Improved FastMap

by

Reza Mashayekhi

A thesis submitted in partial fulfillment of the requirements for the degree of

Master of Science

Department of Computing Science

University of Alberta

Abstract

Pathfinding has been an interesting research area throughout the years. Heuristic search algorithms are used to find a path with the minimum length between a start and a goal in a graph, which has applications in GPS navigation and video games. There are different ways to create a heuristic for these search algorithms. Using embeddings is one of the ways to create a heuristic. In this method, an algorithm builds an embedding space for a given graph. Then, distances in the embedding space can be used to compute the heuristic between any two arbitrary vertices in the graph. Embeddings have other applications, like finding the midpoint between states, but here we focus on their use of heuristics. Cohen et al. (2017) introduced FastMap embedding, which provides a consistent heuristic.

This thesis shows that the FastMap heuristic is not as strong as it was originally shown to be. While the median performance is strong, the average is not. We then analyze the FastMap approach. We show that FastMap is an additive heuristic. Also, we generalize the FastMap embedding by generalizing its embedding function and pivot selection methods. We introduce several new embedding functions and pivot selection methods. We show that the new pivot selection methods enable us to use multiple FastMap embeddings together, which was not effective before. Finally, we evaluate the newly created heuristics and show that using differential heuristics and the Heuristic Error pivot selection method improves the FastMap heuristic.

Acknowledgements

Special thanks to my supervisor, professor Nathan Sturtevant who guided and helped me through this research. I also want to thank:

- The other committee members, Jonathan Schaeffer and Levi Lelis, for evaluating my thesis and giving suggestions for improving the thesis.
- My family for always supporting me.
- My writing advisor, Antonie Bodley, who helped to improve my writing for this thesis.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

A person can find their way home quickly if they already know a path home. However, for someone who does not know a path to a destination, it is hard to find the way. It can get even harder if they do not even know where the destination is and can only find the place of the destination by arriving there and checking it. To make this more clear, imagine someone is left in a city, given only a picture of a house, with the goal of finding the house. Thus, they have to search the whole city and compare every house to the picture they have until they find the house. Now, in order to help them, we give them a tool that beeps more quickly when they move towards the house. This tool gives them an estimate (or heuristic) of their distance to the house so they can find the house faster.

If we want to build the beeping tool, we notice that it has to know its distance to the house in order to beep with respect to the distance. One way to calculate an estimate of distance is by using GPS coordinates and considering the Euclidean distance between two points as the measure of distance. The Euclidean distance between two points in 2D is the length of the straight line connecting them. The length of the black line between the two red points in Figure 1.1 is their Euclidean distance. Then, the beeping tool beeps with respect to its Euclidean distance to the goal house (i.e., it beeps more quickly if the Euclidean distance to the goal house gets shorter on the map).

Although the Euclidean distance can be used in the beeping tool as the measure of closeness, it may not be accurate enough in some cases. For ex-

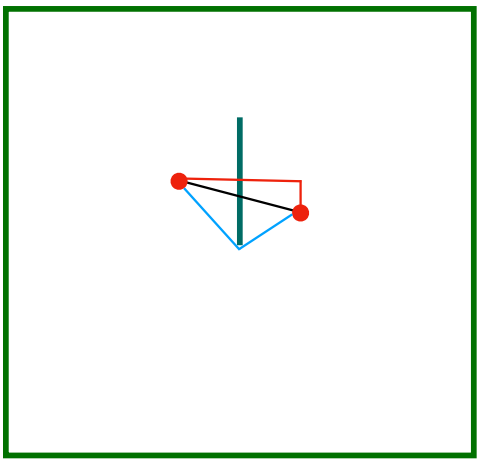
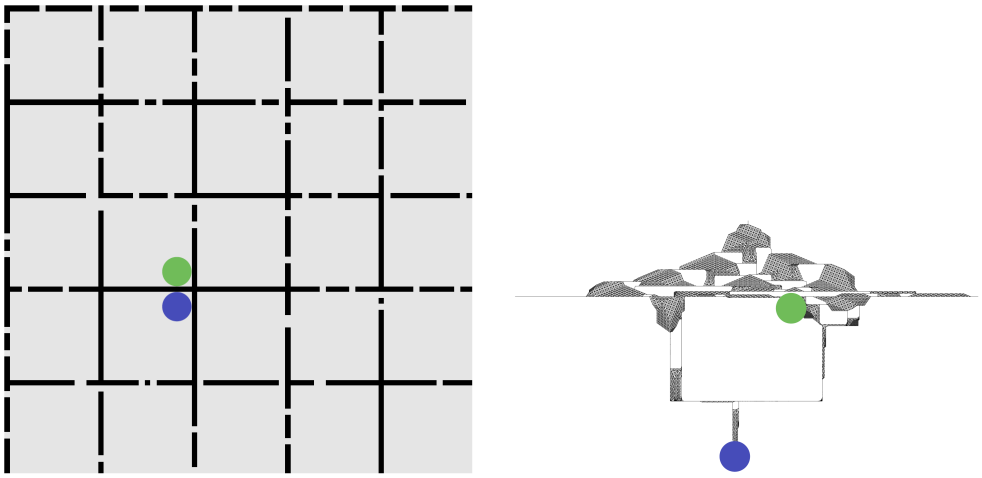


Figure 1.1: This image shows the shortest path length (the length of the blue path), Euclidean distance (the length of the black line), and the L_1 distance (the length of the red path) between the two red points.

ample, the Euclidean distance between the two points in Figure 1.2(a) is not a good estimate of the shortest path length between them, which in this case needs to go around the walls. The measure of distance in the beeping tool needs to be a better estimate of the shortest path length between the two points so the person can find the goal house sooner. For instance, we can create a new map like Figure 1.2(b), representing the map 1.2(a), in which the Euclidean distance between these two points is a better estimate of the shortest path length between the two points in Figure 1.2(a). Therefore, if we use Euclidean distance on this new map inside the beeping tool, the person can likely reach the destination faster.

This real-life problem and its solution are similar to computer science pathfinding problems. In computer science pathfinding problems, there is an area to search, a start, and a goal. The problem is to find a path from the start to the goal. If the computer does not know a path, it can use an algorithm to find the path. Search areas can be represented as a graph with vertices connected by edges. There are many different pathfinding algorithms [6, 11, 18], and in this thesis, our focus is on the class that finds a path with the minimum length. Dijkstra's algorithm, one of the early algorithms that find the shortest path, was published in 1959 [6]. One of the challenges since then has been finding the shortest paths in a shorter amount of time. In 1968,



(a) (b)

Figure 1.2: (a) A sample map and (b) a 2-dimensional re-embedding of the map.

the A^* algorithm [11] was introduced, which uses an admissible heuristic to guide the search for finding the shortest path between the start and goal more quickly. The heuristic between two vertices is an estimate of the path length between them. The more accurate the heuristic is, a path will likely be found sooner. Using an embedding is one way to build a heuristic for a given graph. Embeddings [21] are built in a preprocessing step on the graph in which all vertices would be assigned coordinates. Then, during the search, these coordinates assigned in the embedding step can be used for computing a distance heuristic. For instance, the map in Figure 1.2 (b) is an embedding of the map in Figure 1.2 (a). Each vertex of the map in Figure 1.2 (a) is assigned new 2d coordinates, which are used to display the embedding as a 2d graph in Figure 1.2 (b). There are many methods for building embeddings. This example, Figure 1.2 (b), is built with the FastMap algorithm [2]. FastMap [2] uses a new way to embed vertices.

In this thesis, we analyze FastMap and improve the embedding it provides. The main contributions are as follows:

- We show that FastMap is an additive heuristic. While many additive heuristics statically partition costs, FastMap dynamically partitions them as each dimension of the embedding is built.

- FastMap is generalized, and its different optimizations are explored. We explored how to optimize using separate FastMap heuristics together or how to improve the FastMap heuristic itself.

- Among all the explored optimizations, we found that using a differential heuristic at the last dimension of FastMap and using heuristic error for pivot selection improves the FastMap heuristic. Heuristic error enabled the effective use of multiple FastMap embeddings.

- Extensive results on many maps reveal that FastMap is not as strong as previously discussed. It was shown that FastMap has a strong median performance; however, we show that it does not have a strong average performance.

Chapter 2

Background

As mentioned in Chapter 1, a class of search algorithms called heuristic search algorithms, like the A^* algorithm [11], as their name suggests, use heuristics to make their search faster.

This chapter describes related work to the FastMap heuristic and gives a formal definition of the problem addressed in this thesis. Then, it gives preliminary background, including what a heuristic is, and describes the FastMap heuristic itself.

2.1 Related Work

Heuristics that do not have any preprocessing step are not accurate in many cases, such as Euclidean distance in Figure 1.2 (a). On the other hand, a large group of heuristics require a preprocessing step to become more accurate [2, 23, 4]. In the preprocessing step, the state space can be analyzed, and useful information can be stored in limited memory. Then, the stored information is used to return a heuristic between any pair of states. In this section, only heuristics with preprocessing are described.

Two of the domains that heuristics can be applied to are (1) exponential domains, in which the state space grows exponentially with respect to the solution cost, and (2) polynomial domains, in which the state space grows polynomially with respect to the solution cost [8]. Pattern Databases (PDBs) [4] are a dominant heuristic approach in exponential domains (like the Sliding Tile puzzle). PDBs, as an abstraction-based approach, create multiple ab-

stract spaces from the original state space. Then, in the additive PDBs [7], the heuristic in each abstract space is added together to compute the heuristic in the original state space. However, abstraction-based approaches are generally not effective in polynomial domains [8]. In polynomial domains (like grid pathfinding), usually, the heuristics that rely on true distances perform better, like ALT (A^*, Landmarks, and the Triangle Inequality) [9], differential heuristics [23], and optimal Euclidean embeddings [21].

True-distance based heuristics [9, 23, 21] use embeddings. ALT and differential heuristics embed the state space into a k-dimensional state space and use the L_∞ norm to compute the heuristic. Optimal Euclidean embeddings embed the state space into a high dimensional state space, then based on the optimization, they keep only k of the dimensions and use the L_2 norm to compute the heuristic. FastMap [2] is also based on true distances. FastMap embeds the original state space into a new k-dimensional state space and uses the L_1 norm to compute the heuristic. There are two other variants of FastMap [14, 12]. Li et al.'s (2019) work [14] does a euclidean embedding and uses the L_2 norm for computing the heuristic, which can also be used in other areas like block modeling [1]. Nonetheless, since it is useful for suboptimal solutions in heuristics search, we do not analyze it in this thesis. Another version of FastMap [12] reduces the preprocessing time of the original FastMap but has the same runtime performance when it is used as the heuristic of A^*.

There are other approaches that do not use embeddings, like bounding boxes [19] and reach [10]. They prune unnecessary states during the search. However, since they need more computational time for the preprocessing, they are not compared with FastMap. Here in this thesis, consistent heuristics using embeddings with the same precomputation time as FastMap are compared.

2.2 Problem Definition

The tuple $P = \{G, C, h_{in}\}$ defines the problem. The state space is represented as the undirected graph $G = \{V, E\}$ where V and E are the sets of vertices and edges, respectively. $C : E \to \mathbb{R}$ is the edge cost function for the graph

G. h_{in} is the given initial heuristic. A heuristic is a function that gives an estimate of the shortest path length between two states, i.e., $h : V \times V \rightarrow \mathbb{R}$.

The output of the problem is a metric embedding (described in Section 2.3) of the graph g. The heuristic h built by the embedding and h_{in} is used in the A^* algorithm (described in Section 2.3) to find the shortest paths between a set of $(start, goal)$ pairs in the graph G. h should be either better than the existing heuristics in some applications or different from them. A heuristic is better than previous ones if it results in finding the shortest paths faster or with less memory usage.

2.3 Preliminary Background

Here we describe the notations used in this thesis, including the FastMap heuristic.

2.3.1 Heuristic Background

Each edge $e \in E$ that connects two vertices $v_i, v_j \in V$ is shown as $e = (v_i, v_j)$.

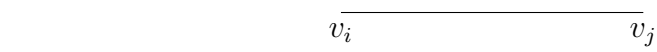


Figure 2.1: An edge $e = (v_i, v_j)$

Also, the edge cost of $e = (u, v) \in E$ is $C(e)$ or $C(u, v)$, where C is the edge cost function.

The neighbors of a vertex v are the vertices connected by an edge to v.

Definition 2.3.1 (Neighbors of a vertex)**.** The neighbors of a vertex v are the members of $neighbors(v) = \{u|(v, u) \in E\}$ set.

In search algorithms, we have to find a path from the start to the goal. Below are the definitions of walk (a more general form of a path), path, and path length.

