The FastMap Pipeline for Facility Location Problems

Omkar Thakoor[1], Ang Li[1], Sven Koenig[1], Srivatsan Ravi[2], Erik Kline[2], and T. K. Satish Kumar[2]

[1] Department of Computer Science, University of Southern California
{othakoor, ali355, skoenig}@usc.edu
[2] Information Sciences Institute, University of Southern California
{sravi, kline}@isi.edu, tkskwork@gmail.com

Abstract. Facility Location Problems (FLPs) involve the placement of facilities in a shared environment for serving multiple customers while minimizing transportation and other costs. FLPs defined on graphs are very general and broadly applicable. Two such fundamental FLPs are the Vertex K-Center (VKC) and the Vertex K-Median (VKM) problems. Although both these problems are NP-hard, many heuristic and approximation algorithms have been developed for solving them in practice. However, state-of-the-art heuristic algorithms require the input graph G to be complete, in which the edge joining two vertices is also the shortest path between them. When G doesn't satisfy this property, these heuristic algorithms have to be invoked only after computing the metric closure of G, which in turn requires the computation of all-pairs shortest-path (APSP) distances. Existing APSP algorithms, such as the Floyd-Warshall algorithm, have a poor time complexity, making APSP computations a bottleneck for deploying the heuristic algorithms on large VKC and VKM instances. To remedy this, we propose the use of a novel algorithmic pipeline based on a graph embedding algorithm called FastMap. FastMap is a near-linear-time algorithm that embeds the vertices of G in a Euclidean space while approximately preserving the shortest-path distances as Euclidean distances for all pairs of vertices. The FastMap embedding can be used to circumvent the barrier of APSP computations, creating a very efficient pipeline for solving FLPs. On the empirical front, we provide test results that demonstrate the efficiency and effectiveness of our novel approach.

1 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. 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. From an agent-centric perspective, FLPs serve the purpose of orchestrating shared resources between multiple agents. FLPs can be defined on geometric spaces or on graphs, on continuous or discrete spaces, and with a variety of distance metrics and objectives. A compendium of FLPs along with various algorithms and case studies can be found in [24].

FLPs defined on graphs are very general and broadly applicable. The Vertex K-Center (VKC) problem and the Vertex K-Median (VKM) problem are two such fundamental FLPs defined on graphs. The VKC (VKM) problem seeks K vertices on the

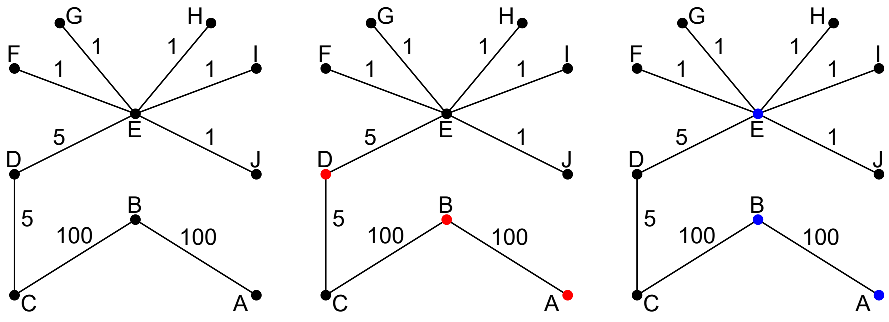


Fig. 1. Examples of the VKC and the VKM problems on the same input graph: The two problems have different optimal solutions for the same value of K. The left panel shows the input graph. The middle panel shows the optimal solution in red for the VKC problem with $K = 3$. The right panel shows the optimal solution in blue for the VKM problem with $K = 3$.

input graph for the placement of facilities so as to minimize the farthest (aggregate) distance of all vertices to their nearest facility. Both the VKC and the VKM problems have many real-world applications—often in the same domain but with slightly different objective functions. For example, in urban development, they can be used to optimally place various public service centers within a city. In communication networks, they can be used to determine the optimal placement of computation sites for critical multiplexing and the optimal placement of traffic merging sites while deploying network coding.

Formally, in both the VKC and the VKM problems, we are given an undirected edge-weighted graph $G = (V, E)$ and seek a subset of vertices $S \subseteq V$ of cardinality K. In the VKC problem, we are required to minimize $\max_{v \in V} \min_{u \in S} d(v, u)$ while, in the VKM problem, we are required to minimize $\sum_{v \in V} \min_{u \in S} d(v, u)$. Here, $d(u, v) = d(v, u)$ is the shortest-path distance between u and v in G. Figure 1 shows examples of both these problems posed on the same graph for $K = 3$.

Both the VKC and the VKM problems are computationally NP-hard to solve optimally [33]. However, many heuristic and approximation algorithms have been developed for solving them in practice. For example, the Gonzalez (GON) algorithm [21,31] is among the fastest algorithms proposed to solve the VKC problem in $O(K|V|)$ time, achieving a factor-2 approximation. Similarly, Partition Around Medoids (PAM) [55], a local search procedure proposed for the VKM problem [5] arrives at a near-optimal solution by repeatedly swapping a vertex from its current solution S with a vertex in $V \setminus S$. It converges very quickly; and by restricting the number of swaps to a large enough constant, it terminates in $O(K^2|V|^2)$ time. We discuss more algorithms for these two problems in the Related Work section.

Despite the existence of many previous works, one of the drawbacks of the existing heuristic algorithms for the VKC and the VKM problems is that they require the input graph G to be complete, in which the edge joining two vertices is also the shortest path between them. This assumption is primarily made so that the heuristic algorithms can focus on the combinatorially hard part of the problem. If G doesn't satisfy this property, these heuristic algorithms can still be effective but should be invoked only after computing the metric closure of G.

Technically, the metric closure of G can be computed in polynomial time by calculating the all-pairs shortest-path (APSP) distances. However, APSP algorithms, such as the Floyd-Warshall algorithm [25], are computationally expensive with their running time complexity typically being cubic in $|V|$. Because of these limitations, APSP algorithms quickly become a computational bottleneck for deploying heuristic algorithms on large VKC and VKM instances.

Some APSP algorithms are based on fast matrix multiplication and achieve sub-cubic running time complexities, but these are better than the Floyd-Warshall algorithm only for very large values of $|V|$. There also exist several other algorithms with better running time complexities [3,35], but these are much more complicated than the Floyd-Warshall algorithm and rely on complicated data structures. Hence, in most cases, the Floyd-Warshall algorithm is still the APSP algorithm of choice, notwithstanding the issue of being the bottleneck for solving large VKC and VKM instances.

