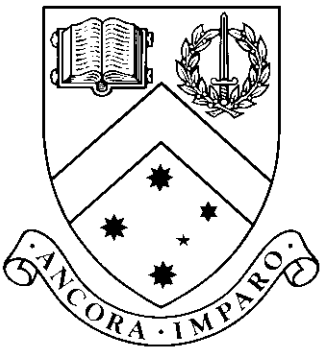
School of Computer Science (BICA)
Monash University



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 **1**

2 Background **2**
2.1 Single-agent pathfinding 2
2.2 Multi-agent pathfinding 2
2.3 Bounded-suboptimal variants 3

3 Problem definition **3**

4 Suboptimal multi-agent pathfinding algorithms **4**
4.1 Cooperative A* 4
4.2 MAPP 4

5 Optimal multi-agent pathfinding algorithms **5**
5.1 Centralized A* 5
5.2 Conflict based search 6
5.3 Increasing Cost Search Tree 6
5.4 Combined-target assignment and pathfinding 7

6 Independence detection and operator decomposition **8**

7 Discussion **8**

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 will be exploring 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 more hard problems to solve than MAPF. Scheduling occurs when storage pods run low on products or drive unit agents run out of battery. Both of these have to be replenished, hence a schedule needs to be determined. Another example is a robotics problem described in the Amazon picking challenge (Correll et al. (2016)). The motivation of this challenge is replace the human worker with a robotic arm to perform the picking and packing and hence improve the reliability and speed of picking.

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 to see what has the most effect but is out of the scope of this project. We will solely be focusing on the MAPF aspect of this system.

We see that a MAPF problem arises when a drive unit agent retrieves, delivers or returns a storage pod. Here we need a solution to coordinate drive unit agents so they can move around the warehouse without colliding with other drive units. 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 and hence scaling up to that scale optimally on a NP-hard problem. Later in Section 5, we look at the performance of these optimal algorithms. 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.

The main contribution of this project is identifying which algorithms have been applied to a Kiva system setting, what conclusions they have drawn and what literature gaps exists. Additionally, we will compare the scalability of optimal and suboptimal MAPF algorithms and see how optimal algorithms have tried to bridge this gap.

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*, two techniques which can be applied to most MAPF with significant benefits. Finally Section 7 compares suboptimal and optimal algorithms with a focus on putting them into the context of a Kiva system environment.

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 used in many MAPF benchmarks. These 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)). Botea et al. (2013) outlines both these algorithms as other single-agent search techniques in the context of computer games.

2.2 Multi-agent pathfinding

Here we briefly describe the general problem of MAPF, in Section 3 we will formally define the problem in the context of Kiva systems as well as outlining terminology. The task of the MAPF problem is to coordinate multiple agents moving around an environment. Each agent is a goal location and the algorithm is required to find a path for each agent while ensuring that agent collides with another. A common secondary objective is to minimize either the makespan of the system or the sum of individual costs (SIC). 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 [5, 4]. Below are some important properties of MAPF algorithms. Table 1 shows a comparison of these properties across the MAPF algorithms discussed in this literature review.

Completeness: the algorithm will return valid solution to the MAPF problem if one exists
Solution quality: solution quality is usually measured as the objective of makespan or sum of individual costs (SIC). Makespan is the number of timestep required from the start state to the goal state where all agents are at their goals. SIC 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 sub-optimal 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

Anonymous: In an anonymous MAPF problem the goal location is interchangeable and any agent can be assigned to a goal in order to satisfy the MAPF problem.
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: A centralized algorithm looks at all agents together in planning paths, whereas in a decentralized algorithm, paths are found for one agent at time. A centralized approach is usually used in finding an optimal solution but 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 in 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 for Kiva systems. The optimal algorithms in 5 will mention any bounded suboptimal variants if they exist.
Integer Linear Programming. Integer Linear Programming (ILP) is an optimization technique. It looks at an objective function subject to a list of constraints and finds the optimal solution for each variable. **not sure what to write here?**

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 in order 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 $goal(k) \in V$

An aspect of Kiva systems which makes the MAPF problem hard is that there are a large number of robots within them, up to a density of 1 in 6. The space within the warehouse is also very constrained as the storage pods are setup in narrow aisles where only 1 drive unit may move down at a time. However, collision occurs between a drive unit and a storage pod only if a drive unit is carrying a storage pod. In other words, drive units are capable of maneuvering underneath storage pods. Additionally collision always occurs between drive units, picking stations and other drive units.