Definition 2.3.2 (Walk)**.** A walk from a vertex $u_1 \in V$ to a vertex $u_k \in V$ in graph G is a list of vertices $[u_1, u_2, ..., u_k]$ where $u_i \in V, (u_i, u_{i+1}) \in E, 1 \leq i \leq$

$k-1$. The walk w can also be represented as a list of edges $w = [e_1, e_2, ..., e_{k-1}]$ where $e_i = (u_i, u_{i+1}), 1 \leq i \leq k-1$.

Definition 2.3.3 (Path)**.** A walk without repeated vertices is called a path.

Since each edge has a cost, a path, which is a set of edges, can have a cost too.

Definition 2.3.4 (Cost (length) of a path)**.** The cost of a path $p = \{e_1, e_2, ..., e_k\}$ in graph G is computed as $C(p) = \sum_{i=1}^{k} C(e_i)$, where C is the edge cost function of G.

The shortest path length between two vertices u and v is shown with $C^*(u, v)$.

A useful property is triangle inequality in a graph, which will be used in many proofs of this thesis.

Definition 2.3.5 (Triangle inequality)**.** For any three vertices $u, v, w \in V$

$$C^*(u, v) \leq C^*(u, w) + C^*(w, v) \tag{2.1}$$

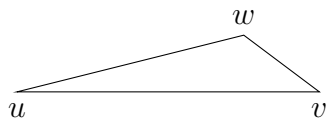


Figure 2.2: Triangle inequality.

Remark. *Directly driven from lemma 2.3.5, for any edge $e = (u, v) \in E$ with the cost $C(e)$ and any arbitrary vertex $a \in V$*

$$C^*(a, u) \leq C^*(a, v) + C(e) \tag{2.2}$$

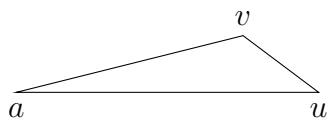


Figure 2.3: Triangle inequality for an edge.

Now that we know what the state space and path cost are, we can define the heuristic.

Definition 2.3.6 (Heuristic)**.** A heuristic function for the graph G can be defined as $h\colon V \times V \to \mathbb{R}$, where for any two $u, v \in V$, $h(u, v)$ is an estimate of the length of the path between u and v. Also, $h(e)$, for making notations shorter, is used to denote the heuristic between two vertices of an edge; i.e., $h(e) := h(u, v), e = (u, v)$.

Heuristics can have different properties, which makes them useful for different applications. Two of the most common properties are admissibility and consistency, each of which can be either global or local.

Definition 2.3.7 (admissible heuristic)**.** A heuristic is globally admissible if

$$\forall u, v \in V \quad h(u, v) \leq C(u, v) \tag{2.3}$$

Definition 2.3.8 (Locally consistent heuristic)**.** In undirected graphs, a heuristic is locally consistent if

$$\forall g \in V, (u, v) \in E \quad |h(u, g) - h(v, g)| \leq C(u, v) \tag{2.4}$$

Definition 2.3.9 (Globally consistent heuristic)**.** In undirected graphs, a heuristic is globally consistent if

$$\forall g, u, v \in V \quad |h(u, g) - h(v, g)| \leq C^*(u, v) \tag{2.5}$$

One application of consistent (or admissible) heuristics is in the A^* algorithm. If the heuristic is consistent (or admissible), the A^* algorithm using that heuristic is guaranteed to find the optimal path between any two vertices [11].

Assume we have a set of heuristics, and we want to use these heuristics within the given limited memory to optimize a heuristic for the graph. Subset selection is one of the methods that can be used for this optimization.

The optimal form of subset selection [20] chooses a subset of heuristics (with size d) among the n existing heuristics that optimize the total heuristic between

all pairs of a sample. The sample can be a random subset of the graph vertices. For finding the optimal subset of heuristics, all combinations of the existing heuristics with size d should be checked ($min(O(n^d), O(n^{n-d}))$); therefore, it will be time-consuming. However, there is an approximate solution [20] that greedily chooses one heuristic at a time. The chosen heuristic should maximize the total heuristic between all pairs of the sample vertices compared to the heuristic captured by the previous steps. This thesis will use subset selection for optimizing multiple FastMap embeddings in Chapter 3.

Definition 2.3.10 (Greedy subset selection)**.** Assume H is a set of existing heuristics with size n. $P(H)_d$ would be the set of all possible subsets of H with size d i.e.,

$$P(H)_d = \{A | A \subseteq H, |A| = n\}$$

In order to select a member of $P(H)_d$ that will likely give us a more accurate heuristic than other members, we take a sample of vertices in the state space called S. Then, we find the d heuristics, one at a time, in a way that maximizes the heuristic between all pairs of the sample. i.e.

$$subH_{i+1} = subH_i \cup \underset{h \in H - subH_i}{\operatorname{argmax}} \left(\sum_{u,v \in S} \max(\max_{h_0 \in subH_i} h_0(u,v), h) \right)$$

Where $0 \leq i \leq d-1$, $SubH_0 = \{\}$ and $SubH_d$ would be the selected subset of heuristics with size d. The heuristic built by subset selection is [20]:

$$h(v_1, v_2) = \max_{h_i \in SubH_d} (h_i(v_1, v_2))$$

2.3.2 Embeddings Background

Different distance measures are used in heuristic computations. Here we describe L_p distances, which are used in this thesis.

Definition 2.3.11 (L_p distance)**.** L_p distance of a vector $U = (u_1, u_2, ..., u_n)$ to a vector $V = (v_1, v_2, ..., v_n)$ is $(\sum_{i=1}^{n} |v_i - u_i|^p)^{\frac{1}{p}}$ [15].

Three common L_p distances are derived below.

Definition 2.3.12 (L_∞ distance)**.** L_∞ distance of a vector $U = (u_1, u_2, ..., u_n)$ to a vector $V = (v_1, v_2, ..., v_n)$ is the maximum element of $(|u_1 - v_1|, |u_2 - v_2|, ..., |u_n - v_n|)$.

Definition 2.3.13 (L_1 distance)**.** L_1 distance of a vector $U = (u_1, u_2, ..., u_n)$ to a vector $V = (v_1, v_2, ..., v_n)$ is $|u_1 - v_1| + |u_2 - v_2| + ... + |u_n - v_n|$.

Definition 2.3.14 (L_2 (Euclidean) distance)**.** L_2 distance of a vector $U = (u_1, u_2, ..., u_n)$ to a vector $V = (v_1, v_2, ..., v_n)$ is

$$\sqrt{(u_1 - v_1)^2 + (u_2 - v_2)^2 + ... + (u_n - v_n)^2}$$

Since FastMap embedding is a pseudo-metric embedding (as shown in Lemma 2.3.2), and pseudo-metric embeddings have a pseudo-metric space, here they are defined.

Definition 2.3.15 (Metric space)**.** A pair (V, d), where V is the set of coordinates (coordinate space) and d is a function $d : VV :\rightarrow \mathbb{R}^{\geq 0}$ is a metric space if:

- $d(u, v) = 0 \iff u = v$
- $d(u, v) = d(v, u)$
- $d(u, v) + d(v, w) \geq d(u, w)$

where $u, v, w \in V$ [3].

Definition 2.3.16 (Pseudo-Metric space)**.** A metric space (V, d) that can have $d(u, v) = 0$ where $u! = v$ [3].

Embeddings are a way to build heuristics for a given graph. In the embedding process, each vertex is mapped to a coordinate in coordinate space. Then, distances in this embedded space (coordinate space) can be used to compute the heuristic.

Definition 2.3.17 (Metric embedding)**.** In this thesis, an embedding is a 1-1 mapping function that maps a graph $G = \{V, E\}$ into a metric space (U, d); i.e.,

$$f : V \rightarrow U$$

Two of the heuristics built by using embeddings are the differential heuristic [9, 16] and FastMap [2], which are further explored in this thesis.

Definition 2.3.18 (differential heuristic (DH_P))**.** For a given subset of vertices $P = \{p_1, p_2, ..., p_n\} \subseteq V$, where each member is called a pivot, single source shortest paths from all pivots are computed separately. The computed single source shortest paths are stored as a differential embedding. i.e.,

$$DE_P(v) = (C^*(p_1, v), C^*(p_2, v), ..., C^*(p_n, v))$$

Then, the differential heuristic between two vertices in the graph is computed as the l_∞ distance in the embedding space ($DH_P(V)$) [9], i.e.,

$$DH_P(v_1, v_2) = \max_{p_i \in P} |C^*(p_i, v_1) - (C^*(p_i, v_2)|$$

The embedding of the differential heuristic is a pseudo-metric embedding, as shown in the lemma below.

Lemma 2.3.1. *A differential heuristic is a pseudo-metric embedding.*

Proof. The differential embedding space ($DH_P(V)$) and the differential heuristic DH_P are respectively coordinate space and the distance measure of the pseudo-metric space. Here we show that the three conditions of a pseudo-metric embedding are met.

The first condition requires checking that self distances are 0:

$$DH_P(u, u) = |d(p, u) - d(p, u)| = 0 \tag{2.6}$$

where $p \in P$ is the pivot that is furthest from u.

The second condition requires distances to be symmetric:

$$DH_P(u, v) = |d(p, u) - d(p, v)| = |d(p, v) - d(p, u)| = DH_P(v, u) \tag{2.7}$$

where $p \in P$ is the pivot that has the maximum $DH_p(u, v)$.

The third condition requires the triangle inequality to hold:

$$DH_P(u, w) = |d(p, u) - d(p, w)| \tag{2.8}$$

$$\leq |d(p,u) - d(p,v)| + |d(p,v) - d(p,w)| \tag{2.9}$$

$$\leq DH_P(u,v) + DH_P(v,w) \tag{2.10}$$

□

A differential heuristic embedding is shown in Figure 2.4.

FastMap embedding, which is also a pseudo-metric embedding, is described below.

Definition 2.3.19 ($FastMap^n$ (FM^n))**.** FM^n is an n-dimensional heuristic created by adding n 1-dimensional FastMap heuristics together [2]. In each dimension, an embedding is done. Based on that embedding, a 1-dimensional FastMap heuristic is created. Then, the edge costs will get updated by subtracting the captured heuristic from that edge. Finally, by adding each dimension's FastMap heuristic together, the FM^n heuristic is built. Below is algorithm 1 for doing the embeddings $(f_1, f_2, ..., f_n)$ at each dimension.

In more detail, at each dimension, the two furthest pivots $a_i, b_i \in V$ are selected. Then, all vertices are embedded with the formula at line 6. FastMap heuristic of each dimension is $FM_i(u,v) = |f_i(u) - f_i(v)|$ where f_i is the embedding of dimension i (here, the FastMap embedding function is used, but it can be other functions, which will be discussed in Chapter 3). Edge costs are changed by deducting the captured heuristic between the edges. The edge cost function at each dimension is represented as R_i, and R_0 is the graph's original edge cost.

The graph after doing each dimension's embedding, which has a new edge cost function, is called the **residual** graph of that dimension. Finally, the FM^n heuristic is:

$$FM^n(u,v) = \sum_{1 \leq i \leqq n} FM_i(u,v) \tag{2.11}$$

To make notations shorter, we show the heuristic between two vertices of an edge $e_i = (u,v)$ in i-th dimension by $FM_i(e) = |FM_i(u) - FM_i(v)|$.

Lemma 2.3.2 proves that FastMap is a pseudo-metric embedding.

Lemma 2.3.2. *The FastMap embedding is a pseudo-metric embedding.*

Algorithm 1 Embeddings of FM^n [2]

Input: $G = \{V, E\}, R_0, n$. R_0 is the original edge cost function of the grapgh G.
Output: $FM^n(V) = (f_1(V), f_2(V), ..., f_n(V))$. $FM^n(V)$ is the embedded space and f_i is the embedding function at dimension i
$i = 1$;
while $i \leq n$ **do**
    Select a random vertex $r \in V$;
    Let a_i be the furthest pivot from r;
    Let b_i be the furthest pivot from a_i;
    For each $v \in V$ $f_i(v) = \frac{d(a_i,v)+d(a_i,b_i)-d(v,b_i)}{2}$;
    For each edge $(u, v) \in E$, $C_i(u, v) = C_{i-1}(u, v) - |f_i(u) - f_i(v)|$;
    $i++$;
end while

Proof. The embedding space of FastMap $FM^n(V)$ and the FastMap heuristic FM^n are the coordinate space and the distance measure of the pseudo-metric space. Here we show that the three conditions of a pseudo-metric embedding are met.

