Improved Solvers for Bounded-Suboptimal Multi-Agent Path Finding*

Liron Cohen, Tansel Uras, T. K. Satish Kumar, Hong Xu, Nora Ayanian, Sven Koenig
Department of Computer Science , University of Southern California
{lironcoh, turas}@usc.edu tkskwork@gmail.com {hongx, ayanian, skoenig}@usc.edu

Abstract

Multi-Agent Path Finding (MAPF) with the objective to minimize the sum of the travel times of the agents along their paths is a hard combinatorial problem. Recent work has shown that bounded-suboptimal MAPF solvers, such as Enhanced Conflict-Based Search or ECBS(w_1) for short, run often faster than optimal MAPF solvers at the cost of incurring a suboptimality factor w_1, that is due to using focal search. Other recent work has used experience graphs to guide the search of ECBS(w_1) and speed it up, at the cost of incurring a separate suboptimality factor w_2, that is due to inflating the heuristic values. Thus, the combination has suboptimality factor w_1w_2. In this first feasibility study, we develop a bounded-suboptimal MAPF solver, Improved-ECBS or iECBS(w_1) for short, that has suboptimality factor w_1 rather than w_1w_2 (because it uses experience graphs to guide its search without inflating the heuristic values) and can run faster than ECBS(w_1). We also develop two first approaches for automatically generating experience graphs for a given MAPF instance. Finally, we observe heavy-tailed behavior in the runtimes of these MAPF solvers and develop a simple rapid randomized restart strategy that can increase the success rate of iECBS(w_1) within a given runtime limit.

1 Introduction

Given a directed or undirected graph and a set of agents with unique start and goal vertices, the Multi-Agent Path Finding (MAPF) problem is to find collision-free paths for all agents from their respective start vertices to their respective goal vertices. The agents traverse edges in unit time but can also wait in vertices. The MAPF problem arises in application domains such as automated Kiva-like (now: Amazon Robotics) warehousing [Wurman *et al.*, 2008] and airport surface operations [Morris *et al.*, 2016]. Minimizing the solution cost given by the sum of the travel times of the agents along their paths is NP-hard [Yu and LaValle, 2013]. Optimal MAPF solvers, such as Conflict-Based Search or CBS for short [Sharon *et al.*, 2015], are thus often slow. Recent work has shown that w-suboptimal (also called bounded-suboptimal) MAPF solvers run faster than optimal MAPF solvers at the cost of incurring a suboptimality factor $w \geq 1$. In other words, they are guaranteed to find solutions whose solution costs are at most w times the minimum solution costs. Enhanced Conflict-Based Search or ECBS (w_1) for short [Barer *et al.*, 2014] is a w_1-suboptimal variants of CBS (for parameter $w_1 \geq 1$). Its suboptimality factor is due to it using focal search [Pearl and Kim, 1982] with parameter w_1. Larger values of w_1 result in greedier searches. CBS+HWY(w_2) [Cohen *et al.*, 2015] is a w_2-suboptimal variant of CBS (for parameter $w_2 \geq 1$). Its suboptimality factor is due to it inflating some of the admissible heuristic values by factor w_2, making use of experience graphs [Phillips *et al.*, 2012] in form of human-generated directed highways. Larger values of w_2 allow the paths of agents to conform more to the highways. ECBS(w_1)+HWY(w_2) [Cohen *et al.*, 2015] combines both of the above approaches by using focal searches with highways and has suboptimality factor w_1w_2. In this first feasibility study, we develop a w_1-suboptimal variant of ECBS(w_1)+HWY(w_2), Improved-ECBS or iECBS(w_1) for short, that also uses focal searches with highways but has suboptimality factor w_1 rather than w_1w_2. Thus, iECBS(w_1) has only one parameter, which makes parameter-tuning easy. This parameter is, like the parameter of ECBS(w_1), the suboptimality factor, which makes it easy to compare both MAPF solvers fairly. Our experimental results show that iECBS(w_1) can run faster than ECBS(w_1) despite both MAPF solvers having the same suboptimality factor. So far, the highways used with MAPF solvers have been human-generated. We also develop two first approaches for automatically generating experience graphs for a given MAPF instance that can make ECBS(w_1)+HWY(w_2) and iECBS(w_1) about as fast as human-generated experience graphs in Kiva-like warehousing domains, thus potentially reducing the dependency on human expertise for highway generation. Finally, we observe heavy-tailed behavior in the runtimes of these MAPF solvers

*Our research was supported by by NASA via Stinger Ghaffarian Technologies as well as NSF under grant numbers 1409987 and 1319966. The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the sponsoring organizations, agencies or the U.S. government.

and develop a simple rapid randomized restart strategy that can increase the success rate of iECBS(w_1) within a given runtime limit.

2 Background

We now define the MAPF problem formally: We are given a directed or undirected graph $G = (V, E)$ and a set of K agents $1, \ldots, K$. Each agent j has a unique start vertex $s^j \in V$ and a unique goal vertex $g^j \in V$. At each time step, each agent can either move to a neighboring vertex or wait at its current vertex, both with cost one. A *solution* of a MAPF instance is a set of feasible paths, one path $\{s_0^j, \ldots, s_{T_j}^j, s_{T_j+1}^j, \ldots\}$ for each agent $j \in \{1, \ldots, K\}$, such that no two paths collide. A path for agent j is *feasible* if and only if 1) it starts at the start vertex of agent j, that is, $s_0^j = s^j$; 2) it ends at the goal vertex of agent j and remains there, that is, there exists a smallest T_j such that $s_{T_j}^j = g^j$ and, for each $t > T_j$, $s_t^j = g^j$; and 3) every action is a legal move or wait action, that is, for all $t \in \{0, 1, \ldots, T_j - 1\}$, $\langle s_t^j, s_{t+1}^j \rangle \in E$ or $s_t^j = s_{t+1}^j$. A *collision* between the paths of agents j and k is either a *vertex collision* (j, k, s, t), that is, $s = s_t^j = s_t^k$, or an *edge collision* (j, k, s_1, s_2, t), that is, $s_1 = s_t^j = s_{t+1}^k$ and $s_2 = s_{t+1}^j = s_t^k$. The travel time of agent j is the number of time steps T_j until it reaches its goal vertex. Our objective is to minimize the solution cost, given as the sum of the travel times of all agents, which is a common objective in the literature [Yu and LaValle, 2013; Sharon *et al.*, 2015]. A variety of MAPF solvers have been developed by different research groups, see [Wagner, 2015] and [Hoenig *et al.*, 2016] for overviews. In the following, we describe the ones that are important for this paper.