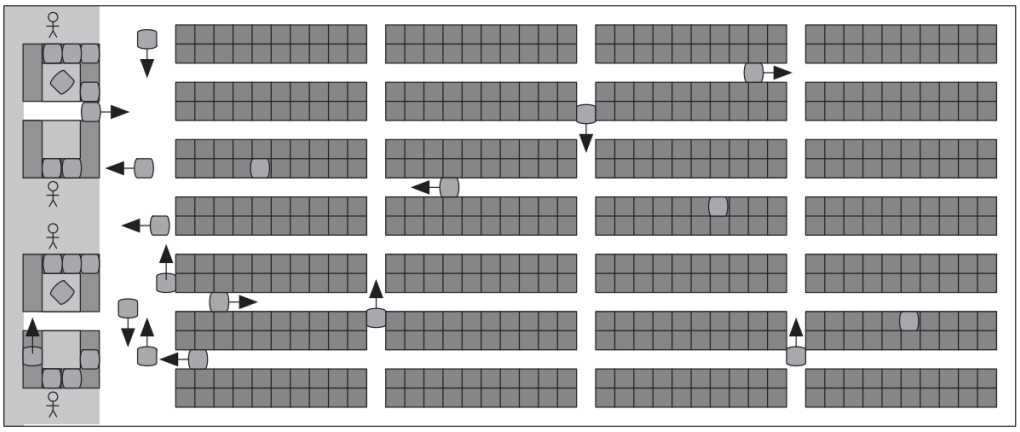


Figure 2: Kiva system represented on an orthagonal grid map (Wurman et al. (2008))

Different to traditional the MAPF problem, the objective of a Kiva system is to minimize the down-time of human pickers. Kiva systems run 24 hours a day hence the Makespan and SIC are not quite representative as they require a goal-state. For MAPF algorithms which were applied to Kiva systems, we look at whether new objectives were used to measure performance.

4 Suboptimal multi-agent pathfinding algorithms

Suboptimal MAPF algorithms usually use a decentralized approach. In a decoupled approach, paths are found for one agent at time as oppose to a coupled approach which looks at all agents globally. These algorithms usually have the properties of being suboptimal and incomplete.

4.1 Cooperative A*

Cooperative A* Exponential in number of agents

Holte et al. (1995)

Silver (2005) gives an overview of these three algorithms.

4.2 MAPP

MAPP (Wang and Botea (2011)) is a suboptimal algorithm which main benefit is its polynomial 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 and is complete for slidable problems if there exists an Ω path for each agent (Figure ??).

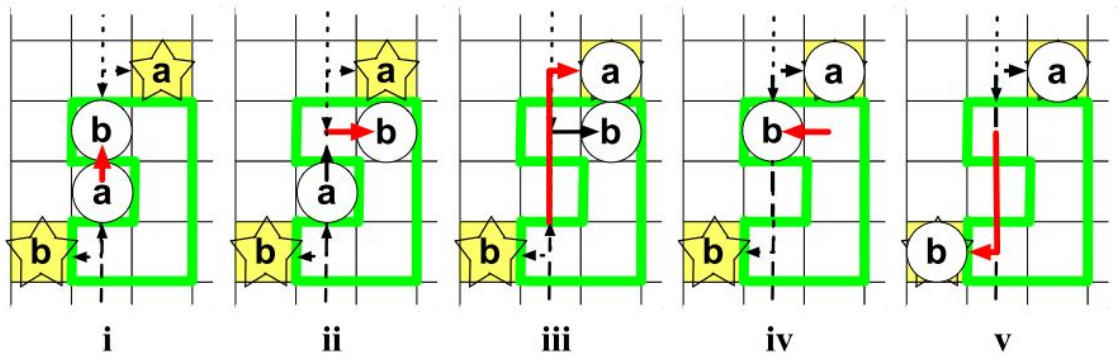


Figure 3: Example of how MAPP works (Wang and Botea (2011))

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 is 2.18 times slower than FAR and 4.8–5.2 times faster than WHCA*.

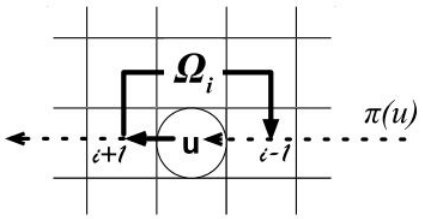


Figure 4: An example of an alternate path, Ω (Wang and Botea (2011))

5 Optimal multi-agent pathfinding algorithms

Optimal MAPF algorithms usually use a coupled approach also known as centralized approach. A coupled approach finds paths for all agents at once.