The first condition requires checking that self distances are 0:

$$FM^n(u, u) = |FM^n(u) - FM^n(u)| = 0 \tag{2.12}$$

The second condition requires distances to be symmetric:

$$FM^n(u, v) = |FM^n(u) - FM^n(v)| = |FM^n(v) - FM^n(u)| = FM^n(v, u) \tag{2.13}$$

The third condition requires the triangle inequality to hold:

$$FM^n(u, w) = |FM^n(u) - FM^n(w)| \tag{2.14}$$

$$\leq |FM^n(u) - FM^n(v)| + |FM^n(v) - FM^n(w)| \tag{2.15}$$

$$= FM^n(u, v) + FM^n(v, w) \tag{2.16}$$

□

A FastMap embedding is shown in Figure 2.4.

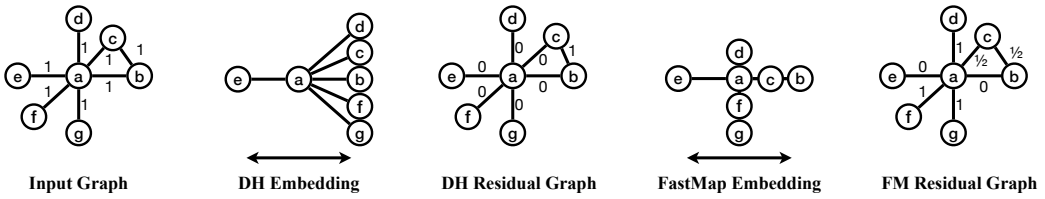


Figure 2.4: Embeddings of differential heuristic and FastMap along with the residual graphs after doing the embeddings.

2.3.3 Abstraction Background

A branch of heuristics is built on the abstraction(s) of the state space, like the Pattern Databases heuristic. First, they abstract the state space into other state spaces, and then they use the heuristics built on the abstract state spaces to compute a heuristic for the original state space. Below are definitions regarding abstraction that are taken from Yang et al. (2008) [24] with minor changes.

Definition 2.3.20 (Abstract state space)**.** An abstract state space of a graph G is an undirected graph $A_i = (V_i, E_i, C_i, R_i)$, which is constructed by an abstraction mapping (Definition 2.3.21) from G to A_i. V_i and E_i are, respectively, the set of abstract vertices and abstract edges. A_i has two edge cost functions: the primary edge cost function $C_i : E_i \to \mathbb{R}$ and the residual edge cost function $R_i : E_i \to \mathbb{R}$. Thus, each abstract edge $e_i \in E_i$ is assigned with two costs which are a primary cost $C_i(e_i)$ and a residual cost $R_i(e_i)$ [24].

Definition 2.3.21 (Abstraction mapping)**.** An abstraction mapping from G to A_i is a function $\psi_i : G \to A_i$ that maps vertices of G to vertices of A_i, $\psi_i : V \to V_i$, while satisfying the two following conditions [24]:

1. For each edge in the original graph G, there is an edge in the abstract graph A_i. Thus connectivity in the original graph G is preserved:

$$\forall (u, v) \in E, (\psi_i(u), \psi_i(v)) \in E_i$$

or

$$\forall e \in E, \psi_i(e) \in E_i$$

2. The cost of an abstract edge must be less than the cost of the edge they correspond to in the original graph:

$$\forall e \in E, C_i(e_i) + R_i(e_i) \leq C(e)$$

To make notations easier, the abstractions of a vertex $v \in V$ and an edge $e \in E$, which are $\psi_i(v)$ and $\psi_i(e)$, are represented as $v_i \in V_i$ and $e_i \in E_i$ respectively.

Definition 2.3.22 (Cost of an abstract path)**.** The cost of an abstract path $p_i = \{e_i^1, e_i^2, ..., e_i^j\}$ in space A_i is [24]

$$Cost_i(p_i) = C_i(e_i^k) + R_i(e_i^k), 1 \leq k \leq j$$

The below definition does not exist separately in Yang et al. (2008) work [24]. Nonetheless, it can be inferred directly.

Definition 2.3.23 (Primary cost of an abstract path)**.** The primary cost of an abstract path $p_i = e_i^1, e_i^2, ..., e_i^j$ in space A_i is

$$C_i(p_i) = \sum_{k=1}^{j} C_i(e_i^k)$$

The below definition is different from Yang et al. (2008) work [24].

Definition 2.3.24 (Abstract heuristic)**.** The abstract heuristic h_i, in space A_i, between two vertices $u_i, v_i \in V_i$ is defined as

$$h_i(u_i, v_i) = \min_{p_i \in Paths(A_i, u_i, v_i)} C_i(p_i)$$

where $Paths(A_i, u_i, v_i)$ is the set of all possible paths between u_i and v_i in space A_i.

Another notation is:

$$C_i^*(u_i, v_i) := \min_{p_i \in Paths(A_i, u_i, v_i)} C_i(p_i)$$

Definition 2.3.25 (Abstraction)**.** An abstraction of a graph G is a $< A_i, \psi_i >$ pair where A_i is the abstract state space of G constructed by the abstraction mapping ψ_i from G to A_i [24].

Definition 2.3.26 (Abstraction system)**.** An abstraction system is a pair (G, Ξ), where G is an undirected graph and Ξ is a set of abstractions $\Xi = \{< A_i, \psi_i > | \psi_i : G \to A_i, 1 \leq i \leq n\}$ [24].

Definition 2.3.27 (Additive heuristic)**.** The additive heuristic between $u, v \in V$ in an abstraction system (G, Ξ) is

$$h_{add}(u, v) = \sum_{i=1}^{n} h_i(u_i, v_i)$$

where h_i is the abstract heuristic in space A_i [24].

Definition 2.3.28 (Additive abstraction system)**.** An abstraction system is additive if [24]

$$\forall e \in E, \sum_{i=1}^{n} C_i(e_i) \leq Cost(e)$$

One method for building a heuristic is using additive heuristics. Two kinds of additive heuristics are static and dynamic ones, as defined below.

Definition 2.3.29 (Static additive heuristic)**.** In a static additive heuristic, the abstract state spaces are defined before building the heuristic.

A good example of a static additive heuristic can be shown on a Sliding Tile Puzzle which is one of the famous search problems.

An $n \times n$ Sliding Tile Puzzle has an $n \times n$ board of tiles where each tile contains a number from 1 to $n \times n - 1$. Each tile's number differs from the others, and one tile is blank. The player can swap the blank tile at each step with one of its neighbors. The goal is to have an ascending sort of the numbers in the sliding tile puzzle. The blank tile should be at $(1, 1)$ location of the board, and the number i should be at location $(i/n+1, i\%n+1)$. Location (i, j) of the board is at the i-th row and the j-th column of the board where i, j are from 1 to n beginning from the top left of the board as shown in Figure 2.5(a).

For example, in a 3×3 sliding tile puzzle shown in Figure 2.5(a), the blank tile can be swapped with one of the 3, 4, 6 tiles (Figure 2.5(b)). By swapping

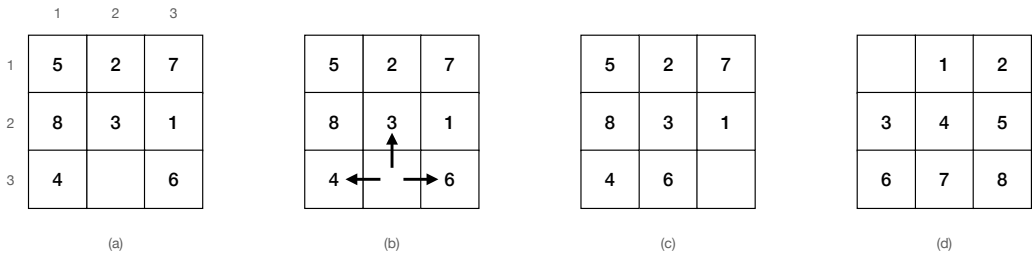


Figure 2.5: 3*3 Sliding tile puzzle.

the blank tile with tile 6, the state in Figure 2.5(c) is reached. Then the goal is to reach the state in Figure 2.5(d).

To show a static additive heuristic, consider a 2×2 Sliding Tile Puzzle, which has the state space as a 2×2 board Figure 2.6(a). Two abstract state spaces can be Figure 2.6(b) and Figure 2.6(c). Yellow tile moves have a primary cost of 0 and a residual cost of 1. Thus, the minimum costs of solving the puzzles in sub-figures (a), (b), and (c) of Figure 2.6 are respectively 6, 4, and 2. Since the minimum cost of a solution in abstract space is considered as the abstract heuristic, Figure 2.6(b) and Figure 2.6(c) have abstract heuristics of 4 and 2. Moreover, since this abstraction system is additive (each edge has cost 1 only in one of the abstractions), the additive heuristic of state A in Figure 2.6(a) is the sum of the abstract heuristics in Figure 2.6(b) and (c); i.e., $h_{add}(stateA) = 4 + 2 = 6$.

Definition 2.3.30 (Dynamic additive heuristic)**.** In a dynamic additive heuristic, the abstract state spaces are defined while building the heuristic.

2.3.4 Search Algorithms Background

Heuristics are used in search algorithms that use a heuristic during the search. A^* is one of the most popular heuristic search algorithms. Before explaining A^*, we explain the Best First Search algorithm (Algorithm 2)[5], which is a general form of A^*.

The best first search algorithm is a greedy algorithm in which there is an open and a closed list. At each step in the algorithm, the first element with

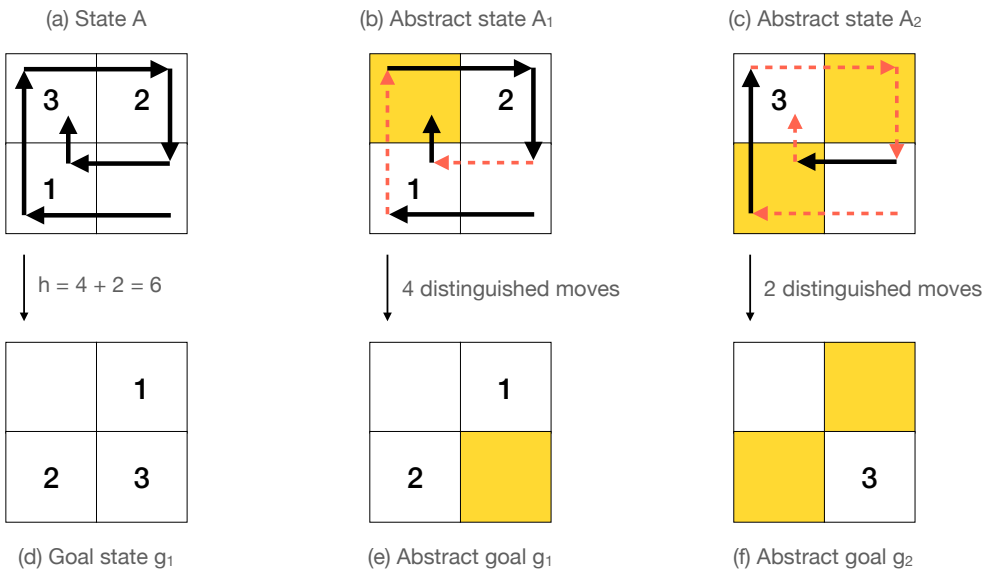


Figure 2.6: 2*2 Sliding tile puzzle abstraction.

the lowest cost in the open list is expanded and is added to the closed list after expansion. The algorithm is shown below. The cost function of this algorithm can be various functions. For example, Dijkstra, a simple best first search algorithm, uses g as the cost function, where g is the path cost.

Algorithm 2 Best First Search algorithm [5]

Input: $G = \{V, E\}, C, start, goal.$ the graph, the edge cost function, the start, and the goal states
Output: $p = \{start, u_1, u_2, ..., u_n, goal\}$ A path with the minimum cost from the start to the goal
OPEN = []
CLOSED = []
Add start to OPEN
u = start
while OPEN is not empty and u is not the goal **do**
    u = pop the state with min cost from OPEN
    for v in NEIGHBORS(U) **do**
        if v is not in CLOSED **then**
            if (v in OPEN and $f(v)$ reached from u $\leqq f(v)$ already in OPEN) or (v not in OPEN) **then**
                update f(v)
                parent(v) = u
            end if
            add v to the OPEN if it is not already in the OPEN
        end if
    end for
    add u to the CLOSED list
end while
Extract the path and return it

There are several best first search algorithms [6, 18, 13]. In this thesis, we use the A^* algorithm. A^* as a best first search algorithm has the cost function being $f(v) = g(v) + h(v)$ where for each state $v \in V$, $g(v)$ is the path cost from start to v and $h(v)$ is the heuristic between v and the goal.