CBS [Sharon *et al.*, 2015] is an optimal MAPF solver. It performs high-level and low-level searches. Each high-level node contains a set of constraints and, for each agent, a feasible path that respects the constraints. The high-level root node has no constraints. The high-level search of CBS is a best-first search that uses the costs of the high-level nodes as their f-values. The cost of a high-level node is the sum of the travel times along its paths. When CBS expands a high-level node N, it checks whether the node is a goal node. A high-level node is a goal node if and only if none of its paths collide. If N is a goal node, then CBS terminates successfully and outputs the paths N as solution. (Thus, the fewer collisions there are in high-level nodes, the faster CBS terminates.) Otherwise, at least two paths collide. CBS chooses a collision to resolve and generates two high-level children of N, called N_1 and N_2. Both N_1 and N_2 inherit the constraints of N. If the chosen collision is a vertex collision (j, k, s, t), then CBS adds the vertex constraint (j, s, t) to N_1 (that prohibits agent j from occupying vertex s at time step t) and the vertex constraint (k, s, t) to N_2. If the chosen collision is an edge collision (j, k, s_1, s_2, t), then CBS adds the edge constraint (j, s_1, s_2, t) to N_1 (that prohibits agent j from moving from vertex s_1 to vertex s_2 between time steps t and $t + 1$) and the edge constraint (k, s_2, s_1, t) to N_2. During the generation of high-level node N, CBS performs a low-level search for the agent i affected by the added constraint. The low-level search for agent i is a (best-first) A* search that ignores all other agents and finds a minimum-cost path from the start vertex of agent i to its goal vertex that is both feasible and respects the constraints of N that involve agent i.

ECBS(w_1) [Barer *et al.*, 2014] is a w_1-suboptimal variant of CBS whose high-level and low-level searches are focal searches rather than best-first searches. A focal search, like A*, uses an OPEN list whose nodes n are sorted in increasing order of their f-values $f(n) = g(n) + h(n)$, where $h(n)$ are the primary heuristic values. Unlike A*, a focal search with suboptimality factor w_1 also uses a FOCAL list of all nodes currently in the OPEN list whose f-values are no larger than w_1 times the currently smallest f-value in the OPEN list. The nodes in the FOCAL list are sorted in increasing order of their secondary heuristic values. A* expands a node in the OPEN list with the smallest f-value, but a focal search expands instead a node in the FOCAL list with the smallest secondary heuristic value. Thus, the secondary heuristic values should favor a node in the FOCAL list that is close to a goal node to speed up the search and thus exploit the leeway afforded by w_1 that A* does not have available. If the primary heuristic values are admissible, then a focal search is w_1-suboptimal. The secondary heuristic values can be inadmissible. The high-level and low-level searches of ECBS(w_1) are focal searches. During the generation of a high-level node N, ECBS(w_1) performs a low-level focal search with OPEN list OPEN$^i(N)$ and FOCAL list FOCAL$^i(N)$ for the agent i affected by the added constraint. The high-level and low-level focal searches of ECBS(w_1) use measures related to the number of collisions as secondary heuristic values. Thus, the high-level nodes of ECBS(w_1) with reasonably small values of $w_1 > 1$ can have fewer collisions than the high-level nodes of CBS, and ECBS(w_1) then runs faster. On the other hand, the path costs can become large for ECBS(w_1) with large values of w_1 due to the larger leeway afforded by w_1. The agents might then move around in wiggly lines, which increases the chance of collisions, thus increasing the number of collisions in the high-level nodes of ECBS(w_1) and slowing it down. Thus, larger values of w_1 do not necessarily entail smaller runtimes of ECBS(w_1).

ECBS(w_1)+HWY(w_2) [Cohen *et al.*, 2015] is a w_1w_2-suboptimal variant of ECBS(w_1) that inflates some of the primary admissible heuristic values of the low-level search by a factor of w_2, making use of experience graphs [Phillips *et al.*, 2012] in form of a human-generated set of directed edges, called highway. In particular, it inflates the costs of wait actions and move actions along edges that do not belong to the highway by a factor of w_2 during the computation of the low-level heuristic values, resulting in the highway heuristic values from [Cohen *et al.*, 2015]. The highway heuristic values bias the low-level search to find paths that use the highway edges, which tends to reduce the number of head-on collisions (where the colliding agents move in opposite directions) in dense environments, such as Kiva-like warehousing domains [Cohen *et al.*, 2015]. Thus, the high-level nodes of ECBS(w_1)+HWY(w_2) with $w_2 > 1$ can have fewer collisions than the high-level nodes of ECBS(w_1), and ECBS(w_1)+HWY(w_2) then runs faster.

