The FastMap Algorithm for Shortest Path Computations

Liron Cohen[1], **Tansel Uras**[1], **Shiva Jahangiri**[2], **Aliyah Arunasalam**[1],
Sven Koenig[1] **and T. K. Satish Kumar**[1]

[1]University of Southern California

[2]University of California, Irvine

{lironcoh, turas, arunasal, skoenig}@usc.edu, shivaj@uci.edu, tkskwork@gmail.com

Abstract

We present a new preprocessing algorithm for embedding the nodes of a given edge-weighted undirected graph into a Euclidean space. The Euclidean distance between any two nodes in this space approximates the length of the shortest path between them in the given graph. Later, at runtime, a shortest path between any two nodes can be computed with an A* search using the Euclidean distances as heuristic. Our preprocessing algorithm, called FastMap, is inspired by the data-mining algorithm of the same name and runs in near-linear time. Hence, FastMap is orders of magnitude faster than competing approaches that produce a Euclidean embedding using Semidefinite Programming. FastMap also produces admissible and consistent heuristics and therefore guarantees the generation of shortest paths. Moreover, FastMap applies to general undirected graphs for which many traditional heuristics, such as the Manhattan Distance heuristic, are not well defined. Empirically, we demonstrate that A* search using the FastMap heuristic is competitive with A* search using other state-of-the-art heuristics, such as the Differential heuristic.

1 Introduction and Related Work

Shortest path computations commonly occur in the inner procedures of many AI programs. In video games, for example, a large fraction of CPU cycles is spent on shortest path computations [Uras and Koenig, 2015]. Many other tasks in AI, including motion planning, temporal reasoning, and decision making [Russell and Norvig, 2009], also involve finding and reasoning about shortest paths. While Dijkstra's algorithm [Dijkstra, 1959] can be used to compute shortest paths in polynomial time, speeding up shortest path computations allows one to solve the aforementioned tasks faster. One way to do that is to use A* search with an informed heuristic [Hart *et al.*, 1968].

A perfect heuristic is one that returns the true distance between any two nodes in a given graph. A* with such a heuristic and proper tie-breaking is guaranteed to expand nodes only on a shortest path between the given start and goal nodes. In general, computing the perfect heuristic between two nodes is as hard as computing the shortest path between them. Hence, A* search benefits from a perfect heuristic only if it is computed offline. However, precomputing all pairwise distances is not only time-intensive but also requires a prohibitive $O(N^2)$ memory where N is the number of nodes. The memory requirements for storing all-pairs shortest paths data can be somewhat addressed through compression [Botea and Harabor, 2013; Strasser *et al.*, 2015].

Existing methods for preprocessing a given graph (without precomputing all pairwise distances) can be grouped into the following categories: Hierarchical abstractions that yield suboptimal paths have been used to reduce the size of the search space by abstracting groups of nodes [Botea *et al.*, 2004; Sturtevant and Buro, 2005]. More informed heuristics [Björnsson and Halldórsson, 2006; Cazenave, 2006; Sturtevant *et al.*, 2009] focus A* searches better, resulting in fewer expanded nodes. Hierarchies can also be used to derive heuristics during the search [Leighton *et al.*, 2008; Holte *et al.*, 1994]. Dead-end detection and other pruning methods [Björnsson and Halldórsson, 2006; Goldenberg *et al.*, 2010; Pochter *et al.*, 2010] identify areas of the graph that do not need to be searched to find shortest paths. Search with contraction hierarchies [Geisberger *et al.*, 2008] is an optimal hierarchical method, where every level of the hierarchy contains only a single node. It has been shown to be efficient on road networks but seems to be less efficient on graphs with higher branching factors, such as grid-based game maps [Storandt, 2013]. N-level graphs [Uras and Koenig, 2014], constructed from undirected graphs by partitioning the nodes into levels, also allow for significant pruning during the search.

A different approach that does not rely on preprocessing of the graph is to use some notion of a geometric distance between two nodes as a heuristic of the distance between them. One such heuristic for gridworlds is the Manhattan Distance heuristic.[1] For many gridworlds, A* search using

---

[1]In a 4-neighbor 2D gridworld, for example, the Manhattan Distance between two cells (x_1, y_1) and (x_2, y_2) is $|x_1-x_2|+|y_1-y_2|$. Generalizations exist for 8-neighbor 2D and 3D gridworlds.

the Manhattan Distance heuristic outperforms Dijkstra's algorithm. However, in complicated 2D/3D gridworlds like mazes, the Manhattan Distance heuristic may not be sufficiently informed to focus A* searches effectively. Another issue associated with Manhattan Distance-like heuristics is that they are not well defined for general graphs.[2] For a graph that cannot be conceived in a geometric space, there is no closed-form formula for a "geometric" heuristic for the distance between two nodes because there are no coordinates associated with them.

For a graph that does not already have a geometric embedding in Euclidean space, a preprocessing algorithm can be used to generate one. As described before, at runtime, A* search would then use the Euclidean distance between any two nodes in this space as an estimate for the distance between them in the given graph. One such approach is Euclidean Heuristic Optimization (EHO) [Rayner *et al.*, 2011]. EHO guarantees admissiblility and consistency and therefore generates shortest paths. However, it requires solving a Semidefinite Program (SDP). SDPs can be solved in polynomial time [Vandenberghe and Boyd, 1996]. EHO leverages additional structure to solve them in cubic time. Still, a cubic preprocessing time limits the size of the graphs that are amenable to this approach.