Chapter 3

Contribution

This chapter shows that FastMap is an additive heuristic. It generalizes FastMap embedding. Finally, it provides new embedding functions and pivot selection methods.

3.1 FastMap Is an Additive Heuristic

Since the publication of the FastMap paper, the connection between FastMap and additive heuristics has not been made. This section provides lemmas that prove the heuristic built by FastMap is an additive heuristic.

FastMap embedding builds an n-dimensional embedding space. Each dimension of the embedding space can be considered an abstract space of the original graph. Then, the heuristic in each abstract space is added together to build the FastMap heuristic. Therefore, FastMap seems to have both abstraction and the additive properties of an additive heuristic. In the next paragraph, in more detail, we describe the abstraction system of FastMap, which is a required part of defining an additive heuristic [24].

Define the abstraction system of the FM^n as (G, Ξ), where G is the state space and $\Xi = \{(A_i, \psi_i) | \psi_i : G \rightarrow A_i, 1 \leq i \leq n\}$ is the set of abstractions and their mapping functions pairs. The mapping functions of FM^n are defined as $\psi_i(v) = v_i^{f_i(v)}, 1 \leq i \leq n, v \in V$ and f_i is the embedding function of FastMap's i-th dimension (f_i in Algorithm 1). The i-th abstract state space of FastMap is $A_i = \{V_i, E_i, C_i, R_i\}$; the set of vertices is $V_i = \psi_i(V)$ and the set of edges is $E_i = \{(u_i, v_i) | u_i, v_i \in V_i\}$ (a complete graph). In the abstract

space, each edge has two cost functions: primary and residual cost functions. The primary edge cost function is $C_i(e_i) = FM_i(e_i),\ e_i \in E_i$. The residual edge cost function for edges that do not have a corresponding edge in E is $R_i(e_i) = 0,\ \nexists\ e \in E\ \psi_i(e) = e_i$, and the residual edge cost function for edges that have a corresponding edge in E is $R_i(e_i) = R_{i-1}(e_{i-1}) - FM_i(e_i),\ \exists\ e \in E\ \psi_i(e) = e_i,\ \psi_{i-1}(e) = e_{i-1}$. Also, we define $\mathrm{R}_0(e_0) = C(e),\ e_0 := e \in E$.

Mapping functions of the abstraction system have to follow two conditions (Definition 2.3.21). The first condition of FastMap's mapping function is satisfied as edges in the abstractions are a superset of the edges in the original state space. The second condition is also satisfied, as shown in the lemma below.

Lemma 3.1.1 (FastMap's mapping function is an abstraction mapping (second condition))**.** $\forall e \in E, C_i(e_i) + R_i(e_i) \leq C(e)$

Proof. By the definition of R_0

$$R_0(e_0) = C(e) \tag{3.1}$$

By the definition of R_i

$$\forall 1 \leq i \leq n, R_i(e_i) = R_{i-1}(e_{i-1}) - FM_i(e_i) \tag{3.2}$$

$$\implies \forall 1 \leq i \leq n, R_i(e_i) \leq R_{i-1}(e_{i-1}) \tag{3.3}$$

$$\implies \forall 1 \leq i \leq n, R_i(e_i) \leq R_0(e_0) = C(e) \tag{3.4}$$

$$\implies \forall 1 \leq i \leq n, R_i(e_i) \leq C(e) \tag{3.5}$$

$$\implies \forall 1 \leq i \leq n, R_{i-1}(e_{i-1}) \leq C(e) \tag{3.6}$$

$$\implies \forall 1 \leq i \leq n, (R_{i-1}(e_{i-1}) - FM_i(e_i)) + FM_i(e_i) \leq C(e) \tag{3.7}$$

By the definition of $R_i(e_i)$ and $C_i(e_i)$

$$\implies \forall 1 \leq i \leq n, R_i(e_i) + C_i(e_i) \leq C(e) \tag{3.8}$$

□

So far, we have defined an abstraction system for FastMap. In order to prove FastMap is an additive heuristic, we have to show both that FastMap's abstraction system is additive and that FastMap's abstraction system heuristic is equal to the FastMap heuristic. Theorem 3.1.2 shows that FastMap has an additive abstraction system. Theorem 3.1.4 completes the proof that FastMap is an additive heuristic.

We can show that the FastMap abstraction system is additive, knowing that FastMap is locally admissible [2].

Theorem 3.1.2. *FastMap abstraction system is additive.*

Proof. From the definition of FastMap heuristic

$$h(e) = \sum_{i=1}^{n} h_i(e_i) \tag{3.9}$$

Also, from the definition of abstraction system primary edge costs, we know that

$$\forall e_i \in E_i \quad h_i(e_i) = C_i(e_i) \tag{3.10}$$

$$\implies h(e) = \sum_{i=1}^{n} C_i(e_i) \tag{3.11}$$

Since the FastMap heuristic is locally admissible [2]

$$\forall e \in E \quad h(e) \leq C(e) \tag{3.12}$$

$$\implies \sum_{i=1}^{n} C_i(e_i) = h(e) \leq C(e) \tag{3.13}$$

$$\implies \sum_{i=1}^{n} C_i(e_i) \leq C(e) \tag{3.14}$$

Which is the definition of an additive abstraction system. □

Lemma 3.1.3. *The shortest path cost between two vertices (u_i, v_i) in the defined abstract space of FastMap, A_i, is equal to the FastMap heuristic at dimension i (FM_i); i.e.,*

$$\forall u_i, v_i \in V_i,\ C_i^*(u, v) = FM_i(u, v) \tag{3.15}$$

Proof. The edge between u_i and v_i is a path with cost $FM_i(u, v)$, which implies

$$C_i^*(u, v) \leq FM_i(u, v)$$

To prove the lemma, we use a proof by contradiction. Assume: $C_i^*(u, v) < FM_i(u, v)$.

Then there would be a shortest path $p = \{u, u_i^1, u_i^2, ..., u_i^k, v\}$ with length shorter than $FM_i(u, v)$.

Each edge cost in the abstract space is the difference between its vertices embedding coordinates. Therefore, according to the path p, the embedding $\phi_i(v)$ would be at most $C_i^*(u, v)$ apart from $\phi_i(u)$. This is a contradiction since they are, by definition, $FM_i(u, v)$ apart. □

Theorem 3.1.4. *FastMap is an additive heuristic.*

Proof. In Theorem 3.1.2, we showed that the FastMap abstraction system is additive. This abstraction system's heuristic is

$$\forall u, v \in V \quad h_{add}(u, v) = \sum_{i=1}^{n} h_i(u_i, v_i) = C_i^*(u_i, v_i) \tag{3.16}$$

Lemma 3.1.3 showed that

$$\forall u, v \in V \quad C_i^*(u_i, v_i) = FM_i(u, v) \tag{3.17}$$

$$\implies \forall u, v \in V \quad h_{add}(u, v) = \sum_{i=1}^{n} FM_i(u, v) \tag{3.18}$$

Therefore, the FastMap heuristic is equal to the newly defined additive abstract system's heuristic, which implies that FastMap is an additive heuristic. □

Additive PDBs are static and create all abstract spaces simultaneously at the beginning. However, FastMap creates abstract spaces through the embedding process dynamically. Each abstract space of FastMap is built after creating its previous abstract space and is dependent on the previous one.

3.2 General FastMap Embeddings

FastMap embedding can be generalized by generalizing its embedding function and its pivot selection method.

Theorem 3.1.2 relies on the local admissibility of the FastMap heuristic. Therefore, if we substitute the FastMap original embedding function with a new embedding function that results in a locally admissible heuristic, FastMap with the new embedding function will remain an additive heuristic.

Also, pivots other than the furthest pivots can be selected at each dimension since the proofs are independent of the pivots. Section 3.4 will explain pivot selection methods.

3.2.1 Conditions of General FastMap Embedding Functions

Assume a and b are the embedding pivots and ϕ is an embedding function. In order to have a locally admissible heuristic, ϕ has to have Condition 3 below.

1. $\phi(a) = 0$
2. $\phi(b) = d_{ab}$
3. $\forall e = (i, j) \in E \quad |\phi(i) - \phi(j)| \leq C(e)$

The first two conditions are unnecessary; However, they can be used to normalize the coordinates of the pivots. Condition 3 is the local admissibility of each dimension's heuristic. The lemma below shows that if the heuristic of each dimension is locally admissible, the whole heuristic of general FastMap will be locally admissible too.

Lemma 3.2.1. *If each dimension's heuristic is locally admissible in the residual graph of that dimension, regardless of the embedding function, $FastMap^n$ will be locally admissible in the original graph.*

Proof. From line 8 of the FastMap Algorithm 1

$$R_i(e) = R_{i-1}(e) - h_i(e) \tag{3.19}$$

where $R_i(e)$ is the edge cost of the edge $e \in E$ at dimension i of the FastMap, and $h_i(e)$ is the captured heuristic of the edge e at dimension i of the FastMap heuristic.

$$\implies R_i(e) = (R_{i-2}(e) - h_{i-1}(e)) - h_i(e) \tag{3.20}$$

$$\implies R_i(e) = R_{i-2}(e) - h_{i-1}(e) - h_i(e) \tag{3.21}$$

With proof by induction we have

$$\implies R_i(e) = R_0(e) - h_1(e) - h_2(e) - ... - h_i(e) \tag{3.22}$$

$R_0(e)$ is the initial edge cost of e. Therefore, the edge cost of an edge at a dimension is the initial edge cost subtracted by the sum of all captured heuristics of the edge in previous dimensions.

$$R_i(e) = R_0(e) - (h_1(e) - h_2(e) - ... - h_i(e)) \tag{3.23}$$

On the other hand, according to the assumption that the heuristic of each dimension is locally admissible, we have

$$\forall\ 1 \leq i \leq n \quad \forall e \in E \quad h_k(e) \leq R_k(e) \tag{3.24}$$

Considering Equation 3.24 for $i = n$, we have

$$\forall e \in E \quad h_n(e) \leq R_n(e) \tag{3.25}$$

Considering Equation 3.23 for $i = n$ and Equation 3.25, we have

$$h_n(e) \leq R_0(e) - (h_1(e) + h_2(e) + ... + h_n(e)) \tag{3.26}$$

$$\implies (h_1(e) + h_2(e) + ... + h_n(e)) + h_n(e) \leq R_0(e) \tag{3.27}$$

$$\implies h_1(e) + h_2(e) + ... + h_n(e) \leq R_0(e) \tag{3.28}$$

From FastMap formula Equation 2.11

$$\implies h(e) \leq R_0(e) \tag{3.29}$$

$$\implies \forall e \in E \quad h(e) \leq R_0(e) \tag{3.30}$$

□

The A^* algorithm with a consistent heuristic is guaranteed to find optimal solutions. Therefore, to use the general FastMap heuristic in the A^* algorithm, we have to show that the general FastMap heuristic is consistent. The lemma below shows the consistency of the general FastMap heuristic since we have already shown its local admissibility.

Lemma 3.2.2. *If the general FastMap heuristic is locally admissible, regardless of the embedding function, it will be consistent too.*

Proof. Let the general FastMap has n dimensions. For any dimension i and three vertices u, v, g in the V_i where $(u, v) \in E$, because of triangle inequality, we have

$$|C_i^*(u, g) - C_i^*(v, g)| \leq C_i^*(u, v) \tag{3.31}$$

We know from Lemma 3.1.3 that for any two vertices in V_i, $C_i^*(u, g) = FM_i(u, g)$

$$\implies |FM_i(u, g) - FM_i(v, g)| \leq FM_i(u, v) \tag{3.32}$$

Summing both sides for all dimensions, we get

$$\sum_{i=1}^{n} |FM_i(u, g) - FM_i(v, g)| \leq \sum_{i=1}^{n} FM_i(u, v) \tag{3.33}$$

Since $|(a_1 - b_1) + (a_2 - b_2)| \leq |a_1 - b_1| + |a_2 - b_2|$ we can make the left side of the equation smaller

$$|\sum_{i=1}^{n} FM_i(u, g) - \sum_{i=1}^{n} FM_i(v, g)| \leq \sum_{i=1}^{n} FM_i(u, v) \tag{3.34}$$

From the definition of FastMap, we have

$$|FM(u, g) - FM(v, g)| \leq FM(u, v) \tag{3.35}$$

Since we know the general FastMap is locally admissible, we have

$$|FM(u, g) - FM(v, g)| \leq C(u, v) \tag{3.36}$$

The equation above proves the local consistency of the general FastMap heuristic. Also, we know that local consistency implies global consistency [17]. □

3.3 New Embedding Functions

Any embedding function following the three mentioned conditions can be used instead of FastMap's original embedding function. In this section, we introduce new embedding functions.