	Low-level (for Agent i in High-level node N)		High-level		SUB-OPT
	Sort $n \in$ **OPEN**$^i(N)$	Sort $n \in$ **FOCAL**$^i(N)$	Sort $N \in$ **OPEN**	Sort $N \in$ **FOCAL**	
ECBS(w_1)	1. $g^i(n) + h^i_{sp}(n)$ 2. $g^i(n)$	1. $h^i_{c(N)}(n)$ 2. $g^i(n) + h^i_{sp}(n)$ 3. $g^i(n)$	1. $\sum_i \min_{n \in \text{OPEN}^i} (g^i(n) + h^i_{sp}(n))$	1. $h_c(N)$ 2. $g(N)$	w_1
ECBS(w_1)+**HWY**(w_2)	1. $g^i(n) + h^i_{hwy}(n)$ 2. $g^i(n)$	1. $h^i_{c(N)}(n)$ 2. $g^i(n) + h^i_{hwy}(n)$ 3. $g^i(n)$	1. $\sum_i \min_{n \in \text{OPEN}^i} (g^i(n) + h^i_{hwy}(n))$	1. $h_c(N)$ 2. $g(N)$	$w_1 \cdot w_2$
iECBS(w_1)	1. $g^i(n) + h^i_{sp}(n)$ 2. $g^i(n)$	1. $h^i_{c(N)}(n)$ 2. $g^i(n) + h^i_{hwy}(n)$ 3. $g^i(n)$	1. $\sum_i \min_{n \in \text{OPEN}^i} (g^i(n) + h^i_{sp}(n))$	1. $h_c(N)$ 2. $g(N)$	w_1

Figure 1: Summarizes the differences in the high-level and low-level searches for various MAPF solvers. The main criterion is labeled 1, the first tie-breaking criterion is labeled 2, and the second tie-breaking criterion is labeled 3. Smaller values are preferred for all comparisons except for all $g^i(n)$ values in the low-level search.

3 iECBS

We now develop a w_1-suboptimal MAPF solver, iECBS(w_1), that, like ECBS(w_1)+HWY(w_2), uses a focal search with highways but has suboptimality factor w_1 rather than $w_1 w_2$ because it uses the highways to compute the secondary heuristic values for the low-level search rather than the primary ones and thus does not inflate the primary ones. The w_1-suboptimality proofs of ECBS(w_1) apply to iECBS(w_1) since both MAPF solvers construct their FOCAL lists for the high-level and low-level searches identically.

The low-level search of ECBS(w_1) uses as secondary heuristic values triples of the number of vertex and edge collisions with the paths of other agents, the f-value (as primary tie-breaking criterion) and the g-value (as secondary tie-breaking criterion). The low-level search of iECBS(w_1), on the other hand, uses as secondary heuristic values triples of the number of vertex and edge collisions with the paths of other agents, the highway heuristic value (defined below) and the g-value, which are again compared lexicographically. Smaller values are preferred for all comparisons except for all $g^i(n)$ values in the low-level search, for which larger values are preferred.

Figure 1 summarizes the differences between ECBS(w_1), ECBS(w_1)+HWY(w_2) and iECBS(w_1). Here, $g^i(n)$ is the g-value of low-level node n for agent i. $h^i_{sp}(n)$ is the admissible h-value of low-level node n for agent i, defined to be the cost of a minimum-cost path from n to the goal vertex of agent i that is feasible but ignores all other agents and does not necessarily respect all constraints of the corresponding high-level node. $h^i_{sp}(n)$ is the inadmissible highway heuristic value of low-level node n for agent i, defined to be, after inflating all costs of non-highway edges by a factor of w_2, the cost of a minimum-cost path from n to the goal vertex of agent i that is feasible but ignores all other agents and does not necessarily respect all constraints of the corresponding high-level node. $h^i_{c(N)}(n)$ is the inadmissible collision heuristic value of low-level node n for agent i during the generation of high-level node N, defined to be the number of vertex and edge collisions on the path from the start vertex of agent i to n with the paths of the other agents in N. $g(N)$ is the g-value of high-level node N, defined to be the sum of the travel times along all paths in N. Finally, $h_c(N)$ is the inadmissible collision heuristic value of high-level node N, defined to be the number of vertex and edge collisions between the paths in N.

		1	2	3	4	5	6	7	8	9	10
ECBS(1.5)	Time	287	300	92	300	300	300	**71**	33	58	54
	Cost	9357	n/a	9234	n/a	n/a	n/a	9267	9349	9179	9676
iECBS(1.5)	Time	**28**	**15**	**7**	**7**	**41**	**8**	77	**8**	**7**	**17**
	Cost	**9141**	**9002**	**8930**	**8925**	**9143**	**8967**	**9070**	**9138**	**8785**	**9181**

Figure 2: Compares the runtimes (in seconds) and solution costs of iECBS(1.5) with a human-generated highway and ECBS(1.5) in a Kiva-like warehousing domain. Each column corresponds to a randomly generated MAPF instance. n/a indicates that the 300-second runtime limit is reached.

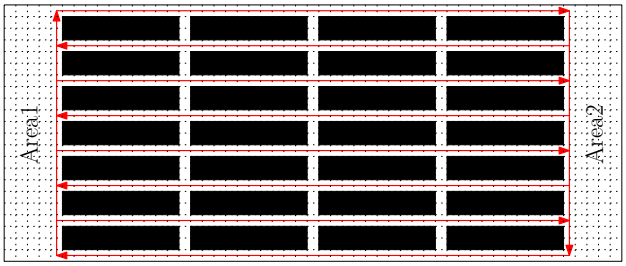


Figure 3: Shows a Kiva-like warehousing domain and a human-generated highway in red [Cohen et al., 2015].

