Online Fleet Management using Reinforcement Learning in UPPAAL STRATEGO

Anonymous Author(s)
Affiliation
Address
e-mail

Abstract: We consider dynamic route planning for a fleet of Autonomous Mobile Robots (AMRs) doing fetch and carry tasks on a shared factory floor. We use the tool UPPAAL STRATEGO that employs Q-Learning for the synthesis of near-optimal plans. Furthermore, we deploy the tool in an online and distributed fashion to facilitate scalable, rapid replanning. While executing their current plan, each AMR generates a new plan incorporating updated information about the other AMRs positions and plans. We propose a two-layer Model Predictive Controller-structure (way-point and station planning), each individually solved by the Q-learning-based solver in UPPAAL STRATEGO. We simulate the AMR movement with Argos3 to show that the learned plans result in faster total completion times than a greedy approach to planning, with up to 25% savings on the average makespan. In addition, we construct a benchmark platform for comparing planning techniques and provide this under the MIT open-source license.

Keywords: Uppaal Stratego, Q-Learning, Fetch and Carry, Argos3 Simulation

1 Introduction

In modern industrial production, robots are increasingly working in close collaboration with humans in shared workspaces. In recent years, increased flexibility in manufacturing environments has, among other approaches, been enabled through the introduction of automated guided vehicles (AGV) and autonomous mobile robots (AMR). AMR and AGV systems have thus become paramount to modern production paradigms where increased flexibility in logistics applications are enabled through such technologies [1, 2, 3, 4]. Fleet management, in general, exhibits challenges such as being able to adapt running tasks to environment disturbances and resource availability. Furthermore, the fleet manager should also allocate priorities and coordinate tasks for multiple AMRs, while supervising the state of the individual AMRs, e.g., battery levels and subsequent automatic recharging. Finally, to avoid deadlocks in critical areas of the manufacturing environment, the fleet manager must also coordinate traffic in areas where the paths of multiple robots intersect.

In a fetch and carry application, each AMR has to pick up an order of items from different work stations throughout the factory floor [5]. Traditionally, the coordination is handled in a centralized manner through a simple fleet manager, leaving the low-level coordination and behavior to each robot, e.g., when one AMR meets another AMR on its path. This coordination, however, becomes intractable when the fleet starts to grow, and plans for an entire shift must be planned ahead and must have the capability to be adapted in real-time during execution.

In this paper, we consider a complex planning scenario entailing 10, 15, and 20 AMRs doing fetch and carry operations. The primary contribution is to demonstrate an approach to efficient online replanning of the entire fleet during runtime. We formulate the planning architecture as a dual Model-Predictive Control (MPC) approach, with each MPC solved by online Q-learning. The Q-learning-based solver synthesizes near-optimal plans for visiting stations and way-points. Our simulation architecture is based on Argos3 framework [6]. We demonstrate that our approach can learn policies that outperform a greedy approach by up to 25% on average. A secondary contribution of the paper is the developed simulation platform. It is released as open-source under the MIT license and can be used to compare different planning approaches against each other.

Submitted to the 5th Conference on Robot Learning (CoRL 2021). Do not distribute.
2 Related Work

Optimal fleet management, and herein task planning and replanning, is a fundamental challenge in multi-AMR systems. In recent years artificial intelligence methodologies have become prevalent in this domain of robotics research. We propose a combined learning and Model-Predictive Control (MPC) based approach to the challenge.

Recent research has proposed a Deep Q-Network-based approach to dispatch/navigation plans to serve randomly and constantly incoming orders and reduce time consumption for AGVs [7]. Ridesharing platforms share many of the same challenges to fleet management in industrial AMR scenarios, e.g., how waiting time is reduced through proactive dispatching strategies. Reinforcement Learning has been applied to specifically optimize dispatching strategies hence learning policies under uncertainty in the environment [8]. In this paper, we consider a stream of incoming orders, each comprising a set of items to collect from a factory floor. We assume that a simple global algorithm solves the allocation of AMRs to orders. The aim is to reduce the total completion time of the order queue. We differ from [7, 8] both in the method applied and by considering plans where multiple points have to be visited, allowing for a more considerable degree of choice of the AMRs. In particular we leverage recent advances in near-optimal synthesis for Priced Time Markov Decision Processes using UPPAL STRATEGO [9, 10]. Our method automatically incorporates rapid replanning, something we believe will lead to faster response time to rare and unpredictable events, situations that require special care when applying Neural Network-based approaches [11, 12]. We apply the tool UPPAL STRATEGO [13] in the setting of MPC - a combination which has recently seen encouraging performance in various domains [14, 15, 16].

