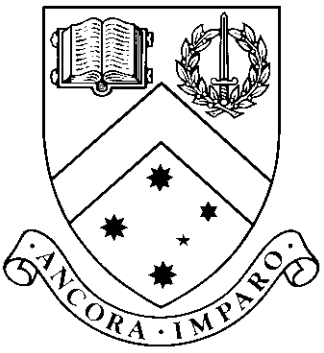
School of Computer Science (BICA)
Monash University



Literature Review, 2017

Review of optimal multi-agent pathfinding algorithms and usage in warehouse automation

Phillip Wong

Supervisors: Daniel Harabor,
Pierre Le Bodic

Contents

1 Introduction

The order picking process is the number one expense in the operating cost of warehouse systems (De Koster et al. (2007)). Order picking involves the retrieval of inventory from around the warehouse. This project looks at the use of multi-agent pathfinding (MAPF) algorithms within order picking. In particular, we explore Kiva systems (Wurman et al. (2008)) where the order-picking process is performed by autonomous vehicles.

Firstly we define the components making up a Kiva system (Figure 1). *Drive units* are our autonomous vehicles which move around the warehouse, retrieving and delivering shelving units. These shelving units are known as *storage pods* and are usually organized in aisles. Lastly we have the *picking station* where a human workers are situated. With these defined, the process of order picking in a Kiva system is as follows:

1. An order is received requesting one or more products
2. The system identifies storage pods containing the said products
3. The system schedules drive units to pickup the storage pods
4. Once a drive unit is ready, it will move to the storage pod, pick it up and deliver it to the correct picking station
5. Once the drive unit has delivered the storage pod, a human worker situated at the picking station will pick the product off the storage pod and pack it into a box
6. After the storage pod has been picked, the drive unit agent returns it to a suitable location
7. Finally once all the products for the order have been packed, the box it is processed and sent off

As this process describes, Kiva systems have a number of inter-dependent problems to solve. step 1. requires that the system assigns an order to a picking station; step 3. involves identifying a suitable drive unit to picking a storage pod. Other notable areas include: the replenishment requirement for the batteries of drive units and the products stored in the storage pods; a robotics problem described in the Amazon picking challenge (Correll et al. (2016)) which aims to make the picking process autonomous.

Figure 1: A worker picking an order from a storage pod. The orange robot underneath is the drive unit. (Al Dekin (2014))

Solving all these problems and applying them in one solution has been done by the creators of Kiva systems but is unfortunately is not described in any literature. The consequences of optimizing these systems together would be very useful but is out of the scope of this project. Here we will be focusing on heart of the order-picking process which is the MAPF problem.

We see that a MAPF problem arises when a drive unit agent retrieves, delivers or

returns a storage pod (steps 4 and 6). Here we need a solution to coordinate drive unit agents so they can move around the warehouse while avoiding collisions. This is not a simple task as solving MAPF optimally is an NP-hard problem (Yu and LaValle (2013b)). We can conclude that Kiva systems in production are using a suboptimal algorithms as these warehouses run over 1000 agents. This forms the motivation behind our project, to look at the benefits we may find in Kiva systems by using an optimal approach and what we will have to sacrifice for scalability. Later in Section 5, we look at the performance of these optimal algorithms.

Our main contribution in this literature review is comparing the scalability of optimal and suboptimal MAPF algorithms and seeing how past work has tried to bridge this gap. Additionally, we look at any algorithms which were applied to a Kiva system setting; what simplifications they assumed and what conclusions they have drawn to the usability.

Section 2 describes in detail, the MAPF problem and according terminology. Section 3 formally outlines the Kiva system problem. Section 4 describes some state-of-the-art suboptimal MAPF algorithms. Section 5 looks at optimal MAPF algorithms. Section 6 looks at *operator decomposition* and *independence detection*. Finally, Section 7 discusses the findings of this literature review, identifying possible research gaps in improving scalability and applications to Kiva systems.