In this paper, we address this issue by using a novel algorithmic pipeline based on a graph embedding algorithm called FastMap. In general, graph embeddings have been used in many different contexts such as for shortest-path computations [15], multi-agent meeting problems [46], community detection and block modeling [45], and social network analysis [51]. They are useful as they facilitate geometric interpretations and algebraic manipulations in vector spaces. FastMap [15,46] is a recently developed graph embedding algorithm that runs in near-linear time[3]. It embeds the vertices of a given undirected graph into a Euclidean space such that the pairwise Euclidean distances between vertices approximate the shortest-path distances between them in the graph.

We use the FastMap embedding as an alternative to APSP algorithms, creating a very efficient pipeline for solving FLPs on graphs. We provide empirical results demonstrating the efficiency and effectiveness of our proposed FastMap pipeline for the VKC and the VKM problems. We show that, for the same or similar qualities of solutions, the FastMap pipeline is significantly faster than the Floyd-Warshall pipeline.

2 FastMap

FastMap [23] was introduced in the Data Mining community for automatically generating Euclidean embeddings of abstract objects. For many real-world objects such as DNA strings, multi-media datasets like voice excerpts or images, medical datasets like ECGs or MRIs, there is no geometric space in which they can be naturally visualized. However, there is often a well-defined distance function for every pair of objects in the problem domain. For example, the edit distance[4] between two DNA strings is well defined although an individual DNA string cannot be conceptualized in geometric space.

FastMap embeds a collection of abstract objects in an artificially created Euclidean space to enable geometric interpretations, algebraic manipulations, and downstream Machine Learning algorithms. It gets as input a collection of abstract objects $\mathcal{O}$, where

[3] linear time after ignoring logarithmic factors

[4] The edit distance between two strings is the minimum number of insertions, deletions, or substitutions that are needed to transform one to the other.

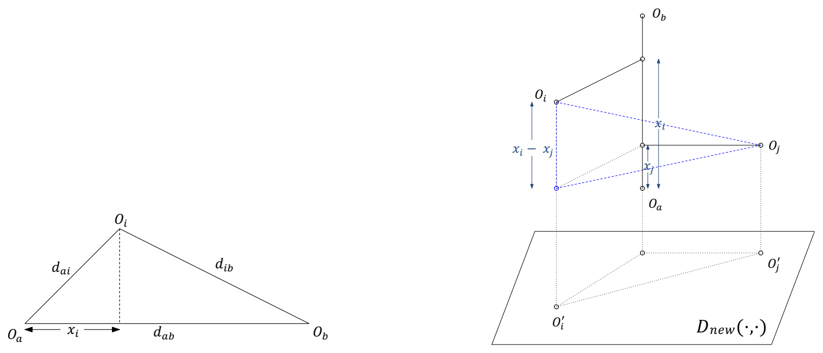


Fig. 2. Illustration of how coordinates are computed in FastMap, borrowed from [15]: The left panel illustrates the "cosine law" projection in a triangle. The right panel illustrates the process of projecting onto a hyperplane that is perpendicular to $\overline{O_a O_b}$.

$D(O_i, O_j)$ represents the domain-specific distance between objects $O_i, O_j \in \mathcal{O}$. A Euclidean embedding assigns a κ-dimensional point $p_i \in \mathbb{R}^\kappa$ to each object O_i. A good Euclidean embedding is one in which the Euclidean distance χ_{ij} between any two points p_i and p_j closely approximates $D(O_i, O_j)$. For $p_i = ([p_i]_1, [p_i]_2 \dots [p_i]_\kappa)$ and $p_j = ([p_j]_1, [p_j]_2 \dots [p_j]_\kappa)$, $\chi_{ij} = \sqrt{\sum_{r=1}^{\kappa} ([p_j]_r - [p_i]_r)^2}$.

FastMap creates a κ-dimensional Euclidean embedding of the abstract objects in $\mathcal{O}$, for a user-specified value κ. In the very first iteration, it heuristically identifies the farthest pair of objects O_a and O_b in linear time. 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 $d_{ib} = D(O_i, O_b)$, as shown in Figure 2 (left panel). The sides of the triangle define its entire geometry, and the projection of O_i onto the line $\overline{O_a O_b}$ is given by

$$x_i = (d_{ai}^2 + d_{ab}^2 - d_{ib}^2)/(2d_{ab}). \quad (1)$$

FastMap sets the first coordinate of p_i, the embedding of O_i, to x_i. In the subsequent $\kappa - 1$ iterations, the same procedure is followed for computing the remaining $\kappa - 1$ 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 domain-specific distance between these two objects, from the second iteration onward, 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 $\overline{O_a O_b}$, as shown in Figure 2 (right panel). Although the hyperplane is never constructed explicitly, its conceptualization implies that the distance function for the second iteration should be changed for all i and j in the following way:

$$D_{new}(O_i', O_j')^2 = D(O_i, O_j)^2 - (x_i - x_j)^2. \quad (2)$$

Here, O_i' and O_j' are the projections of O_i and O_j, respectively, onto this hyperplane, and $D_{new}(\cdot, \cdot)$ is the new distance function.