As the fleet of agents starts to grow into large-scale systems, the fleet management problem becomes increasingly challenging. Traffic congestion and transportation efficiency become important factors, which have been investigated through multi-agent deep reinforcement learning methods [17]. To combat scalability issues, we enable each AMR to optimize its own plan while taking the future plans of the other AMRs into account. This allows the individual AMRs to adapt locally to (temporal) congestion to mitigate the issue. An alternative approach is taken by [18], who rely on a centralized computation of navigation plans, leading to scalability issues.

In traditional guided path-based transport systems such as AGVs, MPC [19] schemes have been proposed to simplify the traffic scheduling problem [20, 21]. Our approach builds on similar concepts, where our scheme relies on a distributed (individually for each AMR) MPC and a traditional Q-learning architecture. This approach enables optimization of the distributed online planning of the entire fleet of AMRs in a highly efficient manner, relying on near-optimal solutions to the optimization problem rather than computationally expensive analytical globally optimal solutions.

In contrast to [7, 8, 11, 18, 20, 21], we publish our evaluation platform in a manner where others can reproduce the results and compare with their own approach. Among the related work in this section, only [17] has made their simulation environment available to other researchers.

3 Case

As a motivational case, we consider a setup inspired by a world-leading industrial company with many custom orders apart from their primary production. These orders are currently hand-picked by shopfloor workers working in three shifts. We envision a future where this operation can be fully automated using larger fleets of AMRs, which would open up the possibility of producing custom orders to a much larger degree with increased profitability. Currently, humans pick a series of components to fulfill a single order, however, with no particular restriction on the specific sequence the components are packed in. They are self-managed and rely on intuition to avoid congestion at shelves with items in high demand. When all the items have been picked up, they must be delivered to one of the drop-off stations.

The floorplan, shown in Fig. 1, contains a connected graph of workstations – where items can be picked up, drop-off stations – where the finished orders are delivered and waypoints – helping navigating around obstacles. Only one AMR can work at any station at any given time, resulting in congestion if several AMRs need to access the same station. A job is a set of workstations that need to be visited in any sequence; thus, there are no dependencies between the stations. We consider a
job completed when 1. all its stations have been visited, and 2. the worker with the job arrives at a drop-off station.

4 Planning Architecture

As depicted in Fig. 3, we consider the planning problem independently for each AMR and restrict ourselves from a global planning procedure. We do so to avoid an explosion in computation-time with a growing fleet of AMRs where the global optimization problem, even for the fully controllable setting, can be shown to be NP-complete [22], implying an exponential growth in running-time assuming $P \neq NP$. We also note that the problem appears even less tractable, possibly undecidable in the face of stochastic behavior [23, 24]. This leads us to select an approach based on near-optimal plans obtained with machine learning techniques.

Figure 3: Architectural overview of a dual MPC approach for fleet management. The inner workings of each planner instance is shown in Fig. 4.

Figure 4: The data-flow of each of the two types of planners: Station planner and Way-point planner.
Towards the AMR’s general navigational capabilities, we assume that collision avoidance and path-following primitives are available such that high-level plans from the planners can be followed.

Firstly, an abstraction of the floorplan is represented by a weighted connectivity graph with three node types: way-points, work stations, drop-off stations. We partition the optimization problem into two simpler sub-problems using the abstracted floorplan. On the topmost level, we let the AMR plan (and re-plan) the sequence in which items are picked up, yielding what we denote a station plan. However, several routes exist between two stations, and optimizing over these gives the second optimization problem, namely which way-points to visit to reach the current next station, yielding a way-point plan. In particular, the way-point planner can micro-optimize to avoid triggering the (possibly expensive in terms of movement) collision avoidance mechanism.

To facilitate cooperation of the AMRs, we allow for the AMRs to communicate the following data:

1. current location (we assume a reliable and accurate positioning-system),
2. expected sequence of stations to visit (the station plan), and
3. expected sequence of way-points to visit (the way-point plan).

Our approach uses online planning in the sense that each AMR calculates a new way-point plan every time it reaches a node (point) in the graph and a new station plan every time it reaches a station.

