Multi-Directional Heuristic Search

Dor Atzmon[1] , **Jiaoyang Li**[2] , **Ariel Felner**[1] , **Eliran Nachmani**[1] ,
Shahaf Shperberg[1] , **Nathan Sturtevant**[3] and **Sven Koenig**[2]
[1]Ben-Gurion University
[2]University of Southern California
[3]University of Alberta
dorat@post.bgu.ac.il, jiaoyanl@usc.edu, felner@bgu.ac.il, nachamni@post.bgu.ac.il,
shperbsh@post.bgu.ac.il, nathanst@ualberta.ca, skoenig@usc.edu

Abstract

In the *Multi-Agent Meeting* problem (MAM), the task is to find a meeting location for multiple agents, as well as a path for each agent to that location. In this paper, we introduce MM*, a *Multi-Directional Heuristic Search* algorithm that finds the optimal meeting location under different cost functions. MM* generalizes the *Meet in the Middle* (MM) bidirectional search algorithm to the case of finding an optimal meeting location for multiple agents. Several admissible heuristics are proposed, and experiments demonstrate the benefits of MM*.

1 Introduction

MM [Holte *et al.*, 2016] is a bidirectional heuristic search algorithm with a unique priority function; nodes are ordered in OPEN according to $\max(2g, g+h)$. MM is guaranteed to *meet in the middle* but the practical meaning of this property is that the two search frontiers never venture further than $C^*/2$ from their start locations (where C^* is the cost of the shortest path). Nevertheless, it is important to note that a meeting location is not returned by MM; a shortest path from a start location to a goal location is returned. In fact, a meeting location is not even defined in MM.

In this paper, we deal with the "real" problem of *meeting in the middle*, where a meeting location is returned. Furthermore, while MM is designed for a problem that is defined on start and goal locations, our problem is defined on $k \geq 2$ start locations where we search for an optimal meeting location with respect to the shortest path from each of these k start locations to that meeting location. We call this problem the *Multi-Agent Meeting* problem (MAM). We focus on minimizing the following two common cost functions defined on possible solutions. **(1)** *Sum-Of-Costs* (SOC): the sum of the costs of the paths to the meeting location. **(2)** *Makespan* (MKSP): the cost of the longest path to the meeting location. SOC may corresponds to the energy (fuel) consumed until the agents meet, and MKSP corresponds to the time elapsed until the agents meet. MAM has many real-life applications, such as choosing a gathering point for multiple traveling agents (humans, cars, or robots), or suggesting a location that needs to be close to important surrounding locations.

To find the optimal meeting location, we introduce the *Multi-Directional Meet in the Middle* (MM*) algorithm. MM* is a best-first search algorithm that progresses in k directions until a meeting location is found. We provide a unique priority function for each of the SOC and MKSP cost functions and prove the optimality of MM* for these cases. MM* is strongly related to MM — its priority function for MKSP is a generalization of that of MM, although their halting conditions are different (so as to return the actual meeting location). MM* relies on an admissible heuristic function and we propose a number of such heuristic functions for SOC and MKSP. We then provide experimental results that demonstrate the benefits of MM* with these heuristic functions.

MAM is also known as *Optimal Meeting Point* [Yan *et al.*, 2015], *Smallest Enclosing Discs* [Welzl, 1991], and *1-Center problem* [Megiddo, 1983] in other areas of computer science. In the field of computational geometry, MAM is known as the *Weber problem* [Cooper, 1968]. Many efficient algorithms exist for MAM in continuous Euclidean spaces [Ostresh Jr, 1977; Chen, 1984; Radó, 1988; Rosing, 1992] that calculate a geometric point that satisfies the relevant constraints. The problem has also been investigated on general graphs by applying variants of *Dijkstra's algorithm* [Dijkstra, 1959] in parallel, one for each agent. Once a path from all start locations to all locations in the graph is known, the best meeting location can be chosen by iterating over all relevant meeting locations, exhaustively. Many improvements have been suggested, such as pruning areas of the state space [Lanthier *et al.*, 2005; Xu and Jacobsen, 2010; Yan *et al.*, 2015; Li *et al.*, 2019; Geisberger *et al.*, 2008]. We continue this direction and, to the best of our knowledge, are the first ones to add admissible heuristics and study MAM in the context of heuristic search.

2 Problem Definition

The *Multi-Agent Meeting* problem (MAM) receives as input a weighted, undirected graph $G = (V, E)$ and a set of k start locations $S = \{s_1, \ldots, s_k\} \subseteq V$ for k agents $A = \{a_1, \ldots, a_k\}$. The cost of edge $(v, v') \in E$ is denoted by $c(v, v') > 0$. A solution is a target location $t \in V$, indicating a meeting location for the agents, plus a set of shortest paths from each s_i to t. Let $d(v, u)$ be the cost of a shortest

path from v to u. The cost functions for SOC and MKSP for a meeting location t are calculated as follows. For SOC:

$$C_{SOC}(t) = \sum_{a_i \in A} d(s_i, t). \quad (1)$$

This corresponds to the sum of the costs of the paths to the meeting location. For MKSP:

$$C_{MKSP}(t) = \max_{a_i \in A} d(s_i, t). \quad (2)$$

This is the cost of the longest path to the meeting location. An *optimal* solution has the lowest cost among all possible solutions. Its cost is denoted by C^* and its meeting location by t^*. In this paper, we focus on optimal solutions.