2 Background

2.1 Single-agent pathfinding

The single-agent pathfinding problem aims to find a path from start node to goal node. A detailed review of single agent search algorithms can be found in the *grid-based path planning competition* (Sturtevant et al. (2015)). The paper overviews all the entries to the competition and their relative performance. The algorithms are run on a number of game maps from Sturtevant (2012) which are used for benchmarking and these maps are also used in many MAPF benchmarks. The results from Sturtevant et al. (2015) are a few years old but we are expecting new approaches from this year's GPPC.

In this project, we aim to look at two single-agent search algorithms: *jump point search* (Harabor et al. (2011)) and *compressed path databases* (Strasser et al. (2015)).

2.2 Multi-agent pathfinding

In this section, we describe the problem of *multi-agent pathfinding* (MAPF). The MAPF problem is a generalization of the single-agent pathfinding problem for multiple agents and single agent pathfinding algorithms are often used a sub-solver within the MAPF algorithm. The task of MAPF is to coordinate a number of agents moving around an environment. Each agent must find a path to their goal while ensuring that no agent collides with another. A common secondary objective is to minimize either the makespan of the system or the sum of costs. As mentioned in Section 1, solving MAPF optimally is an NP-hard problem (Yu and LaValle (2013b)). Suboptimal and optimal MAPF have vastly different properties in scalability and solution quality. Thus they are compared in different sections [4, 5] of this literature review. Later in Section 3 we formally define the MAPF problem in the context of Kiva systems.

Below are some important properties of MAPF algorithms. Table 1 shows a comparison of these properties across the all MAPF algorithms discussed in this literature review.

Completeness: A complete algorithm will always return valid solution to the MAPF problem if one exists.

Solution quality: Solution quality is usually measured in makespan or sum of costs. Makespan is the number of timestep required from the start state to the goal state where

all agents are at their goals. Sum of costs is the sum of the number of timesteps taken for each individual agent to reach their goal.

- Optimal: finds the optimal path to the goal minimizing the objective
- Bounded suboptimal: given a optimal path cost of C, the cost of a bounded suboptimal solution, B is guaranteed to be within C and $w * C$ where w is a user set parameter. Hence, $C \leq B \leq wC$.
- Suboptimal: finds a path to the goal with no guarantee of solution cost

Anytime: The search can spend more computation to improve the solution quality or number of solutions, it can stop earlier at any time and return a path.
Centralized / Coupled: In a centralized approach, all agents are looked at globally and a decision is made with all paths in mind. In a decentralized approach, paths are found for one agent at time. A centralized approach is usually has the properties of an optimal solution and is not scalable. Accordingly, a decentralized approach scales up to many agents but is suboptimal and often non-complete.
Success rate: Success rate is often used as a measure of performance for optimal algorithms. It describes the percentage of instances which were capable of being solved within a fixed time t, usually set at 5 minutes.

2.3 Bounded-suboptimal variants

A *bounded suboptimal variant* is a variant of an optimal MAPF algorithm. The defining feature of a bounded suboptimal variant is that we are able to guarantee that the cost of the bounded suboptimal solution, B lies within the optimal cost C and $w * C$ where w is a user-defined parameter. A bounded suboptimal variant provides a way of trading speed for solution quality and is especially relevant this project as we want to find a middle ground between optimality and speed Kiva systems. The optimal algorithms in 5 will mention any bounded suboptimal variants if they exist.

3 Problem definition

This section will formally define the MAPF problem in the context of Kiva Systems. The goal of MAPF is to coordinate agents and find a path for each agent to their goal while ensuring that no path conflicts. Here, a path is a sequence of actions, where an action is one of $\{up, down, left, right, wait\}$. A path is said to conflict with another when on the same timestep: two agents share the same node or two agents cross the same edge.

