Analyzing and Improving the Use of the FastMap Embedding in Pathfinding Tasks

Reza Mashayekhi[1], **Dor Atzmon**[3, 4], **Nathan R. Sturtevant**[1, 2]

[1]Department of Computing Science, University of Alberta, Canada
[2]Alberta Machine Intelligence Institute (Amii)
[3]Ben Gurion University of the Negev, Israel
[4]Royal Holloway, University of London
{rmashaye, nathanst}@ualberta.ca, dorat@post.bgu.ac.il

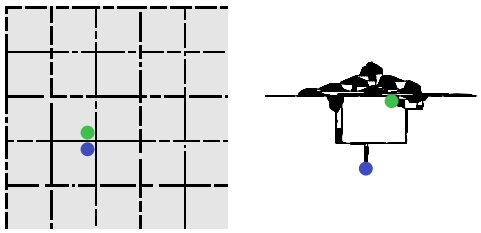
Abstract

The FastMap algorithm has been proposed as an inexpensive metric embedding which provides admissible distance estimates between all vertices in an embedding. As an embedding, it also supports additional operations such as taking the median location of two vertices, which is important in some problems. This paper studies several aspects of FastMap embeddings, showing the relationship of FastMap to general additive heuristics. As an admissible heuristic, FastMap is not as strong as previous suggested. However, by combining FastMap with the ideas of differential heuristics, we can significantly improve the performance of FastMap heuristics. We show the impact of these ideas in both single-agent pathfinding and the Multi-Agent Meeting problem, where the performance of algorithms using our improved FastMap embedding is improved by up to a factor of two.

Introduction

The idea of re-embedding a state space to improve heuristic estimates has received moderate research interest over the last decade (Rayner, Bowling, and Sturtevant 2011; Chen, Weinberger, and Chen 2013; Cohen et al. 2018). Figure 1 illustrates the power of an embedding. The original map is in Figure 1(a), while the graph representation of this map has been re-embedded in Figure 1(b). The distance between points in both representations can be used as an admissible heuristic for the true distances. In Figure 1(a) the two points marked with circles seem to be nearly adjacent. But, the embedding is able to capture these distances more accurately, placing these same vertices farther apart. Embeddings can be used as improved heuristics for algorithms like A*, or can be used for operations such as finding the median location between vertices (Atzmon et al. 2020). We are particularly interested in the FastMap embedding, which has been adapted and studied in different applications (Li et al. 2019; Gopalakrishnan et al. 2020; Li et al. 2022).

There are many open questions about the L^1 variant of FastMap (Cohen et al. 2018), particularly when it is used as an admissible heuristic. These include how FastMap performs across different problem types, how FastMap relates to other heuristics, whether FastMap can use alternate embedding functions, and whether FastMap pivot selection can be optimized as in differential heuristics (DHs) can (Goldberg and Harrelson 2005a; Sturtevant et al. 2009; Rayner, Sturtevant, and Bowling 2013).



(a) (b)

Figure 1: (a) Two points in a sample map that appear close and (b) a 2-dimensional re-embedding of the map where distances are represented more accurately.

This paper is the result of deep study of the FastMap heuristic, making the following contributions. First, we present validation results of FastMap, showing that FastMap is much worse in practice that previous results suggested. Next, we provide pedagogical results establishing the relationship between FastMap and other additive heuristics. Then, a general framework shows how to (1) combine FastMap and DHs into a new FM+DH approach, and (2) optimize the selection of pivots in FastMap to allow multiple independent FastMap heuristics to be used together.

Experimental results give insight into the performance of the A* and MM* algorithms using our improved FastMap embedding. In A*, our improved embeddings are up two 2x better than the original FastMap, when measured by average expansions, while there is minimal difference in runtime speed. Extensive experimentation shows that the relative strength of FastMap depends highly on the type of maps being tested. These results also carry over to MM* (Atzmon et al. 2020) in the Multi-Agent Meeting problem, which can take advantage of an L^1 embedding to estimate the best meeting point when solving this problem.

Related Work and Background

For many years, research in heuristic search focused on exponential domains, where Pattern Databases (Culberson and Schaeffer 1998) are a dominant heuristic approach. A

broader range of approaches have been developed in planning, such as merge-and-shrink (Helmert et al. 2014; Sievers and Helmert 2021) among many others. These abstraction-based approaches are generally not effective in polynomial domains (Felner, Sturtevant, and Schaeffer 2009) where heuristics based on true distances (Sturtevant et al. 2009) are more effective. This has motivated approaches such as ALT (A*, Landmarks, and the Triangle Inequality) (Goldberg and Harrelson 2005a), differential heuristics (DHs) (Sturtevant et al. 2009), and optimal Euclidean embeddings (Rayner, Bowling, and Sturtevant 2011).

Most of the true-distance approaches can be seen as embedding the state space into a new metric space where distances in the embedding can be used as heuristics in the original state space. Optimal Euclidean embeddings are built based on an optimization that builds a single high-dimensional embedding and then discards all but k of the dimensions to construct the final embedding; L^2 (Euclidean) distance can be used to compute heuristic values. ALT and DHs build k-dimensional embeddings incrementally, using the L^∞ norm (max) of several different embeddings to compute a heuristic. FastMap (Cohen et al. 2018) is also an incremental approach that builds a k-dimensional embedding, one dimension at a time. The original FastMap approach, adapted from work in data mining (Faloutsos and Lin 1995), uses L^1 (Manhattan) distance to compute heuristics, which are also admissible using an L^2 norm (Li et al. 2019). FastMap and DHs both require $O(N)$ time (as measured by expansions) and memory to compute and store a single dimension, where N is the size of the input graph. A formal definition of these approaches will be provided later in the paper. Euclidean embeddings also require $O(N)$ memory per dimension, but require $O(N^3)$ time to compute.