Figure 1(a) illustrates a MAM problem instance with three agents a_1, a_2, and a_3 with start locations s_1, s_2, and s_3, respectively. Edges are labeled with their costs. Consider location t as a meeting location. Since $d(s_1, t) = 5$, $d(s_2, t) = 5$, and $d(s_3, t) = 5$, $C_{SOC}(t) = 15$ while $C_{MKSP}(t) = 5$. Now, consider v. Since $d(s_1, v) = 8$, $d(s_2, v) = 2$, and $d(s_3, v) = 2$, $C_{SOC}(v) = 12$ and $C_{MKSP}(v) = 8$. v is optimal for minimizing SOC and t for MKSP. Next, we present the MM* algorithm that solves MAM for both cost functions.

3 Multi-Directional MM (MM*)

MM* is a multi-directional best-first search algorithm that guarantees to return an optimal MAM solution for both SOC and MKSP. As will be shown below, the only difference in MM* between SOC and MKSP is the priority function used.

A node in MM* is a pair (a_i, v) representing an agent and its location. MM* organizes nodes in a single open-list (denoted OPEN) and a single closed-list (denoted CLOSED).[1] OPEN is initialized with k root nodes: (a_i, s_i) representing each of the k agents and its start location. Each node is associated with a g-value. Naturally, $g(a_i, s_i) = 0$. Let $N(v)$ represent the successors of v. Expanding a node (a_i, v) has two parts: **(1)** Generating a node (a_i, v') for each $v' \in N(v)$, setting $g(a_i, v') = g(a_i, v) + c(v, v')$, and inserting it into OPEN. **(2)** Moving (a_i, v) to CLOSED.

Recall that, in heuristic search, given a node n in the search tree, $f^*(n)$ is defined to be the cost of the optimal solution that passes through n, and $f(n)$ is defined to be a lower bound on $f^*(n)$. This terminology is migrated to MM*. Let $f^*(a_i, v)$ be the cost of the optimal MAM solution (for either SOC or MKSP) such that a_i passes through v on its way to the meeting location. $f(a_i, v)$ is a lower bound on $f^*(a_i, v)$, i.e., $f(a_i, v) \leq f^*(a_i, v)$. In Section 4, we define $f(a_i, v)$ for either SOC or MKSP by exploiting admissible heuristic functions that estimate the remaining cost of all agents (including that of a_i) that can be added to $g(a_i, v)$ (the cost of the path from s_i to v along the search tree). Each of these f-values can be plugged into MM*.

[1] Bidirectional searches usually maintain two open-lists, one for each search direction, but the priority function can choose a node from either one of them. This is logically equivalent to a single open-list which contains nodes from both directions. MM* uses a single open-list, which is equivalent to k open-lists, one for each agent, that use the same priority function.

Algorithm 1: The MM* Algorithm

```
1  Main(MAM problem instance)
2      Init OPEN, CLOSED; U ← ∞
3      foreach a_i ∈ A do
4          Insert (a_i, s_i) into OPEN
5      while OPEN is not empty do
6          Extract (a_i, v) from OPEN // with lowest f(a_i, v)
7          if f(a_i, v) ≥ U then
8              return U
9          foreach v' ∈ N(v) do
10             if CLOSED contains (a_i, v') then
11                 if g(a_i, v') ≤ g(a_i, v) + c(v, v') then
12                     continue
13                 Remove (a_i, v') from CLOSED
14             else if OPEN contains (a_i, v') then
15                 if g(a_i, v') ≤ g(a_i, v) + c(v, v') then
16                     continue
17             Insert (a_i, v') into OPEN // possibly overwriting
18             Update U
19         Insert (a_i, v) into CLOSED
20     return U
```

There is no notion of a goal node in MM* but instead we have a goal condition on each location v. We say that v is a *possible goal* iff it has been generated from all directions, i.e., $\forall a_i \in A$: $(a_i, v) \in$ OPEN $\cup$ CLOSED. To manage this in practice, for each location v, we keep a k bit-vector, where bit i is set when node (a_i, v) is generated. If v is a possible goal, its cost $C(v)$ depends on the cost function and is:

$C_{SOC}(v) = \sum_{a_i \in A} g(a_i, v)$ and

$C_{MKSP}(v) = \max_{a_i \in A} g(a_i, v)$.[2]

Let U be the cost of the *incumbent solution*, i.e., U is the minimum $C(v)$ among all possible goals (initially $U = \infty$). U is an upper bound on C^*. The halting condition for MM* is to halt if $fmin \geq U$, where $fmin$ is the minimum f-value in OPEN. This guarantees that U cannot be further improved.

Algorithm 1 gives the pseudo-code of MM*. First, MM* initializes OPEN and CLOSED, and sets $U = \infty$ (line 2). Then, the initial nodes (a_i, s_i) are inserted into OPEN (lines 3-4). MM* performs a best-first search as follows. While OPEN is not empty (line 5), it extracts (a_i, v), the best node (with the lowest f-value) from OPEN (line 6). Then, it checks the halting condition on location v, i.e., whether $fmin = f(a_i, v) \geq U$ (lines 7-8). Otherwise, it performs the expansion cycle on (a_i, v) (lines 9-18). MM* performs *duplicate detection and pruning* on CLOSED (lines 10-13) and OPEN (lines 14-16). As a result, MM* always keeps the lowest seen g-value for each generated node (a_i, v'). In general, MM* allows nodes in CLOSED to be re-opened (line 13). But, this will never happen for consistent heuristics (such as all heuristics that we suggest and experiment with below). If (a_i, v') is not a duplicate node, then (a_i, v') is inserted into OPEN (line

[2] In Equations 1 and 2, we used $d(s_i, v)$. Here, we use $g(a_i, v)$ because the path is the path from s_i to v along the search tree.

