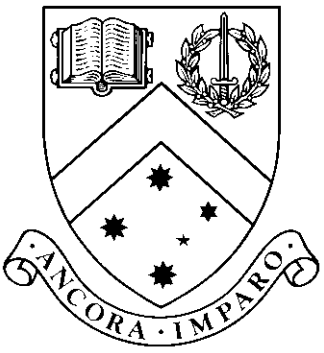
School of Computer Science (BICA)
Monash University



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). This project looks at the use of multi-agent pathfinding (MAPF) within warehouse automation. In particular we will we will be exploring Kiva systems, whereby the order-picking process is performed by automated vehicles. More detail about Kiva systems is provided in Section 2. Our work looks at finding an optimal solution multi-agent pathfinding. In order to find an optimal solution we will be using formulating the MAPF problem as a mixed integer program. Section 5 will look at past approaches taken to MAPF. The results of this literature review will help identify how we should position storage and picking stations in a warehouse. Additionally, we will be looking at developing a MAPF method which uses a pre-computed path oracle.

2 Background

In this project, we look at Kiva Systems (now known as Amazon Robotics). In Kiva systems, products are stored in mobile shelves known as storage pods. Robots known as drive units are responsible for retrieving and delivering storage pods to picking stations. A human worker is stationed at each picking station who picks the item off the pod before processing it (Figure 1). Once the pod has been processed, the drive unit will return the pod to an appropriate location in the warehouse.

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

Single-agent pathfinding. Single-agent pathfinding 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 the entries to the competition and their relative performance. The algorithms are run on a number of game maps which are used for benchmarking Sturtevant (2012). While these results are a few years old, we are expecting to new approaches from this year's GPPC (2017).

In this project, we aim to look at two single-agent search algorithms: *jump point search* and *compressed path databases*. The authors of Jump Point Search and Compressed Databases provide an overview of a large number of search algorithms (Botea et al. (2013)). In it they outline these algorithms and techniques in the context of computer games. cover heuristics, symmetry elimination which is the core of Jump Point Search. and specific gaps in the literature between video games.

Multi-agent pathfinding. Solving MAPF optimally is an NP-hard problem (Yu and LaValle (2013b)). As a result, Kiva systems are generally looked at with suboptimal algorithms with scalability in mind. Here we are focusing on an optimal approach, while it is not scalable to the desired size (1000+ agents), we will be looking at making a trade-off between speed and solution quality by creating bounded-suboptimal algorithm.
Bounded-suboptimal varient. A *bounded suboptimal variant* is a variant of an optimal MAPF algorithm. The defining feature of a bounded suboptimal variant is that we can guarantee that the cost of the solution, B lies within the optimal cost C and $w * C$ where w is a user-defined parameter. Hence:

$$C \leq B \leq w * C$$

Our project is aiming to make a bounded suboptimal variant of our algorithm due to the large-scale of the simulation (Section 2). By adjusting w we are able to relax the problem and find a solution faster at the loss of solution quality.

3 Problem definition

This section will formally define the MAPF problem in the context of Kiva Systems. The goal of MAPF is to find a path for each agent to their goal while ensuring that no path conflicts.

- A path is a list of action a at timestep t where $a \in \{up, down, left, right, wait\}$
- A path is said to conflict with another when on the same timestep: two agents share the same node or the agents cross the same edge.

In order to reduce the complexity of the MAPF algorithm, the environment of a Kiva systems is modeled as a square grid map (Figure 2). On this map we have k drive unit agents.

- Graph $G(V, E)$ where V is the set of nodes on the grid map and E is the set of edges between nodes.
- A set of agents, K, where for all $k \in K$, $location(k) \in V$ and $goal(k) \in V$

Each action

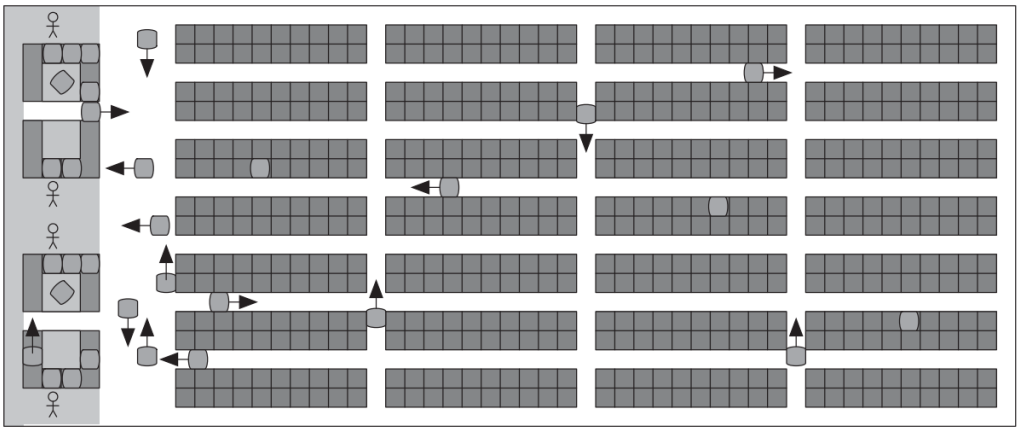


Figure 2: Kiva system represented on an orthagonal grid map (Wurman et al. (2008))