3.3.1 FastMap Variants

Let a and b be the pivots of the embedding. Different variants of FastMap can be shown as

$$\phi(i) = qd_{ai} + rd_{ab} + sd_{bi} \tag{3.37}$$

where q, r, and s are the parameters. According to conditions in Section 3.2.1, f has to meet the conditions below.

1.

$$\phi(a) = 0 \tag{3.38}$$

$$\implies rd_{ab} + sd_{ab} = 0 \tag{3.39}$$

$$\implies r = -s \tag{3.40}$$

2.

$$\phi(b) = d_{ab} \tag{3.41}$$

$$\implies qd_{ab} + rd_{ab} = d_{ab} \tag{3.42}$$

$$\implies q + r = 1 \tag{3.43}$$

3.

$$\forall (i,j) \in E \quad |\phi(i) - \phi(j)| \leq e \tag{3.44}$$

From the embedding function formulas, we have

$$|\phi(i) - \phi(j)| = \tag{3.45}$$

$$|(qd_{ai} + rd_{ab} + sd_{bi}) - (qd_{aj} + rd_{ab} + sd_{bj})| = \tag{3.46}$$

$$|(q(d_{ai} - d_{aj}) + s(d_{bi} - d_{bj})| \leq \tag{3.47}$$

From the remark 2.3.1

$$|qe + se| \leq \tag{3.48}$$

$$|q + s|e \leq e \tag{3.49}$$

$$\implies |q + s| \leq 1 \tag{3.50}$$

The FastMap original embedding has $q = r = 1/2$ and $s = -1/2$. There are other combinations that can be used, like $q = 3/4$, $r = 1/4$, and $s = -1/4$. However, we have not tested them extensively.

3.3.2 Proportion Embedding Function

This new embedding function, in comparison to FastMaps's original embedding function, has a different combination of the arguments.

$$\phi_p(i) = \frac{d_{ai}}{d_{ai} + d_{bi}} d_{ab}$$

where d_{ai} is the shortest path cost from a to i, and a and b are two pivots of the embedding.

Theorem 3.3.1. *The heuristic defined by ϕ is locally admissible*

Description:

Assume i and j are neighbors, and e is the edge cost between them. The proposition is equivalent to:

$$|\phi_p(i) - \phi_p(j)| < e$$

Proof.

$$|\phi_p(i) - \phi_p(j)| = \tag{3.51}$$

From the embedding function's formula

$$\left| \frac{d_{ai}}{d_{ai} + d_{bi}} d_{ab} - \frac{d_{aj}}{d_{aj} + d_{bj}} d_{ab} \right| = \tag{3.52}$$

$$\left| \frac{d_{ai}(d_{aj} + d_{bj}) - d_{aj}(d_{ai} + d_{bi})}{(d_{ai} + d_{bi})(d_{aj} + d_{bj})} d_{ab} \right| = \tag{3.53}$$

$$\left| \frac{d_{ai}d_{bj} - d_{aj}d_{bi}}{(d_{ai} + d_{bi})(d_{aj} + d_{bj})} d_{ab} \right| \leq \tag{3.54}$$

From remark 2.3.1

$$\left| \frac{d_{ai}(d_{bi} + e) - (d_{ai} - e)d_{bi}}{(d_{ai} + d_{bi})(d_{aj} + d_{bj})} d_{ab} \right| = \tag{3.55}$$

$$\left| \frac{d_{ai}e + d_{bi}e}{(d_{ai} + d_{bi})(d_{aj} + d_{bj})} d_{ab} \right| = \tag{3.56}$$

$$\left| \frac{(d_{ai} + d_{bi})e}{(d_{ai} + d_{bi})(d_{aj} + d_{bj})} d_{ab} \right| = \tag{3.57}$$

$$\left|\frac{d_{ab}}{d_{aj}+d_{bj}}e\right| \leq \tag{3.58}$$

From lemma 2.3

$$|e| = e \tag{3.59}$$

$$\implies |\phi_p(i) - \phi_p(j)| \leq e \tag{3.60}$$

□

3.3.3 Shrinking Function

$$f_s(i) = c.\phi(i) \; for \; 0 \leq c \leq 1 \tag{3.61}$$

Since $\forall \, i f_s(i) \leq \phi(i)$ the heuristic made by this embedding is locally admissible too. After doing the first dimension of the FastMap embedding, the captured heuristic at each edge will be deducted from the edge cost. Therefore, edge weights are reduced (or unchanged) after doing each dimension's embedding. Consequently, in the second dimension of the FastMap, these reduced edges appear in the shortest paths from the pivots. Since the FastMap embedding function uses the shortest path costs, the range of the embedding values will become smaller. Therefore, the captured heuristic, which is the difference between the embedding values, will become less. By shrinking the captured heuristic in the first dimension, the reduction of edge weights would be less; hence, the residual would be greater than the original FastMap. This method can be used in any dimension, and it helps to leave more residual for the following dimensions. However, using this embedding in the last dimension will not be useful since it results in capturing less heuristic.

3.3.4 Differential Heuristic As The Last Dimension

By doing DH in one of the embedding steps, the residual graph would have the shortest paths with cost zero between any two vertices (Figure 3.1).

Lemma 3.3.2. *After doing the embedding by DH, the shortest path cost between any two vertices in the graph would be zero.*

Proof. First, we prove that after doing the embedding by DH, every edge on the shortest path from the DH pivot to any vertex in the residual graph would have the cost of zero, i.e.,

$$\forall e \in p,\ C(e) = 0 \tag{3.62}$$

where $p = \{v_0, v_1, v_2, ..., v_n\}$ is the shortest path from v_0 (the DH pivot) to an arbitrary vertex v_n in the graph before doing the embedding.

From the definition of the shortest path, we know that the shortest path from v_0 to any vertex v_i on p is $\{v_0, v_1, v_2, ..., v_{i-1}, v_i\}$. Therefore, for every vertex on the shortest path from the pivot, its preceding vertex is also on the same shortest path except for the last edge connecting them. Thus, the difference in the differential heuristic embeddings of these two vertices is their edge cost, which results in a residual of 0 for this edge. As a consequence, all edges on the shortest paths from the pivot would have a residual of 0 after doing differential heuristic embedding. Therefore, if we do an embedding after doing the differential heuristic, we will not capture any new heuristic as the embedding of all the vertices would be zero. □

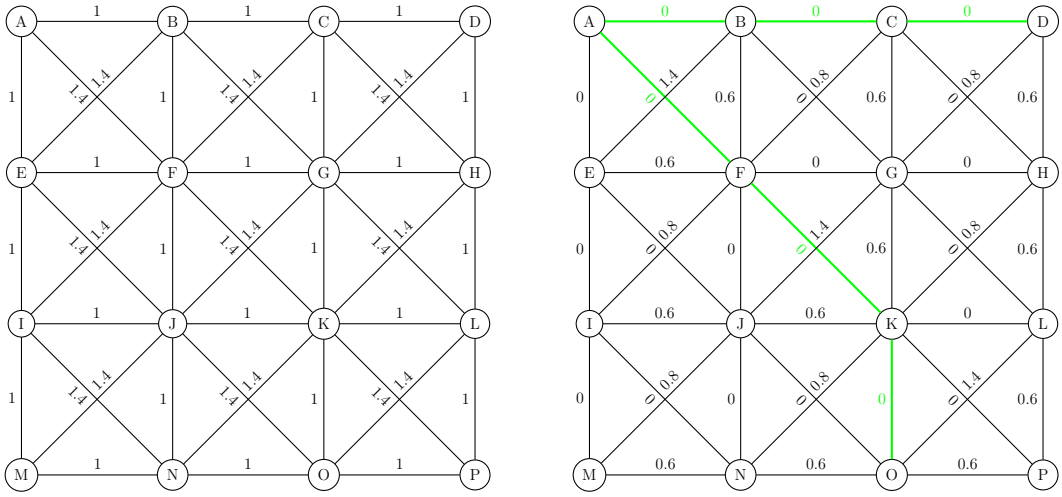


Figure 3.1: Differential heuristic as the embedding function. The left graph is the state space before doing differential heuristic embedding. The right graph is the state space after doing differential heuristic embedding with A as the pivot. A shortest path from D to O is shown in green with the cost zero.

Capturing the most out of the residual at the last dimension is good since the less residual remains, the more heuristic is captured, at least between the

adjacent vertices. Note that the optimal heuristic has a residual of zero, but a residual of zero does not imply an optimal heuristic. By using a differential heuristic for embedding the last dimension, more heuristic is captured in comparison to FastMap as shown in Figure 4.2.

3.4 New Pivot Selection Methods

Instead of one n-dimensional FastMap heuristic, there can be multiple different heuristics with fewer dimensions but having n dimensions altogether. For example, there can be $\frac{n}{2}$ different 2-dimensional heuristics. Therefore, the general case of the heuristic is

$$h(v_1, v_2) = \max_{h_i \in H}(h_i(v1, v2))$$

where H with $|H| = m$ is the set of the m heuristics with each $h_i \in H$ having $1 \leq n_i \leq n$ dimensions such:

$$\sum_{1 \leq i \leq t} n_i = n$$

However, since the original FastMap uses the furthest method for selecting pivots, it is possible that the first dimensions of two separate FastMap heuristics on a graph become the same. Thus, the challenge is ensuring different pivots are used in each dimension so that heuristics become different and capture other parts of the graph too. In order to optimize pivots selection at each dimension's embedding, two ways are suggested. Below is the optimization of pivot placement of the $j-th$ dimension of $h_k \in H$ by using a baseline heuristic h_0 where $1 \leq j \leq n_k$.

The Baseline Heuristic

The baseline heuristic h_0 can be built in different ways.

- h_0 = Zero heuristic.
- h_0 = The default embedding distance (e.g. Euclidean).
- h_0 can be the heuristic of previous dimensions inside the h_k

- h_0 can be the maximum of the heuristic of previous dimensions inside the h_k and $h_1, h_2, ..., h_{k-1}$

Using Subset Selection

Here we use the approximate version of subset selection [20] to find pivots of each dimension while maximizing the captured heuristic at each dimension approximately. To do this, choose different candidate pivots $CP \subseteq V$. Embed the $j-th$ dimension of h_k using each of these candidate pivots separately. Build different heuristics using these different embeddings at $j-th$ dimension on top of h_0. Greedily find the one heuristic that maximizes the sample's total heuristic. Use the pivot(s) of the founded heuristic for embedding $j-th$ dimension. This process can go on for the following dimensions similarly by choosing candidate pivots for each dimension.

The optimal version of subset selection [20] can be used to find multiple dimensions pivots. To use subset selection, We have to find a combination of pivots for these multiple dimensions (among many candidates for each dimension) that maximize the sample heuristic. Note the difference with the approximate version: here, we do not choose candidate pivot at each dimension greedily; we compare whole dimensions' heuristics to find the optimal heuristic and consequently pivots.

Using Heuristic Error (HE)

Instead of selecting pivots that are farthest in distance, pivots with the maximum heuristic error can be selected. The heuristic error between two states means that we have an uncaptured heuristic between them. Therefore, finding the state pair with the maximum heuristic error is good; by placing pivots there, we can capture more heuristics. To use heuristic error, select a random vertex $v \in V$, find the vertex that has the maximum $\alpha \cdot d(r, v) + \beta \cdot (d(r, v) - h_0(r, v))$ in the original graph G (not the residual) where d is the distance function, h_0 is the baseline heuristic, and α, β are arbitrary constants. Use the founded vertex as a pivot for $j-th$ dimension of h_k.

Figure 3.2 shows candidate pivots for the sample map inside the figure.

Table 3.1 shows the perfect heuristic, Octile heuristic, and different FastMap heuristics between three pairs of (E, F), (C, D), and (A, D). Each row shows the maximum heuristic of Octile and that row's heuristic. For example, the $FM(A, D)$ row shows the maximum of the heuristic of Octile and a FastMap embedding with pivots being A and D. The Octile heuristic between (A, D) is perfect. However, there is a heuristic error from the perfect heuristic if we use Octile between (E, F), or (C, D). Using $FM(A, D)$ captures the heuristic between (E, F) better. $FM(B, C)$ is similar to $FM(A, D)$ since the map is symetric. $FM(E, F)$ captures the heuristic between (E, F) perfectly. $FM(C, F)$ (and $FM(D, E)$) capture heuristic between (E, F) better than Octile. $FM(C, D)$ captures the heuristic between (E, F) and (C, D) better than Octile.