3.1 Experimental Results

We now show that iECBS(w_1) can run faster than ECBS(w_1) in the Kiva-like warehousing domain with narrow corridors of width one shown in Figure 3. We compare iECBS(1.5) with the human-generated highway shown in Figure 3 and ECBS(1.5). We choose suboptimality factor 1.5 since it is sufficiently small to result in a large runtime for ECBS(w_1). We use 10 randomly generated MAPF instances with 130 agents each, half of which are assigned a random start vertex in Area 1 and a random goal vertex in Area 2, and vice-versa for the other half of the agents. We run all experiments in

this paper on a PC with a 3.2GHz Intel Core i7 CPU and a 300-second runtime limit per MAPF instance, unless stated otherwise. Figure 2 reports the runtimes (in seconds) and solution costs of both MAPF solvers on each MAPF instance. The runtime is more important for us than the solution cost since both MAPF solvers have the same suboptimality factor. The runtimes of iECBS(1.5) are smaller than the ones of ECBS(1.5), showing that highways can guide the search of iECBS(1.5) well.

We do not compare ECBS(w_1)+HWY(w_2) and iECBS(w_1). If one wants both of them to have suboptimality factor w, then one needs to choose w_1 with $w_1 \leq w$ for iECBS(w_1) as well as w_1 and w_2 with $w_1 w_2 < w$ for ECBS(w_1)+HWY(w_2). As argued earlier, the runtimes are not necessarily minimized when $w_1 = w$ for iECBS(w_1) and $w_1 w_2 = w$ for ECBS(w_1)+HWY(w_2). Thus, there are many choices of parameters and it is unclear how to compare both MAPF solvers fairly. When trying several combinations of w_1 and w_2, we notice that they are often too small for ECBS(w_1)+HWY(w_2) to run sufficiently fast.

4 Automatically Generating Highways

We now develop two approaches for automatically generating highways. We develop two approaches rather than one because none exist so far and we thus pursue different ideas to increase the chances to find a strong one. The idea behind both approaches is simple: Solving the MAPF problem optimally is NP-hard but computing the minimum-cost paths for all agents independently is fast. The information in these minimum-cost paths can be used heuristically to automatically generate highways.

For a given MAPF instance, first a highway is automatically generated by one of the approaches and then the instance is solved by ECBS(w_1)+HWY(w_2) or iECBS(w_1) with the generated highway. We compare the automatically generated highways to human-generated highways and a Criss-Cross (CC) highway as baselines, The movement direction of a CC highway is from west to east in odd-numbered west-east rows and from south to north in odd-numbered north-south columns (and the opposite directions in even-numbered rows and columns). The CC highway is inspired by MAPF research done in the context of avoiding deadlocks [Wang and Botea, 2008] and used here since it avoids head-on collisions.

4.1 Graphical Model-Based Approach

We now use a Graphical Model (GM) to obtain an approach for automatically generating highways for undirected two-dimensional four-neighbor grids with missing vertices that correspond to blocked cells. A GM is a graph whose nodes represent variables and whose edges represent interactions between variables, resulting in a factored representation of such interactions [Koller and Friedman, 2009]. We associate two variables X_i and Y_i with each unblocked cell i. Figure 4(a) illustrates a GM. Solid circles represent the X_i variables, and hollow circles represent the Y_i variables.

The X_i variables are observable and correspond to the following statistics gathered for cell i when computing the minimum-cost paths for all agents independently and assuming that the agents follow these paths: 1) whether a collision occurs between any two agents in cell i (to be precise: whether a vertex collision occurs in the cell or an edge collision occurs in any adjacent edge), 2) whether the magnitude of the direction vector DV(i) is greater than 0.5 and 3) the direction of DV(i) discretized into the four compass directions. The two-dimensional direction vector DV(i), inspired by [Jansen and Sturtevant, 2008], is computed by generating two unit vectors whenever an agent traverses i, namely one in its entry direction and one in its exit direction. DV(i) is the average of all such unit vectors.

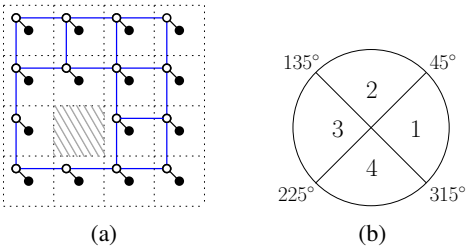


Figure 4: Shows a GM over a 4x4 grid (a) and its mapping from angles to discrete values (b).

The Y_i variables are hidden and indicate the compass direction to the neighboring cell, if any, to which a highway edge leaving i should be generated. If the value of Y_i is 'None' or the indicated neighboring cell is blocked, then no highway edge is generated. The values of the Y_i variables are determined by solving the maximum-a-posteriori estimation problem for the GM with the factors shown in Figure 5. For example, $f(X_i = 111, Y_i = None) = 1$. The rationale behind the values of the factors is as follows: 1) Consider an unblocked cell i. If a collision occurs in i, then there is a preference to have a highway edge leave i in the discretized direction of DV(i). A larger discretized magnitude of DV(i) indicates a stronger preference. These factors $f(X_i, Y_i)$ are represented by the black edges in Figure 4(a). 2) Consider two unblocked cells i and j that are horizontal [vertical] neighbors. There is a preference to have highway edges leave them in the same horizontal [vertical] direction (to construct a longer highway) or leave them in opposite vertical [horizontal] directions (to construct two highways in opposite directions next to each other to allow for two-way traffic). These factors $f_H(Y_i, Y_i)$ [$f_V(Y_i, Y_i)$] are represented by the blue edges in Figure 4(a). We do not completely optimize the values of the factors. While solving the maximum-a-posteriori estimation problem is itself NP-hard even for the special case of two-dimensional grids, there are approaches that work well in practice. For example, our problem corresponds to a two-dimensional hidden Markov model, for which specialized approaches exist. Since we derive only heuristic values from the highways, we do not require an exact solution and thus use Gibbs sampling [Koller and Friedman, 2009] with $5,000$ samples. The GM-based approach runs in about one second in our experiments and thus is fast compared to running the MAPF solvers. Our implementation of the GM-based approach uses libDAI [Mooij, 2010].