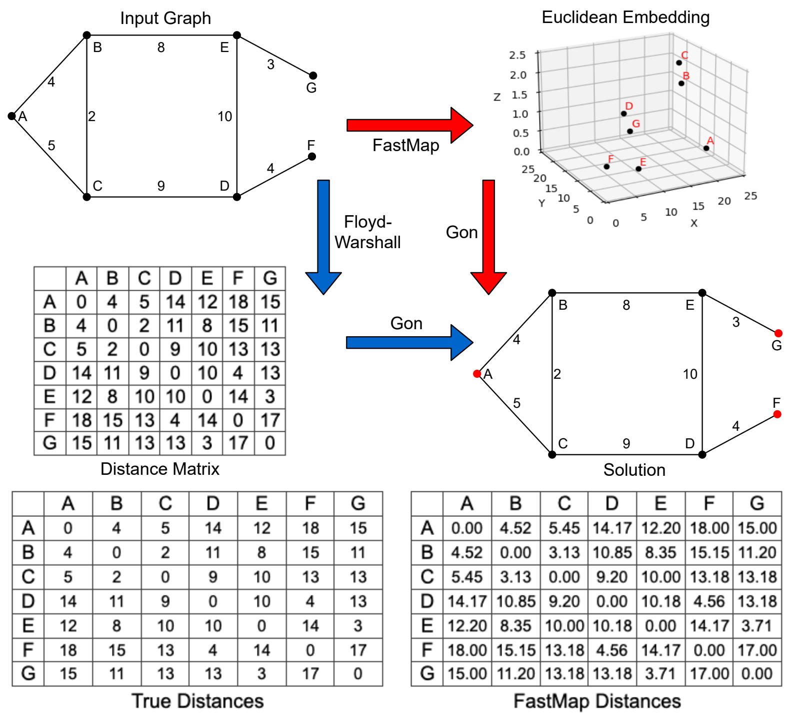
Algorithm 1: FastMap: A near-linear-time graph embedding algorithm.

```
Input: $G = (V, E)$, $\kappa$, and $\epsilon$.
Output: $p_i \in \mathbb{R}^r$ for all $v_i \in V$.
1  for $r = 1, 2 \ldots \kappa$ do
2      Choose $v_a \in V$ randomly and let $v_b = v_a$;
3      for $t = 1, 2 \ldots C$ do                              // $C$ is a constant.
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     $\{d_{ai} : v_i \in V\} \leftarrow$ ShortestPathTree$(G, v_a)$;
11     $\{d_{ib} : v_i \in V\} \leftarrow$ ShortestPathTree$(G, v_b)$;
12     $d'_{ab} \leftarrow d_{ab}^2 - \sum_{j=1}^{r-1}([p_a]_j - [p_b]_j)^2$;
13     if $d'_{ab} < \epsilon$ then
14         Break;
15     for each $v_i \in V$ do
16         $d'_{ai} \leftarrow d_{ai}^2 - \sum_{j=1}^{r-1}([p_a]_j - [p_i]_j)^2$;
17         $d'_{ib} \leftarrow d_{ib}^2 - \sum_{j=1}^{r-1}([p_i]_j - [p_b]_j)^2$;
18         $[p_i]_r \leftarrow (d'_{ai} + d'_{ab} - d'_{ib})/(2\sqrt{d'_{ab}})$;
19 return $p_i$ for all $v_i \in V$.
```

FastMap can also be used to embed the vertices of a graph in a Euclidean space to preserve the pairwise shortest-path distances between them. The idea is to view the vertices of a given graph $G = (V, E)$ as the objects to be embedded. As such, the Data Mining FastMap algorithm cannot be directly used for generating an embedding in linear time. This is because it assumes that the distance d_{ij} between any two objects O_i and O_j can be computed in constant time, independent of the number of objects in the problem domain. However, computing the shortest-path distance between two vertices depends on the size of the graph.

The issue of having to retain (near-)linear time complexity can be addressed as follows: In each iteration, after we heuristically identify the farthest pair of vertices O_a and O_b, the distances d_{ai} and d_{ib} need to be computed for *all* other vertices O_i. Computing d_{ai} and d_{ib} for any single vertex O_i can no longer be done in constant time but requires $O(|E| + |V| \log |V|)$ time instead [27]. However, since we need to compute these distances for all vertices, computing two shortest-path trees rooted at each of the vertices O_a and O_b yields all necessary distances in one shot. The complexity of doing so is also $O(|E| + |V| \log |V|)$, which is only linear in the size of the graph[5]. The

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



Distance Matrix

	A	B	C	D	E	F	G
A	0	4	5	14	12	18	15
B	4	0	2	11	8	15	11
C	5	2	0	9	10	13	13
D	14	11	9	0	10	4	13
E	12	8	10	10	0	14	3
F	18	15	13	4	14	0	17
G	15	11	13	13	3	17	0

	A	B	C	D	E	F	G
A	0	4	5	14	12	18	15
B	4	0	2	11	8	15	11
C	5	2	0	9	10	13	13
D	14	11	9	0	10	4	13
E	12	8	10	10	0	14	3
F	18	15	13	4	14	0	17
G	15	11	13	13	3	17	0

True Distances

	A	B	C	D	E	F	G
A	0.00	4.52	5.45	14.17	12.20	18.00	15.00
B	4.52	0.00	3.13	10.85	8.35	15.15	11.20
C	5.45	3.13	0.00	9.20	10.00	13.18	13.18
D	14.17	10.85	9.20	0.00	10.18	4.56	13.18
E	12.20	8.35	10.00	10.18	0.00	14.17	3.71
F	18.00	15.15	13.18	4.56	14.17	0.00	17.00
G	15.00	11.20	13.18	13.18	3.71	17.00	0.00

FastMap Distances

Fig. 3. The FastMap pipeline and its comparison with the Floyd-Warshall pipeline: The FastMap pipeline uses a Euclidean embedding instead of an APSP distance matrix (left). The distortion in the APSP distances implicitly produced by FastMap (right) rarely affects the quality of the final solution.

amortized complexity for computing d_{ai} and d_{ib} for any single vertex O_i is therefore near-constant time.

The foregoing observations are used in [46] to build a graph-based version of FastMap that embeds the vertices of a given undirected graph in a Euclidean space in near-linear time. The Euclidean distances approximate the pairwise shortest-path distances between vertices. Algorithm 1 presents the pseudocode for this algorithm.

A slight modification of this FastMap algorithm, presented in [15], can also be used to preserve *consistency* and *admissibility* of the Euclidean distance approximation used as a heuristic in A* search for shortest-path computations. In both [15] and [46], κ is user-specified, but a threshold parameter ϵ is introduced to detect large values of κ that have diminishing returns on the accuracy of approximating pairwise shortest-path distances.

3 The FastMap Pipeline

We will now exploit the efficiency of the FastMap algorithm towards APSP computations. After FastMap computes the Euclidean embedding of the given graph in near-