There are other approaches that also use memory constant in the size of the input graph, however these require more computation to construct. These include Bounding Boxes (Rabin and Sturtevant 2016; Hu et al. 2021), which store a bounding box on each edge. This bounding box contains all possible goals that can be reached optimally by following an edge. Reach (Goldberg, Kaplan, and Werneck 2007) stores with each state the maximum distance to a start/goal that can be reached optimally via that state. Both of these approaches provide constraints that prune states from the search. While both can be very effective, they are more expensive to compute exactly, and cannot be improved with further memory. This contrasts with heuristics. DHs store, in the limit, a perfect heuristic (Sturtevant et al. 2009), and monotonically improve with more memory (Rayner, Sturtevant, and Bowling 2013). While FastMap will monotonically improve with more memory, it will not approach a perfect heuristic.

Note that the FastMap algorithm has different embedding variants (Cohen et al. 2018; Li et al. 2019). This paper studies the L^1 variant of FastMap and its applications.

Problem Definition

The primary problem studied in this paper is how to build one or more metric embeddings of a graph. In this problem, the input is defined by a graph, a cost function, and a heuristic: (G, c, h). $G = (V, E)$ is a weighted undirected graph, with vertices V and edges E. The cost of all edges are non-negative, and are defined by the cost function $c : E \rightarrow \mathbb{R}^+$. Let a path π be a tuple of vertices $\{v_0, v_1, v_2, \ldots, v_n\}$ where $\{v_i, v_{i+1}\} \in E$. The cost of a path is $c(\pi) = \sum_{i=0}^{n-1} c(\{v_i, v_{i+1}\})$. The heuristic h is an estimate of the cost of the shortest path between two vertices $h : V \times V \rightarrow \mathbb{R}^+$. Let C^* be the cost of an optimal (least-cost) path between v_1 and v_2. A heuristic is *admissible* if for all $e = \{v_1, v_2\} \in E$ $h(v_1, v_2) \leq C^*$ and is *consistent* if for all $e = \{v_1, v_2\} \in E$ and $v \in V$ $|h(v_1, v) - h(v_2, v)| \leq c(v_1, v_2)$.

The output of algorithms that solve this problem is a metric embedding of the vertices in G. The embedding function maps (embeds) each vertex to k-dimensional real-valued coordinates; $\ell : v \rightarrow \mathbb{R}^d$. An embedding is metric if distances are symmetric, self distance is 0, and the triangle inequality holds. We let $\ell_i(v)$ be the ith dimension (or coordinate) of the vertex v in the metric space. Grid graphs are, by definition, already embedded in the metric space defined by their coordinates in the grid. In a 2D grid the embedding for a grid cell would just be the coordinates of that cell; eg $\ell(v) = (3, 4)$ where $l_0(v) = 3$ and $l_1(v) = 4$. The metric embedding is a new tuple of coordinates for each vertex. This embedding can then be used to reason about vertices, such as the average location of a group of vertices, or can be used directly as an admissible heuristic.

The approaches we describe require distance computations to perform embeddings. Thus, we define a distance function d_c as the cost of the shortest path between vertices using cost function c on the input graph. We allow d to return distances between pairs of vertices or from a set of vertices to a single vertex. In the second case it returns the cost of the shortest path to any vertex in the set. In either case the distance function can be computed by a single Dijkstra search.

The quality of an embedding can measured by the performance of algorithms using that embedding. For the purpose of pathfinding, performance is measured by the number nodes expanded by A* when using the embedding as a heuristic. Time is also an important measure, if approaches have different computational overheads. The heuristics compared in this paper perform exactly the same memory accesses and, as we will show, thus have nearly identical time overhead. In the multi-agent meeting problem (Atzmon et al. 2020) performance is measured by the number of expansions performed by MM*.

Pre-Study of FastMap

We began this research by attempting to replicate previous published results that claimed FastMap as an L^1 heuristic is 'competitive with other state-of-the-art heuristics' (Cohen et al. 2018). The distribution of node expansions on a single grid map by A* with a 10-dimensional FastMap embedding (FM10) and DH with 10 heuristics (DH10) is found in Figure 2. This distribution suggests that FastMap is not competitive. There is little difference in performance on 75% of the benchmark problems used, but significant differences on the remaining 25%.

Looking further into the results, there are two primary ex-

Algorithm 1 Generalized Metric Embedding

1: **Input:** $G = \{V, E\}, c, h, k$
2: **Output:** Embedding: $\ell_1(V), \ell_2(V), ..., \ell_k(V)$
3: $i = 1$;
4: $c_1 = c$
5: $P = \varnothing$
6: **while** $i \leq k$ **do**
7: $\quad P \leftarrow$ SELECTPIVOTS(G, c, c_i, h, P)
8: $\quad \forall_{v \in V} \ell_i(v) \leftarrow$ EMBED(v, G, c_i, P)
9: $\quad c_{i+1} \leftarrow$ UPDATECOSTS(G, c_i, ℓ_i)
10: $\quad i \leftarrow i + 1$
11: **end while**
12: **return** ℓ

planations. First, the original paper looked at the percentage of problems where FastMap outperforms DHs, but not the magnitude of the improvement on these problems. Second, the original paper looked primarily at the median node expansions. These alone do not give a full view of the distribution of problem difficulties. We additionally observed that the original experiments were run on random problems; benchmark problems contain a longer tail of hard problems, which further reduce the performance of FastMap. The FM9+DH curve in Figure 2 shows the improved performance that will be achieved from the ideas in this paper.

Unified Description of DH and FastMap

The key idea studied in this paper is the idea of embedding the vertices from the input graph into a metric space to use as heuristics for search. To simplify the description of the contributions of the paper we begin with a unified description of DHs and (L^1) FastMap.

These two approaches are generalized by the pseudo-code in Algorithm 1. That is to say, these approaches take as input a graph, a cost function, possibly a default heuristic, and the number of dimensions for the resulting embedding, k. The embedding process works by first selecting pivots (line 7), using those pivots to compute the embedding (line 8), and then possibly updating the cost function (line 9). These steps are done once per dimension. The pivots are list of vertices $P = (p_0, p_1, \ldots, p_{|P|-1})$.