Dividing the planning into two sub-problems also allows us to experiment with different planner compositions to study the effects of more advanced planners in isolation, which we utilize in our experiments.

### 4.1 Greedy Planner

As a baseline method for comparison to our Q-learning-powered approach, we use two naïve planners based on quick-computing algorithms with no consideration for the movement and location of other AMRs, resulting in what we call a greedy way-point planner and a greedy station planner. Even though both are called greedy, there is a major difference between the two greedy planners:

- The greedy way-point planner only needs to give the path to the next way-point, so its goal is to compute the shortest path to that one point.
- The greedy station planner needs to arrange the remaining stations to visit, decide if any other stations are used as way-points on the way, and end up at a drop-off station at the end.

The connectivity graph used by the station planner contains only the workstations and the shortest distance between them. Before any simulation starts, we calculate the complete distance matrix between all the points using the Floyd-Warshall algorithm with path reconstruction, which is a solution for the All Pairs Shortest Path problem. The result gives us the optimal solution in terms of distance between any two points in the graph, and we can obtain the exact path between two points. However, the station planner must also arrange the sequence in which the stations must be visited. The optimal solution of a restricted traveling salesman problem (we have to end the travel at a drop-off station) is NP-complete, so the greedy station planner uses the Nearest Neighbor approach. The planner will choose the closest remaining station in the job relative to the current location, then the closest of the remaining stations relative to the station just visited, and so forth. The resulting plan consists of the sequence of stations that will be passed through or worked at.

The way-point planner must compute the way-point path to the next station in the station plan. The connectivity graph for this planner uses a modified version of the Floyd-Warshall algorithm with path reconstruction. Two points are only considered connected if there exists a path between the two points that only contain way-points. This restricts the way-point planner from plans moving through irrelevant stations that are not in the station plan. Given this restriction, the way-point planner uses the connectivity matrix to obtain the shortest path to its destination.

### 4.2 Uppaal Stratego Model and Planner

For the UPPAAL STRATEGO-based planner, we develop an MPC-based approach. This implies defining an optimization problem (a model and an objective) and solving the problem using a solver.
In the case of the UPPAAL STRATEGO solver, such models can be encoded as Stochastic Priced Timed Games (STPG) [9], and optimization criteria can be given as an arbitrary function over the system state.

Given the distributed nature of our architecture, we model and optimize the problem as seen from each AMR independently. We state the optimization problem for the two layers, denoted as the way-point model and the station model. The general flow of the solver can be seen in Fig. 4.

From an abstract perspective, we can see the STPG models of the problems as a simplified, lightweight, and abstract model of the real world – or in our experiments, the Argos3 environment. Specifically, as our model does not capture the physical movement of the AMRs, but only their allocation to the edges in the connectivity graph of Fig. 1, we attempt to capture these imprecisions as stochastic behavior of our model, specifically in transit durations. The time it takes for the robots to travel along one of the edges follows a probability distribution, thus introducing stochasticity into the model. Currently, we use a uniform distribution between a min and max travel time, but this can be changed to any distribution at no cost to the planning.

Remark here that plans of the other AMRs are communicated, and thus it is possible to incorporate their current plans into the simplified model. Specifically, in our models, we model their behavior directly and thus implicitly define the (expected) future station and way-point allocation.

In this simplified model, the planner’s goal is to minimize the makespan of the current task of the AMR. In the way-point model, the goal is to reduce the time to reach the next station, while the station model attempts to reduce the makespan of completing an entire order. The planner has the choice of which stations (or way-points respectively) to visit. After a plan has been computed, the expected way-point and station plans are communicated to all other AMRs. Notice that the AMR only commits to the immediate next step of the plan, allowing for re-planning once this next milestone has been reached, i.e., the next way-point for way-point plans or the next station in the station plans.