The Differential heuristic is another state-of-the-art approach that has the benefit of a near-linear runtime. However, unlike the approach in [Rayner *et al.*, 2011], it does not produce an explicit Euclidean embedding. In the preprocessing phase of the Differential heuristic approach, some nodes of the graph are chosen as pivot nodes. The distances between each pivot node and every other node are precomputed and stored [Sturtevant *et al.*, 2009]. At runtime, the heuristic between two nodes a and b is given by $\max_p |d(a, p) - d(p, b)|$, where p is a pivot node and $d(\cdot, \cdot)$ is the precomputed distance. The preprocessing time is linear in the number of pivots times the size of the graph. The required space is linear in the number of pivots times the number of nodes, although a more succinct representation is presented in [Goldenberg *et al.*, 2011]. Similar preprocessing techniques are used in Portal-Based True Distance heuristics [Goldenberg *et al.*, 2010].

In this paper, we present a new preprocessing algorithm, called FastMap, that produces an explicit Euclidean embedding while running in near-linear time. It therefore has the benefits of the small preprocessing time of the Differential heuristic approach and of producing an embedding from which a heuristic between two nodes can be quickly computed using a closed-form formula. Our preprocessing algorithm, dubbed FastMap, is inspired by the data-mining algorithm of the same name [Faloutsos and Lin, 1995]. It is orders of magnitude faster than SDP-based approaches for producing Euclidean embeddings. FastMap also produces admissible and consistent heuristics and therefore guarantees the generation of shortest paths.

The FastMap heuristic has several advantages: First, it is defined for general (undirected) graphs. Second, we observe

empirically that, in gridworlds, A* using the FastMap heuristic runs faster than A* using the Manhattan or Octile distance heuristics. A* using the FastMap heuristic runs equally fast or faster than A* using the Differential heuristic, with the same memory resources. The (explicit) Euclidean embedding produced by FastMap also has representational benefits like recovering the underlying manifolds of the graph and/or visualizing them. Moreover, we observe that the FastMap and Differential heuristics have complementary strengths and can be easily combined to generate an even more informed heuristic.

2 The Origin of FastMap

The FastMap algorithm [Faloutsos and Lin, 1995] was introduced in the data-mining community for automatically generating geometric embeddings of abstract objects. For example, if we are given objects in form of long DNA strings, multimedia datasets such as voice excerpts or images or medical datasets such as ECGs or MRIs, there is no geometric space in which these objects can be naturally visualized. However, there is often a well defined distance function between every pair of objects. For example, the *edit distance*[3] between two DNA strings is well defined although an individual DNA string cannot be conceptualized in geometric space.

Clustering techniques, such as the k-means algorithm, are well studied in machine learning [Alpaydin, 2010] but cannot be applied directly to domains with abstract objects because they assume that objects are described as points in geometric space. FastMap revives their applicability by first creating a Euclidean embedding for the abstract objects that approximately preserves the pairwise distances between them. Such an embedding also helps to visualize the abstract objects, for example, to aid physicians in identifying correlations between symptoms from medical records.

The data-mining FastMap gets as input a complete undirected edge-weighted graph $G = (V, E)$. Each node $v_i \in V$ represents an abstract object O_i. Between any two nodes v_i and v_j there is an edge $(v_i, v_j) \in E$ with weight $D(O_i, O_j)$ that corresponds to the given distance between objects O_i and O_j. A Euclidean embedding assigns to each object O_i a K-dimensional point $p_i \in \mathbb{R}^K$. A good Euclidean embedding is one in which the Euclidean distance between any two points p_i and p_j closely approximates $D(O_i, O_j)$.

One of the early approaches for generating such an embedding is based on the idea of *multi-dimensional scaling* (MDS) [Torgerson, 1952]. Here, the overall distortion of the pairwise distances is measured in terms of the "energy" stored in "springs" that connect each pair of objects. MDS, however, requires $O(|V|^2)$ time and hence does not scale well in practice. On the other hand, FastMap [Faloutsos and Lin, 1995] requires only linear time. Both methods embed the objects in a K-dimensional space for a user-specified K.

FastMap works as follows: In the very first iteration, it heuristically identifies the farthest pair of objects O_a and O_b in linear time. It does this by initially choosing a random object O_b and then choosing O_a to be the farthest object

---

[3]The edit distance between two strings is the minimum number of insertions, deletions or substitutions that are needed to transform one to the other.
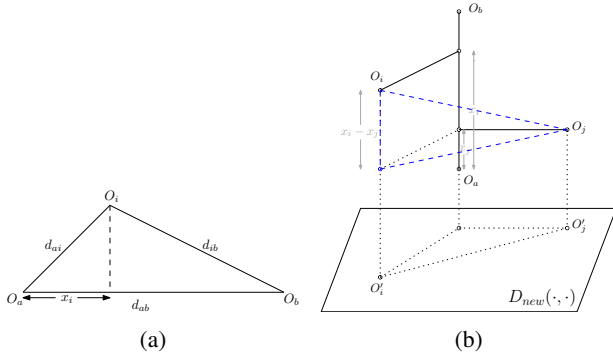
Figure 1: (a) The three sides of a triangle define its entire geometry. In particular, $x_i = (d_{ai}^2 + d_{ab}^2 - d_{ib}^2)/(2d_{ab})$. (b) Shows a geometric conceptualization of the recursive step in FastMap. In particular, $D_{new}(O_i', O_j')^2 = D(O_i, O_j)^2 - (x_i - x_j)^2$.