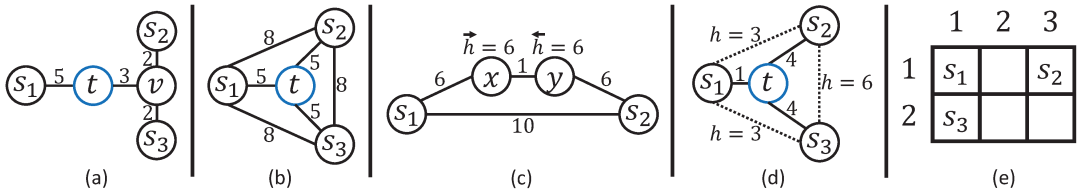

Figure 1: (a,b) f_{MKSP} examples. (c) MM and MM* difference example. (d,e) Heuristics examples.

17). Then, U is updated in case v' is a possible goal with a smaller cost (line 18). After its expansion, (a_i, v) is inserted into CLOSED (line 19). When U is returned (lines 8 and 20), it also includes the meeting location t as well as the paths to it, that can be constructed by parent-pointers (not included in the pseudo-code).

4 MM* Priority Functions

We next define the priority function f for MM* for both SOC and MKSP. In A*, given a node n, a perfect heuristic $h^*(n)$ equals $d(n, goal)$ and a perfect priority function $f^*(n)$ equals $g(n) + h^*(n)$. If $h(n)$ is a lower bound on $h^*(n) = d(n, goal)$, then $f(n) = g(n) + h(n)$ is a lower bound on $f^*(n)$. Next, we generalize this to MM* and define all these functions for both SOC and MKSP.

4.1 SOC

Consider node (a_i, v) in OPEN. $f^*_{SOC}(a_i, v)$ is the cost of the optimal solution such that: **(1)** a_i passes through v (via the path of cost $g(a_i, v)$ along the search tree). **(2)** a_i continues from v to meet the other agents at some location t. **(3)** Each of the other agents a_j travels from s_j to t. Now, $f^*_{SOC}(a_i, v) = g(a_i, v) + h^*_{SOC}(a_i, v)$, where $h^*_{SOC}(a_i, v)$ is the sum of the cost of a_i to get from v to t along a shortest path (item 2), plus the cost of the other agents to get from their start locations to t along shortest paths (item 3):

$$h^*_{SOC}(a_i, v) = \min_{t \in V} \{d(v, t) + \sum_{a_j \in A \setminus \{a_i\}} d(s_j, t)\}. \quad (3)$$

We denote the best meeting location t w.r.t. to node (a_i, v) by $t^*(a_i, v)$. $h_{SOC}(a_i, v)$ is an admissible estimate (lower bound) of $h^*_{SOC}(a_i, v)$, i.e., $h_{SOC}(a_i, v) \leq h^*_{SOC}(a_i, v)$. We propose a number of admissible h-functions for SOC in Section 7. For SOC, naturally,

$$f_{SOC}(a_i, v) = g(a_i, v) + h_{SOC}(a_i, v). \quad (4)$$

4.2 MKSP

The MKSP case is more complicated. Since, in MKSP, we take the maximum among agents (not the sum), we do not know which agent has the path with the highest cost. We begin by defining $f^*_{MKSP}(a_i, v)$, which is the optimal solution given that a_i passes through v, via a path of cost $g(a_i, v)$:

$$f^*_{MKSP}(a_i, v) = \min_{t \in V} \left[\max \begin{Bmatrix} g(a_i, v) + d(v, t) \\ \max_{a_j \in A \setminus \{a_i\}} d(s_j, t) \end{Bmatrix} \right]. \quad (5)$$

For a given possible meeting location t, we want the path of one of the agents with the highest cost. If this is our current agent a_i, then this is given by $g(a_i, v) + d(v, t)$ (top line of the max term). If it is some other agent a_j, then it is given by $d(s_j, t)$ (bottom line).

Next, we need to define f_{MKSP} as a lower bound on f^*_{MKSP}. Here, we do not define h^*_{MKSP} and h_{MKSP} but define $f_{MKSP}(a_i, v)$ in terms of $h_{SOC}(a_i, v)$ as follows:

$$f_{MKSP}(a_i, v) = \max \left\{ g(a_i, v), \frac{g(a_i, v) + h_{SOC}(a_i, v)}{k} \right\}, \quad (6)$$

where k is the number of agents. $g(a_i, v)$ is a lower bound on $f^*_{MKSP}(a_i, v)$ because a_i has already traveled along a path of cost $g(a_i, v)$. Thus, $f^*_{MKSP}(a_i, v) \geq g(a_i, v)$. Now, observe that $\frac{f^*_{SOC}(a_i, v)}{k} \leq f^*_{MKSP}(a_i, v)$ because one of the agents must travel at least $\frac{f^*_{SOC}(a_i, v)}{k}$. Since $f_{SOC}(a_i, v)$ is a lower bound on $f^*_{SOC}(a_i, v)$, dividing it by k will yield a lower bound on $f^*_{MKSP}(a_i, v)$.