We can now provide the specific details of SELECTPIVOTS, EMBED, and UPDATECOSTS for FastMap and DHs, as well as how the final embedding is used in practice.

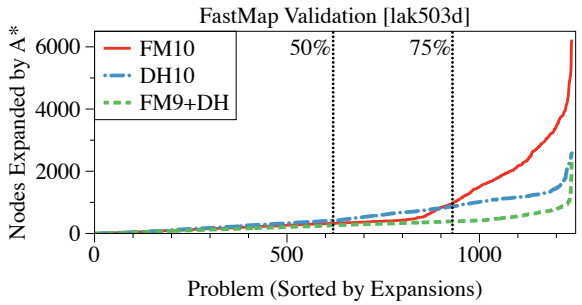


Figure 2: Baseline heuristic performance with A*.

Algorithm 2 SELECTPIVOTS (Farthest) for DH

1: **Input:** $G = \{V, E\}, c_{input}, c_{curr}, h, P$
2: **Output:** list of pivots
3: $S \leftarrow \varnothing$
4: **if** $P = \varnothing$ **then**
5: $\quad S \leftarrow$ random state from V;
6: **else**
7: $\quad S \leftarrow P$
8: **end if**
9: $P \leftarrow P \bigcup \text{argmax}_{v \in V} : d_{c_{input}}(S, v)$
10: **return** P

Algorithm 3 EMBED for DH

1: **Input:** v, G, c, P
2: **Output:** Location of v in embedding
3: **return** $d_c(p_{|P|-1}, v)$

Differential Heuristics

Each dimension of a differential heuristic (DH) is built based on the selection of a single pivot vertex, p_i, which is described below. Once a pivot vertex is selected, the distance $d_c(p_i, v)$ is computed for all $v \in V$ using the input cost function c. Then, in the ith dimension, vertex v is embedded at location $d(p_i, v)$ (Algorithm 3). Let $\ell(v)$ be the embedded coordinate of vertex v and $\ell_i(v)$ be the ith dimension of the embedding of v. Then, the differential heuristic $h_{DH}(v_1, v_2) = \max_i |\ell_i(v_1) - \ell_i(v_2)|$. That is, the DH measures the distance between vertices in each individual dimension and takes the maximum of these distances to return the final heuristic. Using the maximum of the distances in each dimension is equivalent to using L^∞ norm to measure distances between vertices in the embedding.

There are multiple ways to select the pivot locations used for each DH. The simplest approach, called *furthest* (Goldberg and Harrelson 2005b; Sturtevant et al. 2009), chooses the next pivot, p_{i+1}, as the point in the graph that has maximum distance from previous pivots $p_0 \ldots p_i$. In this implementation (Algorithm 2), P is the list of all pivots selected thus far. Each new pivot is added to P so that future pivots can be selected based on the past pivots stored in P, but the embedding function only uses the last pivot selected (Algorithm 3). The first pivot is the location that is farthest from a random point in the graph. A more expensive approach is to use subset selection (Rayner, Sturtevant, and Bowling 2013) to greedily select the best pivots. Our experiments verified that this does result in a better embedding, but it requires significantly more precomputation time and memory, so we only use the simpler and faster approach here.

The DH method always uses distances in the input cost function, c, so the UPDATECOSTS method, not shown here, just returns the original cost function.

We demonstrate a one-dimensional DH embedding in Figure 3. The first portion of the figure shows the input graph

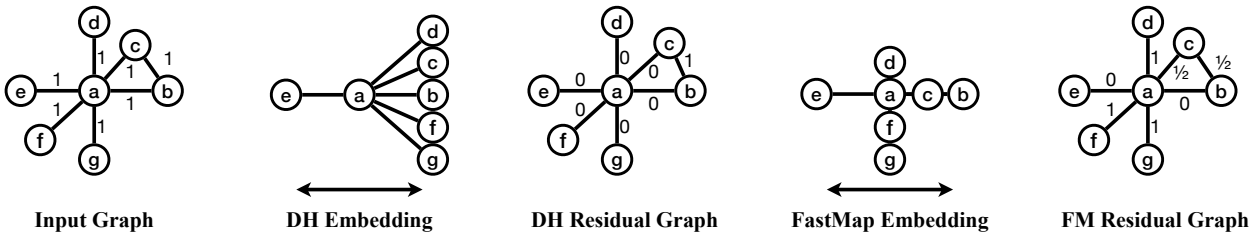


Figure 3: Example of FastMap and DH embeddings

along with the weights in the graph. In the DH embedding, the graph is embedded in one dimension along the x-axis. The separation in the y-axis is only so vertices can be distinguished. In this example, the pivot vertex is e, and all other vertices are embedded according to their distance from e. While all distances to e are captured perfectly, the distance estimate between all remaining pairs vertices (e.g. $h(b, c)$) excluding a is 0, because they are all at the same x coordinate of 2. Subsequent embeddings would in turn select each of these vertices as the next pivot. The DH residual graph will be discussed later.

L^1 FastMap Heuristic

Each dimension of the FastMap heuristic is built based on the selection of two pivot vertices, p_1 and p_2 (Algorithm 4). Once the pivot vertices are selected, the distances $d_c(p_1, v)$ and $d_c(p_2, v)$ are computed for all $v \in V$ using the current residual cost function, which will be described momentarily. Then in the ith dimension, vertex v is embedded at location $(d_c(p_{i_1}, v) + d_c(p_{i_1}, p_{i_2}) - d_c(v, p_{i_2}))/2$ (Algorithm 5). Note that this embedding function could also be used with DHs (using the L^∞ norm), but FastMap updates the cost function in order to use the L^1 norm (Manhattan distance) in the embedding for the heuristic instead.