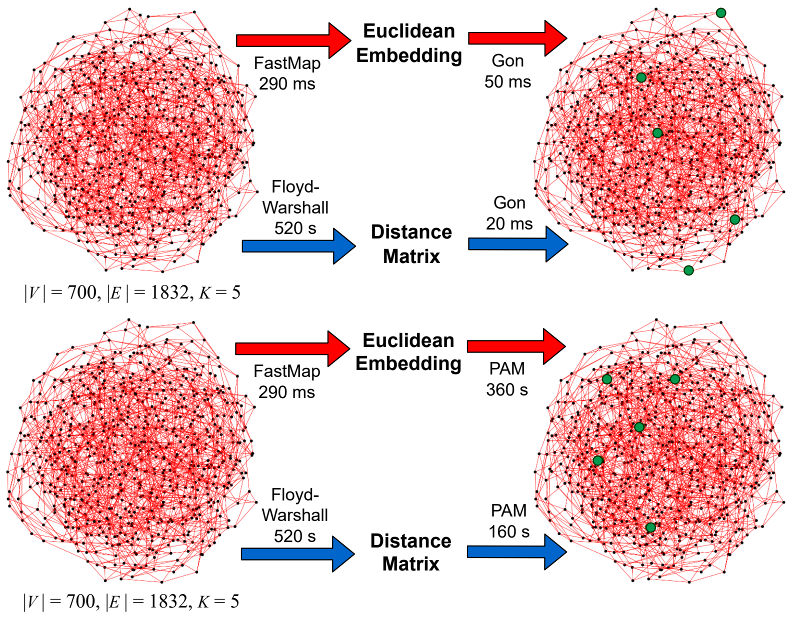


Fig. 4. Efficiency of the FastMap pipeline for solving the VKC (top) and VKM (bottom) problems: For the same quality of the final VKC solution, FastMap+GON takes 0.34 s, while Floyd-Warshall+GON takes 520 s. Here, the FastMap pipeline yields a $1529\times$ speedup. For the same quality of the final VKM solution, FastMap+PAM takes 360 s, while Floyd-Warshall+PAM takes 680 s. Here, the FastMap pipeline yields a $1.887\times$ speedup. The final solutions are marked by the green vertices.

linear time, the Euclidean distance between any pair of its vertices serves to approximate the shortest-path distance between them. Since Euclidean distances can be computed in $O(\kappa)$ time, independent of the size of the graph, FastMap efficiently sets up the groundwork for solving the VKC and VKM problems.

Figures 3 and 4 show the FastMap pipeline in comparison with the Floyd-Warshall pipeline for solving the VKC and VKM problems. While both pipelines can invoke the same heuristic algorithm of choice for solving the VKC or VKM problem, the FastMap pipeline is much faster because of its efficiency in the APSP computations.

For the VKC problem, we can use the GON algorithm [21, 31]. It is simple to implement, has a low running time complexity, and yields a factor-2 approximation. In the first iteration, it picks a random vertex and nominates it as a center. In each subsequent iteration, it picks a vertex that is farthest away from any of the existing centers and nominates it as an additional new center. Thus, a solution is obtained after K iterations. GON runs in $O(K|V|)$ time and produces a factor-2 approximation.

For the VKM problem, we can use the PAM algorithm [55]. Although PAM has several variants, a simple version of it with a naive implementation suffices for demonstrating the effectiveness of our FastMap pipeline. In fact, improved versions of PAM increase the benefits of the FastMap pipeline since the bottleneck of APSP computations

K	Instance	Vertices	Edges	Quality Factor	Speedup
5	queen7_7	49	476	0.800	3.483
	myciel7	191	2360	0.889	50.328
	queen16_16	256	6320	1.167	60.620
	le450_25b	450	8263	1.333	254.702
10	queen8_8	64	728	1.000	3.585
	games120	120	638	0.750	16.598
	myciel7	191	2360	1.500	24.973
	le450_5c	450	9803	1.333	98.959
20	myciel6	95	755	1.000	5.887
	miles1000	128	3216	1.333	4.728
	queen14_14	196	4186	1.500	13.760
	le450_5d	450	9757	1.250	59.687
40	queen10_10	100	1470	1.500	3.348
	games120	120	638	1.167	7.991
	queen12_12	144	2596	1.333	6.263
	queen16_16	256	6320	1.000	13.227

Table 1. Results for VKC on DIMACS.

K	Instance	Vertices	Edges	Quality Factor	Speedup
5	n0100k6p0.6	100	483	1	22.704
	n0300k4p0.3	300	776	1	290.837
	n0600k6p0.6	600	2865	1	810.219
	n0600k4p0.3	600	1564	1	1197.670
10	n0100k4p0.6	100	315	1	12.446
	n0400k4p0.6	400	1277	1.333	267.190
	n0700k6p0.6	700	3355	1	537.126
	n0700k4p0.3	700	1832	1	771.443
20	n0100k4p0.3	100	262	1	9.831
	n0200k4p0.6	200	651	1.5	40.005
	n0600k6p0.6	600	2865	1	278.556
	n0700k6p0.3	700	2705	1.333	472.860
40	n0100k4p0.6	100	315	2	6.049
	n0400k6p0.6	400	1951	1	88.304
	n0600k4p0.3	600	1564	1.333	268.690
	n0700k4p0.6	700	2244	1.333	209.628

Table 2. Results for VKC on Small World.

becomes more pronounced. PAM constructs an initial solution greedily. It then invokes local search to improve the quality of the solution by repeatedly swapping a vertex from its current solution S with a vertex in $V \setminus S$. While a non-trivial bound on the number of iterations required for convergence is not known, fewer than K iterations are usually observed in practice [57, 63]. In addition, with slightly modified swapping conditions, convergence within a polynomial number of iterations can be guaranteed [5].

Of course, the FastMap pipeline introduces some intermediate distortion in the APSP distances. But this distortion is usually not much and is a very small price to pay for huge benefits in running times, both complexity-wise and in actual wall-clock times. In fact, for the examples chosen in Figures 3 and 4, the FastMap pipeline does not change the qualities of the final solutions. But it runs several orders of magnitude faster than the Floyd-Warshall pipeline. The running time of the GON or PAM algorithm is slightly higher in the FastMap pipeline compared to that in the Floyd-Warshall pipeline. This is because, in the FastMap pipeline, the pairwise distances cannot be looked up in a distance matrix but should now be computed using the Euclidean coordinates, requiring $O(\kappa)$ time. However, since κ is a small number, usually less than 5, the enormous savings in the APSP computations continue to be the dominant factor benefiting the FastMap pipeline.

The same patterns in the benefits of the FastMap pipeline are also observed on several kinds of benchmark problem instances, as reported in the next section.

