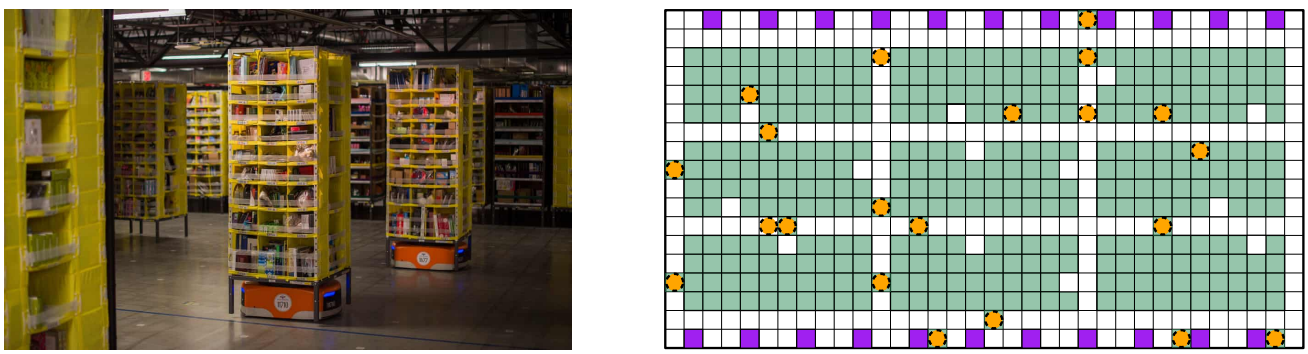
Project description—Efficient MAPF via massive preprocessing

Background

Coordinating the movement of a fleet of agents or robotsis a decades-old family of problems that has been intensively studied by the robotics and artificial intelligence communities. Applications of this family of problems can be found in diverse settings including assembly [12], evacuation [14], formation [13] and search-and-rescue [11].

One specific application of this general problem that gained significant interest in the research community is the warehouse domain. Here, storage locations host inventory pods that hold one or more kinds of goods. A large number (several hundreds and some times even several thousands) of robots operate autonomously in the warehouse picking up, carrying and putting down the inventory pods. The robots move the pods from their storage locations to designated dropoff locations where the needed goods are manually removed from the inventory pods (to be packaged and then shipped to customers). Each pod is then carried back by a robot to a (possibly different) storage location [22]. The successful use of such robots in warehouse applications led to a multi-billion industry led by tech-giants such as Amazon robotics and Alibaba [1]. For a visualization, see Fig. 1.



(a) (b)

Figure 1: Warehouse robots. (a) Amazon robots (orange) moving pods (yellow) containing goods in a warehouse environment. Figure adapted from `http://tiny.cc/ief6cz`. (b) Typical layout of pods in a warehouse. Pods and dropoff locations are depicted by green and purple squares, respectively. Agents are depicted using orange circles. Notice that some agents carry the pods and some don't.

In this work we are interested in one type of problem that can be used to model this application (as well as many others) called *Multi Agent Path Finding* (MAPF). Here, we are given a graph $G = (V, E)$ which, in our motivating example, is a discretization of the warehouse into cells where each cell represents a graph vertex and two vertices are connected in the graph if their corresponding cells are adjacent and do not contain pods. In addition, we are given $s : [1, \ldots, k] \to V$ and $t : [1, \ldots, k] \to V$

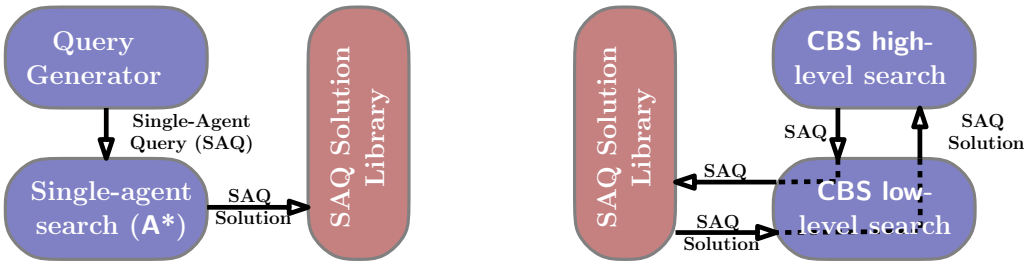
which map each one of the k agents to a start and target vertex, respectively. Time is typically discretized as well and at each time step an agent performs one of two *actions*: it can either wait in its cell or move to an adjacent cell. An action is considered *conflict free* or valid, if two agents do not occupy the same vertex or the same edge at a given timestep. The objective is to find conflict-free paths for the agents from their start to their target locations that minimize some objective. Typically, we wish to minimize the *makespan* which is the latest arrival time of an agent at its target location or the *flowtime* which is the sum of the arrival times of all agents at their target locations.[1]

Problem statement

MAPF problems are typically defined as a one-shot problem where a query is provided to the algorithm and it is tasked with computing a collision-free path for the fleet of agents between their respective start and target locations. However, in many cases the environment is known in advance (e.g., in our motivating warehouse application, the layout of the pods is known in advance) and the system is tasked with answering multiple queries. While this layout is constantly changing (e.g., as agents move from place to place, with and without pods), at any given point of time, the vast majority of the environment remains unchanged (e.g., in terms of pod location), when compared to the original layout. Thus, when solving the MAPF problem, we are often given the additional flexibility of *preprocessing* the environment in order to efficiently answer multiple MAPF problems in a query phase An immediate question that follows is "how can the query phase benefit from preprocessing the environment"?