away from O_b. It then reassigns O_b to be the farthest object away from O_a. Once O_a and O_b are determined, every other object O_i defines a triangle with sides of lengths $d_{ai} = D(O_a, O_i)$, $d_{ab} = D(O_a, O_b)$ and $d_{ib} = D(O_i, O_b)$. Figure 1(a) shows this triangle. The sides of the triangle define its entire geometry, and the projection of O_i onto O_aO_b is given by $x_i = (d_{ai}^2 + d_{ab}^2 - d_{ib}^2)/(2d_{ab})$. FastMap sets the first coordinate of p_i, the embedding of object O_i, to x_i. In particular, the first coordinate of p_a is $x_a = 0$ and of p_b is $x_b = d_{ab}$. Computing the first coordinates of all objects takes only linear time since the distance between any two objects O_i and O_j for $i, j \notin \{a, b\}$ is never computed.

In the subsequent $K - 1$ iterations, the same procedure is followed for computing the remaining $K - 1$ coordinates of each object. However, the distance function is adapted for different iterations. For example, for the first iteration, the coordinates of O_a and O_b are 0 and d_{ab}, respectively. Because these coordinates fully explain the true distance between them, from the second iteration onwards, the rest of p_a and p_b's coordinates should be identical. Intuitively, this means that the second iteration should mimic the first one on a hyperplane that is perpendicular to the line O_aO_b. Figure 1(b) explains this intuition. Although the hyperplane is never constructed explicitly, its conceptualization implies that the distance function for the second iteration should be changed in the following way: $D_{new}(O_i', O_j')^2 = D(O_i, O_j)^2 - (x_i - x_j)^2$. Here, O_i' and O_j' are the projections of O_i and O_j, respectively, onto this hyperplane, and D_{new} is the new distance function.

3 FastMap for Shortest Path Computations

In this section, we provide the high-level ideas for how to adapt the data-mining FastMap algorithm to shortest path computations. In the shortest path computation problem, we are given a non-negative edge-weighted undirected graph $G = (V, E)$ along with a start node v_s and a goal node v_g. As a preprocessing technique, we can embed the nodes of G in a Euclidean space. As A* searches for a shortest path from v_s to v_g, it can use the Euclidean distance from $v \in V$ to v_g as a heuristic for v. The number of node expansions of A*

search depends on the informedness of the heuristic which, in turn, depends on the ability of the embedding to preserve the pairwise distances.

The idea is to view the nodes of G as the objects to be embedded in Euclidean space. As such, the data-mining FastMap algorithm cannot directly be used for generating an embedding in linear time. The data-mining FastMap algorithm assumes that the distance d_{ij} between two objects O_i and O_j can be computed in constant time, independent of the number of objects. Computing the distance between two nodes depends on the size of the graph. Another problem is that the Euclidean distances may not satisfy important properties such as *admissibility* or *consistency*. Admissibility guarantees that A* finds shortest paths while consistency allows A* to avoid re-expansions of nodes as well.

The first issue of having to retain (near-)linear time complexity can be addressed as follows: In each iteration, after we identify the farthest pair of nodes O_a and O_b, the distances d_{ai} and d_{ib} need to be computed for *all* other nodes O_i. Computing d_{ai} and d_{ib} for any single node O_i can no longer be done in constant time but requires $O(|E| + |V| \log |V|)$ time instead [Fredman and Tarjan, 1984]. However, since we need to compute these distances for all nodes, computing two shortest path trees rooted at nodes O_a and O_b yields all necessary distances. The complexity of doing so is also $O(|E| + |V| \log |V|)$, which is only linear in the size of the graph.[4] The amortized complexity for computing d_{ai} and d_{ib} for any single node O_i is therefore near-constant time.

The second issue of having to generate a consistent (and thus admissible) heuristic is formally addressed in Theorem 1. The idea is to use L_1 distances instead of L_2 distances in each iteration of FastMap. The mathematical properties of the L_1 distance can be used to prove that admissibility and consistency hold irrespective of the dimensionality of the embedding.

Algorithm 1 presents data-mining FastMap adapted to the shortest path computation problem. The input is an edge-weighted undirected graph $G = (V, E, w)$ along with two user-specified parameters K_{max} and ϵ. K_{max} is the maximum number of dimensions allowed in the Euclidean embedding. It bounds the amount of memory needed to store the Euclidean embedding of any node. ϵ is the threshold that marks a point of diminishing returns when the distance between the farthest pair of nodes becomes negligible. The output is an embedding $p_i \in \mathbb{R}^K$ (with $K \leq K_{max}$) for each node $v_i \in V$.