Optimization of SPTG is known to be undecidable as it is undecidable even for the untimed fragment [25]. UPPAAL STRATEGO, therefore, utilizes a state-of-the-art partition-refinement Q-learning method [10] to compute near-optimal strategies, which are represented as binary decision trees. In short, the method repeatedly (up to a given budget) samples episodes from the STPG and utilizes Q-learning to derive a near-optimal solution. What is particular for the Q-learning employed by UPPAAL STRATEGO is the ability to conduct partition/refinement of the observed state-variables [10]. This allows the tool to generalize the observations onto sets of states, thereby extending the applicability of the Q-learning methods to the continuous domain. In particular, this allows the planner to generalize over the system’s temporal allocations, namely that two states of the system that are close in time are likely to have similar optimal plans. Even though UPPAAL STRATEGO generates entire strategies, we do not communicate does between the AMRs. The generated strategy is used to obtain the most likely sequence that the stations or way-points will be visited, and it is this sequence (or plan) that is communicated to the other AMRs. The Uppaal models we have created are part of the supplementary material.

5 Experimental Results & Discussion

In the following, we describe the experimental setup, in particular the permutation of the planning architectures from Section 4 which we have evaluated. To evaluate the proposed methods, we have implemented a simulation architecture based on the Argos3 framework [6], which is submitted as supplementary material for this paper.

Since we use Argos3 as our framework, we can simulate with or without visualization of the AMRs and their movement. The AMRs’ implementation is based on the footbot that is part of Argos3, and the simulation framework does not contain any stochastic behavior.

As mentioned in Section 3, a single job is a set of stations that the AMR must visit. Note that jobs cannot be collaboratively completed by several AMRs but must be completed by a single AMR. We assume that the job queue is unobservable, delimiting the AMRs from predictive job allocation based on the queue content. Instead we assign AMR in a first-idle-first-allocated manner. We leave for further work on the intelligent planning of AMR job allocation and predictive job allocation.
Jobs are generated by first (uniformly) sampling the job’s size between 2 and 4 stations (excluding the final drop-off). After the size is determined, we sample the \( n \) stations to visit. We do so according to two distributions to emulate different job loads. The first is a uniform distribution, where all stations have an equal probability of being part of the job. The second is a triangular distribution, where station number \( i \) has the probability \( \frac{2i}{2k+1} \) to be part of the job, assuming \( k \) workstations indexed from 1 to \( k \). The latter distribution represents situations where some items are required more often than others. We sample one fixed queue of 200 jobs for each distribution time, allowing us to compare the obtained makespans of the job queue between the methods directly.

We compare the different planning approaches by the total completion time for all jobs. In an experiment, all the AMRs on the map will have the same combination of planners, such as UPPAAL STRATEGO-based station planning and greedy way-point planner. We study changes to four parameters:

1. the planner configuration,
2. the simulation budget for UPPAAL STRATEGO,
3. and the number of AMRs on the map, and
4. the generated job queues.

We experiment with fleets of 10, 15, and 20 AMRs and study all four combinations of planners as way-point and station planners. The greedy algorithms are deterministic as there is no stochastic behavior in the AMRs’ movement in our simulation, so the total completion time of the 200 jobs is deterministic when both levels of planning are greedy. On the other hand, if one of the planning levels uses UPPAAL STRATEGO, we repeat the experiment 100 times to assess the stability, as the planner is stochastic in nature. We use the greedy/greedy instance as the baseline comparison and report improvements in the makespan relative to that instance. Note here that the logical time of the simulation is paused whenever a plan is generated, emulating an infinitely fast computer. This implies that the AMR gets a plan instantly, seen from the point of view of the simulator, which allows us to evaluate the effectiveness of the synthesized plans directly. Furthermore, we will later discuss the implications of computation times in Section 5.3. We ran the experiments on an AMD EPYC 7642 running Ubuntu 20.04 with hyperthreading and frequency-scaling disabled. We allocate two cores and 3 GB of ram for each experiment instance.

### 5.1 Uniform Distribution Experiments

First, we evaluate experiments run with jobs generated under a uniform distribution. In Figs. 5 to 8, the box plots are grouped by the number of AMRs in the given simulation. Each group contains simulations for greedy station planning and UPPAAL STRATEGO way-point planning (GSUW), UPPAAL STRATEGO station planning, and greedy way-point planning (USGW), and UPPAAL STRATEG station planning and UPPAAL STRATEG way-point planning (USUW). The y-axis values are the simulations’ logical completion time relative to the Greedy approach with the same number of AMRs. The value reported is \( \frac{\text{simulated completion time}}{\text{base line}} \), which give us the relative makespan.

We observe from Fig. 5 that GSUW for all fleet sizes, in general, adds no benefit for the AMRs’ planning but also observe a potential for a marginal improvement. The median of the relative completion times for 10 and 15 AMRs is above 1.0 but with outliers demonstrating improvement. The potential can be better observed in the 20 AMR case of GSUW but yields only an improvement for 75% of the cases.