4 Experimental Results

In this section, we present experimental results comparing the FastMap pipeline and the Floyd-Warshall pipeline for solving VKC and VKM problem instances. For solving the VKC problem instances, the FastMap pipeline uses FastMap+GON and the Floyd-Warshall pipeline uses Floyd-Warshall+GON. For solving the VKM problem instances, the FastMap pipeline uses FastMap+PAM and the Floyd-Warshall pipeline uses Floyd-Warshall+PAM. We implemented all algorithms and experimentation procedures using Python3 with the NetworkX library. For the Floyd-Warshall algorithm, several state-of-

K	Instance	Vertices	Edges	Quality Factor	Speedup
5	pmed2	100	193	0.859	5.550
	pmed9	200	785	1.018	11.435
	pmed29	600	7042	0.943	33.606
	pmed39	900	15896	–	–
10	pmed5	100	196	0.865	3.189
	pmed13	300	1760	1.174	10.573
	pmed21	500	4909	0.962	14.950
	pmed38	900	15898	–	–
20	pmed10	200	786	0.971	3.633
	pmed17	400	3142	1.020	7.908
	pmed29	600	7042	1.105	11.721
	pmed39	900	15896	–	–
40	pmed6	200	786	1.460	2.813
	pmed14	300	1771	1.204	4.100
	pmed28	600	7054	1.129	6.985
	pmed40	900	15879	–	–

Table 3. Results for VKC on ORLib.

K	Instance	Vertices	Edges	Quality Factor	Speedup
5	n0100	100	99	0.981	36.152
	n0300	300	299	1.168	346.328
	n0600	600	599	1.036	1437.121
	n1000	1000	999	–	–
10	n0200	200	199	0.841	83.738
	n0500	500	499	1.027	495.447
	n0700	700	699	0.974	1008.908
	n1000	1000	999	–	–
20	n0300	300	299	1.020	110.636
	n0400	400	399	1.217	200.612
	n0600	600	599	0.888	301.079
	n1000	1000	999	–	–
40	n0300	300	299	1.452	76.859
	n0500	500	499	1.549	220.077
	n0700	700	699	1.051	446.138
	n1000	1000	999	–	–

Table 4. Results for VKC on Tree.

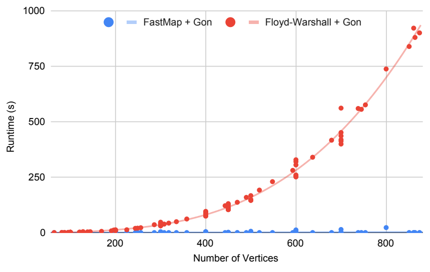


Fig. 5. Runtime scaling for VKC.

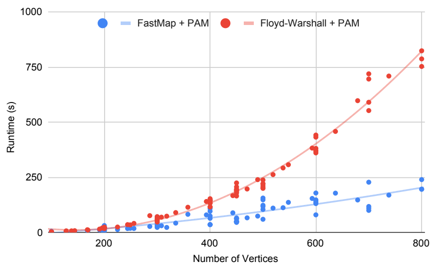


Fig. 6. Runtime scaling for VKM.

K	Instance	Vertices	Edges	Quality Factor	Speedup
5	orz106d	335	602	0.846	316.146
	lak102d	519	920	1.000	839.839
	hrt002d	754	1300	1.192	2093.917
	lak526d	954	1715	–	–
10	orz203d	244	442	1.000	106.603
	den404d	358	632	0.889	219.150
	orz105d	679	1245	1.000	733.192
	lak526d	954	1715	–	–
20	ost102d	249	447	1.000	65.042
	lak105d	443	766	1.167	222.258
	lak104d	851	1570	1.125	822.030
	den009d	1003	1863	–	–
40	den404d	358	632	1.333	97.559
	den408d	548	991	1.250	226.418
	lak104d	851	1570	1.200	582.540
	den009d	1003	1863	–	–

Table 5. Results for VKC on MovingAI.

K	Instance	Vertices	Edges	Quality Factor	Speedup
5	queen7_7	49	476	1.244	0.647
	queen10_10	100	1470	1.064	1.625
	queen14_14	196	4186	1.153	2.116
	queen16_16	256	6320	1.194	2.302
	le450_15c	450	16680	1.178	3.947
	p-hat700-1	700	60999	1.213	5.905
10	queen5_5	25	160	1.136	0.686
	miles500	128	1170	1.208	0.805
	queen14_14	196	4186	1.193	1.199
	le450_5d	450	9757	1.159	1.036
	le450_5a	450	5714	1.183	1.776
	p-hat700-1	700	60999	–	–

Table 6. Results for VKM on DIMACS.

the-art implementations with code-level optimizations are available. They work particularly well for certain classes of graphs. However, due to the inherent difference in the asymptotic complexities of the two pipelines, on large enough instances with no special properties, the Floyd-Warshall pipeline can be shown to be significantly slower than the FastMap pipeline. Hence, for a fair comparison of the two pipelines on benchmark

K	Instance	Vertices	Edges	Quality Factor	Speedup
5	n0300k6p0.3	300	1168	1.133	1.691
	n0400k6p0.3	400	1562	1.120	2.244
	n0500k4p0.3	500	1281	1.241	2.107
	n0600k4p0.3	600	1564	1.253	2.912
	n0700k6p0.3	700	2705	1.173	3.157
	n0800k4p0.6	800	2561	–	–
10	n0200k6p0.3	200	787	1.187	0.971
	n0300k6p0.6	300	1440	1.124	1.008
	n0400k6p0.6	400	1951	1.211	1.492
	n0700k6p0.3	700	2705	1.157	2.200
	n0800k4p0.3	800	2062	1.214	1.781
	n0800k4p0.6	800	2561	–	–

Table 7. Results for VKM on Small World.

K	Instance	Vertices	Edges	Quality Factor	Speedup
5	pmed8	200	792	1.242	0.853
	pmed14	300	1771	1.236	1.139
	pmed20	400	3144	1.410	1.200
	pmed22	500	4896	1.153	1.350
	pmed28	600	7054	1.411	3.035
	pmed39	900	15896	–	–
10	pmed2	100	193	1.162	1.020
	pmed14	300	1771	1.195	1.206
	pmed15	300	1754	1.197	1.152
	pmed17	400	3142	1.261	1.204
	pmed28	600	7054	1.279	1.894
	pmed39	900	15896	–	–

Table 8. Results for VKM on ORLib.

K	Instance	Vertices	Edges	Quality Factor	Speedup
5	n0100	100	99	1.287	1.140
	n0200	200	199	1.398	2.469
	n0300	300	299	1.393	1.624
	n0500	500	499	1.447	3.398
	n0700	700	699	1.361	4.695
	n0900	900	899	–	–
10	n0100	100	99	1.435	1.160
	n0200	200	199	1.484	0.971
	n0300	300	299	1.593	1.354
	n0400	400	399	1.511	1.301
	n0500	500	499	1.405	1.515
	n0800	800	799	–	–

Table 9. Results for VKM on Tree.