X_i	Y_i	$f(X_i, Y_i)$
0??	None	6
0??	East	1
0??	North	1
0??	West	1
0??	South	1
101 ; 111	None	3 ; 1
101 ; 111	East	6 ; 24
101 ; 111	North	1 ; 1
101 ; 111	West	1 ; 1
101 ; 111	South	1 ; 1
102 ; 112	None	3 ; 1
102 ; 112	East	1 ; 1
102 ; 112	North	6 ; 24
102 ; 112	West	1 ; 1
102 ; 112	South	1 ; 1
103 ; 113	None	3 ; 1
103 ; 113	East	1 ; 1
103 ; 113	North	1 ; 1
103 ; 113	West	6 ; 24
103 ; 113	South	1 ; 1
104 ; 114	None	3 ; 1
104 ; 114	East	1 ; 1
104 ; 114	North	1 ; 1
104 ; 114	West	1 ; 1
104 ; 114	South	6 ; 24

Y_i	Y_j	$f_H(Y_i, Y_j)$	$f_V(Y_i, Y_j)$
None	None	6	6
None	East	1	1
None	North	1	1
None	West	1	1
None	South	1	1
East	None	2.8	5
East	East	5	0.2
East	North	1	1
East	West	0.2	2.8
East	South	1	1
North	None	5	2.8
North	East	1	1
North	North	0.2	5
North	West	1	1
North	South	2.8	0.2
West	None	2.8	5
West	East	0.2	2.8
West	North	1	1
West	West	5	0.2
West	South	1	1
South	None	5	2.8
South	East	1	1
South	North	2.8	0.2
South	West	1	1
South	South	0.2	5

Figure 5: Shows the different factors of our GM. The first digit in the value of X_i indicates whether ('1') or not ('0') there is a collision in the cell. The second digit indicates whether the magnitude of DV(i) is greater than 1/2 ('1') or not ('0'). The third digit indicates whether the direction of DV(i) is in the eastern quadrant $[315°, 45°)$ ('1'), northern quadrant $[45°, 135°)$ ('2'), western quadrant $[135°, 225°)$ ('3') or southern quadrant $[225°, 315°)$ ('4'). '?' represents a wildcard.

4.2 Heat Map-Based Approach

We now use a Heat Map (HM) to obtain an approach for automatically generating highways for arbitrary undirected graphs, inspired by an oracular approach for solving multi-commodity flow problems [Leighton *et al.*, 1995]. Different from the GM-based approach, it computes the paths sequentially (by repeatedly picking an agent randomly and computing its minimum-cost path using the current HM costs of the edges), and each path is influenced by the previously computed paths because agents are given an incentive for following previously computed paths until the capacity constraints of edges (here: equal to one) are reached. The edges that are used frequently after $MaxIterations$ path computations (for parameter $MaxIterations > 1$), namely those with small HM costs, become the highway edges.

Algorithm 1 shows the HM-based approach, which first converts each undirected edge to two directed edges with HM costs one and then updates the following values for each directed edge $\langle u, v\rangle$ to update its HM cost: 1) $n(u, v)$ (initialized to zero) is the number of times the directed edge is chosen in any minimum-cost path. 2) $p(u, v)$ is the follow preference of the edge, defined to be $\alpha n(u, v)/MaxIterations$ (for parameter $\alpha \in (0, 1)$). The follow preferences encourage agents to traverse directed edges that appear in previously computed paths. 3) $t(u, v)$ is the interference cost of the edge, defined to be $\beta n(v, u)/MaxIterations$ (for parameter $\beta > 1$). The interference costs discourage agents from traversing opposite directed edges (that correspond to the same undirected edge). 4) $s(u, v)$ is the saturation cost of the edge, defined to be $\gamma^{(n(u,v)+n(v,u))/(2MaxIterations)}$ (for parameter $\gamma > 1$). 5) $c(u, v)$ is the HM cost of the edge, defined to be $1 - p(u, v) + t(u, v) + s(u, v)$. Our implementation uses parameters $\alpha = 0.5$, $\beta = 1.2$, $\gamma = 1.3$ and $MaxIterations = 100,000$. It considers the $1/7^{th}$ of the edges with the lowest HM costs after $MaxIterations$ iterations and then returns $1/5^{th}$ of these edges randomly. We do not completely optimize the values of these parameters. The HM-based approach runs in about six seconds in our experiments and thus is fast compared to running the MAPF solvers.

Algorithm 1: Heat Map-Based Approach.

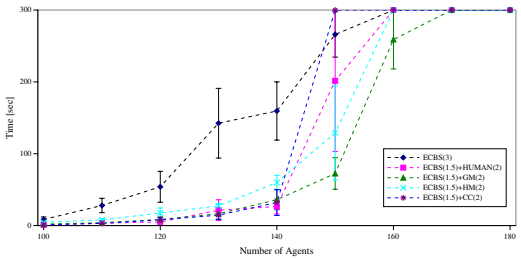
input : MAPF instance
output: Highway edges

1 Convert each undirected edge $(u, v) \in E$ to two directed edges $\langle u, v\rangle$ and $\langle v, u\rangle$ with HM costs one.
2 **for** *iteration := 1 to MaxIterations* **do**
3 Pick a random (s^i, g^i) pair.
4 Compute a minimum-cost path P from s^i to g^i in the directed graph using the current HM costs of the edges.
5 For each edge $\langle u, v\rangle$ on this path P, increase $n(u, v)$ by 1 and then update the follow preference $p(u, v)$, the saturation cost $s(u, v)$, and the interference cost of the opposite edge $t(v, u)$.
6 Return some of the edges with small HM costs as highway edges.