Costs of Subsets

f^*_{MKSP} for k agents is determined by the path of one of the agents with the highest cost. Therefore, f^*_{MKSP} and f_{MKSP} for any subset of these k agents are also lower bounds on f^*_{MKSP} for *all* k agents. Thus, for any subset of $k' < k$ agents, we can compute f_{MKSP} and use it as a lower bound on f^*_{MKSP} for the entire set of k agents. Therefore, while the right-hand side of the max function in $f_{MKSP}(a_i, v)$ (Equation 6) contains all k agents, it can also contain any subset of agents. This can be done by calculating $h_{SOC}(a_i, v)$ for the selected subset of $k' < k$ agents and dividing it by k' instead of k. Figure 1(a,b) shows examples of MAM problem instances for MKSP. In both cases, the optimal MKSP is 5 at location t. Assume a perfect heuristic for SOC. For Figure 1(a), the optimal SOC is 12 at location v. Thus, for each agent a_i, $f_{SOC}(a_i, s_i) = 12$, and, by computing MKSP for all agents, we get $f_{MKSP}(a_i, s_i) = 12/3 = 4$. Now, consider the subset of agents $\{a_1, a_2\}$. Their SOC is 10, and hence $f_{MKSP}(a_i, s_i) = 10/2 = 5$. For Figure 1(b), the optimal SOC is 15 at location t, and thus $f_{SOC}(a_i, s_i) = 15$. By computing MKSP for all agents, we get $f_{MKSP}(a_i, s_i) = 15/3 = 5$ but the SOC of the subset of agents $\{a_1, a_2\}$ is 8, and hence $f_{MKSP}(a_i, s_i) = 8/2 = 4$. This shows that there is no best subset for all cases. In our experiments, we used all combinations of pairs of agents, in addition to the set of all agents. It is future work to investigate additional subset selection policies.

5 MM & MM* - Similarities and Differences

It is very interesting that f_{MKSP} is a generalization of the priority function of the MM algorithm: $pr(n) = \max(2g(n), g(n) + h(n))$ [Holte *et al.*, 2016].

If we divide this expression by two, we get $pr(n) = \max(g(n), \frac{g(n)+h(n)}{2})$. This is a special case of the proposed f_{MKSP} for $k = 2$ ($h(n)$ is equivalent to an estimate of the cost from the start location of the backward agent, the goal in MM, to the current location of the forward agent). MM prioritizes nodes based on MKSP to keep MM *restrained* [Shaham *et al.*, 2017], i.e., to never expand nodes with $g(n) > \frac{C^*}{2}$.

In Figure 1(c), we illustrate the difference between MM and MM* (for MKSP with 2 agents) with regard to halting. Both algorithms start by inserting s_1 and s_2 into OPEN. Next, both algorithms expand s_1 and generate x and s_2. MM sets $pr(x) = 12$ and $pr(s_2) = 10$ (from the forward side; $h_F(x) = 6$ and $h_F(s_2) = 0$), and $U = 10$ because a path of cost 10 has been found. At this point, MM halts, as $U \leq fmin = 10$, so a path of smaller cost cannot be found. By contrast, MM* sets $f_{MKSP}(a_1, x) = g(a_1, x) = 6$ and $f_{MKSP}(a_1, s_2) = g(a_1, s_2) = 10$, and $U = 10$ because a meeting location with cost 10 has been found. Unlike MM, MM* continues to search because a better meeting location might be found as $fmin = f_{MKSP}(a_1, x) = 6$. While U is similar for both algorithms, the priorities of MM* are half of the ones of MM. So, MM* continues and returns either x or y as meeting location. For SOC, it returns s_2 as meeting location with cost 10, as it is on a path of minimal cost.

6 Theoretical Analysis

Lemma 1 (Completeness)**.** *MM* is guarantees to return a solution if one exists, and $U = \infty$ otherwise.*

Proof outline. For each agent a_i, MM* performs a best-first search from s_i. In the worst case, MM* explores every reachable location for each agent. If a solution exists, a location reachable for all agents will be generated from all directions and U will be updated. At some point, either $fmin$ will reach U or the entire graph will have been explored for all agents (OPEN will be empty), and a solution will be returned. If no solution exists, there is no location that is reachable for all agents and U will not be updated and thus remains ∞. □

Lemma 2 (Optimality)**.** *Given an admissible f (i.e., $f(n) \leq f^*(n)$ for all nodes n), MM* is guaranteed to return the optimal location t^* with cost C^*.*

Proof outline. Assume, by contradiction, that MM* returned a sub-optimal location $t \neq t^*$ with cost $C > C^*$. Since MM* has terminated and returned a solution, $fmin \geq C > C^*$. Since MM* terminated without returning an optimal solution, there exists a node $n' = (a_i, v_i)$ in OPEN s.t. v_i is a location on the optimal path of a_i to t^*, and every node before n' on the path has already been expanded. Since n' is the first node on the optimal path that was not expanded, it was generated by a node on the optimal path, and thus $g(n') = d(s_i, v_i)$. By definition, $f^*(n')$ is the cost of the optimal solution that passes through n'. Therefore, since $g(n') = d(s_i, v_i)$, $f^*(n') = C^*$. f is admissible. Thus, $f(n') \leq f^*(n') = C^*$. As $n' \in$ OPEN, $fmin \leq f(n') \leq f^*(n') = C^*$, which contradicts the fact that $fmin \geq C > C^*$. □