Here we look at multi-agent pathfinding algorithms which aim to find the optimal solution based on some objective function. The objective function will be noted per algorithm. Our main aim is to determine the benefits and downsides of each algorithm.

Coupled approach: All agents at one! A* ICTS

Krontiris et al. (2013) looked at the feasibility.

5.1 Centralized A*

Centralized A* is exponential in the number of agents $O(n^k)$

There are a number of algorithms based on Centralized A*,

One variant of is an algorithm called enhanced partial A* expansion (EPEA*). The original algorithm Yoshizumi et al. (2000). An improvement on PEA* is EPEA* (Felner et al. (2012), Goldenberg et al. (2014)).

Another variant is M*.

Independence Detection and Operator Decomposition provides an exponential speed up to Centralized A* as it reduces the number of agents being processed, more details about ID in Section **??** and OD in Section **??**.

Ferner et al. (2013) improves on M* by applying operator decomposition.

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. A constraint prohibits an agent a from being at node n at timestep t, in the form of $\langle a, n, t \rangle$.

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 successor nodes 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 node will add the constraint $\langle a_2, n, t \rangle$. CBS is optimal as it chooses the node to expand searching the constraint tree for the node with the lowest cost.

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 may be *strongly coupled*, that is they frequent have a high rate of path conflicts between one another. On the other hand CBS was found to perform better in maps with bottlenecks, where A* would

CBS is exponential in the number of conflicts In comparison to ICTS and A* CBS was found to be best in maps with bottlenecks (Fig. **??**).

Here we look at a bounded-suboptimal variants of CBS were developed called iECBS Cohen et al. (2016), this algorithm improves on the past work which was known as ECBS (Barer et al. (2014)). iECBS expands on ECBS+HWY with the use of highways. iECBS was tested on a Kiva system domain of size 22×54 with 130 agents. This test did not model the Kiva problem described in Section 3, instead the problem generated half the agents in Area1 and half the agents in Area2 of the map (5). Agents were then given a random goal location in the other area. Testing was performed against ECBS and found speedups in runtime of approximately 10 times.

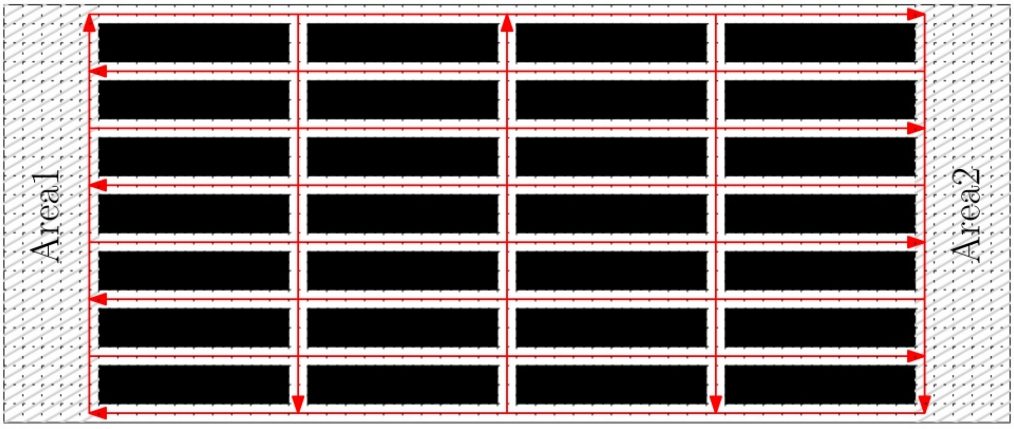


Figure 5: Kiva-like domain with Area1 on the LHS and Area2 on the RHS (Cohen et al. (2016))

Here CBS was compared against ICTS, EPEA*.

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 finds a valid solution when every agent a_i is conflict-free and it generates these paths based off the cost described in the node by exploring all paths which are

equal to C_i. To build the tree, the root starts containing the an optimal cost disregarding any conflicts for each agent. When it does not find a valid solution, the algorithm will branch and generate k successors, one for each agents. 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.

