Solving Facility Location Problems via FastMap and Locality Sensitive Hashing

Ang Li[1], **Peter Stuckey**[2,3], **Sven Koenig**[1], **T. K. Satish Kumar**[1]

[1]University of Southern California, USA
[2]Monash University, Australia
[3]OPTIMA ARC Industrial Training and Transformation Centre, Australia
ali355@usc.edu, peter.stuckey@monash.edu, skoenig@usc.edu, tkskwork@gmail.com

Abstract

Facility Location Problems (FLPs) arise while serving multiple customers in a shared environment, minimizing transportation and other costs. Hence, they involve the optimal placement of facilities. They are defined on graphs as well as in Euclidean spaces with or without obstacles; and they are typically NP-hard to solve optimally. There are many heuristic algorithms tailored to different kinds of FLPs. However, FLPs defined in Euclidean spaces without obstacles are the most amenable to efficient and effective heuristic algorithms. This motivates the idea of quickly reformulating FLPs on graphs and in Euclidean spaces with obstacles to FLPs in Euclidean spaces without obstacles. Towards this end, we propose a new approach that uses FastMap and Locality Sensitive Hashing. FastMap is a near-linear-time algorithm that embeds the vertices of a graph in a Euclidean space while approximately preserving graph-based distances as Euclidean distances for all pairs of vertices. Through extensive experiments, we show that our approach significantly outperforms other state-of-the-art competing algorithms on a variety of FLPs: the Multi-Agent Meeting, Vertex K-Median (VKM), Weighted VKM, and the Capacitated VKM problems.

Introduction

Facility Location Problems (FLPs) are constrained optimization problems that seek the optimal placement of facilities for providing resources and services to multiple customers in a shared environment. That is, FLPs serve the purpose of orchestrating shared resources between multiple customers. They are used to model decision problems related to transportation, warehousing, polling, and healthcare, among many other tasks, for maximizing efficiency, impact, and/or profit. FLPs can be defined on graphs or in geometric spaces, in continuous or discrete environments, and with a variety of distance metrics and objectives. A compendium of FLPs along with various algorithms and case studies can be found in (Farahani and Hekmatfar 2009).

FLPs defined on graphs as well as in Euclidean spaces with or without obstacles are NP-hard to solve optimally (Guo-Hui and Xue 1998; Owen and Daskin 1998). Nonetheless, there are many heuristic algorithms tailored to different kinds of FLPs. FLPs defined on graphs are broadly applicable, since most environments can be represented as a graph (even if discretization is required). Modulo discretization, they are more general compared to FLPs defined in Euclidean spaces with obstacles, which, in turn, are more general compared to FLPs defined in Euclidean spaces without obstacles. However, FLPs defined in Euclidean spaces without obstacles are the most amenable to efficient and effective heuristic algorithms. In fact, FLPs in Euclidean spaces without obstacles are definitionally very close to clustering problems, which, in turn, are amenable to popular clustering algorithms such as the K-means and Gaussian Mixture Model (GMM) clustering (Murphy 2012).

The foregoing summary motivates the idea of quickly reformulating FLPs on graphs and in Euclidean spaces with obstacles to FLPs in Euclidean spaces without obstacles. In this paper, we propose a new approach towards this end that uses FastMap and Locality Sensitive Hashing (LSH). FastMap (Cohen et al. 2018; Li et al. 2019) is a near-linear-time algorithm that embeds the vertices of a graph in a Euclidean space while approximately preserving graph-based distances as Euclidean distances for all pairs of vertices. We use FastMap to efficiently reformulate an FLP on a graph to an FLP in a Euclidean space without obstacles.[1] LSH (Datar et al. 2004) is a hashing technique that maps similar input items to the same hash buckets with high probability. It is designed to answer nearest-neighbor queries efficiently, with time complexity close to $O(\log n)$ (even in very high-dimensional spaces), where n is the total number of items. We use LSH to efficiently interpret a solution found in the FastMap embedding as a solution on the original graph.

We address four well-known FLPs in this paper: the Multi-Agent Meeting (MAM) problem, the Vertex K-Median (VKM) problem, the Weighted VKM (WVKM) problem, and the Capacitated VKM (CVKM) problem. Below, we briefly describe each of these problems on graphs. Their counterparts in Euclidean spaces with or without obstacles have analogous definitions. Moreover, we assume that the graphs are undirected for two reasons: for the ease of exposition and to preserve the analogy in Euclidean spaces where distances are inherently symmetric.

In the MAM problem (Atzmon et al. 2023), the input is a

[1]As explained later, an FLP in Euclidean space with obstacles is also amenable to a similar reformulation.

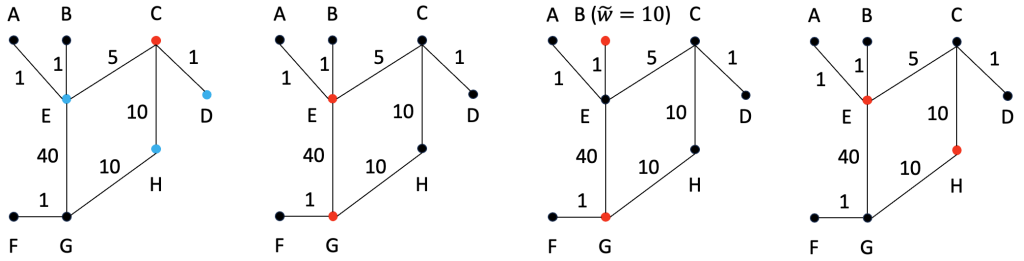

Figure 1: Illustrates the MAM, VKM, WVKM, and the CVKM problems. The first, second, third, and the fourth panels show the optimal solution in red for the MAM, VKM, WVKM, and the CVKM problems, respectively, on the same input graph. In the MAM problem, there are 3 agents, each initially on a different vertex shown in blue. The VKM, WVKM, and the CVKM problems have different optimal solutions for the same value of $K = 2$. In the WVKM problem, all vertices have weight 1; except 'B' has weight 10. In the CVKM problem, $\tau = 4$.

graph and a set of agents, each initially on a different start vertex. The task is to find a common vertex where all the agents should meet so as to minimize the sum of the agents' shortest-path distances to it.[2] The VKM problem seeks K vertices on the input graph for the placement of facilities so as to minimize the sum of the shortest-path distances over each vertex to its nearest facility. The WVKM problem is similar to the VKM problem, except that the objective is to minimize the sum of the weighted shortest-path distances over each vertex to its nearest facility. Here, each vertex is given a weight that measures its importance. The CVKM problem is also similar to the VKM problem, except that no facility is allowed to serve more than τ vertices.[3]

The MAM, VKM, WVKM, and the CVKM problems have many real-world applications. For example, in multi-agent coordination tasks (Atzmon et al. 2023), they can be used to choose a gathering point; in urban development (Farahani et al. 2019), they can be used to optimally place various public service centers within a city; and in communication networks (Monge and Contractor 2003), they can be used to determine the optimal placement of computation sites for critical multiplexing. Figure 1 shows examples of these four problems posed on the same input graph.