The algorithm maintains a working graph $G' = (V, E, w')$ initialized to G. The nodes and edges of G' are always identical to those of G but the weights on the edges of G' change with every iteration. In each iteration, the farthest pair (n_a, n_b) of nodes in G' is heuristically identified in near-linear time (line 4). The Kth coordinate $[p_i]_K$ of each node v_i is computed using a formula similar to that for x_i in Figure 1(a). However, that formula is modified to $(d_{ai} + d_{ab} - d_{ib})/2$ to ensure admissibility and consistency of the heuristic. In each iteration, the weight of each edge

---

[4]unless $|E| = O(|V|)$, in which case the complexity is near-linear in the size of the input because of the $\log |V|$ factor

ALGORITHM 1: Shows the FastMap algorithm. $G = (V, E, w)$ is a non-negative edge-weighted undirected graph; K_{max} is the user-specified upper bound on the dimensionality; ϵ is a user-specified threshold; $K \leq K_{max}$ is the dimensionality of the computed embedding; p_i is the Euclidean embedding of node $v_i \in V$. Line 11 is equivalent to $w'(u, v) = w(u, v) - \|p_u - p_v\|_1$.

Input: $G = (V, E, w)$, K_{max} and ϵ.
Output: K and $p_i \in \mathbb{R}^K$ for all $v_i \in V$.

1   $w' = w$; $K = 1$;
2   **while** $K_{max} > 0$ **do**
3   Let $G' = (V, E, w')$;
4     $(n_a, n_b) \leftarrow$ GetFarthestPair(G');
5   Compute shortest path trees rooted at n_a and n_b on G' to obtain d_{ab}, d_{ai} and d_{ib} for all $v_i \in V$;
6   **if** $d_{ab} < \epsilon$ **then**
7   Break;
8   **for** *each* $v \in V$ **do**
9       $[p_v]_K = (d_{av} + d_{ab} - d_{vb})/2$
10   **for** *each edge* $(u, v) \in E$ **do**
11      $w'(u, v) = w'(u, v) - |[p_u]_K - [p_v]_K|$;
12    $K = K + 1$; $K_{max} = K_{max} - 1$;

is decremented to resemble the update rule for D_{new} in Figure 1(b) (line 11). However, that update rule is modified to $w'(u, v) = w'(u, v) - |[p_u]_K - [p_v]_K|$ to use the L_1 distances instead of the L_2 distances.

The method GetFarthestPair(G') (line 4) computes shortest path trees in G' a small constant number of times, denoted by τ.[5] It therefore runs in near-linear time. In the first iteration, we assign n_a to be a random node. A shortest path tree rooted at n_a is computed to identify the farthest node from it. n_b is assigned to be this farthest node. In the next iteration, a shortest path tree rooted at n_b is computed to identify the farthest node from it. n_a is reassigned to be this farthest node. Subsequent iterations follow the same switching rule for n_a and n_b. The final assignments of nodes to n_a and n_b are returned after τ iterations. This entire process of starting from a randomly chosen node can be repeated a small constant number of times.[6]

Figure 2 shows the working of our algorithm on a small gridworld example.

3.1 Proof of Consistency

In this section, we prove the consistency of the FastMap heuristic. Since consistency implies admissibility, this also proves that A* with the FastMap heuristic returns shortest paths. We use the following notation in the proofs: w^i_{xy} is the weight on the edge between nodes x and y in the ith iteration; d^i_{xy} is the distance between nodes x and y in the ith iteration (using the weights w^i); p_x is the vector of coordinates produced for node x, and $[p_x]_j$ is its jth coordinate;[7]

---

[5] $\tau = 10$ in our experiments.
[6] This constant is also 10 in our experiments.
[7] The ith iteration sets the value of $[p_x]_i$.

h^i_{xy} is the FastMap heuristic between nodes x and y after i iterations. Note that h^i_{xy} is the L_1 distance between p_x and p_y at iteration i, that is $h^i_{xy} := \sum_{j=1}^{i} |[p_x]_j - [p_y]_j|$. We also define $\Delta^{i+1}_{xy} := d^i_{xy} - d^{i+1}_{xy}$. In the following proofs, we use the fact that $|A| + |B| \geq |A + B|$ and $|A| - |B| \leq |A - B|$.

Lemma 1. *For all x, y and i, $d^i_{xy} \geq 0$.*

Proof. We prove by induction that, in any iteration i, $w^i_{uv} \geq 0$ for all $(u, v) \in E$. Thus, the weight of each edge in the ith iteration is non-negative and therefore $d^i_{uv} \geq 0$ for all u, v. For the base case, $w^1_{uv} = w(u, v) \geq 0$. We assume that $w^i_{uv} \geq 0$ and show that $w^{i+1}_{uv} \geq 0$. Let n_a and n_b be the farthest pair of nodes identified in the ith iteration. From lines 9 and 11, $w^{i+1}_{uv} = w^i_{uv} - |(d^i_{au} - d^i_{av})/2 + (d^i_{vb} - d^i_{ub})/2|$. To show that $w^{i+1}_{uv} \geq 0$ we show that $w^i_{uv} \geq |(d^i_{au} - d^i_{av})/2 + (d^i_{vb} - d^i_{ub})/2|$. From the triangle inequality, for any node l, $d^i_{uv} + \min(d^i_{ul}, d^i_{lv}) \geq \max(d^i_{ul}, d^i_{lv})$. Therefore, $d^i_{uv} \geq |d^i_{ul} - d^i_{lv}|$ and thus also $2d^i_{uv} \geq |d^i_{ua} - d^i_{av}| + |d^i_{ub} - d^i_{bv}|$. This means that $d^i_{uv} \geq |d^i_{au} - d^i_{av}|/2 + |d^i_{vb} - d^i_{ub}|/2$. Therefore, $d^i_{uv} \geq |(d^i_{au} - d^i_{av})/2 + (d^i_{vb} - d^i_{ub})/2|$. This concludes the proof since $w^i_{uv} \geq d^i_{uv}$. $\square$