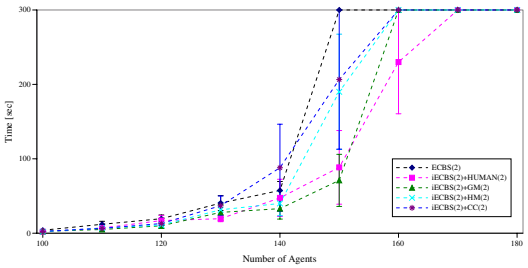
4.3 Experimental Results

We now show that the GM-based and HM-based approaches can make ECBS(w_1)+HWY(w_2) and iECBS(w_1)+HWY about as fast as a human-generated highway in Kiva-like warehousing domains. We report the median over 10 trials for the same experimental set-up as before. Figure 6(a) shows the median runtimes of the ECBS-based MAPF solvers with suboptimality factor 3. We choose this suboptimality factor since it has been used before [Cohen *et al.*, 2015]. The runtime on a MAPF instance counts as 300 if the 300-second runtime limit is reached. We do not include the runtime needed to generate the highways since it is fast compared to running the MAPF solvers. In Figure 6(a), ECBS(1.5)+HWY(2) runs faster and has a smaller median absolute deviation than ECBS(3) in many cases, despite having the same suboptimality factor. Furthermore, ECBS(1.5)+HWY(2) with automatically generated highways runs about as fast as with human-generated highways in many cases. Figure 6(b) shows a similar behavior for the iECBS-based MAPF solvers with suboptimality factor $w_1 = 2$. We choose this suboptimality factor since it minimizes the median runtime of ECBS($w1$). Yet, iECBS(2) runs faster than ECBS(2) in many cases despite having the same suboptimality factor.

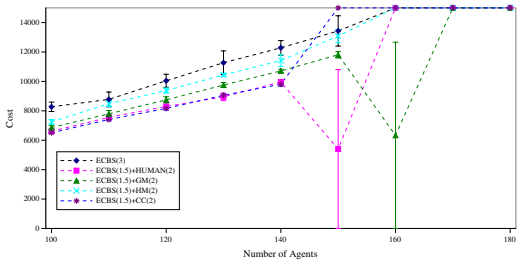
Figure 7 repeats the experiment for 150 agents but we now



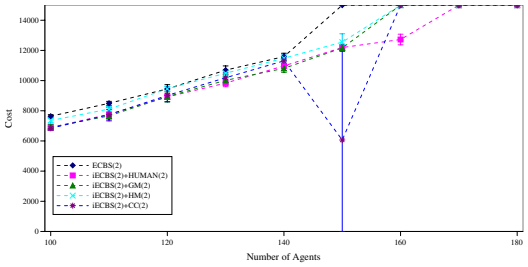
(a) Median runtimes (in seconds) of ECBS(1.5)+HWY(2) with different highways (HUMAN, GM, HM and CC) and ECBS(3).



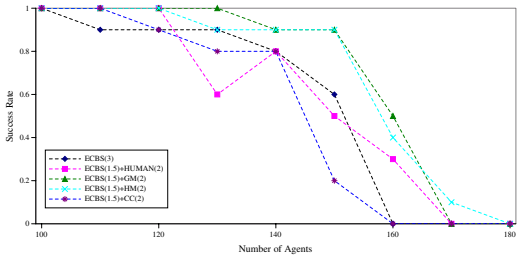
(b) Median runtimes (in seconds) of iECBS(2) with different highways (HUMAN, GM, HM and CC) and ECBS(2).



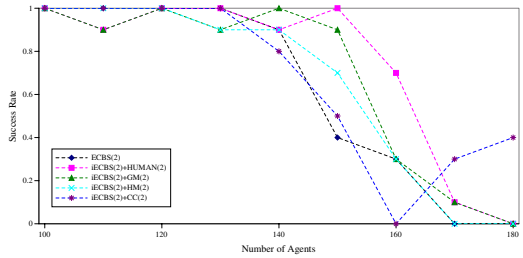
(c) Median solution costs of ECBS(1.5)+HWY(2) with different highways (HUMAN, GM, HM and CC) and ECBS(3).



(d) Median solution costs iECBS(2) with different highways (HUMAN, GM, HM and CC) and ECBS(2).



(e) Success rates of ECBS(1.5)+HWY(2) with different highways (HUMAN, GM, HM and CC) and ECBS(3).



(f) Success rates of iECBS(2) with different highways (HUMAN, GM, HM and CC) and ECBS(2).

Figure 6: Shows the median runtimes (in seconds), median solution costs and success rates of various MAPF solvers in a Kiva-like warehousing domain for various numbers of agents. Each data point is the median or average over 10 trials. A vertical bar around a data point indicates the median absolute deviation.

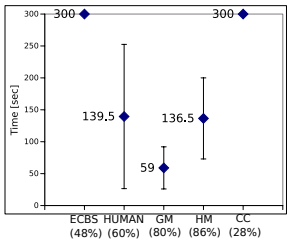
report the median over 50 (rather than only 10) trials with different instances than before. We choose 150 agents because the 300-second runtime limit is reached for many MAPF instances with 150 and 160 agents and the MAPF solvers seem to run faster with human-generated highways than automatically generated ones in some of these cases. Here, ECBS(1.5)+HWY(2) and iECBS(2) with automatically generated highways run about as fast as with human-generated highways but iECBS(2) also runs fast with the CC highway while ECBS(1.5)+HWY(2) does not. However, we can slow down iECBS(2) with the CC highway substantially with small changes to the map, such as changing the spacing of the corridors, due to its fixed layout.