On the other hand, USGW shows a general improvement for 10 AMRs, with nearly all experiments improving over the baseline. At 20 AMRs, USGW has a median improvement of 22.9%. The USUW controller, which uses UPPAAL STRATEG for both levels of planning, does not indicate any benefit compared to USGW and is noticeable worse in the case for 10 and 15 AMRs.

We also observe by comparing Figs. 5 and 6 that increasing the simulation budget offers no significant advantage or just a minor improvement of the median. An example is that USGW’s median changes from 22.9% to just 23.4%, even though the budget is 5 times larger.
5.2 Triangular Distribution Experiments

Compared to the experiments presented in Section 5.1, the non-uniform job creation is expected to lead to a higher degree of congestion potential, thus increasing the need for intelligent planning. The experimental results are shown in Figs. 7 and 8.

Similar to our observations in Section 5.1, the GSUW configuration for all fleet sizes shows little to no improvement when studying the median value but demonstrates potential in the outliers. On the other hand, USGW shows some improvements at 10 AMRs and noticeable improvements for 15 and 20 AMRs by up to a 25% reduction in the median makespan measured. The USUW controller indicates a benefit compared to USGW only when the simulation budget is 5000 and with 15 or more AMRs.

5.3 Computation Time

The CPU time spent by UPPAAL STRATEGO on synthesizing the individual way-point or station plan is shown in Figs. 9 and 10. We observe that the CPU time for way-point planning grows significantly faster than CPU time spent for station planning. This is detrimental to the USUW and GSUW approaches that rely on the frequent replanning using the UPPAAL STRATEGO way-point planner. We also note that the best performing combination (USGW) utilizes the greedy way-point planner and thus avoids this particular runtime overhead. However, the USGW planner still needs to compute the station plans.

Way-point plans are computed at every node in the graph, but the CPU time makes it infeasible to deploy neither USUW nor GSUW in a real environment. Having computation times about 10 seconds means that an AMR must hold still at the way-point while computing, thus blocking other AMRs.
On the contrary, we see the USGW as a feasible candidate for deployment. We anticipate that station plans can be computed simultaneously as background tasks with other tasks during loading and unloading. In addition, when an AMR works at a station, one can imagine that the AMR makes a guess on the map’s state 10 seconds from now and computes a strategy based on the guessed state. This strategy does not extend to the way-point planning as no movement idle-time can be expected between two adjacent way-points, and the non-movement of an AMR could block other AMRs.

6 Conclusion & Future Work

In this work, we studied the application of Q-learning in the context of Model Predictive Control (MPC) for Autonomous Movable Robot (AMR) fleet management.

To achieve scalability, we propose a distributed architecture and further subdivide the planning problem into two parts: 1. the sequence of work stations to visit, and 2. the sequence of way-points to visit between two work stations.

We demonstrate how such planning problems can be translated into Stochastic Priced Time Games and (approximately) solved by the tool Uppaal Stratego. To establish a baseline, we introduce two naïve planners (one for each sub-planning problem) based on shortest path algorithms and the Nearest Neighbor approach.

We demonstrate the performance of the proposed method in a controlled experimental setup based on the Argos3 framework. The proposed method demonstrates an increasing (with the size of the fleet) reduction in the makespan of a series of jobs by up to 25% compared to the greedy baseline. Lastly, we argue for the feasibility of deploying our method by studying the computation time and proposing concrete improvements.

6.1 Future Work

We collaborate with a local robotics startup XX and another department at XX University to get AMRs produced to test the online replanning system in a real-world setting.

Even though our approach currently scales well to 20 AMRs, it is likely that it, in its current iteration, would still suffer from some scalability issues related to the size of the map and the number of other AMRs to consider during the planning. Therefore, it is worth considering solutions where the individual AMR can delimit the observed area such that far away AMRs and distant future tasks are ignored in the optimization problem.

We also aim to work at other types of maps, where one has corridors where AMRs cannot move past each other. One feature of UPPAAL STRATEGO is that it can handle safety properties by finding safe control strategies that with 100% certainty avoid specified error conditions. An error condition could be that no AMR is allowed to enter an edge if another AMR is already on the edge moving in the opposite direction.
References