On each of the FLPs described above, including their Euclidean variants, we demonstrate the efficiency and effectiveness of our approach through extensive experimentation: We show that our approach significantly outperforms other state-of-the-art competing algorithms. In discretized Euclidean spaces, we also show that it is possible to combine FastMap with an any-angle path planner, such as Anya (Harabor et al. 2016).

Preliminaries

In this section, we define the MAM, VKM, WVKM, and the CVKM problems. We first define the graph variants of these problems. We then briefly describe their Euclidean variants.

[2]There are a few other variants of the MAM problem described in (Atzmon et al. 2023). These differ in being conflict-tolerant or conflict-free and having different objective functions.

[3]In a more general version of the CVKM problem, there is a supply and a demand associated with each facility and vertex, respectively. No facility is allowed to serve a total demand that exceeds its supply.

The MAM problem is as follows: Given an undirected edge-weighted graph $G = (V, E, w)$, where $w(e)$ is the non-negative weight on edge $e \in E$, and the start vertices $s_1, s_2 \ldots s_k \in V$ of k agents, the task is to find a vertex $v^* \in V$ such that $v^* = \arg\min_{v \in V} \sum_{i=1}^{k} d_G(s_i, v)$. Here, $d_G(v_i, v_j)$, for $v_i, v_j \in V$, is the shortest-path distance between v_i and v_j in G with respect to the edge weights.

The VKM problem is as follows: Given an undirected edge-weighted graph $G = (V, E, w)$, where $w(e)$ is the non-negative weight on edge $e \in E$, and a positive integer K, the task is to find a subset of vertices $U^* \subseteq V$ of cardinality K such that $U^* = \arg\min_U \sum_{v \in V} \min_{u \in U} d_G(v, u)$.

The WVKM problem is as follows: Given an undirected vertex-weighted and edge-weighted graph $G = (V, E, \tilde{w}, w)$, where $\tilde{w}(v)$ is the non-negative weight on vertex $v \in V$ and $w(e)$ is the non-negative weight on edge $e \in E$, and a positive integer K, the task is to find a subset of vertices $U^* \subseteq V$ of cardinality K such that $U^* = \arg\min_U \sum_{v \in V} \min_{u \in U} \tilde{w}(v) d_G(v, u)$.

The CVKM problem is as follows: Given an undirected edge-weighted graph $G = (V, E, w)$, where $w(e)$ is the non-negative weight on edge $e \in E$, and positive integers K and τ, the task is to find a subset of vertices $U^* \subseteq V$ of cardinality K and an assignment function $f^* : V \to U^*$ such that $(U^*, f^*) =$

$$\begin{aligned} &\arg\min_{U,f} \quad \sum_{v \in V} d_G(v, f(v)) \\ &\text{subject to} \quad \forall u \in U : |\{v \in V : f(v) = u\}| \leq \tau. \end{aligned} \tag{1}$$

The Euclidean variants of the MAM, VKM, WVKM, and the CVKM problems are defined in Euclidean spaces, which are continuous. In a Euclidean space without obstacles, a given set of N points corresponds to V; and the straight-line distances between pairs of these points correspond to the shortest-path distances between the pairs of vertices. However, the solution may be allowed to contain points in the Euclidean space outside of the given N points. In a Euclidean space with obstacles, shortest-path distances via free space, that is, avoiding obstacle regions, replace straight-line distances; and the solution can only include points in free space.

Solving FLPs via FastMap and LSH

In this section, we present our approach for solving FLPs via FastMap and LSH. We illustrate it on the MAM, VKM,

Algorithm 1: FASTMAP: A near-linear-time graph embedding algorithm.

Input: $G = (V, E)$, κ, and ϵ
Output: $p_i \in \mathbb{R}^r$ for all $v_i \in V$

1: **for** $r = 1, 2 \dots \kappa$ **do**
2: Choose $v_a \in V$ randomly and let $v_b = v_a$.
3: **for** $t = 1, 2 \dots C$ (a small constant) **do**
4: $\{d_{ai} : v_i \in V\} \leftarrow$ ShortestPathTree(G, v_a).
5: $v_c \leftarrow \text{argmax}_{v_i}\{d_{ai}^2 - \sum_{j=1}^{r-1}([p_a]_j - [p_i]_j)^2\}$.
6: **if** $v_c == v_b$ **then**
7: Break.
8: **else**
9: $v_b \leftarrow v_a$; $v_a \leftarrow v_c$.
10: **end if**
11: **end for**
12: $\{d_{ai} : v_i \in V\} \leftarrow$ ShortestPathTree(G, v_a).
13: $\{d_{ib} : v_i \in V\} \leftarrow$ ShortestPathTree(G, v_b).
14: $d'_{ab} \leftarrow d_{ab}^2 - \sum_{j=1}^{r-1}([p_a]_j - [p_b]_j)^2$.
15: **if** $d'_{ab} < \epsilon$ **then**
16: $r \leftarrow r - 1$; Break.
17: **end if**
18: **for** each $v_i \in V$ **do**
19: $d'_{ai} \leftarrow d_{ai}^2 - \sum_{j=1}^{r-1}([p_a]_j - [p_i]_j)^2$.
20: $d'_{ib} \leftarrow d_{ib}^2 - \sum_{j=1}^{r-1}([p_i]_j - [p_b]_j)^2$.
21: $[p_i]_r \leftarrow (d'_{ai} + d'_{ab} - d'_{ib})/(2\sqrt{d'_{ab}})$.
22: **end for**
23: **end for**
24: **return** $p_i \in \mathbb{R}^r$ for all $v_i \in V$.

WVKM, and the CVKM problems. The FastMap component of our approach allows us to quickly render the FLP in Euclidean space without obstacles: This enables efficient and effective geometric and analytical techniques for solving the problem. The LSH component of our approach allows us to quickly interpret the solution obtained in Euclidean space as a viable solution on the original graph.

FastMap on Graphs

FastMap (Cohen et al. 2018; Li et al. 2019) is used to embed the vertices of a given undirected edge-weighted graph in a κ-dimensional Euclidean space. The Euclidean distance between any two vertices in this space approximates the shortest-path distance—or any other graph-based distance—between them in the given graph. FastMap approximates a quadratic number of pairwise distances between all the vertices without investing quadratic time for doing so. Instead, it runs in only $O(\kappa(|E| + |V| \log |V|))$ time, where V and E are the vertices and the edges of the graph, respectively. This time complexity is the same as that of Dijkstra's single-source shortest-path tree algorithm (Fredman and Tarjan 1987)[4] and is only linear in the size of the graph.[5]

Algorithm 1 (Li et al. 2019) presents the pseudocode for FastMap on graphs. Here, κ is user-specified, but a threshold parameter ϵ is introduced to detect large values of κ that have diminishing returns on approximating the pairwise shortest-path distances. Algorithm 1 can also be easily adapted to other graph-based distance functions (Li et al. 2022, 2023).