K	Instance	Vertices	Edges	Quality Factor	Speedup
5	lak110d	168	290	1.070	1.452
	orz203d	244	442	1.037	2.013
	den404d	358	632	0.988	1.239
	ht_store	490	910	1.030	4.428
	orz105d	679	1245	1.027	4.059
	den405d	925		–	–
10	lak110d	168	290	1.018	1.342
	orz203d	244	442	1.041	1.688
	den404d	358	632	1.004	1.716
	lak107d	393	710	1.038	2.272
	den408d	548	991	1.025	1.476
	orz102d	738	1359	–	–

Table 10. Results for VKM on MovingAI.

problem instances, while focusing on the quality of the solutions produced, we used a vanilla implementation of the Floyd-Warshall algorithm meant for general graphs. For the FastMap algorithm in Algorithm 1, we used $\kappa = 4$ and $\epsilon = 10^{-4}$. All experiments were conducted on a laptop with a 1.6GHz Intel Core i5 processor and 8GB 1600MHz DDR3 memory.

We used both VKC and VKM problem instances derived from different benchmark datasets. These include the DIMACS[6], Small World, ORLib[7], Tree, and MovingAI[8] datasets. For the DIMACS instances, each edge was assigned an integer weight chosen uniformly at random from the interval $[1, 10]$. The Small World instances were generated using the Newman-Watts-Strogatz graph generator in NetworkX. All edges were assigned a unit weight. The names of these instances indicate the parameter values for n (the number of vertices), k (the number of neighbors in a ring), and p (the probability of adding a new edge). The original ORLib graphs already have weights on the edges, although different values of K were chosen for the experiments. The Tree instances were generated using NetworkX. Each edge was assigned an integer weight chosen

[6] The DIMACS instances were generated using the DIMACS graphs from `http://networkrepository.com/dimacs.php` and `https://mat.tepper.cmu.edu/COLOR/instances.html`.

[7] The ORLib instances were generated using the ORLib graphs from `http://people.brunel.ac.uk/~mastjjb/jeb/orlib/pmedinfo.html`.

[8] The MovingAI instances were generated using the MovingAI graphs from `https://movingai.com/benchmarks`.

uniformly at random from the interval $[1, 10]$. For the MovingAI instances, all edges were assigned a unit weight.

Because various components of the two pipelines use randomization, we used 5 trials for each pipeline on each problem instance and compared the best solutions found by them. This comparison is reported as a "Quality Factor". In essence, the Quality Factor is the cost of the solution found by the FastMap pipeline divided by the cost of the solution found by the Floyd-Warshall pipeline. If the Quality Factor $= 1$, the FastMap pipeline retains the same quality of the final solution as the Floyd-Warshall pipeline. If it is > 1, the FastMap distortion in the APSP distances produces a costlier solution compared to the Floyd-Warshall pipeline. Sometimes, the Quality Factor can even be < 1, indicating that the FastMap pipeline produces a better solution compared to the Floyd-Warshall pipeline. This can happen because of randomization and other heuristic components in the two pipelines. We also report a "Speedup" factor, which is the time taken by the Floyd-Warshall pipeline in the 5 trials divided by the time taken by the FastMap pipeline in the 5 trials.

Tables 1, 2, 3, 4, and 5 show the results on some representative VKC problem instances derived from the DIMACS, Small World, ORLib, Tree, and MovingAI datasets, respectively. In these tables, a "–" indicates that the Floyd-Warshall pipeline timed out after $1000\,\mathrm{s}$ on each of the 5 trials. In such cases, the FastMap pipeline still generated a solution. The FastMap pipeline yields significant speedup on all the datasets for only marginal compromises on the solution qualities. In fact, the FastMap pipeline is orders of magnitude faster than the Floyd-Warshall pipeline for larger problem instances. Figure 5 visualizes and compares the running times of the two pipelines on all the problems instances, barring the ones on which the Floyd-Warshall pipeline timed out.

Tables 6, 7, 8, 9, and 10 show the results on some representative VKM problem instances derived from the DIMACS, Small World, ORLib, Tree, and MovingAI datasets, respectively. In these tables, a "–" indicates that the Floyd-Warshall pipeline timed out after $1000\,\mathrm{s}$ on each of the 5 trials. In such cases, the FastMap pipeline still generated a solution. The FastMap pipeline yields significant speedup on all the datasets for only marginal compromises on the solution qualities. Figure 6 visualizes and compares the running times of the two pipelines on all the problems instances, barring the ones on which the Floyd-Warshall pipeline timed out. The speedup for VKM problem instances is less than that for VKC problem instances since the PAM algorithm is not as efficient as the GON algorithm.

5 Related Work

The VKC problem we considered in this paper is the *uncapacitated unweighted* version. Several other variants have been studied. These include the *capacitated* VKC problem [43], where each center can serve only a fixed number of vertices, the *heterogeneous capacitated* VKC problem [8], which is similar to the capacitated VKC problem, except that the capacities of different centers may be different, the *aligned* K-center problem in Euclidean space [6], where the centers must be selected from a line or a polygon, the *edge-dilation* VKC problem [44], where the goal is to minimize the maximum ratio of the distance between two vertices via their respective centers to their shortest-path distance, the *fault-tolerant* VKC problem [42], where each selected center must have a set of $\alpha \leq K$ centers close to it, the *p-neighbor* VKC problem [12], where, given an integer p, the goal is to minimize the maximum distance of any non-center vertex to its p^{th} closest center, among many other variants.

Several exact algorithms have been proposed for the VKC problem considered in this paper. They are primarily based on Integer Programming or Mixed Integer Programming formulations [1, 7, 13, 16, 18, 22, 49]. None of them run in polynomial time since the VKC problem is NP-hard to solve optimally. Several meta-heuristic algorithms have also been proposed, such as Tabu Search [48], Variable Neighborhood Search [37, 48], Scatter Search [50], GRASP [50], Memetic Genetic Algorithms [53], Harmony Search [41], and Bee Colony Optimization [19]. While these algorithms may provide better performance in practice, they are not guaranteed to converge quickly or to find optimal solutions. The VKC problem is factor-2 approximable in polynomial time. The polynomial-time algorithms that guarantee this approximation include the SH algorithm [52, 59], and its refinements—the GON algorithm [21, 31], and the HS algorithm [36, 52]. The greedy GR algorithm [54, 56], the SCR algorithm [56], and the CDSH algorithm [30] are among the polynomial-time heuristic algorithms that yield the best empirical performance. However, these algorithms are significantly slower than the GON algorithm. In this paper, we used the GON algorithm because of its lower runtime.