7 Heuristics for MM*

We now introduce a number of heuristics for SOC and prove their admissibility. They are plugged directly into $f_{SOC}(a_i, v) = g(a_i, v) + h_{SOC}(a_i, v)$ and used indirectly for f_{MKSP} as shown in Equation 6. Let $t^*(a_i, v)$ be the optimal meeting location where a_i passes through v. For simplicity, we use $\hat{t^*}$ to denote $t^*(a_i, v)$ and $h(a_i, v)$ to denote $h_{SOC}(a_i, v)$. Recall that S is the set of all start locations. Let $S_i(v)$ be the set of all start locations in S, except for s_i, which is replaced with v (the current location of a_i). Formally, $S_i(v) = S \setminus \{s_i\} \cup \{v\}$. Then, $h^*_{SOC}(a_i, v) = \sum_{v' \in S_i(v)} d(v', \hat{t^*})$ (from Equation 3), and we want to find a lower bound on it.[3]

7.1 h_1 : Clique Heuristic

We assume that, for every pair of locations (v_1, v_2), there exists a classic admissible heuristic h (e.g., straight-line distance or Manhattan distance), such that $h(v_1, v_2) \leq d(v_1, v_2)$.

Based on the triangle inequality, for every pair of locations $v_1, v_2 \in S_i(v)$ (with $v_1 \neq v_2$), we have that:

$$d(v_1, v_2) \leq d(v_1, \hat{t^*}) + d(v_2, \hat{t^*}). \tag{7}$$

By summing over all such pairs, we get:

$$\sum_{\substack{\{v_1,v_2\} \in 2^{S_i(v)} \\ v_1 \neq v_2}} d(v_1, v_2) \leq \sum_{\substack{\{v_1,v_2\} \in 2^{S_i(v)} \\ v_1 \neq v_2}} \left[d(v_1, \hat{t^*}) + d(v_2, \hat{t^*}) \right]. \tag{8}$$

As each $v' \in S_i(v)$ is paired with $k-1$ other locations in $S_i(v)$, we can rewrite the right-hand side of Equation 8 as $(k-1) \cdot \sum_{v' \in S_i(v)} d(v', \hat{t^*})$. Therefore:

$$\sum_{\substack{\{v_1,v_2\} \in 2^{S_i(v)} \\ v_1 \neq v_2}} \frac{d(v_1, v_2)}{k-1} \leq \sum_{v' \in S_i(v)} d(v', \hat{t^*}) = h^*(a_i, v). \tag{9}$$

Now, since $h(v_1, v_2) \leq d(v_1, v_2)$, we get that:

$$h_1(a_i, v) = \sum_{\substack{\{v_1,v_2\} \in 2^{S_i(v)} \\ v_1 \neq v_2}} \frac{h(v_1, v_2)}{k-1} \leq h^*(a_i, v). \tag{10}$$

This heuristic h_1 is called the *Clique heuristic*, as it combines the heuristic values of *every* (unordered) pair of locations in $S_i(v)$. Figure 1(d) presents an example of the clique heuristic for three agents. For node (a_1, s_1), $S_1(s_1) = \{s_1, s_2, s_3\}$. Therefore, $h(a_1, s_1) = \frac{h(s_1,s_2)+h(s_1,s_3)+h(s_2,s_3)}{2} = \frac{3+3+6}{2} = 6$.

For each start location $s_i \in S$, $S_i(s_i) = S$ and hence h_1 can be calculated once for all start locations. For each location v that is not a start location, all locations in $S_i(v)$ except for v remain the same. So, h_1 can be calculated incrementally in time that is linear in the number of agents. If $h(v_1, v_2)$ is consistent, then h_1 is also consistent.

[3] This is a form of a front-to-end heuristic. A front-to-front heuristic needs to estimate the remaining costs when all other agents are in their current locations, but these locations thus need to be specified for a given node (a_i, v). This is left for future work.

7.2 h_2 : Median Heuristic

For a set of numbers $B \subset \mathbb{R}$, the median of B provably minimizes the sum of the absolute deviations, i.e., $\text{argmin}_{r\in\mathbb{R}} \sum_{b\in B} |b - r| = median(B)$. Inspired by this property, we design the *Median heuristic* (h_2) for 4-neighbor 2D grids. On such a grid, each location has two coordinates – x and y. For a set of locations, we can find the median over the x-coordinates (dimension 1) of all locations and the median over the y-coordinates (dimension 2) of all locations. Let tm_d be the median of dimension d. This creates a potential meeting location $tm = (tm_1, tm_2)$, that minimizes the sum of the absolute deviations over both dimensions. Namely, if there are no obstacles on the grid, tm will be the optimal meeting location for minimizing SOC. This is due to the fact that the distance between any two locations is their L_1-*distance* (also known as Manhattan distance on 2D grids).