With a runtime complexity close to that of merely reading the input, FastMap produces a Euclidean embedding and a geometric interpretation of graph problems. In doing so, it delegates the combinatorial heavy-lifting of solving graph problems to analytical and geometric techniques that are better equipped for absorbing large input sizes; this is illustrated in (Li et al. 2019, 2022, 2023). The properties of the resulting Euclidean space can be leveraged in many ways. First, the Euclidean space empowers analytical methods to set up and solve equations or establish other conditions of optimality. Second, the Euclidean space is a metric space that empowers geometric methods to conceive of objects like straight lines, angles, and bisectors, which facilitate visual intuition and the design of efficient algorithms that utilize Euclidean interpretations.

Solving FLPs in Euclidean Space without Obstacles

Most FLPs defined on graphs are also defined in Euclidean spaces without obstacles. As described before, the MAM, VKM, WVKM, and the CVKM problems are defined in such a space using Euclidean distances instead of graph-based distances. There are two benefits of using FastMap to convert an FLP specified on a graph to an FLP specified in the Euclidean embedding of that graph. First, FLPs defined in Euclidean spaces without obstacles are the most amenable to efficient and effective heuristic algorithms. Second, invoking FastMap with an intelligently designed distance function can simplify the problem even more.

For illustration of the above arguments, we consider the MAM problem on an input graph with k agents. Solving it optimally requires the computation of k shortest-path trees rooted at the individual start vertices of the agents. The same problem in Euclidean space without obstacles is referred to as the Fermat-Weber problem (Durier and Michelot 1985). This problem is also NP-hard to solve optimally but is amenable to very effective heuristics (Fekete, Mitchell, and Beurer 2005). Moreover, if FastMap is invoked on the input graph to preserve the square-roots of the shortest-path distances—instead of the shortest-path distances—the problem in the resulting Euclidean space becomes one of finding a point that minimizes the sum of the squared distances to k given points. This is a significantly easier problem since the required point is the centroid of the k given points.

Algorithm 1 can be easily modified to incorporate the square-root of the shortest-path distance function $\sqrt{d_G(\cdot, \cdot)}$ between vertices. This is done by returning the square-roots of the shortest-path distances found by the procedure ShortestPathTree() on Lines 4, 12, and 13.

The VKM, WVKM, and the CVKM problems can also utilize the FastMap embedding with the square-root of the shortest-path distance function. Doing so makes them very similar to clustering problems popularly studied in Machine Learning. For example, in a Euclidean space without obstacles, clustering algorithms such as the K-means algorithm intend to minimize the sum of the squared Euclidean distances over each data point to its nearest centroid. With

[4] since κ is usually designated as a constant

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

the Euclidean distances representing the square-roots of the shortest-path distances in the input graph, this is equivalent to solving the VKM problem of minimizing the sum of the shortest-path distances over each vertex to its nearest facility. Similarly, the WVKM problem can also be solved by invoking the K-means algorithm in the FastMap embedding that preserves the square-roots of the shortest-path distances: The K-means algorithm is also given a weight associated with each data point that measures its importance. Finally, the CVKM problem can also be solved by invoking the constrained K-means algorithm (Bradley, Bennett, and Demiriz 2000) in the FastMap embedding that preserves the square-roots of the shortest-path distances: The constrained K-means algorithm restricts the size of each cluster to be no more than a user-specified parameter τ.

Although we have described how to solve FLPs in the Euclidean space generated by FastMap on an input graph, the solutions produced reside in the Euclidean space and are not yet interpretable on the original graph. As described in the next subsection, we use LSH towards this end.

LSH

While the input graph is a discrete structure and has only a finite number of vertices, the Euclidean space generated by FastMap on it is a continuous space and has an infinite number of points. Therefore, while every vertex of the graph maps to a point in the Euclidean space, not every point in the Euclidean space maps to a vertex. In fact, a point in the Euclidean space deemed as belonging to a solution—or any other point of interest in the Euclidean space—may not map to a vertex of the original graph.

To address the foregoing problem, we assign any point of interest in the Euclidean space to its nearest neighbor that maps to a vertex of the original graph. This requires us to answer nearest-neighbor queries very efficiently. Fortunately, this problem is well studied in Computational Geometry. For example, in a 2-dimensional Euclidean space with straight-line distances, nearest-neighbor queries can be answered in logarithmic time using Voronoi diagrams. In higher dimensions, we use LSH (Datar et al. 2004). LSH is a hashing technique that maps similar input items to the same hash buckets with high probability. It answers nearest-neighbor queries in logarithmic time (ignoring small complexity factors).

FastMap with Anya

Compared to FLPs in Euclidean spaces without obstacles, FLPs in Euclidean spaces with obstacles are much harder to solve. Primarily, this is because the shortest-path distances in the latter are no more straight-line distances. In fact, even by itself, computing the shortest path between two points in a Euclidean space with obstacles may be very hard. Even shortest-path algorithms that operate in a Euclidean space with obstacles have to make various kinds of assumptions on the nature of the obstacles and the acceptable paths that maneuver through them. For the same reason, we define FLPs in Euclidean spaces with obstacles only when the environment also supports Anya (Harabor et al. 2016), a popular any-angle path planner. We note that this is not a restriction on the kinds of FLPs that can be discussed but is a standardization of the input environment that is also applicable to the state-of-the-art shortest-path algorithms.

Anya (Harabor et al. 2016) is a recent any-angle shortest-path algorithm for grid-worlds. Given any two discrete points on a 2-dimensional grid-world, Anya finds a shortest any-angle path between them, if one exists. It uses a variant of A^* search over sets of states represented as intervals. Anya is very efficient since it does not require preprocessing or the introduction of additional memory overheads.

In our FastMap-based approach for solving FLPs, the FastMap component always generates a Euclidean space without obstacles. It can be used to transform an input Euclidean space with obstacles to an output Euclidean space without obstacles if the straight-line distances in the output space preserve the shortest-path distances in the input space. Towards this end, we can use the any-angle shortest-path distance function on the discrete points of a 2-dimensional grid-world, as generated by Anya. However, using the any-angle shortest-path distance function with FastMap has the same fundamental challenge as using the regular shortest-path distance function with FastMap: To retain the near-linear time complexity of FastMap, the distances should not be computed from a root vertex to all other vertices independently. The computations have to be amortized to yield all of them simultaneously, as shown on Lines 4, 12, and 13 of Algorithm 1. Thus, we modify Anya to compute the entire tree of any-angle shortest-path distances from a root vertex to all other vertices. We call this version as Anya-Dijkstra. Hence, FastMap with Anya is similar to Algorithm 1, except that it replaces the procedure ShortestPathTree() by Anya-Dijkstra and returns the square-roots of the any-angle shortest-path distances found by Anya-Dijkstra on Lines 4, 12, and 13.