FastMap computes a new cost function c_{i+1} for each edge (v_1, v_2) as $c_{i+1}(v_1, v_2) = c_i(v_1, v_2) - |\ell_i(v_1) - \ell_i(v_2)|$ (Algorithm 6). That is, after each embedding, the distances captured in that embedding are subtracted from the cost function. This results in a new graph with the same edge structure, but only the residual costs that were not captured in previous dimensions. Subsequent embeddings are performed based on distances using c_i instead of c. The final heuristic is $h_{FM}(v_1, v_2) = \sum_i |\ell_i(v_1) - \ell_i(v_2)|$, or the L^1 norm.

FastMap chooses pivots similarly to the furthest differential heuristic pivot selection scheme, but does not need to maintain previous pivots, as they are already captured through the updated cost function. The pivot selection method, shown in Algorithm 5, begins by selecting a random vertex in the graph. The first pivot, p_{i_1} is the vertex with maximum distance from the random point. The second pivot, p_{i_2}, is the vertex with maximum distance from p_{i_1}, where distances for pivot i are computed using c_i. We call this pivot selection scheme the maximum embedding distance, or ED, as it attempts to select as pivots the vertices separated by maximum distance in the graph.

We demonstrate a one-dimensional FastMap embedding in Figure 3. The first portion of the figure shows the input graph along with the weights in the graph. In the FastMap embedding the graph is embedded in one dimension along the x-axis. The separation in the y-axis is only so vertices can be distinguished. In this example, the pivot vertices are e and b. Because a, d, f, and g are equidistant from e and b, they are embedded at location 1 between them. Because c is distance 2 from e and distance 1 from b, it is embedded at location $2 + 2 - 1/2 = 1.5$. In this embedding, the heuristic between vertices a, d, f, and g are all 0, because they are embedded at the same coordinate. The FastMap residual graph shows the new cost function after building the first embedding. The edge costs on the shortest path between the pivots have been captured in the first embedding, but no other edge costs have been fully captured. The remaining residual can be captured in the next dimension of the embedding.

Algorithm 4 SELECTPIVOTS$_{\text{ED}}$ (Embedding Distance) for FastMap

1: **Input:** $G = \{V, E\}, c_{input}, c_{curr}, h, P$
2: **Output:** list of pivots
3: $t =$ random state from V
4: $P \leftarrow \text{argmax}_{v \in V}\, d_{c_{curr}}(t, v)$
5: $P \leftarrow P \bigcup \text{argmax}_{v \in V}\, d_{c_{curr}}(p_0, v)$
6: **return** P

Algorithm 5 EMBED for FastMap

1: **Input:** v, G, c, P
2: **Output:** Location of v in embedding
3: **return** $(d_c(p_0, v) + d_c(p_0, p_1) - d_c(v, p_1))/2$

Comparing DH and FastMap Embeddings

DHs and FastMap both produce a k-dimensional embedding. In DHs the heuristic is based on L^∞ distance between the points, while in FastMap the heuristic is the L^1 distance.

The DH pivot selection schema puts the next pivot in the

Algorithm 6 UPDATECOSTS for FastMap

1: **Input:** G, c, ℓ
2: **Output:** Updated cost function
3: $c_{new} \leftarrow c$
4: $\forall_{e=\{v_1,v_2\} \in E} c_{new}(e) \leftarrow c(e) - |\ell(v_1) - \ell(v_2)|$
5: **return** c_{new}

furthest location from all previously placed pivots, so as more dimensions are added to the embedding, pivots will never be duplicated. While this is also true for a single FastMap embedding, as a result of the modified cost function, it is not true if we wish to build multiple independent FastMap embeddings and take the maximum of the heuristics returned. That is, given storage to store 10 embedding dimensions we could choose to build five 2-dimensional embeddings instead of a single 10-dimensional embedding.

Suppose we choose to build five independent 2-dimensional FastMap embeddings. With the existing pivot selection scheme, there is a chance that the same pivots would be selected in all five embeddings, since the selection of the first pivot in the embedding is independent of pivots in other embedding. In this case, there would be no benefit to using more than one embedding. After showing the connection between FastMap and additive heuristics, we will discuss alternate methods of selecting pivots and embeddings for FastMap.

FastMap as an Additive Heuristic

One might wonder why FastMap can add each of the embedding dimensions, while DHs must take the maximum. This is not discussed in the original FastMap paper; our goal here is to connect the FastMap approach to the general theory behind additive heuristics (Yang et al. 2008).

Yang et. al.'s analysis assumes that the costs in the original state space, as defined by c, are divided up into k separate abstractions, where the cost in each abstraction is defined by c_i. In our context we can assume that the abstraction only partitions the costs between abstractions but does not change the structure of the underlying graph. Then, they show that as long as the distances in the abstraction are admissible and consistent, they will also be additive if the costs are partitioned across the abstractions such that they do not exceed the costs in the original graph. That is, $\forall_{e \in E} \sum_{i=1}^{n} c_i(e_i) \leq c(e)$.

In this context, FastMap does not *a priori* divide the costs between abstractions, as is typically done in additive pattern databases (Felner, Korf, and Hanan 2004). Instead, it builds a new dimension to the embedding, measures the costs that were captured, and then removes them from the residual graph. This ensures that the sum of edge costs across all abstractions is additive, but allows costs to be partitioned dynamically as each dimension of the embedding is built. This is related to approaches in planning (Seipp, Keller, and Helmert 2020) that similarly partition costs between actions.

Alternate FastMap Embedding Functions

FastMap's greedy strategy is not optimal. As an exercise to the reader, this can be seen by considering a 4-connected grid map without obstacles. Manhattan distance is an optimal heuristic in the default grid embedding, but FastMap is unable to reproduce this embedding when embedding relative to a pair of vertices. As such, it is worthwhile to consider both alternate strategies for selecting pivots and alternate embedding functions, as both can improve the performance of FastMap. Note that in this specific case the original grid embedding can be reproduced by embedding relative to a path instead of a pair of vertices, something that cannot be explored further here. We propose a simpler approach that is effective in practice.