Assume that the input graph $G = (V, E)$ is a 4-neighbor 2D grid where every location $v \in V$ is represented by its coordinates $\vec{v} = (v_1, v_2)$. The L_1-distance for any two locations $u, v \in V$ is defined as $||\vec{u} - \vec{v}||_1 = |u_1 - v_1| + |u_2 - v_2|$. Due to the existence of obstacles, for any pair of locations $u, v \in V$, $||\vec{u} - \vec{v}||_1 \leq d(u, v)$. So, for a given node (a_i, v),

$$\sum_{v'\in S_i(v)} ||\vec{v'} - \vec{\hat{t}^*}||_1 \leq \sum_{v'\in S_i(v)} d(v', \hat{t^*}) = h^*(a_i, v). \quad (11)$$

Therefore, by modeling the problem in an empty 2D L_1-space (i.e., without obstacles), we introduce a new admissible (and consistent) heuristic, called *Median heuristic*:

$$h_2(a_i, v) = \min_{\vec{t}\in\mathbb{R}^2}\{\sum_{v'\in S_i(v)} ||\vec{v'} - \vec{t}||_1\} \quad (12)$$

$$= \sum_{v'\in S_i(v)} ||\vec{v'} - \vec{tm}||_1 \quad (13)$$

$$= \sum_{v'\in S_i(v)} [|v'_1 - tm_1| + |v'_2 - tm_2|]. \quad (14)$$

It is admissible because the right-hand side of Equation 12 is no larger than the left-hand side of Equation 11.

We use the Quick-select algorithm for finding medians [Hoare, 1961], which runs in $\Theta(k)$ time, to compute $h_2(a_i, s_i)$ for all root nodes. Then, for every non-root node (a_i, v) (i.e., $v \neq s_i$), $k - 1$ locations have not changed, and we only need to update the median based on the single location that has changed. This can be done in $O(1)$ time.

Figure 1(e) shows an example of a MAM problem instance with three agents (s_1, s_2, and s_3) that are located in $(1, 1), (3, 1)$, and $(1, 2)$. The x-coordinates are {1,3,1} and the y-coordinates are {1,1,2}. Thus, the median location is $(1, 1)$. By computing the Manhattan distances from $(1, 1)$ to each start location, we get $h_2(a_i, s_i) = 3$ for each agent a_i.

h_2 can be generalized easily for 6-neighbor 3D grids and similar graphs of higher dimensions.

7.3 h_3 : FastMap Heuristic

The Median heuristic (h_2) only works for graphs that have coordinates. In addition, h_2 ignores obstacles, which may introduce inaccuracies to the heuristic. The *FastMap heuristic* (h_3) handles this. *FastMap* [Cohen *et al.*, 2018; Li *et al.*, 2019] is a near-linear preprocessing algorithm that embeds the locations of a given edge-weighted undirected connected graph $G = (V, E)$ into a D-dimensional L_1-space $\mathbb{R}^D$. The dimension D of the L_1-space is user-specified. Each location $v_i \in V$ is mapped to a D-dimensional point $\vec{p_i} \in \mathbb{R}^D$. The length of a shortest path $d(v_i, v_j)$ between any two locations $v_i, v_j \in V$ is approximated by the L_1-distance $||\vec{p_i} - \vec{p_j}||_1$ between the corresponding two points $\vec{p_i}, \vec{p_j} \in \mathbb{R}^D$ in this space. FastMap ensures that the L_1-distance in $\mathbb{R}^D$ can be used as an admissible and consistent heuristic for the shortest path computation in G. See [Cohen *et al.*, 2018] for more details on FastMap. To compute h-values for MAM, h_3 applies the Median heuristic on the generated embedding $\mathbb{R}^D$. For any node (a_i, v), its FastMap heuristic is defined as:

$$h_3(a_i, v) = \min_{\vec{t}\in\mathbb{R}^D}\{\sum_{v'\in S_i(v)} ||\vec{p'} - \vec{t}||_1\}, \quad (15)$$

where $\vec{p'} \in \mathbb{R}^D$ is the point of the embedding of location v' generated by FastMap. By the same analysis as for h_2, we can show that the FastMap heuristic (h_3) is admissible (and consistent), and we can compute it for every root node in $\Theta(kD)$ time and for any non-root node in $O(D)$ time.

There are many other approaches for embedding a graph in a continuous space, such as [Ng and Zhang, 2002; Shavitt and Tankel, 2004; Rayner *et al.*, 2011]. However, most of them use L_2-distances, which are not applicable here because the L_2-distance version of Equation 15 is NP-hard to solve optimally [Hoare, 1961].

8 Experimental Results

We experimented with MM* on an Intel® Xeon E5-2660 v4 @2.00GHz processor with 16GB of RAM. We compared all new heuristics to the Dijkstra version of MM*, i.e. where $h = 0$ (denoted by h_0), on different grids while minimizing both SOC and MKSP. For h_1, we used the Manhattan Distance (MD) as a classic admissible heuristic between any two locations. The number of dimensions D for h_3 was always set to 10, as suggested by Li *et al.* [2019].

8.1 SOC

We experimented on a 500×500 grid with 0% – 30% obstacles. Table 1(left) shows the average over 50 instances of: the solution cost, the initial h-value, the number of expansions, and the CPU time for 5 randomly placed agents. h_2 had the best initial h-value, had the lowest number of expansions, and was the fastest. The initial h-values of h_1 (MD) and h_2 remain constant as more obstacles are added since they both ignore obstacles. By contrast, h_3 increases as more obstacles are added. So, h_2 degrades while h_3 improves, in terms of number of expansions and CPU time. h_3 incurred a preprocessing time of $\approx$ 30s, which is incurred only once per grid and thus amortized over multiple problem instances.

We also fixed the number of obstacles at 10% while varying the number of agents from 3 to 9. Table 2(left) shows the CPU time for SOC. Here, too, h_2 was the best heuristic with only 1.27s for 9 agents because h_2 is suitable for grids with small numbers of obstacles.