Algorithmic background Before we describe our approach, we provide some algorithmic background on state-of-the-art MAPF algorithms. One can apply A* [10] to optimally solve the MAPF problem by treating the fleet of agents as one composite system. However, this approach does not scale with the number of agents as the search space and branching factor is exponential in the number of agents. An alternative state-of-the-art approach that is not based (directly) on A* is Constraint-Based Search (CBS) [3, 4, 17]. In the basic version of CBS, agents are associated with constraints indicating that at a given timestep, an agent cannot occupy a given vertex. The algorithm builds a *Constraint Tree* which is a binary tree where each node contains a set of constraints, and a cost of a solution that is not in conflict with theses constraints. Given a node in the Constraint Tree, a low-level search (e.g., an A*-based search) is run for the individual agents that is consistent with the constraints (namely, constraints are treated by the search as moving obstacles). Once a consistent path has been found for each agent by the low-level search, these paths are validated with respect to the other agents by simulating the movement of the agents along their planned paths. If all agents reach their target without any conflict, the node is declared as the goal node. If a conflict is found for two (or more) agents, the validation halts and the node is split by adding two new nodes with new constraints.

[1] For a complete taxonomy of different MAPF problems, including conflicts types, objective functions and more, see [20].



(a) Preprocessing phase (b) Query phase

Figure 2: Ideal approach to dramatically improve the runtime of CBS by preprocessing the environment to efficiently answer single-agent queries (SAQs) by the low-level search of CBS. (a) All $O(n^2)$ different queries together with all possible constraints are generated by the *query generator*. A solution for each SAQ is computed by a single-agent search algorithm such as A* and all solutions are stored in the solution library. (b) In the query phase, when the high-level search of CBS needs a solution to a SAQ, the low-level search simply retrieves the precomputed solution from the library.

Efficient MAPF by preprocessing. The lion's share of the running time of CBS-based algorithms resides in solving the individual-agent search problems using algorithms such as A*. This gives a possible answer to the question posed—we are in need for a data structure that can efficiently answer single-agent shortest-path queries that will allow us to replace these time-consuming A* searches. Ideally, we would like to precompute a shortest path for each possible single-agent query that the algorithm may encounter. In the query phase, given a single-agent query, the low-level search of CBS would then only need to perform a lookup to retrieve the precomputed shortest path. For a visualization, see Fig. 2.

In this project you will have to implement the proposed approach and suggest improvements to it. You do *not* need to write everything from scratch. C++ implementations of MAPF algorithms can be found at `http://mapf.info/index.php/Main/Software`.

References

[1] Three ways Amazon, Alibaba, and Ocado benefit from warehouse robots. `https://www.innovecs.com/ideas-portfolio/amazon-alibaba-ocado-use-warehouse-robots/`. Accessed: 19-07-19.

[2] Max Barer, Guni Sharon, Roni Stern, and Ariel Felner. Suboptimal variants of the conflict-based search algorithm for the multi-agent pathfinding problem. In *ECAI*, volume 263, pages 961–962, 2014.

[3] Eli Boyarski, Ariel Felner, Roni Stern, Guni Sharon, David Tolpin, Oded Betzalel, and Solomon Eyal Shimony. ICBS: improved conflict-based search algorithm for multi-agent pathfinding. In *IJCAI*, pages 740–746, 2015.

[4] Peter E. Hart, Nils J. Nilsson, and Bertram Raphael. A Formal Basis for the Heuristic Determination of Minimum Cost Paths. *IEEE Transactions on Systems, Science, and Cybernetics*, 4(2):100–107, 1968.

[5] James S Jennings, Greg Whelan, and William F Evans. Cooperative search and rescue with a team of mobile robots. In *International Conference on Advanced Robotics (ICAR)*, pages 193–200, 1997.

[6] Bartholomew O Nnaji. *Theory of automatic robot assembly and programming.* Springer Science & Business Media, 1993.

[7] Sameera Poduri and Gaurav S Sukhatme. Constrained coverage for mobile sensor networks. In *ICRA*, volume 1, pages 165–171, 2004.

[8] Samuel Rodriguez and Nancy M Amato. Behavior-based evacuation planning. In *ICRA*, pages 350–355, 2010.

[9] Guni Sharon, Roni Stern, Ariel Felner, and Nathan R. Sturtevant. Conflict-based search for optimal multi-agent path finding. In *AAAI*, 2012.

[10] Roni Stern, Nathan R. Sturtevant, Ariel Felner, Sven Koenig, Hang Ma, Thayne T. Walker, Jiaoyang Li, Dor Atzmon, Liron Cohen, T. K. Satish Kumar, Roman Barták, and Eli Boyarski. Multi-agent pathfinding: Definitions, variants, and benchmarks. In *SOCS*, pages 151–159, 2019.

[11] Peter R Wurman, Raffaello D'Andrea, and Mick Mountz. Coordinating hundreds of cooperative, autonomous vehicles in warehouses. *AI Magazine*, 29(1):9–9, 2008.