On the map of Figure 3.2, if we use FastMap [2] pivot selection method, which chooses pivots that are farthest from each other, we get $FM(A, D)$ or $FM(B, C)$ which only have improvements on $h(E, F)$ over Octile. If we use heuristic error, with $\alpha = 0, \beta = 1$, to find the pivots we get one of $FM(A, D)$, $FM(B, C)$, $FM(E, F)$, $FM(C, F)$, or $FM(E, D)$ which have better heuristic in general in comparison to original FastMap embeddings. If we use heuristic error, with $\alpha = 1, \beta = 2$, to find the pivots we get one of $FM(A, D)$, $FM(B, C)$, $FM(E, F)$, $FM(C, F)$, $FM(E, D)$, or $FM(C, D)$ which have better heuristic in general in comparison to previous HE formula and the original FastMap because it has a high possibility to get $FM(C, D)$ as the embedding which capture heuristics better between (E, F) and (C, D).

By using heuristic error, we put the pivots in places with a higher heuristic error, which results in capturing the heuristic better, at least between the pivots. If we use heuristic error with $\alpha = 0, \beta = 1$, pivots likely end up being close to each other, as shown for the map of 3.2. Close pivots result in having heuristics that are small because the largest possible heuristic to capture is the distance between the pivots. However, if we use heuristic error with $\alpha = 1, \beta = 2$, we get pivots that are further, and still, there is a heuristic error between them, which gives us a higher range of possible heuristics.

Since both of these methods find better pivots for a dimension, they can

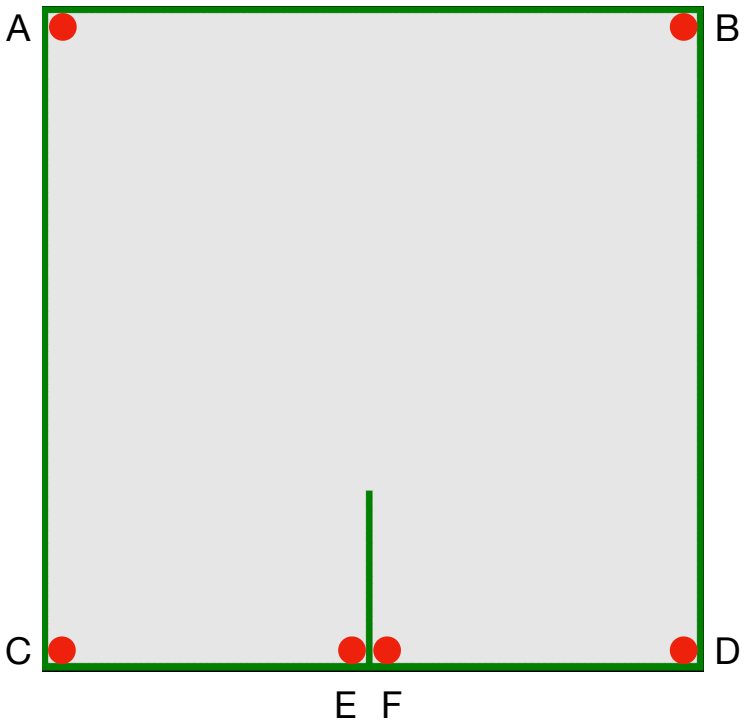


Figure 3.2: A, B, C, D, E, and F are candidate pivots for this map.

Heuristic	h(E, F)	h(C, D)	h(A, D)
h^*	54.0	118.5	137.2
Octile	2.0	97.0	137.2
$FM(A, D)$	19.8	97.0	137.2
$FM(B, C)$	19.8	97.0	137.2
$FM(E, F)$	54.0	97.0	137.2
$FM(C, F)$	46.4	97.0	137.2
$FM(D, E)$	46.4	97.0	137.2
$FM(C, D)$	38.7	118.5	137.2

Table 3.1: The perfect heuristic, Octile heuristic, and different FastMap heuristics between (E, F), (C, D), and (A, D).

either be used to improve the heuristic of a single general FastMap embedding or the heuristic of multiple general FastMap embeddings used together.

Chapter 4

Experiments and Results

This chapter is divided into the four sections below,

- Analyzing the results of the FastMap paper and trying to replicate it.
- Evaluating subset selection method used for optimizing pivots placements.
- Comparing newly introduced heuristics on different sets of maps, including Sturtevant (2012) benchmarks [22] and multi-dimensional graphs.
- Calculating the total captured heuristics for FastMap and General FastMap with differential heuristic as the last dimension.

4.1 FastMap Paper Results Analysis

The FastMap paper ran the A* algorithm with three heuristics on three maps (lak503d, brc300d, maze512-32-0) for 1000 random search problems. The heuristics used by the paper are:

FM^{10}: A 10-dimensional FastMap.

DH^{10}: A differential heuristic with ten furthest pivots.

$max(FM^5, DH^5)$: The maximum heuristic of a 5-dimensional FastMap and a differential heuristic with five furthest pivots.

They showed the results in 3 ways (one of them is duplicated in Table 4.1) [2]. However, their ways of comparing heuristics have four main common

Map	'lak503d'						'brc300d'						'maze512-32-0'					
	FM-WINS 570		DH-WINS 329		FM+DH-WINS 101		FM-WINS 846		DH-WINS 147		FM+DH-WINS 7		FM-WINS 382		DH-WINS 507		FM+DH-WINS 111	
	Med	MAD	Med	MAD	Med	MAD	Med	MAD	Med	MAD	Med	MAD	Med	MAD	Med	MAD	Med	MAD
FM(10)	261	112	465	319	2,222	1,111	205	105	285	149	894	472	1,649	747	11,440	9,861	33,734	13,748
DH(10)	358	215	278	156	885	370	217	119	200	129	277	75	3,107	2,569	2,859	2,194	8,156	4,431
FM(5)+DH(5)	303	160	323	170	610	264	206	105	267	135	249	73	2,685	2,091	3,896	2,992	7,439	4,247

Table 4.1: FastMap paper results [2]. Med is the median of the number of expansions. MAD is the median absolute deviation.

issues. First, they are showing the results only on three maps, which is inadequate for comparing different heuristics. The heuristics should be compared on more maps with different topologies. Second, the results are created for random problems, which makes them hard to replicate. Third, random problems are not a good benchmark since they will not necessarily have a uniform distribution of the problems according to their hardness. Most random problems have short path lengths that cannot evaluate the heuristics thoroughly. Fourth, the median (or the number of wins) is not a good measure for comparing the heuristics. There are cases where one heuristic can have a lower median (or more number of wins) than another heuristic. However, since the average number of expansions of the former one is higher, it takes more time to solve all the problems with the former one.

4.1.1 Validation of FastMap Paper Results

Since the FastMap paper has used random problems, the replication of results is impossible without having the embedding pivots and the random problems. To replicate their result as much as possible, we ran A^* with the same heuristics and settings they used (i.e., 1000 random problems on the three maps). Through the experiments, we found out the pivots have a considerable impact on the embedding, so our results differ. However, it generally matches their result where significant differences exist between the heuristics. For example, in map lak503d, the paper shows in 570 problems, FM performs lower node expansions, which is observable in Figure 4.1(a), where in around the first 570 problems, FM plot is so close to other heuristics. The closeness shows that the number of expansions by the heuristics are close to each other. Therefore, it is possible that with a different set of pivots and random problems, one

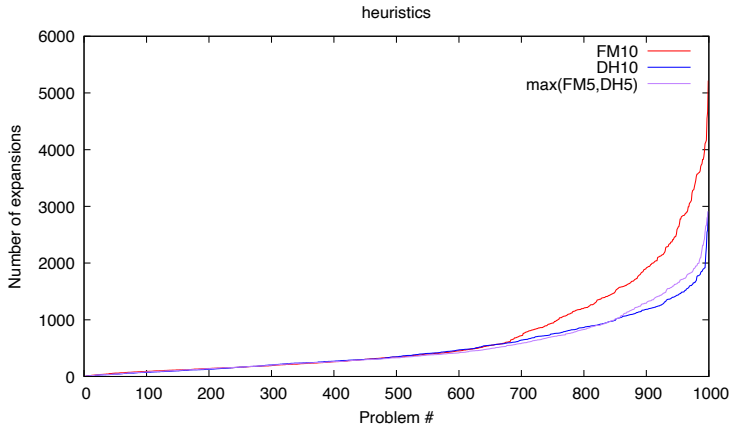
Heuristics	lak503d		brc300d		maze512-32-0	
	Md	Mn	Md	Mn	Md	Mn
FM^{10}	342	704	176	234	9684	17742
DH^{10}	347	506	187	223	7538	11929
$max(FM^5, DH^5)$	329	517	174	196	6387	11006

Table 4.2: Results of running A^* with different heuristics on three maps. Md and Mn are the median and mean of the number of expansions.

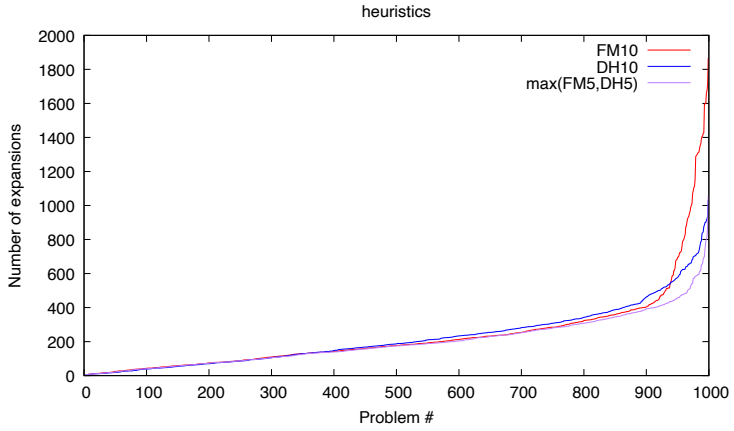
heuristic performs lower node expansions than the other and wins. Also, in problems where FM5+DH5 (i.e., $max(FM^5, DH^5)$ in Figure 4.1) wins, there is a huge gap between FM and FM5+DH5 which is observable in both Table 4.1 and Figure Figure 4.1. In map brc300d, the paper shows in 846 problems, FM wins, which is observable in Figure 4.1(b), where in around the first 870 problems, FM is so close to other heuristics. In maze512-32-0 map, the paper shows in 507 problems, DH wins with a considerable difference in the median compared to FastMap, which is observable in Figure 4.1(c) too. Table 4.2 also verifies that although FM^{10} might win in many problems (or have a lower median), it has a higher mean of the number of expansions in all three maps.

4.2 Evaluating Subset Selection Used for Optimizing Pivots Placement

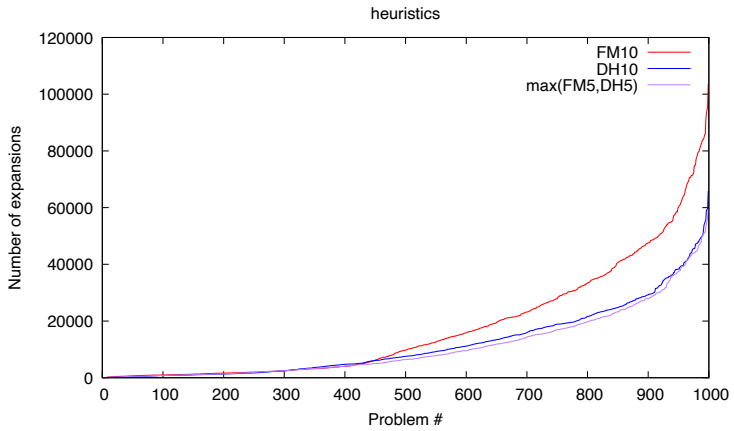
As described in Section 3.4, subset selection can be used to optimize the pivot placement of embeddings. It can be used either in optimizing one single embedding like FM^{10} and DH^{10} or optimizing when multiple embeddings are being used together, like FM^5 and DH^5 being used together. Here we show the results of optimizing the pivots' placement of DH^{10} using subset selection. We take $\sqrt{n}$ random samples on each map. Then we find the 20 furthest pivots on the map. By subset selection, we only choose 10 of them that maximize the heuristic between all pairs of the samples (i.e., $SubDH^{10}(20)$). Table Table 4.3 shows the number of node expansions for these two versions of the differential heuristic. Subset selection's preprocessing takes more time than the default embeddings; therefore, it will not be compared with them. How-



(a)



(b)



(c)

Figure 4.1: The number of expansions on three maps for 1000 random problems. (a)lak503d (b)brc300d (c)maze512-32-0.