The VKM problem is closely related to the general uncapacitated facility location problem [17], with a restriction on the number of facilities (centers) that can be opened, but with no costs for opening them. One of the early proposals for solving the VKM problem was a reverse greedy algorithm [14]. It starts by opening all vertices as centers and, in each iteration, closes a center that increases the total cost by the least amount, until K of them remain. It achieves an $O(\log |V|)$ approximation factor. A frequently used local search algorithm is the PAM algorithm [55]. It iteratively finds cost-lowering swaps of vertices into and out of the current candidate solution, until convergence to a local optimum is achieved. While we used a simple version of PAM in this paper, a more generalized version with p swaps allowed in each iteration, along with a slight modification in the swapping condition, is presented in [5]. This generalized version of PAM expends $O(|V|^{O(p)})$ time in each iteration but is guaranteed to converge after a polynomial number of iterations, achieving an approximation factor of $3+2/p$. Another polynomial-time algorithm that yields a factor-$6\frac{2}{3}$ approximation is based on a Linear Programming relaxation and rounding scheme [10]. Similar techniques have also been proposed by others. A factor-4 polynomial-time approximation algorithm is presented in [38, 39]. An algorithm that achieves an approximation factor of $3.25(1 + \delta)$ with running time $O(K^3|V|^2/\delta^2)$ is presented in [11]. This algorithm outperforms PAM empirically [20]. However, PAM is still very competitive and is used in this paper for its simplicity. Another algorithm that achieves an approximation factor of $1 + \sqrt{3} + \epsilon$ with running time $|V|^{O(1/\epsilon^2)}$ is presented in [47].

A classical algorithm for APSP computations is the Floyd-Warshall algorithm [25]. It uses dynamic programming and runs in $O(|V|^3)$ time. Johnson's algorithm [40] runs in $O(|V||E| + |V|^2 \log |V|)$ time, but it assumes the absence of negative-cost cycles in the graph. For directed graphs with non-negative weights on the edges, APSP computations are closely related to the *distance product* of two matrices. A popular algorithm that exploits this connection has running time $O(|V|^3(\log\log|V|/\log|V|)^{1/2})$ [26,60]. For graphs with integer weights on the edges, [34] presents an algorithm that runs in $O(|V||E| + |V|^2 \log\log|V|)$ time. For undirected graphs with integer weights on the edges, the running time can be improved to $O(|V||E|)$ [61, 62]. For general graphs, several algorithms have improved logarithmic factors in their running times. For example, [35] achieves a running time of $O(|V|^3 \log\log|V|/\log^2|V|)$. Algorithms for fast matrix multiplication can also be invoked to obtain sub-cubic-time APSP algorithms for a large class of "geometrically weighted" graphs [9]. For graphs embedded in a 2D Euclidean space, such an algorithm has running time $O(|V|^{2.922})$. Several other works, such as [3,4,28,29,58] have shown that APSP computations can be done in $\tilde{O}(M(|V|))$ time for unweighted graphs and in $\tilde{O}(\sqrt{|V|^3 M(|V|)})$ time for weighted graphs, where $M(n)$ is the time complexity of $n \times n$ matrix multiplication, currently known to be $O(n^{2.37286})$ [2].

6 Conclusions and Future Work

FastMap is a near-linear-time algorithm that embeds the vertices of a graph in a Euclidean space while approximately preserving the shortest-path distances as Euclidean distances for all pairs of its vertices. In this paper, we presented a FastMap-based approach for solving FLPs, adding to the list of FastMap's previous applications in multi-agent domains. We demonstrated the efficiency and effectiveness of our novel FastMap pipeline on two fundamental and vital FLPs defined on graphs: the VKC and the VKM problems. Both these problems are NP-hard to solve optimally; but enabling efficient heuristics and approximation algorithms is the key to solving them well in practice. Existing state-of-the-art heuristic algorithms rely on the input being a complete graph with edges representing shortest paths. Consequently, an input graph that doesn't satisfy this property has to be first rendered amenable by computing its metric closure via APSP algorithms like the Floyd-Warshall algorithm, which becomes a critical bottleneck when deploying fast heuristics on large VKC and VKM instances. Our proposed FastMap pipeline circumvents this barrier of APSP computations. Through empirical results on a wide variety of VKC and VKM instances, we showed that the distortion of pairwise distances in the FastMap embedding does not affect the quality of the final output by much: For the same or similar qualities of solutions, the FastMap pipeline is significantly faster than the Floyd-Warshall pipeline.

In future work, we will consider reducing the distortion of APSP distances caused by the FastMap embedding using Machine Learning techniques to learn the correction factors. In fact, the FastMap coordinates can themselves be used as features for the learning, as illustrated in [32]. The key challenge for such a self-supervised approach is to minimize the number of training samples and retain the end-to-end efficiency of the pipeline. Another direction of future work is to apply our efficient FastMap pipeline to other kinds of FLPs.

References

1. Al-Khedhairi, A., Salhi, S.: Enhancements to two exact algorithms for solving the vertex p-center problem. Journal of Mathematical Modelling and Algorithms **4**(2), 129–147 (2005)