4 Suboptimal multi-agent pathfinding algorithms

Suboptimal MAPF algorithms usually use a decentralized approach[1]. 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 FAR

Flow Annotation Replanning (FAR) (Wang and Botea 2008) imposes unidirectional travel on top of an initially undirected gridmap. The travel direction alternates across the rows (and columns), covering the grid with criss- crossing virtual road lanes. Topology-specific additional rules, applied locally, preserve the connectivity across the map. In terms of map abstraction, this boils down to remov- ing some directed edges from a graph, as opposed to splitting a map into subgraphs. FAR computes a path for each unit independently, in a unit-centric decomposition. Conflicts, such as cycles, are addressed at runtime. A common idea with our work is restricting the traffic flow by ignoring some edges and imposing a travel direction on others. The flowin- side SDP high-contention areas is restricted as shown later in this paper. We take advantage of the specific topology of high-contention areas, ensuring that they are free from local deadlocks

4.3 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 **??**).

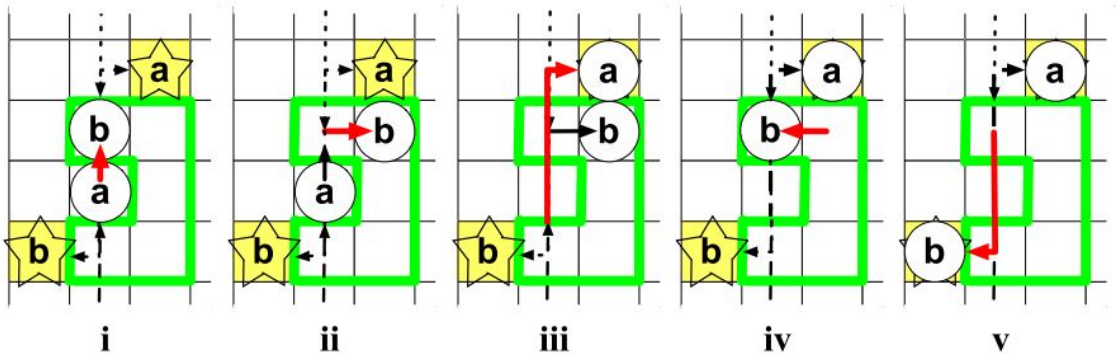


Figure 3: Example of how MAPP works (Wang and Botea (2011))

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

[1] decentralized is also known as decoupled

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*.

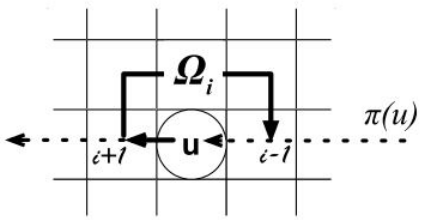


Figure 4: An example of an alternate path, Ω (Wang and Botea (2011))

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 6.2 and OD in Section 6.2.

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. 5).

A bounded-suboptimal variant of CBS, ECBS features Barer et al. (2014) **TODO** Recently iECBS expanded on this with the use of highways and Cohen et al. (2016).

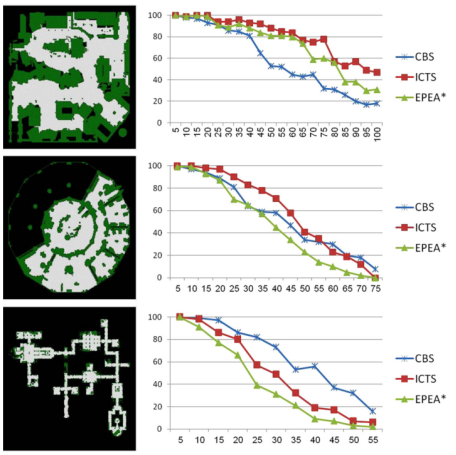


Figure 5: A Small Region of a Kiva Layout (Sharon et al. (2015)). Picking stations located on the left and storage pods laid out in rows.