Lemma 2. *For all x, y and i, $\Delta^{i+1}_{xy} \geq |[p_x]_i - [p_y]_i|$.*

Proof. Let $\langle u_1 = x, \ldots, u_m = y \rangle$ be the shortest path from x to y in iteration i. By definition, $d^i_{xy} = \sum_{j=1}^{m-1} w^i_{u_j u_{j+1}}$ and $d^{i+1}_{xy} \leq \sum_{j=1}^{m-1} w^{i+1}_{u_j u_{j+1}}$. From line 11, $w^{i+1}_{u_j u_{j+1}} = w^i_{u_j u_{j+1}} - |[p_{u_j}]_i - [p_{u_{j+1}}]_i|$. Therefore, $\Delta^{i+1}_{xy} = d^i_{xy} - d^{i+1}_{xy} \geq \sum_{j=1}^{m-1} |[p_{u_j}]_i - [p_{u_{j+1}}]_i|$. This concludes the proof since $\sum_{j=1}^{m-1} |[p_{u_j}]_i - [p_{u_{j+1}}]_i| \geq |\sum_{j=1}^{m-1} [p_{u_j}]_i - [p_{u_{j+1}}]_i| = |[p_x]_i - [p_y]_i|$. $\square$

Lemma 3. *For all x, y, g and i, $d^1_{xy} + h^i_{yg} - h^i_{xg} \geq d^{i+1}_{xy}$.*

Proof. We prove the lemma by induction on i. The base case for $i = 1$ is implied by Lemma 2. We assume that $d^1_{xy} + h^i_{yg} - h^i_{xg} \geq d^{i+1}_{xy}$ and show $d^1_{xy} + h^{i+1}_{yg} - h^{i+1}_{xg} \geq d^{i+2}_{xy}$. We know that $h^{i+1}_{yg} - h^{i+1}_{xg} = h^i_{yg} - h^i_{xg} - (|[p_x]_{i+1} - [p_g]_{i+1}| - |[p_y]_{i+1} - [p_g]_{i+1}|)$. Since $|[p_x]_{i+1} - [p_g]_{i+1}| - |[p_y]_{i+1} - [p_g]_{i+1}| \leq |[p_x]_{i+1} - [p_y]_{i+1}|$, we have $h^{i+1}_{yg} - h^{i+1}_{xg} \geq h^i_{yg} - h^i_{xg} - |[p_x]_{i+1} - [p_y]_{i+1}|$. Hence, $d^1_{xy} + h^{i+1}_{yg} - h^{i+1}_{xg} \geq (d^1_{xy} + h^i_{yg} - h^i_{xg}) - |[p_x]_{i+1} - [p_y]_{i+1}|$. Using the inductive assumption, we get $d^1_{xy} + h^{i+1}_{yg} - h^{i+1}_{xg} \geq d^{i+1}_{xy} - |[p_x]_{i+1} - [p_y]_{i+1}|$. By definition, $d^{i+1}_{xy} = \Delta^{i+2}_{xy} + d^{i+2}_{xy}$. Substituting for d^{i+1}_{xy}, we get $d^1_{xy} + h^{i+1}_{yg} - h^{i+1}_{xg} \geq d^{i+2}_{xy} + (\Delta^{i+2}_{xy} - |[p_x]_{i+1} - [p_y]_{i+1}|)$. Lemma 2 shows that $\Delta^{i+2}_{xy} \geq |[p_x]_{i+1} - [p_y]_{i+1}|$, which concludes the proof. $\square$