5 Rapid Randomized Restarts

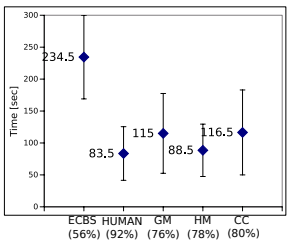
We now develop a simple rapid randomized restart strategy that can increase the success rate of iECBS(w_1) within a given runtime limit. Figures 6 and 7 show large median absolute deviations of the runtime of the MAPF solvers, similar to problems studied in the context of heavy-tailed phenomena in combinatorial search [Gomes *et al.*, 2000]. This observation leads us to develop a rapid randomized restart strategy for MAPF solvers. The idea is for the MAPF solvers to perform multiple short runs rather than one long run. Each run is performed using a random permutation of the initial order in which individual paths are planned for the agents. The low-level searches try to avoid collisions with previously computed paths for other agents, which has a significant effect not only on the paths found in the high-level root node but also on which collisions need to be resolved as the high-level search proceeds. Thus, it has a significant effect on the runtime, which our rapid randomized restart strategy exploits.

5.1 Experimental Results

We now partition the 50 MAPF instances in Figure 7(b) into 38 "easy" MAPF instances that iECBS(2) with GM-based



(a) ECBS(1.5)+HWY(2) with different highways (HUMAN, GM, HM and CC) and ECBS(3)



(b) iECBS(2) with different highways (HUMAN, GM, HM and CC) and ECBS(2).

Figure 7: Shows the median runtimes (in seconds) of various MAPF solvers in a Kiva-like warehousing domain with 150 agents. Each data point is the median over 50 trials. The vertical bar around a data point indicates the median absolute deviation. The percentage indicates the success rate over the 50 trials.

	"easy"	"hard"	"all"
100 sec	97.65%	96.87%	97.60%
60 sec	98.57%	98.81%	98.70%

Figure 8: Shows how a rapidly randomized restart strategy improves the success rate of iECBS(2) with GM-based highways for a given 300-second runtime limit. The rows are the runtime limits. The 100-second row, for example, splits the 300-second runtime limit into 3 runs with a 100-second runtime limit each. The columns partition all 50 MAPF instances into 38 "easy' ones that are solvable without rapid randomized restarts and 12 "hard" ones that are not.

highways solves within the 300-second runtime limit and 12 "hard" MAPF instances that it does not solve. Figure 8 reports the success rates for both partitions of MAPF instances and different time limits. We perform 50 trials for each MAPF instance. If x out of y MAPF instances are solved within the time limit and the time limit allows for z runs, then the success rate is calculated as $1-(1-x/y)^z$. iECBS(2) with five short runs with a 60-second runtime limit each solves 98.70% of all 50 MAPF instances, while iECBS(2) without restarts solves only 76% of them in the previous experiment.

6 Limitations

We now show that both of our first approaches for automatically generating highways have limitations. One limitation of both approaches is their difficulty to generate highways in open areas or non-straight highways, such as in the domain in Figure 9 where two rooms are connected by two zigzag corridors. We compare iECBS(1.5) with the human-generated highway shown in Figure 9 and ECBS(1.5). We use 10 randomly generated MAPF instances with 24 agents each, half of which are assigned a random start vertex in the upper-right room and a random goal vertex in the lower-left room, and vice-versa for the other half of the agents. We use a 60-second runtime limit per MAPF instance. While Figure 10

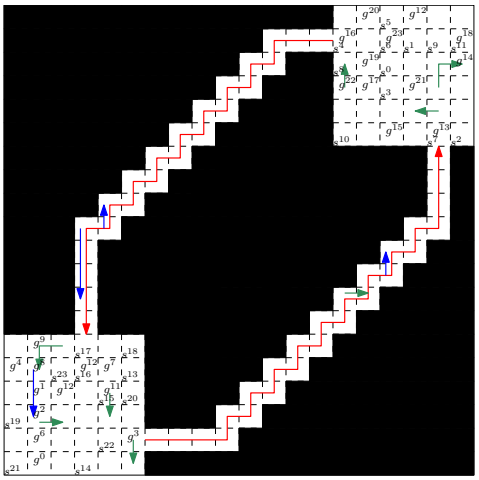


Figure 9: Shows a MAPF instance with zigzag corridors and a human-generated highway in red, a highway generated by the GM-based approach in blue and a highway generated by the HM-based approach in green.

		1	2	3	4	5	6	7	8	9	10
ECBS(1.5)	Time	19.37	0.03	60	1.55	60	0.02	0.11	60	0.02	0.47
	Cost	**904**	**916**	n/a	943	n/a	**896**	**879**	n/a	**912**	959
iECBS(1.5)	Time	**0.10**	**0.01**	**0.4**	**0.25**	60	**0.01**	**0.04**	**60**	**0.01**	**0.09**
	Cost	917	918	**920**	**940**	n/a	897	938	n/a	915	**934**

Figure 10: Compares the runtimes (in seconds) and solution costs of iECBS(1.5) with a human-generated highway and ECBS(1.5) in a domain with zigzag corridors. Each column corresponds to a randomly generated MAPF instance.

shows that iECBS(1.5) with the human-generated highway runs fast, Figure 9 shows that the GM-based and HM-based approaches generate highways that do not look intuitive. Another limitation of the GM-based approach is its difficulty to handle symmetrical start and goal vertices, such as for the MAPF instance in Figure 11. The GM-based approach computes the minimum-cost paths for all agents independently. Since they are mostly symmetrical, the magnitudes of the direction vectors in the upper part of the room are mostly zero and the GM-based approach generates a highway that does not look intuitive, as shown in the figure. Slight pertubations of the start and goal vertices to break the symmetry can help the GM-based approach.