The warehouse environment is modelled as a square grid map to reduce the complexity of the MAPF algorithm. This is a Graph $G(V, E)$ where V is the set of nodes on the grid map and E is the set of edges between nodes. On this map, we have the components of a Kiva system: storage pods, drive units and picking stations (Figure 2), hence these are:

- A set of picking stations, P, where for all $p \in P$, $location(p) \in V$
- A set of storage pods, S, where for all $s \in S$, $location(s) \in V$
- A set of drive units, K, where for all $k \in K$, $location(k) \in V$ and $goal(k) \in V$

Another aspect of Kiva systems which makes the MAPF problem hard is that there are a large number of robots within them, with a density of up to 1 in 6 robots. The space within the warehouse is also very constrained as the storage pods are setup in narrow aisles which only have space for 1 drive unit at a time. However this is relaxed as drive units are able to maneuver under storage pods. In other words, storage pods are only an obstacle to a drive unit if the drive unit is carrying a storage pod. Collision always occurs between drive units, picking stations and other drive units.

Different to traditional the MAPF problem, the objective of a Kiva system is mainly to minimize the downtime of human pickers. This requires the drive units to bring a steady

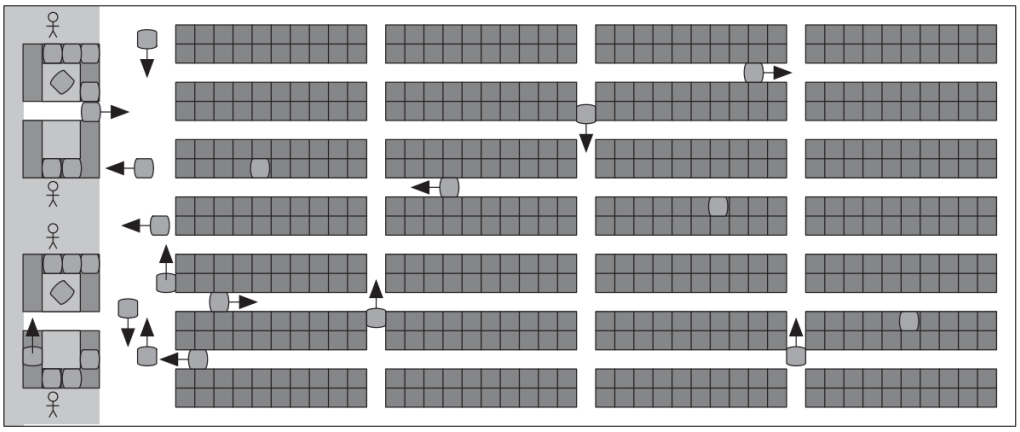


Figure 2: Kiva system representation on square grid map (Wurman et al. (2008))

supply of products to the order picker. Additionally Kiva systems run 24 hours a day hence the measures of makespan and sum of costs are not quite representative as they require a goal-state to be reached. For MAPF algorithms which were applied to a Kiva domain, we look at whether new objectives were used to measure performance.

4 Suboptimal multi-agent pathfinding algorithms

This section looks at suboptimal MAPF algorithms, here ware are mainly focused on makespan and scalability. As these will be helpful for comparison against opitmal MAPF algorithms.

4.1 Cooperative A*

Cooperative A* is a decentralized algorithm where a search is performed for one agent at a time, the resulting path is stored in a *reservation table*. Subsequent agents must treat any stored paths in the reservation table as obstacles when performing their search. The reservation table is often stored as a matrix describing an agent is at a node n at time t.

A variant of this is *hierarchical cooperative A** (HCA*) which relies on an algorithm called *reverse resumable A** in order to calculate a smarter heuristic. A variant of HCA*, *windowed hierarchical cooperative A** (WHCA*) restricts the time window of the reservation table for agents to query. During an agent's path, this window is updated and the agent will see more information in the reservation table. Silver (2005) defines HCA* and WHCA* and provides background of work on CA*.

Cooperative A* and HCA* do not scale well but WHCA* is found a solution in under 0.6 ms per agent and was suggested that it is usable in real-time. For solution quality, the average path length was calculated with the baseline being the shortest path ignoring other agents. WHCA* generally found a path $\frac{3}{2}$ longer than that of the shortest path (with 100 agents in the simulation). This gap scaled as the number of agents increased. WHCA* is often used a benchmarks of newer algorithms and but these algorithms often surpass it in runtime.