Related Work and Competing Algorithms

FastMap has been used as a preprocessing technique to facilitate A^* search for shortest-path computations (Cohen et al. 2018). In this context, it differs slightly from Algorithm 1 to ensure the consistency and admissibility of the heuristics. This version of FastMap has been recently improved by incorporating ideas of Differential Heuristics in the last iteration (Mashayekhi, Atzmon, and Sturtevant 2023). FastMap, as in Algorithm 1, has been used for the MAM problem (Li et al. 2019); but there, it does not use the square-roots of the shortest-path distances and, consequently, uses a heuristically computed solution—instead of the centroid—in the Euclidean space. It has also been used for the VKM problem (Thakoor et al. 2022) but only to the extent of improving the preprocessing phase of other algorithms. FastMap with various novel distance functions has also been used for block modeling (Li et al. 2022) and top-K centrality computations (Li et al. 2023). Moreover, FastMap has been generalized to directed graphs (Gopalakrishnan et al. 2020); and the accuracy of the FastMap embedding has been studied on various undirected and directed graphs (Li et al. 2019; Gopalakrishnan et al. 2020).

The MAM, VKM, WVKM, and the CVKM problems can be formulated using Integer Linear Programming (ILP). The

following template for the CVKM problem can be used.

$$\begin{aligned} \min \quad & \textstyle\sum_{v_i \in V} \sum_{v_j \in V} b_{ij} d_{ij} \\ \text{subject to} \quad & \forall v_i \in V : \textstyle\sum_{v_j \in V} b_{ij} = 1 \\ & \forall v_j \in V : \textstyle\sum_{v_i \in V} b_{ij} \leq \tau \\ & \forall v_i, v_j \in V : b_{ij} \leq c_j \\ & \textstyle\sum_{v_j \in V} c_j = K. \end{aligned} \tag{2}$$

Here, d_{ij} is a shorthand for $d_G(v_i, v_j)$; c_j is a Boolean variable that is '1' iff v_j is a facility; and b_{ij} is a Boolean variable that is '1' iff v_j is the facility assigned to v_i. For the VKM problem, $\tau = |V|$. For the WVKM problem, $d_{ij} = \tilde{w}(v_i) d_G(v_i, v_j)$ and $\tau = |V|$. For the MAM problem, the outer summation of the objective function spans only the start vertices of the agents, $\tau = |V|$, and $K = 1$.

The MAM problem can be solved optimally in polynomial time and does not require the ILP solver: The algorithm computes a shortest-path tree rooted at each of the k start vertices of the agents. When heuristic guidance is available, MM^* (Atzmon et al. 2023) can also be used to solve the MAM problem optimally. For the VKM problem, the state-of-the-art algorithm is FasterPAM (Schubert and Rousseeuw 2021), the successor of the Partition Around Medoids (PAM) algorithm (Kaufman and Rousseeuw 1987). FasterPAM conducts local search by repeatedly swapping a vertex from its current solution S with a vertex in $V \setminus S$. It runs in $O(K|V|^2)$ time. For the WVKM problem, the state-of-the-art implementation of the PAM algorithm is available in the procedure 'wcKMedoids' (Maechler 2018) in R 4.3. The CVKM problem is significantly harder: To the best of our knowledge, there are no good solvers for this problem that scale to the problem sizes discussed in this paper.

The ILP solver and the PAM algorithms require the precomputation of the all-pairs shortest-path distances, which can be done via the Floyd-Warshall algorithm.

Experimental Results

In this section, we provide experimental results that compare our approach to competing algorithms on the MAM, VKM, WVKM, and the CVKM problems.

We implemented our approach in Python 3.9. For the LSH module, we used the 'FALCONN' library (Andoni et al. 2015) that has many code-level optimizations. For the K-means procedure without weights, required for solving the VKM problem, and for the K-means procedure with weights, required for solving the WVKM problem, we used the 'scikit-learn' library (Pedregosa et al. 2011). For the constrained K-means procedure, required for solving the CVKM problem, we used the 'k-means-constrained' library. For the ILP solver, we used the Gurobi Optimizer 10.0 (Gurobi Optimization, LLC 2023). We used the Anya procedure via a Python interface to its implementation in Java. For FasterPAM (Schubert and Rousseeuw 2021), we used the 'kmedoids' library. However, for the PAM clustering of weighted data, we used the procedure 'wcKMedoids' (Maechler 2018) implemented in R 4.3. For the Floyd-Warshall algorithm, we used the 'NetworkX' library (Hagberg, Swart, and S Chult 2008). We conducted all experiments on a laptop with an Apple M2 Max chip and 96 GB RAM. For evaluation purposes, we chose two categories of problem instances, both of which contain realistic FLPs.

In the first category, we chose problem instances representative of FLPs that arise in warehousing, urban planning, and transportation domains, among others. In such cases, the environment is essentially a 2-dimensional map. Moreover, in such cases, both the regular FastMap (FM), that is, FastMap that implements the procedure ShortestPathTree() using Dijkstra's algorithm (DJK), and FastMap with Anya (FMA), that is, FastMap that implements the procedure ShortestPathTree() using Anya-Dijkstra (ADJK), are defined. This enables a more direct comparison of the various algorithms. Such instances are available in the movingAI dataset (Sturtevant 2012): Each instance serves as both a graph instance and a 2-dimensional grid-world instance. In it, each discrete point on a traversable cell[6] is represented as a vertex. Adjacent vertices, corresponding to discrete points on the same traversable cell, are connected by an edge. A horizontal or vertical edge has unit weight but a diagonal edge has weight $\sqrt{2}$. If the graph constructed this way has multiple connected components, only the largest one is used to represent the instance. We used five representative benchmark suites in this category: 'Dragon Age: Origins', 'Warcraft III', 'Baldurs Gate II', 'City/Street Maps', and 'Mazes'. The first three are from commercial game environments; the fourth is from the real world; and the fifth is artificial. Both FastMap and FastMap with Anya use $\kappa = 10$ for these instances.

In the second category, we chose problem instances representative of FLPs that arise in communication networks. In the field of Computer and Communication Networks, Waxman graphs (Waxman 1988) are used as realistic communication networks. Hence, we generated Waxman graph instances using NetworkX (Hagberg, Swart, and S Chult 2008) with commonly used parameter values $\alpha = 0.3$ and $\beta = 0.1$, within a rectangular domain of 100×100, and with the weight on each edge set to the Euclidean distance between its endpoints. FastMap uses $\kappa = 100$ for these instances.[7]

Tables 1-8 show the performance results of various algorithms on the MAM, VKM, WVKM, and the CVKM problems. Although our experiments are extensive and conclusive, we can present only representative results in these tables due to space limitations. In each table, representative results are shown in sets of rows: the first set on instances from 'Dragon Age: Origins', the second set on instances from 'Baldurs Gate II', and the third set on the largest instances from 'City/Street Maps', 'Warcraft III', and 'Mazes', in that order. These three sets are from the first category and serve as both graph and grid-world instances. The odd-numbered tables also have a fourth set of rows from the second category that serve only as graph instances. A '-' is associ-

[6] as defined in (Harabor et al. 2016) for the application of Anya

[7] The Normalized Root Mean Square Deviation, as used in (Li et al. 2019) to measure the accuracy of the FastMap embedding, is much higher for Waxman graphs even with $\kappa = 100$ compared to movingAI instances with $\kappa = 10$.