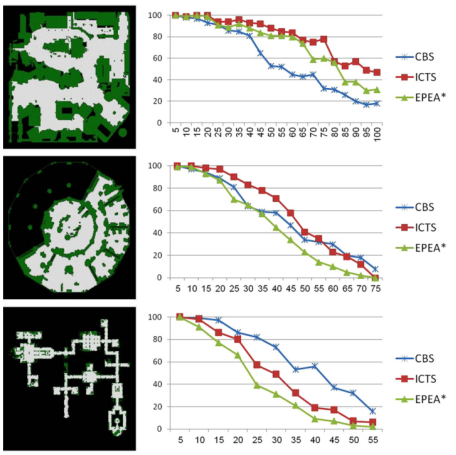


Figure 6: A Small Region of a Kiva Layout (Sharon et al. (2015)). Picking stations located on the left and storage pods laid out in rows.

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 7). 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.

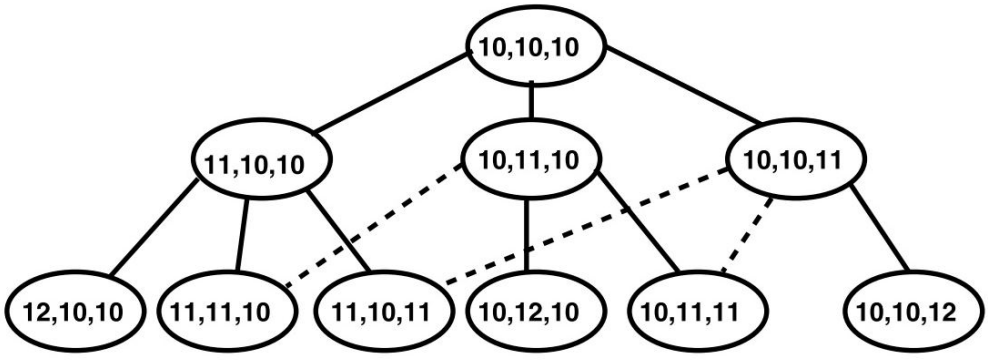


Figure 7: ICT for three agents (Sharon et al. (2011))

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 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.

Ma and Koenig (2016) reduction to the integer multi- commodity flow problem on a time-expanded network.

Yu and LaValle (2016) takes a Network Flow approach. The solver uses the branch-and-bound algorithm. Anytime approach.

Yu and LaValle (2013a) connects multi-agent path planning on graphs (roadmaps) to network flow problems, showing that the former can be reduced to the latter, therefore enabling the application of combinatorial network flow algorithms, as well as general linear program techniques, to multiagent path planning problems on graphs.

Yu and LaValle (2016) looks at Complete Algorithms and Effective Heuristics.

Yu and LaValle (2015) looks at the Structure and Computational Complexity.

6 Independence detection and Operator decomposition

Here we describe any techniques which may be applied to most MAPF algorithms and cause a speedup.

6.1 Independence detection

Standley (2010) Independence detection (ID) allowed the paths of groups of agents to be computed independently, without sacrificing optimality

Standley (Standley 2010) presented (= ?V independence detection (ID) framework. ID partitions the problem into smaller independent subproblems, if possi- ble, and solves each problem separately. Standley also intro- duced operator decomposition (OD) which considers mov- ing a single agent at a time. OD reduces the branching fac- tor at the cost of increasing the depth of the solution in the search tree.

6.2 Operator decomposition

Standley (2010) Operator decomposition (OD) is used to reduce the branching factor of the MAPF algorithms.

7 Discussion

There are a number of algorithms which have been developed in

We should study these algorithm in the warehouse environment. Besides CBS and ILP (TAPF), other algorithms have not been compared against one another. Our research focuses on the area of ILP

[2]BSO: whether the algorithm has a bounded-suboptimal variant
[3]Complexity: runtime complexity
[4]SQ: solution quality
[5]KS: whether the algorithm was tested in a Kiva System environment

Name	BSO[2]	Complete	Complexity[3]	SQ[4]	KS[5]	Comparison[6]
Optimal						
CBS	N	Y	Exp conflicts	Opt	Y	ICTS, EPEA*
Centralized A*	Y	Y	Exp agents	Opt	N	???
ICTS	Y	Y	Exp Δ	Opt	N	A*, A*+OD
ILP (NF)	Y	Y	Exp conflicts	Opt	Y	CBS
ILP (TAPF)	Y	Y	Exp conflicts	Opt	Y	CBS
Suboptimal						
FAR	N	N	Poly	Sub Opt	N	
MAPP	Y	Y	Poly	Sub Opt	N	WHCA*, FAR
WHCA*	N	N	Poly	Sub Opt	N	

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.

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.

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.

[6]Comparison: other MAPF algorithms used for benchmark comparison

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.

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*.

Yu, J. and LaValle, S. M. (2015). Optimal multi-robot path planning on graphs: Structure and computational complexity, *arXiv preprint arXiv:1507.03289* .

Yu, J. and LaValle, S. M. (2016). Optimal multirobot path planning on graphs: Complete algorithms and effective heuristics, *IEEE Transactions on Robotics* **32**(5): 1163–1177.