4.2 MAPP

MAPP (Wang and Botea (2011)) is a suboptimal algorithm which has a polynomial runtime complexity and is complete for slidable problems. It finds a runtime complexity of $O(n^2k^2)$ where n is the number of nodes on the graph and k is the number of agents. It guarantees this by identifying sets of units which are guaranteed to run in polynomial time. A slidable problem is identified when there exists an Ω path for each agent as seen in Figure 3.

In comparison to FAR and WHCA*, MAPP excels in having high completeness at 92%–99.7%, compared to FAR (81.87%) and WHCA* (80.87%). Speed-wise, MAPP is comparable to WHCA* and 10 times slower than FAR but in scenarios where MAPP has Ω paths, MAPP gains significant benefits and is only 2.18 times slower than FAR and 4.8–5.2 times faster than WHCA*.

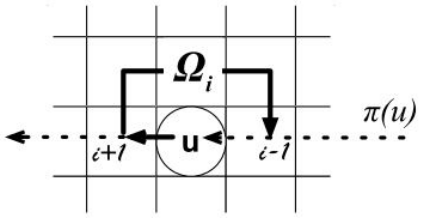


Figure 3: An example of an alternate path, Ω (Wang and Botea (2011))

It should be noted MAPP identifies the situation in which agents attempt to pass a tunnel as MAPP fails otherwise at this case. Their solution, called a *buffer zone extension* is used and importantly it has little performance degrades, with the same runtime complexity as the regular MAPP algorithm and little additional memory requirements. This is relevant as MAPP would not function without in the narrow aisles of Kiva systems without this.

MAPP may be a desirable choice for use in Kiva systems as it scales up to 2000+ agents and is complete. Kiva systems often have high density of agents so MAPP would perform well in finding a solution compared alternative suboptimal algorithms (WHCA* and FAR).

5 Optimal multi-agent pathfinding algorithms

This section looks at three optimal MAPF algorithms, *centralized A**, *conflict based search* and *increasing cost search tree*. We focus on their approach to improving scalability and any applications to Kiva systems.

5.1 Centralized A*

Centralized A* is an optimal algorithm which is exponential in the number of agents. It uses a global heuristic usually the sum of individual costs[1] to lead towards the optimal solution. One recent variant *enhanced partial A* expansion* (EPEA*) by Goldenberg et al. (2014), only generates children which have the same f-cost as their parent node.

Two algorithms, Independence Detection and Operator Decomposition help to reduce the number of agents being processed and hence provides an exponential speed up to Centralized A*. More details about ID and OD is described in Section 6. Both EPEA* and A*+OD were previously state-of-the-art in optimal MAPF and are used as benchmarks for many other optimal MAPF algorithms.

[1] sum of individual costs: the sum of path distances when ignoring collisions with other agents

5.2 Conflict based search

The conflict-based search algorithm (Sharon et al. (2015)) relies on building a binary tree known as a *constraint tree* (CT). Each node in the CT describes a set of constraints, a solution and the cost of this solution (Figure 4). A constraint prohibits an agent a from being at node n at timestep t, in the form of $\langle a, n, t \rangle$.

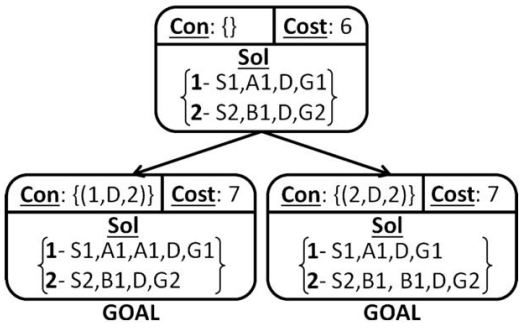


Figure 4: An example of a Constraint Tree (CT) (Sharon et al. (2015))

A node in the CT is processed by finding the shortest path for each agent to their goal making sure to satisfy the constraints described by the node. If this path is not valid and a collision has occurred between two agents: a_1, a_2 then CBS branches and adds two successor nodes. Both successors inherit the constraints of the parent node. Additionally the left successor will adds a new constraint $\langle a_1, n, t \rangle$ while the right successor will add the constraint $\langle a_2, n, t \rangle$. CBS is optimal as it chooses the next node to expand by searching for the lowest cost node in the CT.