Instance	Size ($\|V\|$, $\|E\|$)	Preprocessing for FM: FM_pre (s)	$k = 50$			$k = 100$		
			Time (s)		SO (%)	Time (s)		SO (%)
			DJK	FM	FM	DJK	FM	FM
orz102d	(738, 2632)	0.05	0.09	0.00	1.23	0.19	0.00	0.93
den407d	(852, 3054)	0.07	0.09	0.00	0.28	0.18	0.00	0.98
lak526d	(954, 3329)	0.07	0.09	0.00	0.56	0.19	0.00	1.17
den009d	(1003, 3620)	0.07	0.10	0.00	1.49	0.21	0.00	1.55
AR0512SR	(896, 3275)	0.06	0.10	0.00	6.35	0.19	0.00	2.31
AR0402SR	(1075, 3796)	0.10	0.13	0.00	6.49	0.26	0.00	1.70
AR0517SR	(1083, 4078)	0.09	0.12	0.00	4.47	0.24	0.00	1.17
AR0530SR	(1092, 3885)	0.08	0.11	0.00	6.45	0.22	0.00	0.57
Shanghai_0_256	(48696, 190303)	4.81	6.67	0.01	1.15	13.26	0.00	2.26
blastedlands	(131342, 505974)	12.95	18.58	0.02	5.87	37.18	0.02	0.90
maze512-32-5	(253856, 990715)	21.82	35.25	0.04	1.09	70.54	0.04	2.81
wm00800	(800, 9498)	1.06	0.25	0.00	7.08	0.50	0.00	5.81
wm01000	(1000, 14923)	1.78	0.38	0.00	9.37	0.76	0.00	5.80
wm05000	(5000, 374925)	88.64	13.43	0.00	3.80	26.76	0.00	3.56
wm10000	(10000, 1499713)	360.29	54.00	0.00	1.65	107.38	0.00	1.13

Table 1: Shows the results for the MAM problem on various graph instances. 'FM', 'FM_pre', 'DJK', and 'SO' stand for 'FastMap', 'FastMap preprocessing', 'Dijkstra', and 'suboptimality', respectively.

Instance	Size ($\|V\|$, $\|E\|$)	Preprocessing for FMA: FMA_pre (s)	$k = 50$				$k = 100$			
			Time (s)		SO (%)		Time (s)		SO (%)	
			ADJK	FMA	FM	FMA	ADJK	FMA	FM	FMA
orz102d	(738, 2632)	3.29	5.42	0.00	0.96	0.30	10.86	0.00	1.45	0.46
den407d	(852, 3054)	2.66	3.92	0.00	0.91	0.58	8.07	0.00	0.09	0.13
lak526d	(954, 3329)	3.77	5.12	0.00	2.52	2.52	10.01	0.00	2.90	5.15
den009d	(1003, 3620)	2.00	3.46	0.00	3.65	1.08	6.96	0.00	0.05	0.29
AR0512SR	(896, 3275)	10.81	13.84	0.00	4.91	2.71	27.62	0.00	2.76	3.47
AR0402SR	(1075, 3796)	8.54	10.54	0.00	1.07	2.53	21.33	0.00	0.03	3.53
AR0517SR	(1083, 4078)	9.84	13.89	0.00	6.33	4.28	27.95	0.00	1.82	2.72
AR0530SR	(1092, 3885)	12.09	15.37	0.00	0.64	0.21	30.87	0.00	1.33	0.43
Shanghai_0_256	(48696, 190303)	16.77	16.09	0.01	1.56	1.27	32.12	0.01	1.06	3.07
blastedlands	(131342, 505974)	44.81	35.70	0.02	6.87	7.52	71.59	0.02	3.70	5.50
maze512-32-5	(253856, 990715)	34.69	55.79	0.04	2.68	3.03	111.51	0.04	1.43	1.48

Table 2: Shows the results for the MAM problem on various grid-world instances. 'FMA', 'FMA_pre', and 'ADJK' stand for 'FastMap with Anya', 'FastMap with Anya preprocessing', and 'Anya-Dijkstra', respectively.

Instance	Size ($\|V\|$, $\|E\|$)	Preprocessing for ILP, PAM: FW (s)	Preprocessing for FM: FM_pre (s)	$K = 10$					$K = 20$				
				Time (s)			SO (%)		Time (s)			SO (%)	
				ILP	PAM	FM	PAM	FM	ILP	PAM	FM	PAM	FM
orz102d	(738, 2632)	27.10	0.05	79.34	0.00	0.00	1.47	3.70	175.76	0.00	0.00	1.84	1.01
den407d	(852, 3054)	41.24	0.07	48.70	0.00	0.00	1.34	3.44	132.63	0.00	0.01	1.95	1.78
lak526d	(954, 3329)	56.36	0.07	170.70	0.01	0.02	0.23	1.09	71.53	0.00	0.01	7.43	2.72
den009d	(1003, 3620)	64.38	0.07	83.20	0.12	0.06	0.21	1.23	96.65	0.08	0.06	4.23	1.60
AR0512SR	(896, 3275)	48.20	0.06	52.95	0.00	0.00	54.19	2.52	485.62	0.00	0.01	0.99	2.98
AR0402SR	(1075, 3796)	82.27	0.10	73.03	0.07	0.05	1.85	2.68	74.51	0.11	0.16	0.11	4.06
AR0517SR	(1083, 4078)	85.25	0.09	1556.52	0.15	0.12	0.22	0.80	4467.56	0.19	0.06	0.45	2.66
AR0530SR	(1092, 3885)	84.71	0.08	121.16	0.08	0.03	1.43	0.41	209.84	0.13	0.10	0.12	5.69
Shanghai_0_256	(48696, 190303)	-	4.40	-	-	0.15	-	-	-	-	0.25	-	-
blastedlands	(131342, 505974)	-	12.88	-	-	0.49	-	-	-	-	0.72	-	-
maze512-32-5	(253856, 990715)	-	22.61	-	-	0.65	-	-	-	-	1.00	-	-
wm00800	(800, 9498)	35.94	1.06	1650.12	0.00	0.02	0.03	16.50	3302.97	0.00	0.02	0.04	13.16
wm01000	(1000, 14923)	70.30	1.78	9188.83	0.06	0.04	0.07	15.04	9970.25	0.09	0.07	0.19	16.66
wm05000	(5000, 374925)	-	88.64	-	-	0.12	-	-	-	-	0.16	-	-
wm10000	(10000, 1499713)	-	360.29	-	-	0.33	-	-	-	-	0.49	-	-

Table 3: Shows the results for the VKM problem on various graph instances. 'FW', 'ILP', and 'PAM' stand for 'Floyd-Warshall', 'ILP solver', and 'PAM algorithm', respectively. The ILP solver and the PAM algorithm require the Floyd-Warshall algorithm in a preprocessing phase for the computation of the all-pairs shortest-path distances.

ated with any instance whose preprocessing time exceeds 1 hour. The suboptimality 'SO' columns report (cost - optimal cost)/(optimal cost) as a percentage. In general, our approach is the only one that can scale to large input sizes for the VKM, WVKM, and the CVKM problems.