Theorem 1. *The FastMap heuristic is consistent.*

Proof. For all x, y, g and i: From Lemma 3, we know $d^1_{xy} + h^i_{yg} - h^i_{xg} \geq d^{i+1}_{xy}$. From Lemma 1, we know $d^{i+1}_{xy} \geq 0$. Put together, we have $d^1_{xy} + h^i_{yg} \geq h^i_{xg}$. Finally, $h^i_{gg} = \sum_{j=1}^{i} |[p_g]_j - [p_g]_j| = 0$. $\square$
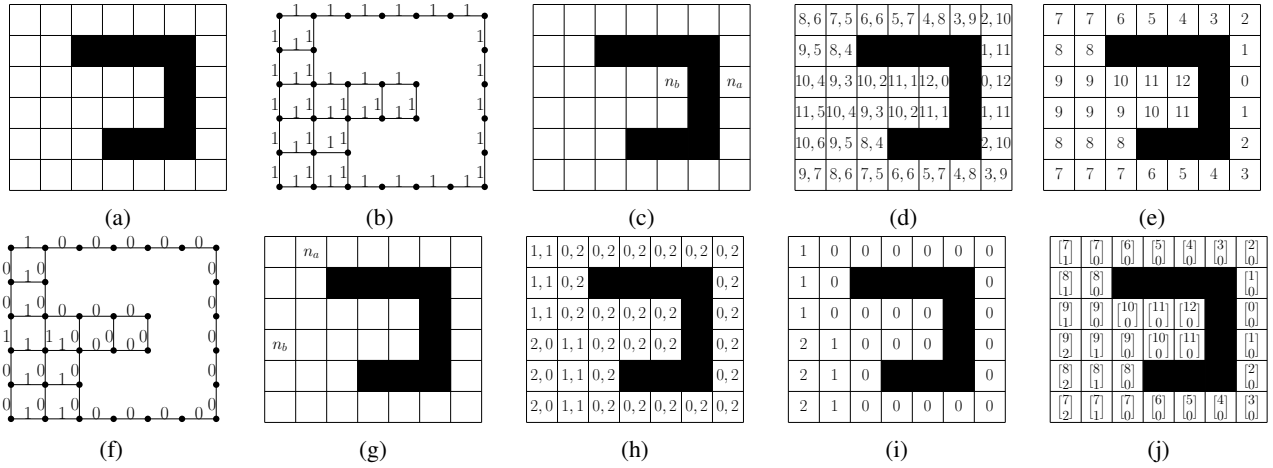
Figure 2: Illustrates the working of FastMap. (a) shows a 4-neighbor gridworld with obstacles in black. (b) shows the graphical representation of (a) with the original unit weights on the edges. (c) shows the identified farthest pair of nodes. (d) shows two numbers in each cell representing the distances from n_a and n_b, respectively. (e) shows the first coordinate produced for each cell. (f) shows new edge weights for the next iteration. (g), (h) and (i) correspond to (c), (d) and (e), respectively, in the second iteration. (j) shows the produced 2D embedding.

Theorem 2. *The informedness of the FastMap heuristic increases monotonically with the number of dimensions.*

Proof. This theorem follows from the fact that for any two nodes x and g, $h_{xg}^{i+1} = h_{xg}^i + |[p_x]_{i+1} - [p_g]_{i+1}| \geq h_{xg}^i$. □

4 Experimental Results

We performed experiments on many benchmark maps from [Sturtevant, 2012]. Figure 3 presents representative results. The FastMap heuristic (FM) and the Differential heuristic (DH) with equal memory resources[8] are compared against each other. In addition, we include the Octile heuristic (OCT) as a baseline, that also uses a closed-form formula for the computation of its heuristic.

We observe that, as the number of dimensions increases, (a) FM and DH perform better than OCT; (b) the median number of expanded nodes when using the FM heuristic decreases (which is consistent with Theorem 2); and (c) the median absolute deviation (MAD) of the number of expanded nodes when using the FM heuristic decreases. When FM's MADs are high, the variabilities can possibly be exploited in future work using Rapid Randomized Restart strategies.

FastMap also gives us a framework for identifying a point of diminishing returns with increasing dimensionality. This happens when the distance between the farthest pair of nodes stops being "significant". For example, such a point is observed in Figure 3(f) around dimensionality 5.[9]

In mazes, such as in Figure 3(g), A* using the DH heuristic outperforms A* using the FM heuristic. This leads us to believe that FM provides good heuristic guidance in domains that can be approximated with a low-dimensional manifold. This observation also motivates us to create a hybrid

FM+DH heuristic by taking the maximum of the two heuristics. Some relevant results are shown in Table 1. We use FM(K) to denote the FM heuristic with K dimensions and DM(K) to denote the DH heuristic with K pivots. For the results in Table 1, all heuristics have equal memory resources. We observe that the number of node expansions of A* using the FM(5)+DH(5) heuristic is always second best compared to A* using the FM(10) heuristic and A* using the DH(10) heuristic. On one hand, this decreases the percentages of instances on which it expands the least number of nodes (as seen in the second row of Table 1). But, on the other hand, its median number of node expansions is not far from that of the best technique in each breakdown.

5 Conclusions

We presented a near-linear time preprocessing algorithm, called FastMap, for producing a Euclidean embedding of a general edge-weighted undirected graph. At runtime, the Euclidean distances were used as heuristic by A* for shortest path computations. We proved that the FastMap heuristic is admissible and consistent, thereby generating shortest paths. FastMap produces the Euclidean embedding in near-linear time, which is significantly faster than competing approaches for producing Euclidean embeddings with optimality guarantees that run in cubic time. We also showed that it is competitive with other state-of-the-art heuristics derived in near-linear preprocessing time. However, FastMap has the combined benefits of requiring only near-linear preprocessing time and producing explicit Euclidean embeddings that try to recover the underlying manifolds of the given graphs.

---

[8]The dimensionality of the Euclidean embedding for FM matches the number of pivots in DH.

[9]The distances between the farthest pair of nodes, computed on line 4 of Algorithm 1, for the first 10 dimensions are: $\langle 581, 36, 22, 15, 14, 10, 6, 6, 5, 4 \rangle$.