As CBS branches at every conflict, it is exponential in the number of conflicts. In comparison to EPEA*, CBS performed worse in maps with open spaces where sets of agents are more likely to be *strongly coupled*, that is they have a high rate of path conflicts between one another. On the other hand CBS was found to perform better in maps with bottlenecks.

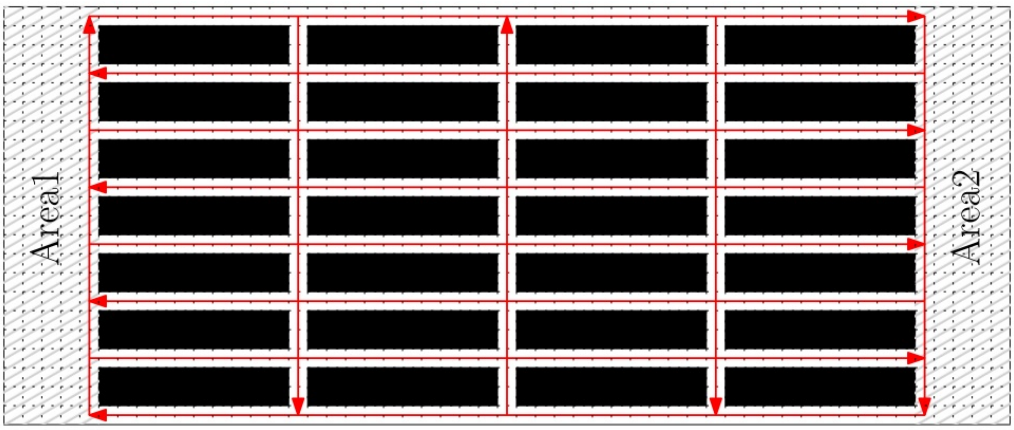


Figure 5: Kiva-like domain with Area1 on the LHS and Area2 on the RHS (Cohen et al. (2016))

A bounded-suboptimal variant of CBS was developed called improved-ECBS (iECBS) Cohen et al. (2016). iECBS was tested on a Kiva system domain of size 22×54 with 130 agents (Figure 5). This test did not model the Kiva problem as described in Section 3, instead the problem generated half the agents in *Area1* and half the agents in *Area2* of the map. Agents were then given the task of moving random goal location in the opposite area. Testing was performed against ECBS and found speedups in runtime of approximately 10

times. The scalability of iECBS was measured in this environment and found it performed reasonably at around 150-160 agents at a success rate of 90% before dropping off. iECBS was not testing aginst CBS but ECBS was tested previously (Barer et al. (2014)). Here they used the DAO game map (BRC202) from Sturtevant (2012) and found that CBS had a steep fall off in success rate at 20-25 agents as compared to ECBS at 130-140 agents. This is a significant increase in scalability and the user set parameter (2.3) for this test: $w = 1.01$ indicates the solution quality was near optimal.

5.3 Increasing Cost Search Tree

This section describes the Increasing Cost Search Tree (ICTS) an optimal, complete MAPF algorithm (Sharon et al. (2011)). The ICTS algorithm generates a tree known as an *increasing cost tree* (Figure 6). Each node in the tree describes a cost C for each agent $[C_1, \ldots, C_k]$. ICTS generates paths based off the cost for each agent described in the node. It explores all permutations of paths which are equal to C_i and a valid solution when a combination of paths is conflict-free. To build the tree, the root starts containing the an optimal cost ignoring any collisions with other agents. When it does not find a valid solution, the algorithm will branch and generate k successors, one for each agent. In each of the successors, the cost of i will increment e.g. for successor 2: $[C_1, C_2 + 1, \ldots, C_k]$. By exploring the tree in breadth-first manner, ICTS ends up with an optimal solution.