To address these limitations, we are developing fast parameter-tuning approaches for the GM-based and HM-based approaches. We are also developing a GM-based approach that first computes the minimum-cost paths for all agents independently, then generates highways based on these paths, then computes new minimum-cost paths for all agents independently by using highway heuristic values that bias the search to find paths that use the highway edges, then generates new highways based on these paths, and so on. Finally, we are developing a GM-based approach that uses abstractions of the domains into larger areas (and their connections) and additional layers of hidden variables to reason more globally.

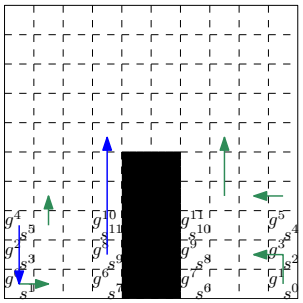


Figure 11: Shows a symmetrical MAPF instance and a highway generated by the GM-based approach in blue and a highway generated by the HM-approach in green.

7 Conclusions

In this paper, we improved bounded-suboptimal MAPF solvers in three ways: 1) We developed iECBS(w_1), a variant of ECBS(w_1)+HWY(w_2) that uses highways to guide its search, has suboptimality factor w_1 rather than $w_1 w_2$ and can run faster than ECBS(w_1). 2) We developed two first approaches for automatically generating highways for a given MAPF instance. 3) Finally, we observed heavy-tailed behavior in the runtimes of these MAPF solvers and developed a simple rapid randomized restart strategy that can increase the success rate iECBS(w_1) within a given runtime limit. In future work, we intend to explore portfolio approaches to combine the strengths of our different MAPF solvers. We also intend to develop more sophisticated rapid randomized restart strategies for different MAPF solvers, for example, by randomizing the high-level and low-level searches, such as the selection of high-level nodes and the collisions to resolve. We also intend to use knowledge-based methods to identify good initial orders in which individual paths should be planned for the agents in the high-level root node.

References

[Barer *et al.*, 2014] Max Barer, Guni Sharon, Roni Stern, and Ariel Felner. Suboptimal variants of the conflict-based search algorithm for the multi-agent pathfinding problem. In *Proceedings of the 7th Annual Symposium on Combinatorial Search*, 2014.

[Cohen *et al.*, 2015] Liron Cohen, Tansel Uras, and Sven Koenig. Feasibility study: Using highways for bounded-suboptimal multi-agent path finding. In *Proceedings of the 8th Annual Symposium on Combinatorial Search*, 2015.

[Gomes *et al.*, 2000] Carla Gomes, Bart Selman, Nuno Crato, and Henry Kautz. Heavy-tailed phenomena in satisfiability and constraint satisfaction problems. *Journal of Automated Reasoning*, 24(1-2):67–100, 2000.

[Hoenig *et al.*, 2016] Wolfgang Hoenig, T. K. Satish Kumar, Sven Koenig, Liron Cohen, Hang Ma, Hong Xu, Nora Ayanian, and Sven Koenig. Multi-agent path finding with kinematic constraints. In *Proceedings of the 26th International Conference on Automated Planning and Scheduling*, 2016. To appear.

[Jansen and Sturtevant, 2008] Renee Jansen and Nathan Sturtevant. A new approach to cooperative pathfinding. In *Proceedings of the 7th International Joint Conference on Autonomous Agents and Multiagent Systems*, pages 1401–1404, 2008.

[Koller and Friedman, 2009] Daphne Koller and Nir Friedman. *Probabilistic Graphical Models: Principles and Techniques*. MIT Press, 2009.

[Leighton *et al.*, 1995] Tom Leighton, Fillia Makedon, Serge Plotkin, Clifford Stein, Eva Tardos, and Sypros Tragoudas. Fast approximation algorithms for multicommodity flow problems. *Journal of Computer and System Sciences*, 50(2):228 – 243, 1995.

[Mooij, 2010] Joris Mooij. libDAI: A free and open source C++ library for discrete approximate inference in graphical models. *Journal of Machine Learning Research*, 11:2169–2173, 2010.

[Morris *et al.*, 2016] Robert Morris, Corina Pasareanu, Kasper Luckow, Waqar Malik, Hang Ma, T. K. Satish Kumar, and Sven Koenig. Planning, scheduling and monitoring for airport surface operations. In *Proceedings of the Workshop on Planning for Hybrid Systems at the 30th AAAI Conference on Artificial Intelligence*, 2016.

[Pearl and Kim, 1982] Judea Pearl and Jin Kim. Studies in semi-admissible heuristics. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 4:392 –399, 1982.

[Phillips *et al.*, 2012] Michael Phillips, Benjamin Cohen, Sachin Chitta, and Maxim Likhachev. E-graphs: Bootstrapping planning with experience graphs. In *Proceedings of the 8th Robotics: Science and Systems Conference*, 2012.

[Sharon *et al.*, 2015] Guni Sharon, Roni Stern, Ariel Felner, and Nathan Sturtevant. Conflict-based search for optimal multi-agent pathfinding. *Artificial Intelligence*, 219:40–66, 2015.

[Wagner, 2015] Glenn Wagner. *Subdimensional Expansion: A Framework for Computationally Tractable Multirobot Path Planning*. PhD thesis, Carnegie Mellon University, 2015.

[Wang and Botea, 2008] Ko-Hsin Wang and Adi Botea. Fast and Memory-Efficient Multi-Agent Pathfinding. In *Proceedings of the 18th International Conference on Automated Planning and Scheduling*, pages 380–387, 2008.

[Wurman *et al.*, 2008] Peter Wurman, Raffaello D'Andrea, and Mick Mountz. Coordinating hundreds of cooperative, autonomous vehicles in warehouses. *AI Magazine*, 29(1):9–20, 2008.

[Yu and LaValle, 2013] Jingjin Yu and Steven LaValle. Structure and intractability of optimal multi-robot path planning on graphs. In *Proceedings of the 27th AAAI Conference on Artificial Intelligence*, pages 1443–1449, 2013.