Table 1 compares FastMap (FM) and the brute-force algorithm (DJK) on the MAM problem. DJK computes the ground truth by rooting a shortest-path tree at each of the k start vertices of the agents. Here, each graph instance is designed by picking the k start vertices at random. The FastMap preprocessing time (FM_pre) refers to the time taken by FastMap to generate the Euclidean embedding of the graph plus the time taken by LSH for the initial indexing. This preprocessing time is required only once per graph, independent of k and the start vertices. We observe that FastMap is significantly faster than the brute-force algorithm on larger instances, up to 4-5 orders of magnitude.[8]

[8]Based on the results reported in (Atzmon et al. 2023), FastMap

Instance	Size ($\|V\|$, $\|E\|$)	Preprocessing for FMA: FMA_pre (s)	$K = 10$					$K = 20$				
			Time (s)		SO (%)			Time (s)		SO (%)		
			ILP	FMA	PAM	FM	FMA	ILP	FMA	PAM	FM	FMA
orz102d	(738, 2632)	3.29	77.38	0.28	6.44	1.69	1.35	214.98	0.28	38.30	1.57	2.87
den407d	(852, 3054)	2.66	49.04	0.28	0.51	3.65	1.32	144.29	0.23	31.33	2.42	4.77
lak526d	(954, 3329)	3.77	170.97	0.28	0.04	1.80	0.12	72.49	0.28	0.31	1.98	6.70
den009d	(1003, 3620)	2.00	83.16	0.28	-0.02	2.41	0.89	95.91	0.28	0.28	3.83	4.14
AR0512SR	(896, 3275)	10.81	55.76	0.27	0.00	1.29	3.26	519.66	0.28	0.99	2.20	4.25
AR0402SR	(1075, 3796)	8.54	74.70	0.29	2.01	1.21	5.10	75.09	0.29	0.51	4.07	4.27
AR0517SR	(1083, 4078)	9.84	1581.05	0.28	3.68	0.78	0.73	5905.02	0.28	0.56	1.50	0.72
AR0530SR	(1092, 3885)	12.09	122.93	0.28	1.90	3.64	3.33	211.75	0.30	1.58	3.91	1.26
Shanghai_0_256	(253856, 990715)	14.07	-	0.10	-	-	-	-	0.30	-	-	-
blastedlands	(131342, 505974)	37.21	-	0.47	-	-	-	-	0.67	-	-	-
maze512-32-5	(253856, 990715)	39.19	-	0.68	-	-	-	-	1.11	-	-	-

Table 4: Shows the results for the VKM problem on various grid-world instances.

Instance	Size ($\|V\|$, $\|E\|$)	Preprocessing for ILP, PAM: FW (s)	Preprocessing for FM: FM_pre (s)	$K = 10$					$K = 20$				
				Time (s)			SO (%)		Time (s)			SO (%)	
				ILP	PAM	FM	PAM	FM	ILP	PAM	FM	PAM	FM
orz102d	(738, 2632)	27.10	0.05	44.49	0.06	0.00	2.61	4.26	95.91	0.14	0.00	4.12	5.50
den407d	(852, 3054)	41.24	0.07	52.29	0.15	0.00	1.69	0.72	74.92	0.18	0.01	4.91	4.90
lak526d	(954, 3329)	56.36	0.07	101.26	0.09	0.02	1.19	2.78	60.25	0.21	0.01	3.35	2.49
den009d	(1003, 3620)	64.38	0.07	79.18	0.11	0.01	1.12	1.63	137.58	0.35	0.01	2.39	4.09
AR0512SR	(896, 3275)	48.20	0.06	146.60	0.08	0.00	1.57	0.72	119.31	0.33	0.01	1.68	3.78
AR0402SR	(1075, 3796)	82.27	0.10	80.94	0.11	0.02	3.92	3.18	92.47	0.25	0.03	4.92	6.26
AR0517SR	(1083, 4078)	85.25	0.09	2907.62	0.19	0.27	0.93	1.09	571.17	0.45	0.02	2.85	1.93
AR0530SR	(1092, 3885)	84.71	0.08	318.33	0.14	0.02	2.15	0.90	127.48	0.28	0.02	3.67	5.11
Shanghai_0_256	(48696, 190303)	-	4.77	-	-	0.17	-	-	-	-	0.25	-	-
blastedlands	(131342, 505974)	-	13.10	-	-	0.38	-	-	-	-	0.70	-	-
maze512-32-5	(253856, 990715)	-	21.89	-	-	0.68	-	-	-	-	1.07	-	-
wm00800	(800, 9498)	35.94	1.06	1946.18	0.12	0.24	0.00	15.84	1388.69	0.21	0.01	0.00	17.19
wm01000	(1000, 14923)	70.30	1.78	5135.40	0.20	0.02	0.00	11.62	15112.05	0.21	0.02	0.01	12.68
wm05000	(5000, 374925)	-	88.64	-	-	0.07	-	-	-	-	0.07	-	-
wm10000	(10000, 1499713)	-	360.29	-	-	0.32	-	-	-	-	0.37	-	-

Table 5: Shows the results for the WVKM problem on various graph instances.

Instance	Size ($\|V\|$, $\|E\|$)	Preprocessing for FMA: FMA_pre (s)	$K = 10$					$K = 20$				
			Time (s)		SO (%)			Time (s)		SO (%)		
			ILP	FMA	PAM	FM	FMA	ILP	FMA	PAM	FM	FMA
orz102d	(738, 2632)	3.29	37.61	0.28	2.63	1.49	2.60	37.60	0.22	3.97	3.51	2.95
den407d	(852, 3054)	2.66	50.66	0.28	1.69	5.39	0.14	61.92	0.22	3.97	2.41	2.18
lak526d	(954, 3329)	3.77	105.26	0.28	1.66	1.61	7.51	72.85	0.28	3.90	3.63	2.45
den009d	(1003, 3620)	2.00	128.13	0.28	2.18	0.81	0.54	91.62	0.28	2.60	4.08	3.88
AR0512SR	(896, 3275)	10.81	53.07	0.28	1.21	0.13	2.82	129.15	0.28	2.70	2.98	5.86
AR0402SR	(1075, 3796)	8.54	84.35	0.29	2.31	4.10	4.63	100.25	0.29	1.90	2.52	5.81
AR0517SR	(1083, 4078)	9.84	92.75	0.28	0.97	0.51	1.11	1358.58	0.28	2.11	1.98	1.67
AR0530SR	(1092, 3885)	12.09	204.61	0.27	1.29	4.39	2.22	139.88	0.28	4.71	3.88	2.83
Shanghai_0_256	(253856, 990715)	14.98	-	0.21	-	-	-	-	0.40	-	-	-
blastedlands	(131342, 505974)	24.34	-	0.22	-	-	-	-	0.45	-	-	-
maze512-32-5	(253856, 990715)	37.99	-	0.67	-	-	-	-	1.03	-	-	-

Table 6: Shows the results for the WVKM problem on various grid-world instances.