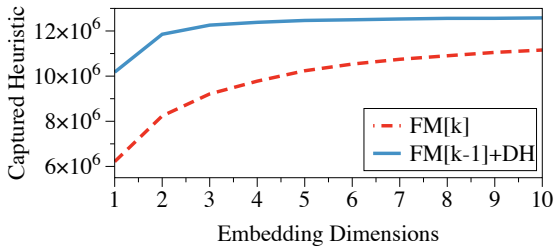


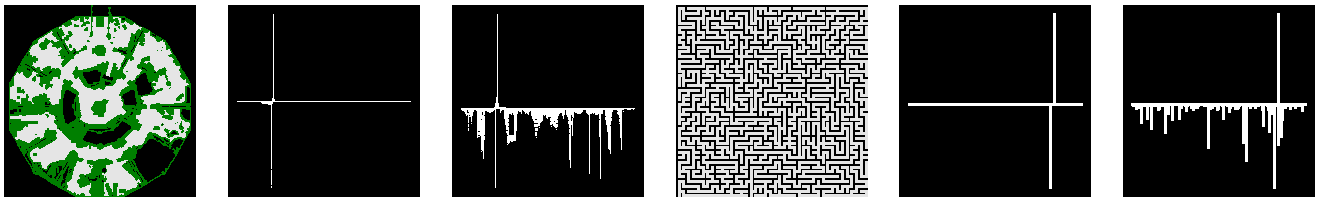
Figure 4: Captured edge costs in embeddings. FM[k] refers to an embedding with k dimensions.

In order to build a metric embedding, the primary requirement is that distances in the embedding obey the triangle inequality. Both the FastMap and DH embeddings meet this property, and there are many other embedding functions that also maintain this property. We focus on combining FastMap and DH embeddings, because this provided the best results.

The primary question we consider is whether we could build a FastMap embedding using the DH embedding function. While this is possible, the DH residual graph in Figure 3 illustrates the problem that arises with the DH embedding. If we compare the FastMap and DH residual graphs, we can see that many more edges were captured in the DH embedding because the sum of edge weights in the residual graph is smaller. But, as a result, the least cost path between vertices c and b is not via the edge cost 1 between them, but is via a, with a total cost of 0. In fact, after performing a DH embedding and computing the residual, the shortest distance between *all* vertices using the residual costs will always be 0. This is because the shortest path between every vertex and the pivot is always fully captured in the DH embedding. So, after building a DH embedding, while residual may remain, it cannot be captured through further embeddings.

If the DH embedding is constructed from the residual graph weights, it is possible to use the DH embedding as part of the additive FastMap heuristic. But, once we have done so, we will not be able to build any additional embeddings from the graph. This suggests that if we aim to build a k-dimensional FastMap embedding, we can use the FastMap embedding function for the first $k-1$ dimensions of the embedding, and the DH embedding function only for the last dimension. While implementation-wise this is very small change to FastMap (essentially one line of code), in practice, it works very well. Experimental results will show that this approach, which we call FM+DH, always gave better aggregate results than using the FastMap embedding.

This is because the FM+DH approach can capture far more residual than the FM approach alone. This is illustrated numerically in Figure 4, where the x-axis shows the number of dimensions in the embedding and the y-axis shows the total edge costs captured in the complete embedding. The more edges captured, the smaller the residual, and the



(a) (b) (c) (d) (e) (f)

Figure 5: (a), (d) Original maps. (b), (e) FM embedding. (c), (f) FMDH embedding.

Algorithm 7 SELECTPIVOTS$_{\text{HE}}$ with Heuristic Error (HE)

1: **Input:** $G = \{V, E\}, c_{input}, c_{curr}, h, P$
2: **Output:** list of pivots
3: **if** $P = \varnothing$ **then**
4: $t =$ random state from V;
5: $p_1 \leftarrow \text{argmax}_{v \in V} : 3d_{c_{input}}(t, v) - 2h(t, v)$
6: $p_2 \leftarrow \text{argmax}_{v \in V} : 3d_{c_{input}}(p_1, v) - 2h(p_1, v)$
7: **return** $p_1 \bigcup p_2$
8: **else**
9: **return** SELECTPIVOTS$_{\text{ED}}(G, c_{input}, c_{curr}, h, P)$
10: **end if**

larger the heuristic. Note that the y-axis does not start at 0. The FM+DH approach captures more information more quickly than the FM approach alone. This can be seen visually by looking at the embeddings in Figure 5. Part (a) and (d) contain the original maps being embedded. Part (b) and (e) are a two-dimensional FM embeddings, while (c) and (f) show the FMDH embedding. The FMDH embedding distinguishes distances between many more states.

FastMap Embedding Based on Heuristic Error

While existing work has taken the maximum of independent FastMap and DHs (Cohen et al. 2018), it did not built more than one FastMap heuristic at a time. This is likely because the current FastMap pivot selection scheme is likely to produce the same pivots if run a second time. Thus, if we wish to use multiple FastMap embeddings, we need to design a new SelectPivots method which takes into account the previous embedding and/or heuristics.

One possible approach is to use subset selection methods (Rayner, Sturtevant, and Bowling 2013; Levi et al. 2016) to find good pivots. However, in our experiments these approaches required significant computational overhead to significantly improve over faster greedy selection schemes. Thus, we focus on linear-time greedy selection schemes to keep the overhead of all methods equal.

The ED pivot selection scheme is built based on the current residual costs in the graph. We propose a new method, Heuristic Error (HE), which selects pivots according to both the embedding distance and heuristic error.