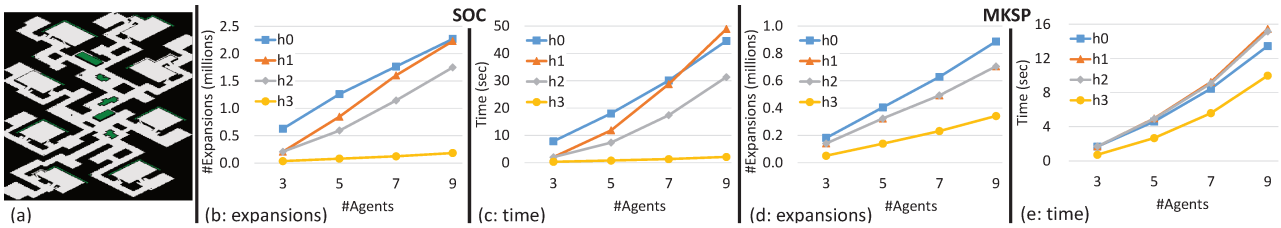

Figure 2: (a) Enigma grid. (b) SOC expansions. (c) SOC time. (d) MKSP expansions. (e) MKSP time.

	SOC				MKSP			
#Obs.	0%	10%	20%	30%	0%	10%	20%	30%
Cost								
	995	1,008	1,033	1,103	292	293	295	305
Initial h-value								
h_0	0	0	0	0	0	0	0	0
h_1	828	828	828	828	166	166	166	166
h_2	**995**	**995**	**995**	**995**	**199**	**199**	**199**	**199**
h_3	643	680	736	778	127	137	140	148
#Expansions (thousands)								
h_0	1,244	1,120	994	856	542	485	420	341
h_1	330	322	320	318	180	159	133	121
h_2	**34**	**58**	**83**	**143**	**179**	**158**	**132**	**119**
h_3	634	561	465	402	308	299	239	197
Time (s)								
h_0	22.05	19.04	16.10	13.04	7.81	6.69	5.47	4.10
h_1	4.50	4.29	4.18	4.01	2.83	2.43	1.94	1.66
h_2	**0.28**	**0.54**	**0.77**	**1.40**	**2.79**	**2.40**	**1.93**	**1.64**
h_3	9.48	8.66	6.81	5.60	6.99	5.44	4.12	4.00

Table 1: Results on 500×500 grids with varying obstacles.

Finally, we experimented on the 768×768 Enigma grid (presented in Figure 2(a)) from the Starcraft video game, available in the `movingai` repository [Sturtevant, 2012]. Figures 2(b) and 2(c) show the average number of expansions and CPU time, respectively, for 3 to 9 agents for SOC. Here, h_3 was the best heuristic in both expansions and time. Since this grid has many obstacles (about 57%), h_1 and h_2 were less effective than h_3, which uses real distances (albeit in the embedded graph). Nevertheless, h_3 required a preprocessing time of 39s for this grid (done once). h_2 was the second best heuristic but h_2 is only suited for grids while h_1 can be used on any graph. For 9 agents, h_1 expanded slightly fewer nodes than h_0 but, since it consumes time for computing the heuristic, was a little slower than h_0.

8.2 MKSP

As described in Section 4, there are different policies for choosing subsets for computing MKSP. We compared two of these policies: (1) selecting all agents and (2) selecting all agents plus all pairs of agents and taking the maximum over all of those. Figure 3 shows the number of expansions and

	SOC				MKSP			
#Agents	3	5	7	9	3	5	7	9
h_0	6.87	18.98	29.46	44.17	1.91	6.73	11.03	18.07
h_1	**0.16**	4.66	16.15	36.29	0.47	2.62	3.57	6.53
h_2	**0.16**	**0.81**	**0.85**	**1.27**	**0.46**	**2.60**	**3.52**	**6.38**
h_3	1.92	8.50	18.66	33.20	1.48	6.57	9.59	16.46

Table 2: Avg. time (s) on 500×500 grids with 10% obstacles

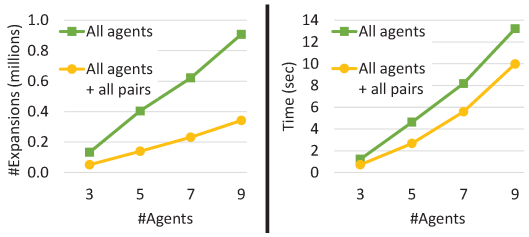

Figure 3: MKSP results for h_3 on the Enigma grid.

CPU time averaged over 50 instances for h_3 on the Enigma grid with 3 to 9 agents. As expected, adding all pairs of agents produces a better heuristic and fewer expansions. It was also better in terms of the CPU time despite the fact that its computation overhead is larger. Therefore, we used this subset selection policy for computing MKSP. It is future work to investigate different policies for choosing subsets.

We repeated the same experiments (reported for SOC) for MKSP. Table 1(right) presents the results. Here, too, h_2 was the best heuristic but here (unlike SOC) h_1 was very close to h_2. This is so probably because the clique heuristic (h_1) for MKSP also guides the agents to the median.

The same trends were observed for different numbers of agents with 10% obstacles and are presented in Table 2(right).

In the Enigma grid for MKSP (Figures 2(d,e)), h_3 was again the best heuristic. h_1 and h_2 (curves cover each other) had fewer numbers of expansion but were slower than h_0.