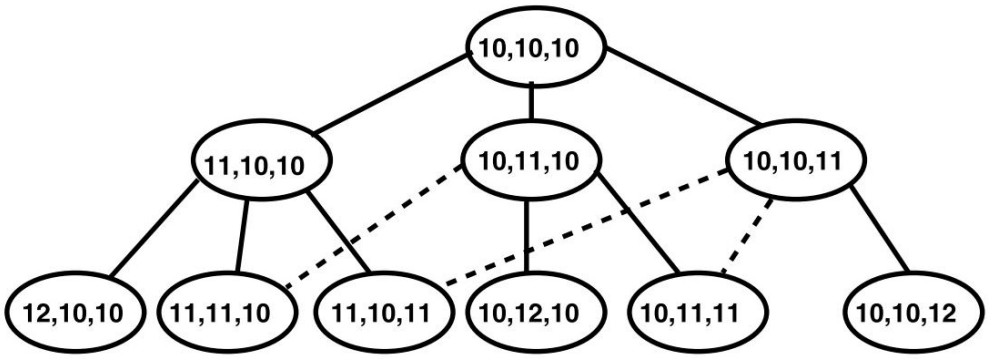


Figure 6: ICT for three agents (Sharon et al. (2011))

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.

ICT was tested on the

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: any agent in a team can be assigned to a goal location

An important property is that anonymous MAPF problems are usually solved in polynomial time. This paper describes two algorithms which are made to solve the TAPF problem. Both of these at the low-level uses a a min-cost max-flow algorithm. First is conflict-based min-cost flow (CBM), a hierarchical algorithm and as the name describes uses CBS (5.2) at the high-level and on the low-level. Next is ILP (TAPF), which is similar to CBM but instead of using CBS at the high level 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.

CBM is applied to Kiva Systems which is the 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.

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 (NF)	Y	Y	Exp conflicts	Y	CBS
ILP (TAPF)	N	Y	Exp conflicts	Y	ILP (NF), IPL (TAPF)
CBM	N	Y	Exp conflicts	Y	ILP (NF), IPL (TAPF)
Suboptimal					
FAR	N	N	Poly	N	WHCA*
MAPP	N	Y	Poly	N	WHCA*, FAR
WHCA*	N	N	Poly	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.*

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 that these two groups 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 agents 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. Regardless they can both be used.

7 Discussion

In regards to suboptimal algorithms, none were applied to Kiva systems. It would be useful to see the relative makespans of the system for a suboptimal solution for comparison against optimal solutions.

We found that most optimal algorithms 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)). These approaches definitely increase the scalability of the optimal algorithm by a factor of up to 5 times in the case of CBS. This allows for a solution to be found with 85% success rate on a graph 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 a 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 which is required by Kiva systems.

We find that in algorithms that 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 perform 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 n number of orders and seeing how long the simulation takes to fulfill these orders, the SIC and the downtime. This approach would allow for the other problems within Kiva systems like scheduling to interact with the MAPF algorithm as described in 1.

Lastly, related to both scalability and the objective of downtime: an optimal algorithm would allow for more shorter paths to be found compared to a suboptimal algorithm hence less agents would required to fulfill the same supply of inventory to human pickers. As less agents are required in the simulation, we can greatly reduce the complexity of the MAPF problem. Therefore there may be a balance of adjusting the solution quality to provide more efficient agents thus keeping the same supply of inventory, while reducing the complexity. We hope to look at this relation in detail and perhaps find a point where it is better to simply add more agents to increase supply or increase the efficiency of each agent.

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*.

Botea, A., Bouzy, B., Buro, M., Bauckhage, C. and Nau, D. (2013). Pathfinding in games, *Dagstuhl Follow-Ups*, Vol. 6, Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik.

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.

Felner, A., Goldenberg, M., Sharon, G., Stern, R., Beja, T., Sturtevant, N. R., Schaeffer, J. and Holte, R. (2012). Partial-expansion a* with selective node generation., *AAAI*.

Ferner, C., Wagner, G. and Choset, H. (2013). Odrm* optimal multirobot path planning in low dimensional search spaces, *Robotics and Automation (ICRA), 2013 IEEE International Conference on*, IEEE, pp. 3854–3859.

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*.

Holte, R. C., Perez, M., Zimmer, R. and MacDonald, A. (1995). Hierarchical a*, *Symposium on Abstraction, Reformulation, and Approximation*.

Krontiris, A., Luna, R. and Bekris, K. E. (2013). From feasibility tests to path planners for multi-agent pathfinding, *Sixth Annual Symposium on Combinatorial Search*.

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.

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

Yoshizumi, T., Miura, T. and Ishida, T. (2000). A* with partial expansion for large branching factor problems., *AAAI/IAAI*, pp. 923–929.

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*.