The HE pivot selection method begins by selecting a random point in the input graph. From that point, we measure the distance to all other vertices in the input graph as with ED. From these, we select the first pivot as the vertex that maximizes the sum of the embedding distance and twice the heuristic error, where the heuristic error is the difference between the true distance in the embedding and the heuristic distance in the full graph. This function can, for instance, take into account any error in the default heuristics. But, we only used this approach to select the first two pivots; remaining pivots are selected using ED, as using different initial pivots significantly changes the residual graph.

Note that other variants are possible, including selecting all pivots using the HE scheme, and using a different equation to balance heuristic error and distance. Empirically, our settings gave the most consistent improvement, so all experiments will use these setting. Pseudo-code for the HE approach, including simple algebraic re-arrangement in lines 5 and 6, is found in Algorithm 7.

Experimental Results

To evaluate the ideas in this paper, we experiment with optimal pathfinding using A* with the FastMap embedding as a heuristic, and then validate the differences in runtime performance between the approaches used in the paper. We then experiment in the Multi-Agent Meeting problem using the MM* algorithm. In this context an L^1 embedding is able to efficiently find potential meeting points and get a lower-bound on the cost for all agents to reach that point. Our experiments illustrate that (1) FastMap on its own gives poor heuristics, (2) adding DHs to FastMap improves performance on both problems, and that (3) the overall performance depends on the nature of the problem set. Our implementation is in C++ in the HOG2 repository[1].

Baseline Performance We begin by evaluating the performance A* using previously published heuristics on a variety of grid maps from the MovingAI pathfinding repository (Sturtevant 2012) and the standard associated benchmark problems. We solve problems on maps from the games Dragon Age: Origins (DAO) and Starcraft (SC1), as well as artificial maps including random maps, room maps, and maze maps. The number of problems used in each set is indicated next to the name of the map set. As all approaches evaluated have approximately the same pre-computation time and runtime overhead, we will use node expansions to compare these approaches.

As previously illustrated in Figure 2, we studied the full distribution of results on individual maps and map sets over-

[1] https://github.com/nathansttt/hog2/tree/PDB-refactor/papers/FMDH

Heuristics	Pivot	DAO (157,905)			SC1 (198,230)			Random (146,220)			Rooms (78,840)		
		Md	Mean	C	Md	Mean	C	Md	Mean	C	Md	Mean	C
DH10	FAR	561	1,581	13	2,219	7,453	58	2,984	**5,345**	32	4,098	**7,700**	66
FM10	ED	516	2,172	21	4,199	18,548	144	5,917	12,455	84	9,471	19,271	165
FM9+DH	ED	381	1,034	11	1,576	15,036	136	4,053	11,069	83	5,783	16,778	163
FM9+DH	HE	370	1,038	12	1,460	13,961	129	4,049	11,089	83	6,219	16,960	161
max[DH5,FM5]	ED	467	1,290	11	2,277	9,029	74	3,452	6,118	36	4,467	8,809	75
max[DH5,FM4+DH]	ED	406	1,036	9	1,511	7,834	70	3,139	5,773	35	3,632	8,030	73
max[FM4+DH,DH5]	HE	399	**1,002**	8	1,441	**7,227**	64	3,079	5,717	34	3,466	**7,687**	70
max[DH5,FM4+DH]	HE	400	**995**	8	1,446	7,399	66	3,025	5,628	34	3,355	**7,598**	70
DH24	FAR	518	1,483	13	1,520	5,678	46	2,387	4,469	28	3,077	5,882	53
FM24	ED	401	1,471	16	1,983	16,120	138	5,257	11,923	83	8,059	18,130	163
FM23+DH	ED	349	887	10	1,198	14,783	135	3,562	10,838	83	5,437	16,471	163
max[DH12+FM12]	ED	355	772	6	916	4,758	46	2,191	4,461	29	2,658	5,930	57
max[DH12,FM11+DH]	ED	327	**646**	5	784	4,314	45	2,011	**4,256**	29	2,152	5,513	57
max[DH12,FM11+DH]	HE	326	**641**	5	773	**4,240**	44	1,995	**4,229**	29	2,127	5,222	53
max[8xFM2+DH]	ED	641	2,036	19	4,924	18,415	145	5,376	11,962	83	7,654	18,200	166
max[8xFM2+DH]	HE	473	1,518	16	1,694	11,337	117	2,638	5,533	37	1,900	**5,139**	56

Table 1: Results reporting expansions by A* including the median node, mean, and the 95% confidence interval on the mean.

Heuristics	Pivot	Mazes (586,370)		
		Md	Mean	C
DH10	FAR	6,554	10,090	27
FM10	ED	8,898	15,311	44
FM9+DH	ED	4,668	**7,067**	19
FM9+DH	HE	4,679	**7,084**	19
max[DH5,FM5]	ED	6,630	10,036	26
max[DH5,FM4+DH]	ED	6,270	9,477	24
max[FM4+DH,DH5]	HE	6,252	9,449	24
max[DH5,FM4+DH]	HE	6,231	9,422	24
DH24	FAR	4,806	7,493	21
FM24	ED	5,557	10,512	32
FM23+DH	ED	3,348	**4,873**	13
max[DH12+FM12]	ED	4,377	6,622	17
max[DH12,FM11+DH]	ED	4,449	5,409	23
max[DH12,FM11+DH]	HE	4,164	6,179	16
max[8xFM2+DH]	ED	10,122	15,047	39
max[8xFM2+DH]	HE	7,313	11,712	32

Table 2: Results on mazes extending Table 1.

all, and summarize these by presenting the median number of expansions, the mean number of expansions, and the 95% confidence interval on the mean. We refer to the FastMap heuristic as FM. FMx or FM[x-1]+DH refers to an embedding with x dimensions. When the maximum of multiple embeddings is used, the order of arguments to max indicates the order in which the embeddings are built. We present results with either 10 dimensions or 24 dimension, as these best illustrate the general trends.