9 Conclusions

We introduced the multi-directional search algorithm MM* that optimally solves MAM. We proved that MM* is complete and optimal and suggested a few admissible heuristics. Experimentally, we showed that MM* performs better with heuristics. For grids with few obstacles, h_2 is best. For grids with many obstacles, h_3 is best but requires preprocessing. The advantage of h_1 is that it is applicable to all domains without the need for preprocessing. Future work will: (1) develop a weighted version of MM* that can find bounded-suboptimal solutions, (2) extend MM* to a Euclidean space, and (3) further investigate subset selection for MKSP.

Acknowledgements

This research was supported by ISF grant 844/17 and BSF grant 2017692 to Ariel Felner, NSF grant 1815660 to Nathan R. Sturtevant and NSF grants 1409987, 1724392, 1817189, 1837779 and 1935712 to Sven Koenig.

References

[Chen, 1984] Reuven Chen. Location problems with costs being sums of powers of Euclidean distances. *Computers & Operations Research*, 11(3):285–294, 1984.

[Cohen *et al.*, 2018] L. Cohen, T. Uras, S. Jahangiri, A. Arunasalam, S. Koenig, and T. K. S. Kumar. The FastMap algorithm for shortest path computations. In *the International Joint Conference on Artificial Intelligence (IJCAI)*, pages 1427–1433, 2018.

[Cooper, 1968] Leon Cooper. An extension of the generalized Weber problem. *Journal of Regional Science*, 8(2):181–197, 1968.

[Dijkstra, 1959] Edsger W Dijkstra. A note on two problems in connexion with graphs. *Numerische Mathematik*, 1(1):269–271, 1959.

[Geisberger *et al.*, 2008] Robert Geisberger, Peter Sanders, Dominik Schultes, and Daniel Delling. Contraction hierarchies: Faster and simpler hierarchical routing in road networks. In *the International Workshop on Experimental and Efficient Algorithms*, pages 319–333, 2008.

[Hoare, 1961] Charles AR Hoare. Algorithm 65: Find. *Communications of the ACM*, 4(7):321–322, 1961.

[Holte *et al.*, 2016] Robert C. Holte, Ariel Felner, Guni Sharon, and Nathan R. Sturtevant. Bidirectional search that is guaranteed to meet in the middle. In *the AAAI Conference on Artificial Intelligence (AAAI)*, pages 3411–3417, 2016.

[Lanthier *et al.*, 2005] Mark A Lanthier, Doron Nussbaum, and Tsuo-Jung Wang. Calculating the meeting point of scattered robots on weighted terrain surfaces. In *the Australasian Theory Symposium*, volume 41, pages 107–118, 2005.

[Li *et al.*, 2019] Jiaoyang Li, Ariel Felner, Sven Koenig, and T. K. S. Kumar. Using FastMap to solve graph problems in a Euclidean space. In *the International Conference on Automated Planning and Scheduling (ICAPS)*, pages 273–278, 2019.

[Megiddo, 1983] Nimrod Megiddo. The weighted Euclidean 1-center problem. *Mathematics of Operations Research*, 8(4):498–504, 1983.

[Ng and Zhang, 2002] T. S. Eugene Ng and Hui Zhang. Predicting internet network distance with coordinates-based approaches. In *the Annual Joint Conference of the IEEE Computer And Communications Societies (INFOCOM)*, pages 170–179, 2002.

[Ostresh Jr, 1977] Lawrence M Ostresh Jr. The multifacility location problem: Applications and descent theorems. *Journal of Regional Science*, 17(3):409–419, 1977.

[Radó, 1988] Francisc Radó. The Euclidean multifacility location problem. *Operations Research*, 36(3):485–492, 1988.

[Rayner *et al.*, 2011] Chris Rayner, Michael Bowling, and Nathan R. Sturtevant. Euclidean heuristic optimization. In *the AAAI Conference on Artificial Intelligence (AAAI)*, pages 81–86, 2011.

[Rosing, 1992] Kenneth E Rosing. An optimal method for solving the (generalized) multi-Weber problem. *European Journal of Operational Research*, 58(3):414–426, 1992.

[Shaham *et al.*, 2017] Eshed Shaham, Ariel Felner, Jingwei Chen, and Nathan R. Sturtevant. The minimal set of states that must be expanded in a front-to-end bidirectional search. In *the International Symposium on Combinatorial Search, SoCS*, pages 82–90, 2017.

[Shavitt and Tankel, 2004] Yuval Shavitt and Tomer Tankel. Big-bang simulation for embedding network distances in Euclidean space. *IEEE/ACM Transactions on Networking*, 12(6):993–1006, 2004.

[Sturtevant, 2012] Nathan R. Sturtevant. Benchmarks for grid-based pathfinding. *Computational Intelligence and AI in Games*, 4(2):144–148, 2012.

[Welzl, 1991] Emo Welzl. Smallest enclosing disks (balls and ellipsoids). In *New Results and New Trends in Computer Science*, pages 359–370. Springer, 1991.

[Xu and Jacobsen, 2010] Zhengdao Xu and Hans-Arno Jacobsen. Processing proximity relations in road networks. In *the ACM SIGMOD International Conference on Management of Data (SIGMOD)*, pages 243–254, 2010.

[Yan *et al.*, 2015] Da Yan, Zhou Zhao, and Wilfred Ng. Efficient processing of optimal meeting point queries in Euclidean space and road networks. *Knowledge and Information Systems*, 42(2):319–351, 2015.