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 1: Shows the median and MAD numbers of A* node expansions for different maps using three different heuristics with equal memory resources on 1000 random instances. FM(10) denotes the FastMap heuristic with 10 dimensions, DH(10) denotes the Differential heuristic with 10 pivots, and FM(5)+DH(5) is a combined heuristic which takes the maximum of a 5-dimensional FastMap heuristic and a 5-pivot Differential heuristic. The results are split into bins according to winners (along with their number of wins).
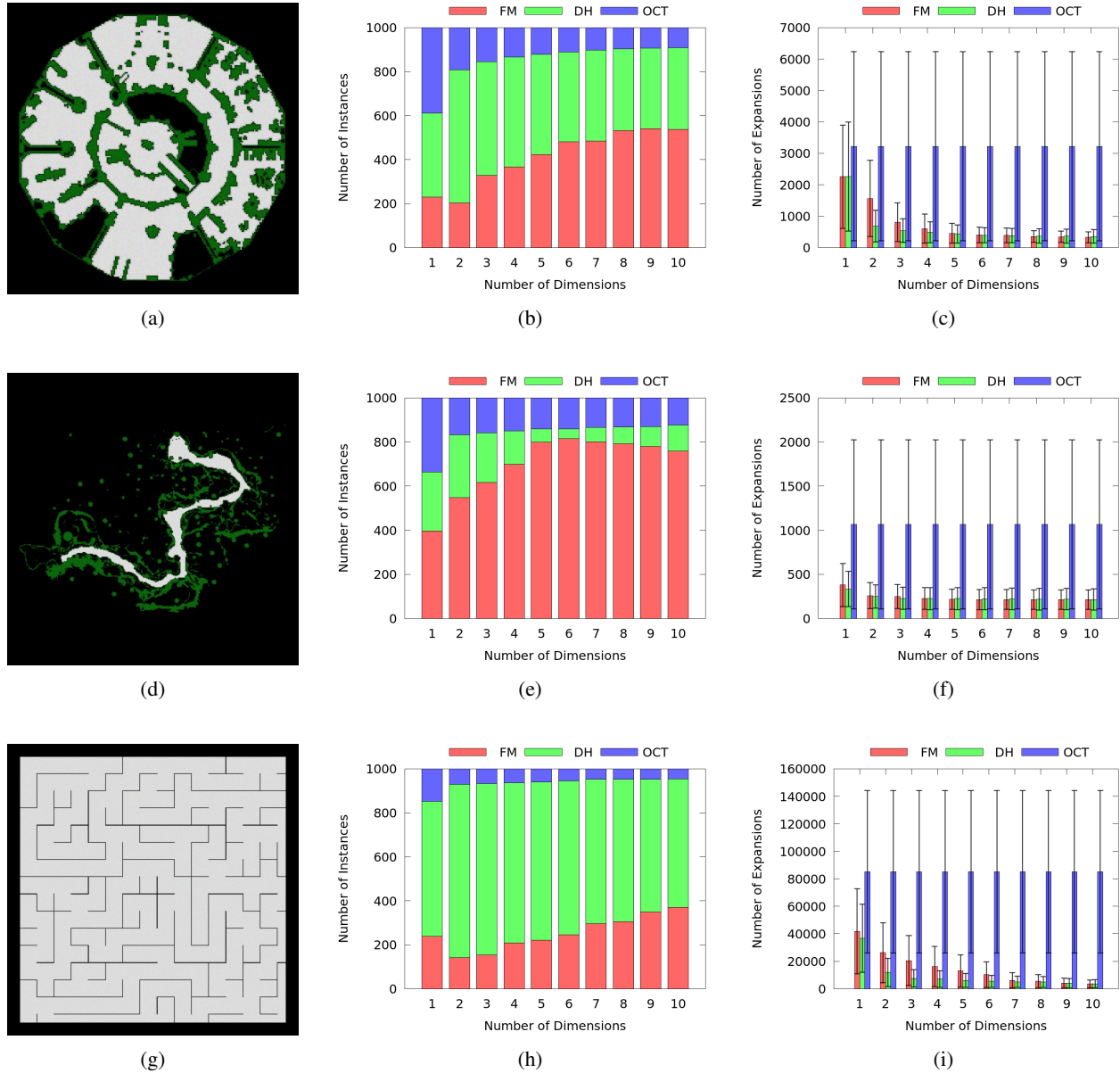

(a)   (b)   (c)

(d)   (e)   (f)

(g)   (h)   (i)

Figure 3: Shows empirical results on 3 maps from Bioware's Dragon Age: Origins. (a) is map 'lak503d' containing $17,953$ nodes and $33,781$ edges; (d) is map 'brc300d' containing $5,214$ nodes and $9,687$ edges; and (g) is map 'maze512-32-0' containing $253,840$ nodes and $499,377$ edges. In (b), the x-axis shows the number of dimensions for the FastMap heuristic (or the number of pivots for the Differential heuristic). The y-axis shows the number of instances (out of $1,000$) on which each technique expanded the least number of nodes. Each instance has randomly chosen start and goal nodes. (c) shows the median number of expanded nodes across all instances. Vertical error bars indicate the MADs. The figures in the second and third rows follow the same order. In the legends, "FM" denotes the FastMap heuristic, "DH" denotes the Differential heuristic, and "OCT" denotes the Octile heuristic.

References

[Alpaydin, 2010] Ethem Alpaydin. *Introduction to Machine Learning*. The MIT Press, 2nd edition, 2010.