The initial results are in the top two rows of Table 1 and 2. The first point, when just comparing previously published approaches, is to notice the difference between the median and mean performance with FastMap. For instance, on the DAO problems FastMap performs 1.4x more expansions on average than DHs, but the median expansions are nearly identical. The problem sets used for testing are explicitly built to include difficult problems, which DHs are better able to solve than FastMap. We also repeated these experiments on purely random problems; FastMap's performance improved relative to DH. Thus, we observed a small tail of very hard problems are not handled well on FastMap.

Algorithm	Median	Mean	95%
FM	271	242	40
FMDH	263	231	37
DH	260	233	37

Table 3: Average total time in microseconds to compute heuristics on 75 SC1 maps.

The second point to notice is that FastMap performs significantly worse than DHs across most problem sets, except when considering the median expansions in DAO. The DAO maps are much more linear in nature, as they come from a single-player game, while the Starcraft maps are designed for two or more players. This can be seen in quantitative analysis of the map sets measuring the underlying dimension of the maps (Sturtevant 2012), which showed that DAO and maze maps have the lowest dimension, which will be reflected by the improved performance of FM+DH.

A Single 10-Dimension Embedding Next we look at the overall results where a single 10-dimension embedding was built. These results are in the top of the table, and include the addition of a single DH embedding (FM9+DH), as well as the HE pivot selection scheme.

Across all maps, FM9+DH improves the performance of FM10, but the gains are particularly large in the Maze and DAO domains – over a 2x improvement in average expansions on both. In these domains FM9+DH outperforms even

	MD		FM10		FM9+DH	
Map Set	Md	Mean	Md	Mean	Md	Mean
DAO	46,904	100,961	6,476	13,626	**4,487**	**8,841**
SC1	451,753	585,071	258,528	468,700	**240,690**	**446,307**
City	**36,904**	**45,598**	95,872	98,518	94,345	96,985
Random	**831**	**1,058**	5,698	6,639	5,622	6,583
Room	4,725	5,242	2,425	3,166	**2,086**	**2,766**
Mazes	52,553	43,735	3,436	3,497	**1,775**	**1,864**

Table 4: Results on different map sets reporting the median and mean node expansions by MM* on MAM problems.

the DH10 approach by a statistically significant margin. This is a surprisingly large impact for what amounts to a very small change in the implementation. The HE pivot selection scheme does not have a significant impact on performance in this context.

Two 5-Dimension Embeddings Next we look at the results using two 5-dimension embeddings, and taking the max of each. The baseline approach is to take the max of a single 5-dimension FastMap embedding and a 5-dimension DH embedding. Replacing the FM5 embedding with a FM4+DH embedding improves the performance on all maps. This gives the best mean expansions on DAO and room maps. The HE pivot selection scheme significantly improves performance on room maps over ED.

24 Dimensions Because FastMap has diminishing returns, it may be less useful when using an embedding with more dimensions. Thus, we also experimented with 24-dimensional embeddings to test this. These results are in the bottom of Table 1 and 2. We see the same trend that FastMap performs poorly across most maps. In maze maps, FM23+DH has the best performance, improving the average nodes expanded by over a factor of two. This is likely because mazes have a tree structure; each embedding is able to capture two branches of this tree. Thus, a single-high dimensional embedding is very effective, particularly because the heuristic is additive.

The other interesting result in the remaining data is that the maximum of 8 3-dimensional embeddings surpassed the performance of DHs on room maps, with a significantly better median, meaning there is still a tail of hard problems. But, this is only with the HE placement scheme; the ED placement scheme performs very poorly.

Heuristic Lookup Speed To isolate details such as the open list implementation and size, we measure the relative cost of heuristic lookups independent of the surrounding search. Our original implementation used somewhat inefficient data structures for storing the heuristic values, which resulted in unstable timing results, likely due to issues such as cache locality. Thus, we re-implemented the algorithms from scratch storing each state's embedding locations adjacently in memory to improve cache locality. We then looked up the heuristic value of every start/goal pair in the benchmark problem sets, and recorded the total time to do this on each map. Then we report the average time per map in microseconds when running on an Apple M1 Pro with 16 GB of RAM on macOS Monterey version 12.6.1 and with the code compiled with Apple clang version 14.0.0. As shown in Table 3, over the 75 maps in the Starcraft map set, all of the mean results fall within the 95% confidence intervals, showing no statistical difference in runtime between the various heuristics.

FastMap in MM*

In the Multi-Agent Meeting (MAM) problem, agents distributed around a map must find the best meeting point. The current state of the art algorithm for MAM is the MM* algorithm (Atzmon et al. 2020). In this work the embedding is used directly to reason about where the meeting point might be. It depends on the fact that the median of each agents' location in an L^1 embedding is the closest point to all agents.

We ran these experiments using the code from the original paper. A larger set of problems is used than in the original work; overall we ran over 338k problems across all maps tested. We created instances for five agents to meet by taking each set of five consecutive start locations from the standard benchmark problems and using them as starting locations for the agents. We compared three L^1 embeddings: Manhattan Distance (MD) in the original maps, the original FastMap, and our improved FM+DH approach. The cost metric optimized was Sum of Costs, and the FastMap embeddings used 10 dimensions.

Experimental results are in Table 4. In these results we find that the MD embedding outperforms FastMap on relatively open maps, such as random maps, where the MD embedding will be more accurate, but FastMap approaches are better overall. The DH approach cannot be used in this problem without solving a relatively expensive LP, so it was not considered. As we saw in single-agent pathfinding, FM+DH always outperforms FM. The data suggests that the better the performance of FastMap over MD, the better the gain of FM+DH over FM.

Conclusions and Future Work

This paper has revisited the FastMap L^1 heuristic and embeddings, showing that their performance isn't as strong as originally suggested. But, combining the DH embedding into the last dimension of a FastMap embedding can significantly improve performance both on A* and MM* pathfinding tasks. Future work will explore additional embedding functions, and integration with other approaches, such as JPS and its variants (Harabor and Grastien 2012, 2014; Sturtevant and Rabin 2016).