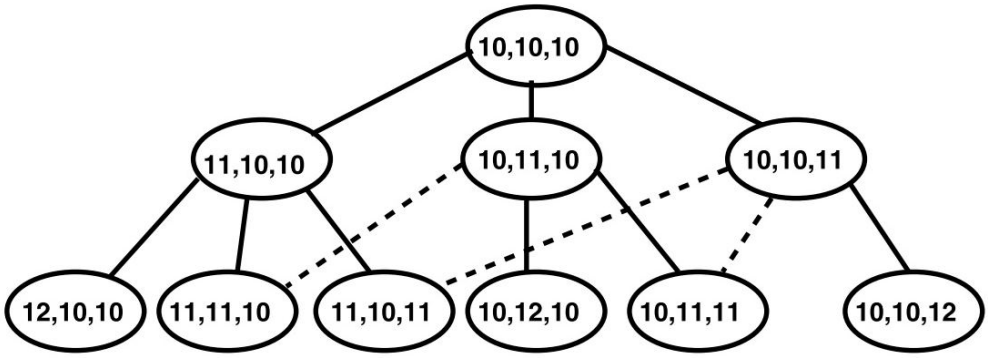


Figure 6: ICT for three agents (Sharon et al. (2011))

The ICT grows exponentially in Δ, where delta is the difference between the optimal collision-free cost and the optimal cost which ignores collision (the root node). Hence the downfall of ICTS is an environment with many conflicts occurs or it is costly to resolve the conflicts, thus having a large Δ. In comparison to $A*$ which is exponential in k, ICTS is exponential in Δ so it benefits most on open maps with many agents.

5.4 Combined-target assignment and pathfinding

This section describes the work by Ma and Koenig (2016), which looks at developing MAPF algorithms for autonomous warehouse robots, they call this a combined assignment and pathfinding (TAPF) problem. Importantly the difference between a MAPF problem and a TAPF problem is that:

- Agents are split into teams and a team of k agents will have k goal locations
- Agents are *anonymous*: the goal location is interchangeable for each agent, so any agent can be assigned to a goal to satisfy the MAPF problem

An important property is that anonymous MAPF problems can be solved in polynomial time. Their paper describes two algorithms which are made to solve the TAPF problem. Both of these use a hierarchical approach with a high-level and a low-level algorithm.

First is conflict-based min-cost flow (CBM), which uses CBS (5.2) at the high-level and a min-cost max-flow algorithm on the low-level. CBM is applied to Kiva Systems which is their motivation behind formulating the TAPF problem. Their problem instances for Kiva Systems contain 420 drive units moving to 7 picking stations, the total warehouse size was not stated. Results suggested that was performed well enough to be used in real-world applications of Kiva systems, with an 80% success rate, mean makespan of 63.83 seconds and mean running time of 91.61 seconds. Importantly the results beat that of bounded suboptimal MAPF algorithms which were designed for Kiva systems.

Next, ILP (TAPF), which is similar to CBM but instead of using CBS at the high level it uses an ILP-based TAPF solver is used at the low-level. This ILP solver is based off the work by (Yu and LaValle (2013a)) and uses a network-flow approach. Results found that this solver performed worse than the ILP MAPF solver in runtime and success rate but won in makespan.

Overall the formulation of the TAPF problem is beneficial in improving scalability between optimal and suboptimal algorithms and will be looked at in further work.

6 Independence detection and operator decomposition

This section looks at two MAPF techniques developed by Standley (2010) which can be applied to most MAPF algorithms and provide significant speedups.

Independence detection. Independence detection (ID) finds independent groups of agents where it can be proven for each of these groups that the agents within them will not be in conflict. These groups are split and treated as seperate subproblems. Importantly this reduces the number of agents and provides an exponential speed up for A* and a polynomial speedup for many non-A* based algorithms.