Heuristics	DAO			SC1			Random			Rooms			Mazes		
	Md	Mn	C	Md	Mn	C	Md	Mn	C	Md	Mn	C	Md	Mn	C
DH^{10}	502	1338	11	1910	6180	47	3065	5164	28	3785	6868	57	6422	9815	26
$SubDH^{10}(20)$	**488**	**1244**	10	**1641**	**5311**	41	**2791**	**4703**	27	**3470**	**6331**	54	**6207**	**9471**	25

Table 4.3: Results of running A^* with different heuristics on different map sets. Md, Mn, and c columns are the median, mean, and 95% confidential interval on the mean of the number of expansions.

ever, we showed that it can improve DH and can be useful in applications that allow enough preprocessing time.

4.3 Comparing Different Heuristics

In order to evaluate heuristics, the A^* algorithm with different heuristics is run on the Sturtevant (2012) benchmarks [22] and multi-dimensional graphs. The heuristics performances and the map sets (and graphs) used are shown in Tables 4.7, 4.8, and 4.9. In the three below subsections, we explain (1) the heuristics, (2) the maps (and their problem instances), then (3) discuss the obtained results in the last subsection.

Pivot Selection Methods and Heuristics

The column pivot in Table 4.7 represents what distance measure is used for selecting the pivots. FAR is used for DH^n, and it finds n furthest pivots in the graph. The ED method (the same pivot selection method of FastMap, lines 5-7 in Algorithm 1) is used to find pivots of one dimension; it can be any dimension of an n-dimensional embedding. The ED method begins with choosing a random vertex r and selects a vertex with the maximum $C^*(r, u)$ in the current dimension embedding (residual), where $C^*(r, u)$ is the shortest path cost between r and u. Then, it selects the vertex v that has the maximum $C^*(u, v)$. Therefore, ED selects u and v as the pivots of that dimension, or in cases like DH as the last dimension of $FM^{n-1}DH$, where only one pivot is needed, u will be selected. Unlike ED, we use the HE method (Heuristic error method in Section 3.4) only for finding the pivots of the first dimension of a multi-dimensional embedding; Through a few experiments, we found out the built heuristic by using HE in all dimensions did not perform

as well as when HE is used only for the first dimension's pivots. HE begins with choosing a random vertex r and selects a vertex with the maximum $C^*(u,v) + 2 * (C^*(u,v) - h_0(u,v))$ in the original graph, where h_0 is the initial heuristic mentioned in Section 3.4. Here we use the maximum of all the existing heuristics as h_0. For a heuristic that consists of m general FastMap heuristics, HE is applied on all the first dimensions of the m FastMaps. For instance, if HE is used to find the first dimension's pivots of the fifth FM^2DH of $8FM^2DH$, h_0 would be the heuristic of the first four FM^2DH.

The heuristics notations in Table 4.7 are already defined in Chapters 2 and 3. All the heuristics in 4.7 use the Euclidean heuristic as their baseline heuristic. I.e., the maximum of them with Euclidean heuristic is used. Also, when HE is used, the Euclidean heuristic is used in constructing h_0. The heuristics with respect to n dimensions are briefly explained below.

Three of them are the baseline heuristics DH^n, FM^n, and $max(FM^{\frac{n}{2}}, DH^{\frac{n}{2}})$ as used in the Cohen et al. (2017) work [2] (explained in Section 4.1 too).

$FM^{n-1}DH$-Far: An n-dimensional general FastMap with the last dimension being a differential heuristic.

$max(FM^{\frac{n}{2}-1}DH, DH^{\frac{n}{2}})$: The maximum heuristic of an $\frac{n}{2}$-dimensional general FastMap with last dimension being differential heuristic and an $\frac{n}{2}$-dimensional differential heuristic. Note if HE is used, the Euclidean heuristic would be used as the h_0 for selecting the first dimension's pivots of $FM^{\frac{n}{2}-1}DH$.

$max(DH^{\frac{n}{2}}, FM^{\frac{n}{2}-1}DH)$: The maximum heuristic of an $\frac{n}{2}$-dimensional differential and an $\frac{n}{2}$-dimensional general FastMap with the last dimension being differential heuristic. Note if HE is used, the heuristic of $DH^{\frac{n}{2}}$ would be used as the h_0 for selecting the first dimension's pivots of $FM^{\frac{n}{2}-1}DH$.

$max(m * FM^{q-1}DH)$, where $m * q = n$: The maximum heuristic of m q-dimensional general FastMap embedding, which is specifically useful when used with HE pivot selection.

Maps and Their Problem Instances

The Sturtevant (2012) benchmarks [22] consist of different map sets, and each map in a map set has a set of problem instances. The heuristics performances, measured by the number of nodes expanded while solving the problem

Map Set	maps	problems
DAO	150	157,750
SC1	75	198,224
Random	70	146,220
Rooms	40	78,840
Mazes	60	586,370
Total	395	1,161,404

Table 4.4: Maps stats: the number of problems and maps of each map set.

instances, are shown in 4.7. To extend our results, we also provide the heuristic performance results on 3D and 4D graphs with different configurations of edge costs. Each vertex in the graphs is connected to two other vertices in each dimension. For instance, 50^3,RE(1-1.5) represents a $50 \times 50 \times 50$ 3D graph with edges having random edge costs between 1 and 1.5. Graphs with $E(1)$ in their name have edge costs of 1. Other graphs' names follow the same pattern. We used 1000 random problem instances for the graphs since there is no standard benchmark for them.

The Sturtevant (2012) benchmarks [22] maps stats are shown in Table 4.4. This benchmark has 395 maps from different categories and 1,161,404 problems. However, Cohen et al.'s (2017) [2] paper only represented results on three maps and for 3000 random problems, which is insufficient for having a strong conclusion on the performance of the heuristics.

Discussion

Tables 4.5 and 4.6 show the time performance and node expansions on DAO and SC1 map sets respectively. These tables show that the mean of the expansions and expansions times are correlated with a correlation of 0.9971. Therefore, we only show the expansions in the other results.

The A^* algorithm's open list has $O(logV)$ time complexity for each time removing the vertex with minimum f-cost from the open list (and finding the next min). The Octile heuristic has less precomputation time than the other heuristics, but it is less accurate than them. Therefore, it (likely) performs more expansions, which include removing the min from the open list. Since

expansions and time are correlated, the Octile heuristic takes more time to solve the whole benchmark. As shown in the tables 4.5 and 4.6, Octile performs ten times worse than the other heuristics.

The tables 4.5 and 4.6 precomputation time column helps us understand when we should use a heuristic. The Octile heuristic performance is ten times worse than the other heuristics. However, the Octile heuristic does not have precomputation time. Therefore, it can be useful in applications that need to solve only a few problem instances. Another example can be DH. DH precomputation time is nearly half the FM precomputation time. Therefore, depending on the number of problems we want to solve, it can be useful to use DH on DAO maps, although it has worse performance in the mean of expansions (and expansions times).

Table 4.7 shows that Cohen et al. (2017) [2] FastMap is not performing as well as it seems in their paper. Their paper only showed results for three maps on random problems, which we discussed in Section 4.1 are not a good evaluation setting. For instance, although FM^n wins most of the problems in DAO maps, as shown in Section 4.1 (or has a median nearly the same as DH^n, as shown in 4.7), FM^n has a higher mean of expansions. This shows that in the problems that are easier to solve, FM^n performs similarly to DH^n, which we also showed in Figure 4.1. However, for the harder problems, DH^{10} performs better, resulting in a lower mean of expansions (4.7).

As a solid result, it is shown that using differential heuristic as the last dimension of general FastMap, improves the heuristic in all the map sets (including 3d and 4d graphs) in comparison to the Cohen et al. (2017) [2] FastMap heuristic. I.e., $FM^{n-1}DH$ and $max(DH^{\frac{n}{2}}, FM^{\frac{n}{2}-1}DH)$ perform better than FM^n and $max(DH^{\frac{n}{2}}, FM^{\frac{n}{2}})$. The improvement is significant in all the maps, especially in the maze map set which the performance is two times better and is the best among all the heuristics. In other map sets, we have the best median in DAO, SC1.

HE method also performs better in all cases in comparison to the old ED pivot selection method. I.e., $FM^{n-1}DH$ and $max(DH^{\frac{n}{2}}, FM^{\frac{n}{2}-1}DH)$ when used with HE method, perform better than their ED version in overall among

Heuristics	Expansions			Time					
	Md	Mean	Conf	PCMean	Md	Mean	Conf	Node/ms	Conf
Octile	5,750	13,300	±91	0	8.793	16.759	±0.113	747.402	±1.036
DH^{10}	499	1,335	±11	**669.693**	0.682	1.688	±0.015	**754.792**	±1.361
FM^{10}	510	2,275	±23	1576.078	0.741	2.958	±0.032	730.266	±1.310
FM^9DH	**375**	**1,052**	±12	1385.684	**0.539**	**1.545**	±0.026	706.716	±1.267

Table 4.5: DAO map set results. Md, Mean, and Conf columns under the Expansions (or Time) column group are the median, mean, and 95% confidence interval on the mean of the number of expansions (or the mean expansions' times or speed). PCMean, and Node/s are the mean of the precomputation times and the number of nodes expanded per second. All times are in milliseconds.

all map sets. HE has made using multiple FastMap together possible, which was not effective before. HE also shows its strength when used for multiple FastMaps; $8 * FM^2DH$ performs as the best heuristic in the Rooms map set, $(50 \times 50 \times 50, E(1))$ graph, $(19 \times 19 \times 19 \times 19, E(1))$ graph, and almost $(19 \times 19 \times 19 \times 19, RE(1_1.5))$ graph. Also, HE performs best in mean in DAO and Maze maps.

The reason why each heuristic performs better than the others needs a deeper study. However, Sturtevant's (2012) work [22] has a measure for the underlying dimension of the map sets, and it seems like in maps with lower underlying dimensions like DAO and Mazes [22], general FastMap embeddings perform better. Also, it seems like in maps with higher underlying dimensions like SC1, random maps [22], and multi-dimensional graphs DH^n performs better. We also could use multiple FastMap embeddings to perform better on maps with high dimensions like Rooms and 4D RE (1,1.5).

4.4 Captured Heuristic

As shown in the results of Section 4.3, using differential heuristic as the last dimension of general FastMap embedding performed better than the FastMap embedding itself in all maps. The following experiment verifies using differential heuristic as the last dimension captures more heuristic (although the more the heuristic is captured, does not mean that it necessarily performs better).

Heuristics	Expansions			Time					
	Md	Mean	Conf	PCMean	Md	Mean	Conf	Node/ms	Conf
Octile	26,389	48,538	±269	0	49.819	83.221	±0.431	**529.964**	±0.653
DH^{10}	1,927	**6,176**	±46	**11589.274**	**3.963**	**9.172**	±0.061	507.759	±0.987
FM^{10}	4,292	18,997	±151	24169.420	7.775	29.517	±0.225	501.973	±0.985
FM^9DH	**1,618**	15,328	±139	24143.241	**3.911**	22.953	±0.202	494.222	±1.007

Table 4.6: SC1 map set results. Md, Mean, and Conf columns under the Expansions (or Time) column group are the median, mean, and 95% confidence interval on the mean of the number of expansions (or the mean expansions' times or speed). PCMean, and Node/s are the mean of the precomputation times and the number of nodes expanded per second. All times are in milliseconds.

Heuristics	Pivot	DAO			SC1			Random			Rooms			Mazes		
		Md	Mean	C	Md	Mean	C	Md	Mean	C	Md	Mean	C	Md	Mean	C
DH^{10}	FAR	503	1,343	11	1,911	**6,251**	48	**3,057**	**5,155**	28	3,805	**6,927**	57	6,422	9,815	26
FM^{10}	ED	513	2,283	23	4,265	19,068	153	5,974	12,472	84	9,656	19,623	169	9,046	15,462	45
FM^9DH	ED	**381**	1,095	14	1,639	16,089	144	4,085	11,146	84	6,252	17,504	169	4,695	7,107	19
FM^9DH	HE	**375**	1,038	12	**1,532**	14,507	135	4,107	11,075	83	5,921	16,275	155	**4,654**	**7,042**	19
max[DH^5, FM^5]	ED	466	1,322	11	2,273	8,413	69	3,616	6,272	36	4,687	8,962	76	6,742	10,246	27
max[DH^5, FM^4DH]	ED	407	1,038	9	1,600	7,469	65	3,285	5,904	34	3,851	8,207	73	6,357	9,634	25
max[FM^4DH, DH^5]	HE	403	**979**	8	1,556	7,411	65	3,279	5,899	34	3,667	7,883	69	6,319	9,562	25
max[DH^5, FM^4DH]	HE	405	**987**	8	1,545	7,385	65	3,135	5,725	34	**3,595**	7,745	69	6,243	9,452	24
DH^{24}	FAR	382	866	6	987	**3,096**	23	**1,799**	**2,940**	16	2,136	**3,729**	30	4,592	6,938	18
FM^{24}	ED	398	1,576	18	1,970	16,557	145	5,236	11,877	83	8,197	18,455	165	5,562	10,515	32
$FM^{23}DH$	ED	344	913	11	1,219	15,019	139	3,508	10,803	83	5,916	17,404	170	**3,336**	**4,843**	12
max[FM^{12}, DH^{12}]	ED	341	693	5	876	3,918	35	2,202	4,067	24	2,453	4,866	42	4,420	6,688	17
max[$FM^{11}DH$, DH^{12}]	ED	**320**	**577**	4	**755**	3,540	34	2,009	3,850	23	2,018	4,466	41	4,230	6,290	16
max[$FM^{11}DH$, DH^{12}]	HE	**318**	**569**	4	**756**	3,406	33	1,993	3,842	23	1,935	4,401	41	4,217	6,264	16
max[$8{\times}FM^2DH$]	HE	412	1,185	14	946	3,755	34	2,066	4,073	26	**1,590**	**3,750**	39	6,489	10,139	27

Table 4.7: Results of running A^* with different heuristics on different map sets. Md, Mn, and c columns are the median, mean, and 95% confidence interval on the mean of the number of expansions.

