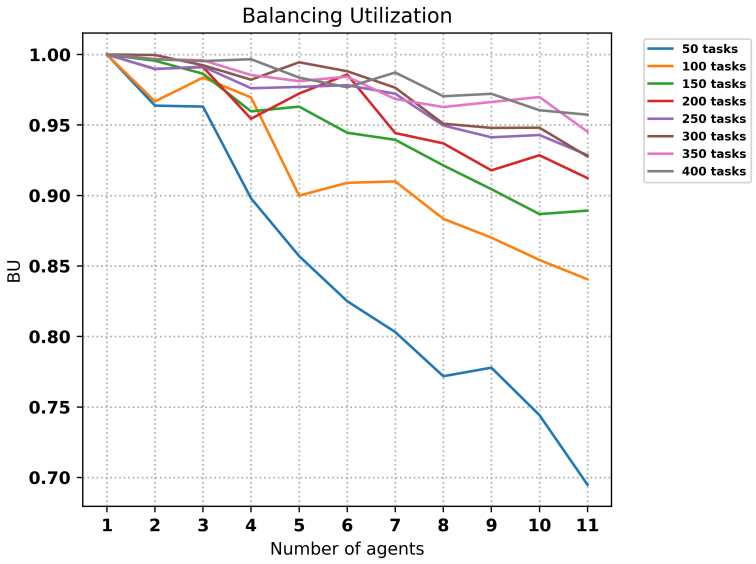


Figure 18: Balancing Utilization for *Greedy 0*, varying the number of agents.

7 Conclusion

This report introduced the MRTA problem and addressed task allocation and collision avoidance possible solutions. Instead of using sub-optimal A* as in [2], we used standard A* to find the shortest paths, preprocessing the routes in order to avoid extra computations and waiting times, allowing the agents to access quickly the paths to follow. We proposed and tested a collision avoidance mechanism with cooperation dynamics within agents and a greedy task allocation approach in which agents can take tasks from each other. Greedy solutions performed better than Genetic Algorithms and Random approaches, although they require the knowledge of the initial environment to compute distances between cells. The customized greedy approach always resulted in similar performances to the standard one. Optimize Instantaneous Allocation requires introducing different policies to address problems, e.g., balancing the number of tasks assigned to the agents and reducing the distance between each task for a single agent, requiring heavy computational resources and time. Introducing an actual collision detection policy and a cooperation mechanism leads to better representation of warehouses, resulting in higher reliability.

In conclusion, we reproduced and improved the results obtained by [2, 3].

7.1 Future works

Future works can address new task allocation (e.g., k-means clustering) and scheduling approaches, refining the collision strategy, introducing new metrics, considering other relevant variables describing an agent (e.g., fuel, battery consumption), and tuning the GA parameters (e.g. *pmutation*). It could also be possible to implement asynchronous and parallel approaches in order to make simulations more realistic, however this might introduce an overhead due to the access to shared resources.

References

[1] B. P. Gerkey and M. Mataric. A formal analysis and taxonomy of task allocation in multi-robot systems. *The International Journal of Robotics Research*, 23:939 – 954, 2004.

[2] L. Zhou, Y. Shi, J. Wang, and P. Yang. A balanced heuristic mechanism for multirobot task allocation of intelligent warehouses. *Mathematical Problems in Engineering*, 2014.

[3] J. Dou, C. Chen, and P. Yang. Genetic scheduling and reinforcement learning in multirobot systems for intelligent warehouses. *Mathematical Problems in Engineering*, 2015.

[4] N. Richards, M. Sharma, and D. Ward. A hybrid a*/automaton approach to on-line path planning with obstacle avoidance. In *AIAA 1st Intelligent Systems Technical Conference*, page 6229, 2004.

[5] S. Russell and P. Norvig. Artificial intelligence: a modern approach. 2002.

[6] P. Larranaga, C. M. H. Kuijpers, Roberto H. Murga, I. Inza, and S. Dizdarevic. Genetic algorithms for the travelling salesman problem: A review of representations and operators. *Artificial intelligence review*, 13(2):129–170, 1999.

[7] B. Yamauchi. Frontier-based exploration using multiple robots. In *Proceedings of the second international conference on Autonomous agents*, pages 47–53, 1998.

[8] F. M. Dekking, C. Kraaikamp, H. P. Lopuhaä, and L. E. Meester. *A Modern Introduction to Probability and Statistics: Understanding why and how.* Springer Science & Business Media, 2005.