Acknowledgements

This work was funded by the Canada CIFAR AI Chairs Program. We acknowledge the support of the Natural Sciences and Engineering Research Council of Canada (NSERC).

References

Atzmon, D.; Li, J.; Felner, A.; Nachmani, E.; Shperberg, S.; Sturtevant, N.; and Koenig, S. 2020. Multi-Directional Heuristic Search. In *International Joint Conference on Artificial Intelligence (IJCAI)*.

Chen, W.; Weinberger, K.; and Chen, Y. 2013. Maximum Variance Correction with Application to A* Search. In Dasgupta, S.; and McAllester, D., eds., *Proceedings of the 30th International Conference on Machine Learning*, volume 28 of *Proceedings of Machine Learning Research*, 302–310. Atlanta, Georgia, USA: PMLR.

Cohen, L.; Uras, T.; Jahangiri, S.; Arunasalam, A.; Koenig, S.; and Kumar, T. K. S. 2018. The FastMap Algorithm for Shortest Path Computations. In *Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence, IJCAI-18*, 1427–1433. International Joint Conferences on Artificial Intelligence Organization.

Culberson, J. C.; and Schaeffer, J. 1998. Pattern databases. *Computational Intelligence*, 14(3): 318–334.

Faloutsos, C.; and Lin, K.-I. 1995. FastMap: A fast algorithm for indexing, data-mining and visualization of traditional and multimedia datasets. In *Proceedings of the 1995 ACM SIGMOD international conference on Management of data*, 163–174.

Felner, A.; Korf, R. E.; and Hanan, S. 2004. Additive pattern database heuristics. *Journal of Artificial Intelligence Research*, 22: 279–318.

Felner, A.; Sturtevant, N.; and Schaeffer, J. 2009. Abstraction-based heuristics with true distance computations. *Symposium on Abstraction, Reformulation and Approximation*, 9.

Goldberg, A. V.; and Harrelson, C. 2005a. Computing the Shortest Path: A Search Meets Graph Theory. In *Proceedings of the Sixteenth Annual ACM-SIAM Symposium on Discrete Algorithms*, 156–165.

Goldberg, A. V.; and Harrelson, C. 2005b. Computing the shortest path: A search meets graph theory. In *SODA*, volume 5, 156–165. Citeseer.

Goldberg, A. V.; Kaplan, H.; and Werneck, R. F. 2007. Better landmarks within reach. In *International Workshop on Experimental and Efficient Algorithms*, 38–51. Springer.

Gopalakrishnan, S.; Cohen, L.; Koenig, S.; and Kumar, T. 2020. Embedding directed graphs in potential fields using fastmap-d. In *International Symposium on Combinatorial Search*, volume 11.

Harabor, D.; and Grastien, A. 2012. The JPS pathfinding system. In *International Symposium on Combinatorial Search*, volume 3.

Harabor, D.; and Grastien, A. 2014. Improving jump point search. In *Proceedings of the International Conference on Automated Planning and Scheduling*, volume 24, 128–135.

Helmert, M.; Haslum, P.; Hoffmann, J.; and Nissim, R. 2014. Merge-and-shrink abstraction: A method for generating lower bounds in factored state spaces. *Journal of the ACM (JACM)*, 61(3): 1–63.

Hu, Y.; Harabor, D.; Qin, L.; and Yin, Q. 2021. Regarding Goal Bounding and Jump Point Search. *Journal of Artificial Intelligence Research*, 70: 631–681.

Levi, L. H.; Franco, S.; Abisrror, M.; Barley, M.; Zilles, S.; and Holte, R. 2016. Heuristic subset selection in classical planning. In *Proceedings of the Twenty-Fifth International Joint Conference on Artificial Intelligence (IJCAI-16)*, 3185–3195. AAAI Press/IJCAI.

Li, A.; Stuckey, P.; Koenig, S.; and Kumar, T. K. S. 2022. A FastMap-Based Algorithm for Block Modeling. In *Integration of Constraint Programming, Artificial Intelligence, and Operations Research*, 232–248. Springer International Publishing. ISBN 978-3-031-08011-1.

Li, J.; Felner, A.; Koenig, S.; and Kumar, T. S. 2019. Using fastmap to solve graph problems in a euclidean space. In *Proceedings of the international conference on automated planning and scheduling*, volume 29, 273–278.

Rabin, S.; and Sturtevant, N. R. 2016. Combining Bounding Boxes and JPS to Prune Grid Pathfinding. In *AAAI Conference on Artificial Intelligence*.

Rayner, C.; Bowling, M.; and Sturtevant, N. 2011. Euclidean Heuristic Optimization. In *AAAI Conference on Artificial Intelligence*, 81–86.

Rayner, C.; Sturtevant, N.; and Bowling, M. 2013. Subset Selection of Search Heuristics. *International Joint Conference on Artificial Intelligence (IJCAI)*, 637–643.

Seipp, J.; Keller, T.; and Helmert, M. 2020. Saturated cost partitioning for optimal classical planning. *Journal of Artificial Intelligence Research*, 67: 129–167.

Sievers, S.; and Helmert, M. 2021. Merge-and-Shrink: A Compositional Theory of Transformations of Factored Transition Systems. *Journal of Artificial Intelligence Research*, 71: 781–883.

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.; Felner, A.; Barer, M.; Schaeffer, J.; and Burch, N. 2009. Memory-based heuristics for explicit state spaces. *International Joint Conference on Artificial Intelligence (IJCAI)*, 609–614.

Sturtevant, N. R.; and Rabin, S. 2016. Canonical Orderings on Grids. *International Joint Conference on Artificial Intelligence (IJCAI)*, 683–689.

Yang, F.; Culberson, J.; Holte, R.; Zahavi, U.; and Felner, A. 2008. A general theory of additive state space abstractions. *Journal of Artificial Intelligence Research*, 32: 631–662.