Instance	Size ($\|V\|$, $\|E\|$)	Preprocessing for ILP: FW (s)	Preprocessing for FM: FM_pre (s)	$K = 10$			$K = 20$		
				Time (s)		SO (%)	Time (s)		SO (%)
				ILP	FM	FM	ILP	FM	FM
orz102d	(738, 2632)	27.10	0.05	101.07	0.09	1.80	162.59	0.13	3.57
den407d	(852, 3054)	41.24	0.07	49.35	0.04	4.02	166.29	0.09	2.88
lak526d	(954, 3329)	56.36	0.07	184.90	0.04	7.82	84.68	0.28	3.34
den009d	(1003, 3620)	64.38	0.07	86.95	0.17	1.45	114.34	0.26	4.10
AR0512SR	(896, 3275)	48.20	0.06	66.96	0.06	2.89	495.82	0.10	3.19
AR0402SR	(1075, 3796)	82.27	0.10	90.72	0.06	3.99	85.61	0.28	5.63
AR0517SR	(1083, 4078)	85.25	0.09	1942.77	0.24	3.17	3615.14	0.29	2.85
AR0530SR	(1092, 3885)	84.71	0.08	175.05	0.30	0.54	242.01	0.23	3.94
Shanghai_0_256	(48696, 190303)	-	4.85	-	6.19	-	-	33.48	-
blastedlands	(131342, 505974)	-	15.18	-	59.19	-	-	62.15	-
maze512-32-5	(253856, 990715)	-	22.50	-	13.18	-	-	20.08	-
wm00800	(800, 9498)	35.94	1.06	2721.67	0.13	48.18	4590.93	0.27	62.34
wm01000	(1000, 14923)	70.30	1.78	10531.79	0.13	35.18	23911.17	0.24	58.05
wm05000	(5000, 374925)	-	88.64	-	1.08	-	-	2.56	-
wm10000	(10000, 1499713)	-	360.29	-	3.07	-	-	6.04	-

Table 7: Shows the results for the CVKM problem on various graph instances.

also seems significantly faster—and more scalable to large graphs with large values of k—compared to MM^*.

Instance	Size ($\|V\|$, $\|E\|$)	Preprocessing for FMA: FMA_pre (s)	$K = 10$				$K = 20$			
			Time (s)		SO (%)		Time (s)		SO (%)	
			ILP	FMA	FM	FMA	ILP	FMA	FM	FMA
orz102d	(738, 2632)	3.29	90.55	0.05	1.52	2.44	192.79	0.19	2.40	3.64
den407d	(852, 3054)	2.66	66.04	0.09	2.12	4.45	153.89	0.10	2.43	2.32
lak526d	(954, 3329)	3.77	187.95	0.09	2.01	2.08	78.51	0.27	3.91	1.65
den009d	(1003, 3620)	2.00	116.48	0.08	1.00	1.05	106.17	0.25	1.59	1.60
AR0512SR	(896, 3275)	10.81	62.47	0.26	5.51	3.47	521.99	0.13	4.24	8.09
AR0402SR	(1075, 3796)	8.54	84.41	0.21	2.02	4.80	90.01	0.14	5.27	7.96
AR0517SR	(1083, 4078)	9.84	1237.97	0.42	3.20	3.22	4324.49	0.35	3.35	2.45
AR0530SR	(1092, 3885)	12.09	138.18	0.11	1.66	3.64	254.33	0.21	5.71	1.68
Shanghai_0_256	(48696, 190303)	17.02	-	12.27	-	-	-	21.37	-	-
blastedlands	(131342, 505974)	42.48	-	33.20	-	-	-	34.67	-	-
maze512-32-5	(253856, 990715)	39.58	-	14.39	-	-	-	24.48	-	-

Table 8: Shows the results for the CVKM problem on various grid-world instances.

In fact, FastMap is very often more efficient even with the preprocessing time included. It also produces solutions that are within just 7% suboptimality on instances from the first category and within 10% suboptimality on instances from the second category. Table 2 shows a similar dominance of FastMap and FastMap with Anya (FMA) over the brute-force algorithm (ADJK) that uses Anya to compute an any-angle shortest-path tree rooted at each of the k start vertices for generating the ground truth on grid-world instances. The FastMap times are excluded from Table 2 since they appear in Table 1. The suboptimality of FastMap is different in Tables 1 and 2, since the quality of a solution is measured using any-angle shortest-path distances in Table 2.

Table 3 compares FastMap, the ILP solver, and FasterPAM for solving graph instances of the VKM problem. Both the ILP solver and FasterPAM use the Floyd-Warshall algorithm (FW) in a preprocessing step to compute the all-pairs shortest-path distances. The preprocessing time of FastMap is significantly smaller than that of the Floyd-Warshall algorithm: The latter is about 3 orders of magnitude slower and not even viable for large graphs. Moreover, at query time, FastMap and FasterPAM are both significantly faster than the ILP solver. FastMap produces solutions within just 6% suboptimality on instances from the first category and within 17% suboptimality on instances from the second category. FasterPAM also produces high-quality solutions but it does so with occasional outliers and does not scale to large instances. Table 4 on grid-world instances shows a similar dominance of FastMap and FastMap with Anya—within similar suboptimality ranges—over the ILP solver and FasterPAM with respect to the preprocessing time and over the ILP solver with respect to the query time. The FastMap times, the Floyd-Warshall preprocessing times, and the FasterPAM query times are excluded from Table 4 since they appear in Table 3.[9] However, the query times of the ILP solver vary across different runs on the same instance and, thus, are included in Table 4. Tables 5 and 6 show the same trends for the WVKM problem,[10] instances of which are generated by attaching to each vertex a weight chosen uniformly at random from the interval $[0, 1)$. Here, the PAM algorithm is the 'wcKMedoids' procedure in R 4.3.

Table 7 compares FastMap and the ILP solver for solving graph instances of the CVKM problem. For representative results, these instances use $\tau = \lceil 2|V|/K \rceil$. Although the CVKM problem renders many approaches ineffective due to its comparative hardness over the VKM and the WVKM problems, FastMap can still solve all the instances, even those with a quarter-million vertices and a million edges, in mere seconds. On smaller instances, FastMap is also significantly faster than the ILP solver, which generates the ground truth. Moreover, FastMap produces solutions within just 8% suboptimality on instances from the first category and within 63% suboptimality[11] on instances from the second category. Table 8 shows the same trends for solving the grid-world instances of the CVKM problem with the same τ, within just 9% suboptimality using FastMap and FastMap with Anya.

Conclusions and Future Work

In this paper, we studied four representative FLPs: the MAM, VKM, WVKM, and the CVKM problems. Like most FLPs, these problems are well defined on graphs as well as in Euclidean spaces with or without obstacles. While they are generally difficult to solve optimally, the ones defined in a Euclidean space without obstacles are akin to clustering problems. We used the idea of FastMap to reformulate FLPs defined on a graph to FLPs defined in a Euclidean space without obstacles. Subsequently, we used standard clustering algorithms to solve the problems in the resulting Euclidean space and LSH to interpret the solutions back on the original graph. End to end, our approach produces high-quality solutions with orders-of-magnitude speedup over state-of-the-art competing algorithms.