Operator decomposition. Operator decomposition (OD) reduces the branching factor by performs one agent's action at a time as oppose to an operator applying to every agent simultaneously. OD gives a much deeper tree but with a much smaller branching factor. At every k levels in the tree we get an equivalent state where OD would not be used and hence the branching factor can be seen as a instead of a^k where a is the number of possible actions and k is the number of agents.

Standley (2010) found that OD had larger benefits and scaled better than ID but they can both be used in conjunction.

7 Discussion

Name	BSV	Complete	Complexity	KS	Comparison
Optimal					
CBS	N	Y	Exp conflicts	Y	ICTS, EPEA*
Centralized A*	Y	Y	Exp agents	N	–
ICTS	Y	Y	Exp Δ	N	A*, A*+OD
ILP (TAPF)	N	Y	Poly conflicts	N	ILP (NF)[2], IPL (TAPF)
CBM	N	Y	Poly conflicts	Y	ILP (NF), IPL (TAPF)
Suboptimal					
MAPP	N	Y	Poly agents, nodes	N	WHCA*, FAR[3]
WHCA*	N	N	Poly agents	N	–

Table 1: Table comparing the stated algorithms and their properties. ***BSV****: Does the algorithm have a bounded-suboptimal variant.* ***Complexity****: Runtime complexity.* ***KS****: has the algorithm been applied to Kiva systems.* ***Comparision****: the algorithms used for benchmark testing.*

In regards to suboptimal algorithms, none were applied to Kiva systems. It would be useful to see the relative performance between suboptimal solutions and optimal solutions within a Kiva system domain. MAPP seems a good choice for comparison against optimal MAPF algorithms. It is far from optimal but is complete and this is especially relevant to Kiva systems where the environment is dense with agents. Other suboptimal algorithms (FARR, WHCA*) would likely fail to find a solution in these situations.

We found that optimal algorithms can improve scalability by relaxing the solution quality and creating a bounded-suboptimal variant. Two examples of these are iECBS (Cohen et al. (2016)) and EPEA* (Goldenberg et al. (2014)). In the case of CBS, this approach increase the scalability of the optimal algorithm by a factor of around 6 times. Hence allowing for a solution to be found with 85% success rate on the DAO game map (BRC202), populated with 200 agents.

Another approach to improving scalability, Ma and Koenig (2016) finds large gains in defining a special case of the MAPF problem, the TAPF problem which applies to Kiva systems. The important difference here is that agents are anonymous. Their algorithm, CBM takes advantage of this formulation and is an optimal algorithm that can scale up to 450 agents with 80% success rate. This optimal algorithm wins over even the bounded-suboptimal variants of CBS.

This is an important development and it would be interesting to see if a bounded-suboptimal variant of CBM can be made and if the variant will provide benefits on the same scale as iECBS gives to CBS. This gap seems like a good fit for our work and we look at these two techniques and applying them together, possibly scaling up to the 1000+ agents required by Kiva systems.

In literature which look at Kiva systems, iECBS (Cohen et al. (2016)) and CBM (Ma and Koenig (2016)), the downtime of human pickers is overlooked. In the case of CBM the algorithm takes a snapshot of the system and performs one cycle of the Kiva process. For ECBS, the algorithm simply looked at a Kiva-like environment and moved agents from one side to the other. For both papers, the objective was on makespan and success rate. Unlike the approach where CBM looks at snapshotting a instance of the Kiva system problem, we may look at generating a fixed number of orders and seeing how long the simulation takes to fulfill these orders, the sum of costs and the downtime. This approach would allow for future work in modelling the other problems within Kiva systems as described in Section 1.

[2]ILP using Network Flow (Yu and LaValle (2013a))
[3]Flow Annotation Replanning (Wang et al. (2008))

In conclusion, we will be aiming formulate the Kiva system problem as a TAPF problem and begin by taking an optimal approach. A bounded-suboptimal variant will be developed to further improve scalability. We expect to be able to scale to equivalent scales as in CBM (450 agents) and possibly up to 1000+ agents with the combination of a bounded-suboptimal variant. Finally we hope to see the consequences when the downtime of the system is the objective.