[Björnsson and Halldórsson, 2006] Yngv Björnsson and Kári Halldórsson. Improved heuristics for optimal path-finding on game maps. In *Proceedings of the 6th AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*, 2006.

[Botea and Harabor, 2013] Adi Botea and Daniel Harabor. Path planning with compressed all-pairs shortest paths data. In *Proceedings of the 23rd International Conference on Automated Planning and Scheduling*, 2013.

[Botea *et al.*, 2004] Adi Botea, Martin Müller, and Jonathan Schaeffer. Near optimal hierarchical path-finding. *Journal of Game Development*, 1:7–28, 2004.

[Cazenave, 2006] Tristan Cazenave. Optimizations of data structures, heuristics and algorithms for path-finding on maps. In *Proceedings of the IEEE Symposium on Computational Intelligence and Games*, 2006.

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

[Faloutsos and Lin, 1995] Christos Faloutsos and King-Ip Lin. Fastmap: A fast algorithm for indexing, data-mining and visualization of traditional and multimedia datasets. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, 1995.

[Fredman and Tarjan, 1984] Michael Fredman and Robert Tarjan. Fibonacci heaps and their uses in improved network optimization algorithms. In *Proceedings of the 25th Annual Symposium on Foundations of Computer Science*, 1984.

[Geisberger *et al.*, 2008] Robert Geisberger, Peter Sanders, Dominik Schultes, and Daniel Delling. Contraction hierarchies: Faster and simpler hierarchical routing in road networks. In *Proceedings of the 7th International Conference on Experimental Algorithms*, 2008.

[Goldenberg *et al.*, 2010] Meir Goldenberg, Ariel Felner, Nathan Sturtevant, and Jonathan Schaeffer. Portal-based true-distance heuristics for path finding. In *Proceedings of the 3rd Annual Symposium on Combinatorial Search*, 2010.

[Goldenberg *et al.*, 2011] Meir Goldenberg, Nathan Sturtevant, Ariel Felner, and Jonathan Schaeffer. The compressed differential heuristic. In *Proceedings of the 25th AAAI Conference on Artificial Intelligence*, 2011.

[Hart *et al.*, 1968] Peter E. Hart, Nils J. Nilsson, and Bertram Raphael. A formal basis for the heuristic determination of minimum cost paths. *IEEE Transactions on Systems, Science, and Cybernetics*, SSC-4(2):100–107, 1968.

[Holte *et al.*, 1994] Robert Holte, Chris Drummond, M. B. Perez, Robert Zimmer, and Alan Macdonald. Searching with abstractions: A unifying framework and new high-performance algorithm. In *Proceedings of the 10th Canadian Conference on Artificial Intelligence*, 1994.

[Leighton *et al.*, 2008] Michael Leighton, Wheeler Ruml, and Robert Holte. Faster optimal and suboptimal hierarchical search. In *Proceedings of the 1st International Symposium on Combinatorial Search*, 2008.

[Pochter *et al.*, 2010] Nir Pochter, Aviv Zohar, Jeffery Rosenschein, and Ariel Felner. Search space reduction using swamp hierarchies. In *Proceedings of the 24th AAAI Conference on Artificial Intelligence*, 2010.

[Rayner *et al.*, 2011] Chris Rayner, Michael Bowling, and Nathan Sturtevant. Euclidean heuristic optimization. In *Proceedings of the 25th AAAI Conference on Artificial Intelligence*, 2011.

[Russell and Norvig, 2009] Stuart J. Russell and Peter Norvig. *Artificial Intelligence: A Modern Approach*. Prentice Hall, 2009.

[Storandt, 2013] Sabine Storandt. Contraction hierarchies on grid graphs. In *Proceedings of KI: the 36th Annual German Conference on Artificial Intelligence*, 2013.

[Strasser *et al.*, 2015] Ben Strasser, Adi Botea, and Daniel Harabor. Compressing optimal paths with run length encoding. *Journal of Artificial Intelligence Research*, 54:593–629, 2015.

[Sturtevant and Buro, 2005] Nathan Sturtevant and Michael Buro. Partial pathfinding using map abstraction and refinement. In *Proceedings of the 20th AAAI Conference on Artificial Intelligence*, 2005.

[Sturtevant *et al.*, 2009] Nathan Sturtevant, Ariel Felner, Max Barrer, Jonathan Schaeffer, and Neil Burch. Memory-based heuristics for explicit state spaces. In *Proceedings of the 21st International Joint Conference on Artificial Intelligence*, 2009.

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

[Torgerson, 1952] Warren S. Torgerson. Multidimensional scaling: I. theory and method. *Psychometrika*, 17(4):401–419, 1952.

[Uras and Koenig, 2014] Tansel Uras and Sven Koenig. Identifying hierarchies for fast optimal search. In *Proceedings of the 28th AAAI Conference on Artificial Intelligence*, 2014.

[Uras and Koenig, 2015] Tansel Uras and Sven Koenig. Subgoal graphs for fast optimal pathfinding. In Steve Rabin, editor, *Game AI Pro 2: Collected Wisdom of Game AI Professionals*, chapter 15, pages 145–159. A K Peters/CRC Press, 2015.

[Vandenberghe and Boyd, 1996] Lieven Vandenberghe and Stephen Boyd. Semidefinite programming. *SIAM Review*, 38:49–95, 1996.