In future work, we will use our approach to solve many other kinds of FLPs. We will also consider FLPs that arise in real-world robotics and communication domains.

[9] The computation of the all-pairs any-angle shortest-path distances is very expensive. Therefore, the Floyd-Warshall algorithm is invoked by treating the grid-world as a graph. This also creates the remote possibility of the ILP solver producing a suboptimal solution, although it is practically still treated as the ground truth.

[10] The ILP solver can also be an anytime solver. However, its performance is nowhere comparable to that of FastMap. For example, on the representative instance 'wm01000' with $K = 20$, compared to FM, ILP takes about $400\times$ time to produce a mere 34% suboptimal solution for both the VKM and the WVKM problems.

[11] 63% suboptimality is still impressive, since the CVKM problem is combinatorially very hard and, till date, there is no known polynomial-time constant-factor approximation algorithm for it.

Acknowledgments

This work at the University of Southern California is supported by DARPA under grant number HR001120C0157 and by NSF under grant number 2112533. The views, opinions, and/or findings expressed are those of the author(s) and should not be interpreted as representing the official views or policies of the sponsoring organizations, agencies, or the U.S. Government. This research is also partially funded by the Australian Government through the Australian Research Council Industrial Transformation Training Centre in Optimisation Technologies, Integrated Methodologies, and Applications (OPTIMA), Project ID IC200100009.

References

Andoni, A.; Indyk, P.; Laarhoven, T.; Razenshteyn, I.; and Schmidt, L. 2015. Practical and optimal LSH for angular distance. *Advances in Neural Information Processing Systems*.

Atzmon, D.; Felner, A.; Li, J.; Shperberg, S.; Sturtevant, N.; and Koenig, S. 2023. Conflict-tolerant and conflict-free multi-agent meeting. *Artificial Intelligence*.

Bradley, P. S.; Bennett, K. P.; and Demiriz, A. 2000. Constrained k-means clustering. Technical report, Microsoft Research, Redmond.

Cohen, L.; Uras, T.; Jahangiri, S.; Arunasalam, A.; Koenig, S.; and Kumar, T. K. S. 2018. The FastMap algorithm for shortest path computations. In *Proceedings of the 27th International Joint Conference on Artificial Intelligence*.

Datar, M.; Immorlica, N.; Indyk, P.; and Mirrokni, V. S. 2004. Locality-sensitive hashing scheme based on p-stable distributions. In *Proceedings of the 20th Annual Symposium on Computational Geometry*.

Durier, R.; and Michelot, C. 1985. Geometrical properties of the Fermat-Weber problem. *European Journal of Operational Research*.

Farahani, R. Z.; Fallah, S.; Ruiz, R.; Hosseini, S.; and Asgari, N. 2019. OR models in urban service facility location: A critical review of applications and future developments. *European Journal of Operational Research*.

Farahani, R. Z.; and Hekmatfar, M. 2009. *Facility location: Concepts, models, algorithms and case studies*. Springer Science & Business Media.

Fekete, S. P.; Mitchell, J. S.; and Beurer, K. 2005. On the continuous Fermat-Weber problem. *Operations Research*.

Fredman, M. L.; and Tarjan, R. E. 1987. Fibonacci heaps and their uses in improved network optimization algorithms. *Journal of the ACM*.

Gopalakrishnan, S.; Cohen, L.; Koenig, S.; and Kumar, T. K. S. 2020. Embedding directed graphs in potential fields using FastMap-D. In *Proceedings of the 13th International Symposium on Combinatorial Search*.

Guo-Hui, L.; and Xue, G. 1998. K-center and K-median problems in graded distances. *Theoretical Computer Science*.

Gurobi Optimization, LLC. 2023. Gurobi Optimizer Reference Manual.

Hagberg, A.; Swart, P.; and S Chult, D. 2008. Exploring network structure, dynamics, and function using NetworkX. Technical report, Los Alamos National Lab, Los Alamos, NM (United States).

Harabor, D. D.; Grastien, A.; Öz, D.; and Aksakalli, V. 2016. Optimal any-angle pathfinding in practice. *Journal of Artificial Intelligence Research*.

Kaufman, L.; and Rousseeuw, P. 1987. Clustering by means of medoids. In *Proceedings of the Statistical Data Analysis Based on the L1 Norm Conference, Neuchatel, Switzerland*.

Li, A.; Stuckey, P.; Koenig, S.; and Kumar, T. K. S. 2022. A FastMap-based algorithm for block modeling. In *Proceedings of the 19th International Conference on the Integration of Constraint Programming, Artificial Intelligence, and Operations Research*.

Li, A.; Stuckey, P.; Koenig, S.; and Kumar, T. K. S. 2023. A FastMap-based framework for efficiently computing top-K projected centrality. In *Proceedings of the 9th International Conference on Machine Learning, Optimization, and Data Science*.

Li, J.; Felner, A.; Koenig, S.; and Kumar, T. K. S. 2019. Using FastMap to solve graph problems in a Euclidean space. In *Proceedings of the 29th International Conference on Automated Planning and Scheduling*.

Maechler, M. 2018. Cluster: Cluster analysis basics and extensions. *R Package Version 2.0. 7–1*.

Mashayekhi, R.; Atzmon, D.; and Sturtevant, N. R. 2023. Analyzing and improving the use of the FastMap embedding in pathfinding tasks. In *Proceedings of the 37th AAAI Conference on Artificial Intelligence*.

Monge, P. R.; and Contractor, N. S. 2003. *Theories of communication networks*. Oxford University Press, USA.

Murphy, K. P. 2012. *Machine learning: A probabilistic perspective*. The MIT Press.

Owen, S. H.; and Daskin, M. S. 1998. Strategic facility location: A review. *European Journal of Operational Research*.

Pedregosa, F.; Varoquaux, G.; Gramfort, A.; Michel, V.; Thirion, B.; Grisel, O.; Blondel, M.; Prettenhofer, P.; Weiss, R.; Dubourg, V.; Vanderplas, J.; Passos, A.; Cournapeau, D.; Brucher, M.; Perrot, M.; and Duchesnay, E. 2011. Scikit-learn: Machine Learning in Python. *Journal of Machine Learning Research*.

Schubert, E.; and Rousseeuw, P. J. 2021. Fast and eager k-medoids clustering: O(k) runtime improvement of the PAM, CLARA, and CLARANS algorithms. *Information Systems*.

Sturtevant, N. 2012. Benchmarks for grid-based pathfinding. *Transactions on Computational Intelligence and AI in Games*.

Thakoor, O.; Li, A.; Koenig, S.; Ravi, S.; Kline, E.; and Kumar, T. K. S. 2022. The FastMap pipeline for facility location problems. In *Proceedings of the 24th International Conference on Principles and Practice of Multi-Agent Systems*.

Waxman, B. M. 1988. Routing of multipoint connections. *IEEE Journal on Selected Areas in Communications*.