References

Al Dekin, A. V. (2014). Kiva systems warehouse automation at quiet logistics. **URL:** *https://youtu.be/3UxZDJ1HiPE?t=2m2s*

Barer, M., Sharon, G., Stern, R. and Felner, A. (2014). Suboptimal variants of the conflict-based search algorithm for the multi-agent pathfinding problem, *Seventh Annual Symposium on Combinatorial Search*.

Cohen, L., Uras, T., Kumar, T. S., Xu, H., Ayanian, N. and Koenig, S. (2016). Improved solvers for bounded-suboptimal multiagent path finding, *International Joint Conference on Artificial Intelligence*, pp. 3067–3074.

Correll, N., Bekris, K. E., Berenson, D., Brock, O., Causo, A., Hauser, K., Okada, K., Rodriguez, A., Romano, J. M. and Wurman, P. R. (2016). Lessons from the amazon picking challenge, *arXiv preprint arXiv:1601.05484* .

De Koster, R., Le-Duc, T. and Roodbergen, K. J. (2007). Design and control of warehouse order picking: A literature review, *European Journal of Operational Research* **182**(2): 481–501.

Goldenberg, M., Felner, A., Stern, R., Sharon, G., Sturtevant, N. R., Holte, R. C. and Schaeffer, J. (2014). Enhanced partial expansion a., *J. Artif. Intell. Res.(JAIR)* **50**: 141–187.

Harabor, D. D., Grastien, A. et al. (2011). Online graph pruning for pathfinding on grid maps., *AAAI*.

Ma, H. and Koenig, S. (2016). Optimal target assignment and path finding for teams of agents, *Proceedings of the 2016 International Conference on Autonomous Agents & Multiagent Systems*, International Foundation for Autonomous Agents and Multiagent Systems, pp. 1144–1152.

Sharon, G., Stern, R., Felner, A. and Sturtevant, N. R. (2015). Conflict-based search for optimal multi-agent pathfinding, *Artificial Intelligence* **219**: 40–66.

Sharon, G., Stern, R., Goldenberg, M. and Felner, A. (2011). The increasing cost tree search for optimal multi-agent pathfinding, *Twenty-Second International Joint Conference on Artificial Intelligence*.

Silver, D. (2005). Cooperative pathfinding., *AIIDE* **1**: 117–122.

Standley, T. S. (2010). Finding optimal solutions to cooperative pathfinding problems., *AAAI*, Vol. 1, pp. 28–29.

Strasser, B., Botea, A. and Harabor, D. (2015). Compressing optimal paths with run length encoding, *Journal of Artificial Intelligence Research* **54**: 593–629.

Sturtevant, N. R. (2012). Benchmarks for grid-based pathfinding, *IEEE Transactions on Computational Intelligence and AI in Games* **4**(2): 144–148.

Sturtevant, N. R., Traish, J., Tulip, J., Uras, T., Koenig, S., Strasser, B., Botea, A., Harabor, D. and Rabin, S. (2015). The grid-based path planning competition: 2014 entries and results, *Eighth Annual Symposium on Combinatorial Search*.

Wang, K.-H. C. and Botea, A. (2011). Mapp: a scalable multi-agent path planning algorithm with tractability and completeness guarantees, *Journal of Artificial Intelligence Research* **42**: 55–90.

Wang, K.-H. C., Botea, A. et al. (2008). Fast and memory-efficient multi-agent pathfinding., *ICAPS*, pp. 380–387.

Wurman, P. R., D'Andrea, R. and Mountz, M. (2008). Coordinating hundreds of cooperative, autonomous vehicles in warehouses, *AI magazine* **29**(1): 9.

Yu, J. and LaValle, S. M. (2013a). Multi-agent path planning and network flow, *Algorithmic Foundations of Robotics X*, Springer, pp. 157–173.

Yu, J. and LaValle, S. M. (2013b). Structure and intractability of optimal multi-robot path planning on graphs., *AAAI*.