Heuristics	Pivot	50^3,E(1)			50^3,RE(1-1.5)			50^3,RE(1-10)		
		Md	Mn	C	Md	Mn	C	Md	Mn	C
FM^{24}	ED	14593	22380	1395	15074	23324	1419	12989	18267	1111
DH^{24}	FAR	49	61	8	**596**	**1104**	87	**779**	**1288**	93
max[FM^{12}, DH^{12}]	ED	49	109	26	898	1948	203	1261	2110	159
$FM^{23}DH$	ED	13700	22007	1437	14300	23171	1446	11841	18067	1142
max[$FM^{11}DH$, DH^{12}]	ED	49	109	26	876	1921	202	1204	2066	158
max[$FM^{11}DH$, DH^{12}]	HE	50	220	59	870	1859	177	1200	2050	160
max[$12 \times FMDH$]	HE	49	49	1	982	1722	131	1231	1976	143
max[$8 \times FM^2DH$]	HE	**49**	**49**	1	954	1681	129	1282	1960	136

Table 4.8: Results of running A^* with different heuristics on 3D graphs with different edge costs. Md, Mn, and c columns are the median, mean, and 95% confidence interval on the mean of the number of expansions.

Heuristics	Pivot	19^4,E(1)			19^4,RE(1-1.5)			19^4,RE(1-10)		
		Md	Mn	C	Md	Mn	C	Md	Mn	C
FM^{24}	ED	14435	22563	1444	16568	25155	1561	12385	19197	1195
DH^{24}	FAR	**27**	245	64	**381**	**1020**	121	**607**	**1124**	526
max[FM^{12}, DH^{12}]	ED	127	1235	218	614	1858	225	987	1993	171
$FM^{23}DH$	ED	14006	22732	1453	16332	24097	1519	12424	19200	1216
max[$FM^{11}DH$, DH^{12}]	ED	101	1151	209	616	1822	221	929	1730	144
max[$FM^{11}DH$, DH^{12}]	HE	66	1051	206	554	1673	212	933	1952	172
max[$12\times FMDH$]	HE	26	26	1	433	907	79	1017	1829	148
max[$8\times FM^2DH$]	HE	**26**	**26**	1	400	**847**	75	977	1659	124

Table 4.9: Results of running A^* with different heuristics on 4D graphs with different edge costs. Md, Mn, and c columns are the median, mean, and 95% confidence interval on the mean of the number of expansions.

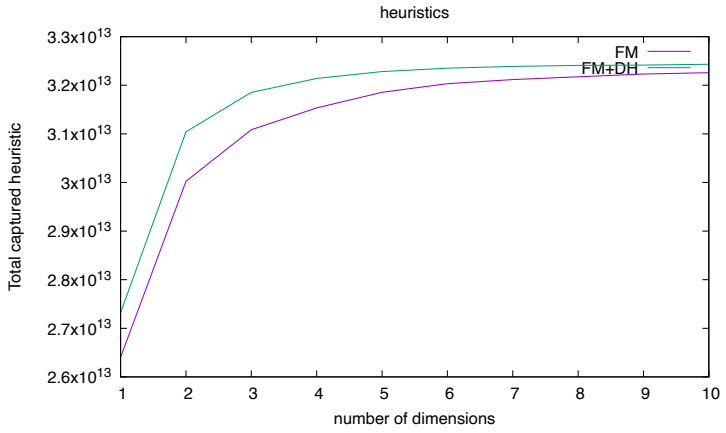


Figure 4.2: Total captured heuristics in all DAO maps by FM and FMDH.

For a single map, we calculate the total captured heuristic between all vertices pairs of the map. Then we sum up all the captured heuristics among all the DAO maps. This is illustrated in Figure 4.2, where the x-axis is the number of dimensions, and the y-axis shows the total captured heuristic among all the maps. For instance, the points with $x = 5$ show the results for FM^4DH and $DH5$. As shown in Figure 4.2, the heuristic captured by $FM^4 + DH$ is greater than FM^{10}. This shows using DH instead of FM as the last dimension of FM^5 captures more heuristics than the last five dimensions of FM^{10}.

Chapter 5

Conclusion and Future Work

In this thesis, we have improved the FastMap embedding. Our contribution can be summarized in the below list:

- "FastMap is an additive heuristic" is proven.
- FastMap embedding is generalized by generalizing its embedding function and pivot selection method.
- Several new embedding functions and pivot selection methods are introduced.
- The introduced pivot selection methods make using multiple FaastMap together possible, which was not effective before.
- It is shown that the results of the FastMap paper [2] are not sufficient and misleading.
- New and old heuristics are compared on a wide range of maps, including muti-dimensional graphs.
- It is shown using differential heuristic as the last dimension of FastMap improves its heuristic.
- Both subset selection and HE pivot selection methods improve the heuristics.

- Our new heuristics perform best in multiple maps like DAO, Mazes, Rooms, 3D (50^3 ,E(1)), 4D (19^4 ,E(1)), and almost in 4D (19^4, RE(1,1.5)) graph.

- It is shown that using DH as the last dimension of FM^4DH general FastMap embedding captures more heuristics than the last five dimensions of FM^{10}.

The future work can be summarized in the below items:

- **Different Configurations of the Introduced Heuristics**: In Chapter 4, we gave our results for selected heuristics. However, we have introduced several methods, and each can have different configurations. For instance, the Heuristic Error method for pivot selection uses a formula that has two arbitrary constants. We only provided results for one configuration, which we thought would be best by testing it through a few maps. Or, there are embedding functions like the shrinking function, which we did not provide results for since, in a few maps, it did not perform well. However, it is possible that different configurations of it perform better.

- **Different Map Sets for the Experiments in Chapter 4**: There are other maps and graphs that can be used for the experiment. For example, there are City maps in the MovingAI repository which can be used. Or, graphs with different ranges of edge costs can be used for testing the heuristics.

- **Deeper study**: The introduced heuristics and their results can be studied deeper to gain a better understanding of them. For instance, it needs to be studied why a heuristic, like DH^n, performs better in a particular type of map, like maps with higher dimensions SC1. Or, it needs to be studied why using the HE method in all dimensions of an embedding does not perform well as we tested on a few maps.

- **New Methods**: New embedding functions and pivot selection methods can be introduced. Even a different admissible FastMap using l_2 embedding may possibly be introduced.

- **New Experiment**: The experiments on the multi-dimensional graphs in chapter 4 do not have a standard benchmark. Since the graphs are a good representation of maps with higher dimensions, testing on them can help with understanding heuristics better. Therefore, creating a standard benchmark for graphs may be useful.

References

[1] S. Koenig A. Li P. Stuckey and S. Kumar. "A FastMap-Based Algorithm for Block Modeling." In: *In Proceedings of the International Conference on the Integration of Constraint Programming, Artificial Intelligence, and Operations Research (CPAIOR)*. 2022.

[2] Liron Cohen et al. "The FastMap Algorithm for Shortest Path Computations." In: *Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence, IJCAI-18*. International Joint Conferences on Artificial Intelligence Organization, July 2018, pp. 1427–1433.

[3] *CSC2414 - Metric Embeddings Lecture 1: A brief introduction to metric embeddings, examples and motivation*. `http://www.cs.toronto.edu/~avner/teaching/S6-2414/LN1.pdf`. Accessed: 2022-09-29.

[4] Joseph C Culberson and Jonathan Schaeffer. "Pattern databases." In: *Computational Intelligence* 14.3 (1998), pp. 318–334.

[5] Rina Dechter and Judea Pearl. "Generalized best-first search strategies and the optimality of A." In: *Journal of the ACM (JACM)* 32.3 (1985), pp. 505–536.

[6] Edsger W Dijkstra et al. "A note on two problems in connexion with graphs." In: *Numerische mathematik* 1.1 (1959), pp. 269–271.

[7] Ariel Felner, Richard E Korf, and Sarit Hanan. "Additive pattern database heuristics." In: *Journal of Artificial Intelligence Research* 22 (2004), pp. 279–318.

[8] Ariel Felner, Nathan R Sturtevant, and Jonathan Schaeffer. "Abstraction-Based Heuristics with True Distance Computations." In: *SARA*. Citeseer. 2009.

[9] Andrew V Goldberg and Chris Harrelson. "Computing the shortest path: A search meets graph theory." In: *SODA*. Vol. 5. Citeseer. 2005, pp. 156–165.

[10] Andrew V Goldberg, Haim Kaplan, and Renato F Werneck. "Better landmarks within reach." In: *International Workshop on Experimental and Efficient Algorithms*. Springer. 2007, pp. 38–51.

[11] Peter E Hart, Nils J Nilsson, and Bertram Raphael. "A formal basis for the heuristic determination of minimum cost paths." In: *IEEE transactions on Systems Science and Cybernetics* 4.2 (1968), pp. 100–107.

[12] Cong Hu et al. "Speeding up FastMap for Pathfinding on Grid Maps." In: *2019 IEEE International Conference on Mechatronics and Automation (ICMA)*. IEEE. 2019, pp. 2501–2506.

[13] Richard E. Korf. "Depth-first iterative-deepening: An optimal admissible tree search." In: *Artificial Intelligence* 27.1 (1985), pp. 97–109.

[14] Jiaoyang Li et al. "Using fastmap to solve graph problems in a euclidean space." In: *Proceedings of the international conference on automated planning and scheduling*. Vol. 29. 2019, pp. 273–278.

[15] Jirı Matoušek. *Lecture notes on metric embeddings*. Tech. rep. Technical report, ETH Zürich, 2013.

[16] TS Eugene Ng and Hui Zhang. "Predicting Internet network distance with coordinates-based approaches." In: *Proceedings. Twenty-First Annual Joint Conference of the IEEE Computer and Communications Societies*. Vol. 1. IEEE. 2002, pp. 170–179.

[17] Judea Pearl. *Heuristics: intelligent search strategies for computer problem solving*. Addison-Wesley Longman Publishing Co., Inc., 1984.

[18] Ira Pohl. "Heuristic search viewed as path finding in a graph." In: *Artificial intelligence* 1.3-4 (1970), pp. 193–204.

[19] Steve Rabin and Nathan R Sturtevant. "Combining bounding boxes and jps to prune grid pathfinding." In: *Thirtieth AAAI Conference on Artificial Intelligence*. 2016.

[20] Chris Rayner, Nathan Sturtevant, and Michael Bowling. "Subset selection of search heuristics." In: *Twenty-Third International Joint Conference on Artificial Intelligence*. 2013.

[21] D Chris Rayner, Michael Bowling, and Nathan Sturtevant. "Euclidean heuristic optimization." In: *Proceedings of the AAAI Conference on Artificial Intelligence*. Vol. 25. 1. 2011.

[22] Nathan R Sturtevant. "Benchmarks for grid-based pathfinding." In: *IEEE Transactions on Computational Intelligence and AI in Games* 4.2 (2012), pp. 144–148.

[23] Nathan R. Sturtevant et al. "Memory-Based Heuristics for Explicit State Spaces." In: *Proceedings of the 21st International Joint Conference on Artificial Intelligence*. IJCAI'09. Pasadena, California, USA: Morgan Kaufmann Publishers Inc., 2009, pp. 609–614.

[24] Fan Yang et al. "A general theory of additive state space abstractions." In: *Journal of Artificial Intelligence Research* 32 (2008), pp. 631–662.