2. Alman, J., Williams, V.V.: A refined laser method and faster matrix multiplication. In: Proceedings of the 2021 ACM-SIAM Symposium on Discrete Algorithms (SODA). pp. 522–539. SIAM (2021)
3. Alon, N., Galil, Z., Margalit, O.: On the exponent of the all pairs shortest path problem. Journal of Computer and System Sciences **54**(2), 255–262 (1997)
4. Alon, N., Naor, M.: Derandomization, witnesses for boolean matrix multiplication and construction of perfect hash functions. Algorithmica **16**(4), 434–449 (1996)
5. Arya, V., Garg, N., Khandekar, R., Meyerson, A., Munagala, K., Pandit, V.: Local search heuristics for k-median and facility location problems. SIAM Journal on computing **33**(3), 544–562 (2004)
6. Brass, P., Knauer, C., Na, H.S., Shin, C.S., Vigneron, A.: The aligned k-center problem. International Journal of Computational Geometry & Applications **21**(02), 157–178 (2011)
7. Calik, H., Tansel, B.C.: Double bound method for solving the p-center location problem. Computers & operations research **40**(12), 2991–2999 (2013)
8. Chakrabarty, D., Krishnaswamy, R., Kumar, A.: The heterogeneous capacitated k-center problem. In: International Conference on Integer Programming and Combinatorial Optimization. pp. 123–135. Springer (2017)
9. Chan, T.M.: More algorithms for all-pairs shortest paths in weighted graphs. In: Proceedings of the Thirty-Ninth Annual ACM Symposium on Theory of Computing. p. 590–598. Association for Computing Machinery (2007)
10. Charikar, M., Guha, S., Tardos, É., Shmoys, D.B.: A constant-factor approximation algorithm for the k-median problem. Journal of Computer and System Sciences **65**(1), 129–149 (2002)
11. Charikar, M., Li, S.: A dependent lp-rounding approach for the k-median problem. In: International Colloquium on Automata, Languages, and Programming. pp. 194–205. Springer (2012)
12. Chaudhuri, S., Garg, N., Ravi, R.: The p-neighbor k-center problem. Information Processing Letters **65**(3), 131–134 (1998)
13. Chen, D., Chen, R.: New relaxation-based algorithms for the optimal solution of the continuous and discrete p-center problems. Computers & Operations Research **36**(5), 1646–1655 (2009)
14. Chrobak, M., Kenyon, C., Young, N.: The reverse greedy algorithm for the metric k-median problem. Information Processing Letters **97**(2), 68–72 (2006)
15. Cohen, L., Uras, T., Jahangiri, S., Arunasalam, A., Koenig, S., Kumar, T.K.S.: The FastMap algorithm for shortest path computations. In: Proceedings of the 27th International Joint Conference on Artificial Intelligence (2018)
16. Contardo, C., Iori, M., Kramer, R.: A scalable exact algorithm for the vertex p-center problem. Computers & Operations Research **103**, 211–220 (2019)
17. Cornuéjols, G., Nemhauser, G., Wolsey, L.: The uncapacitated facility location problem. Tech. rep., Cornell University Operations Research and Industrial Engineering (1983)
18. Daskin, M.S.: A new approach to solving the vertex p-center problem to optimality: Algorithm and computational results. Communications of the Operations Research Society of Japan **45**(9), 428–436 (2000)
19. Davidović, T., Ramljak, D., Šelmić, M., Teodorović, D.: Bee colony optimization for the p-center problem. Computers & Operations Research **38**(10), 1367–1376 (2011)
20. Dohan, D., Karp, S., Matejek, B.: K-median algorithms: Theory in practice. Tech. rep., Princeton University Computer Science (2015)
21. Dyer, M.E., Frieze, A.M.: A simple heuristic for the p-centre problem. Operations Research Letters **3**(6), 285–288 (1985)
22. Elloumi, S., Labbé, M., Pochet, Y.: A new formulation and resolution method for the p-center problem. INFORMS Journal on Computing **16**(1), 84–94 (2004)

23. Faloutsos, C., Lin, K.I.: FastMap: A fast algorithm for indexing, data-mining and visualization of traditional and multimedia datasets. In: Proceedings of the 1995 ACM SIGMOD International Conference on Management of Data (1995)
24. Farahani, R.Z., Hekmatfar, M.: Facility Location: Concepts, Models, Algorithms and Case Studies. Springer Science & Business Media (2009)
25. Floyd, R.W.: Algorithm 97: shortest path. Communications of the ACM **5**(6), 345 (1962)
26. Fredman, M.: New bounds on the complexity of the shortest path problem. SIAM J. Comput. **5**, 83–89 (1976)
27. Fredman, M.L., Tarjan, R.E.: Fibonacci heaps and their uses in improved network optimization algorithms. J. ACM **34**(3), 596–615 (1987)
28. Galil, Z., Margalit, O.: All pairs shortest paths for graphs with small integer length edges. Journal of Computer and System Sciences **54**(2), 243–254 (1997)
29. Galil, Z., Margalit, O.: Witnesses for boolean matrix multiplication and for transitive closure. Journal of Complexity **9**(2), 201–221 (1993)
30. Garcia-Diaz, J., Sanchez-Hernandez, J., Menchaca-Mendez, R., Menchaca-Mendez, R.: When a worse approximation factor gives better performance: a 3-approximation algorithm for the vertex k-center problem. Journal of Heuristics **23**(5), 349–366 (2017)
31. Gonzalez, T.F.: Clustering to minimize the maximum intercluster distance. Theoretical computer science **38**, 293–306 (1985)
32. Gopalakrishnan, S., Cohen, L., Koenig, S., Kumar, T.K.S.: Embedding directed graphs in potential fields using FastMap-D. In: Proceedings of the 13th International Symposium on Combinatorial Search (2020)
33. Guo-Hui, L., Xue, G.: k-center and k-median problems in graded distances. Theoretical computer science **207**(1), 181–192 (1998)
34. Hagerup, T.: Improved shortest paths on the word ram. In: International Colloquium on Automata, Languages, and Programming. pp. 61–72. Springer (2000)
35. Han, Y., Takaoka, T.: An $o(n^3 \log\log n/\log^2 n)$ time algorithm for all pairs shortest paths. Journal of Discrete Algorithms **38-41**, 9–19 (2016)
36. Hochbaum, D.S., Shmoys, D.B.: A best possible heuristic for the k-center problem. Mathematics of operations research **10**(2), 180–184 (1985)
37. Irawan, C.A., Salhi, S., Drezner, Z.: Hybrid meta-heuristics with vns and exact methods: application to large unconditional and conditional vertex p-centre problems. Journal of Heuristics **22**(4), 507–537 (2016)
38. Jain, K., Mahdian, M., Markakis, E., Saberi, A., Vazirani, V.V.: Greedy facility location algorithms analyzed using dual fitting with factor-revealing lp. Journal of the ACM (JACM) **50**(6), 795–824 (2003)
39. Jain, K., Vazirani, V.V.: Approximation algorithms for metric facility location and k-median problems using the primal-dual schema and lagrangian relaxation. Journal of the ACM (JACM) **48**(2), 274–296 (2001)
40. Johnson, D.B.: Efficient algorithms for shortest paths in sparse networks. Journal of the ACM (JACM) **24**(1), 1–13 (1977)
41. Kaveh, A., Nasr, H.: Solving the conditional and unconditional p-center problem with modified harmony search: A real case study. Scientia Iranica **18**(4), 867–877 (2011)
42. Khuller, S., Pless, R., Sussmann, Y.J.: Fault tolerant k-center problems. Theoretical Computer Science **242**(1-2), 237–245 (2000)
43. Khuller, S., Sussmann, Y.J.: The capacitated k-center problem. SIAM Journal on Discrete Mathematics **13**(3), 403–418 (2000)
44. Könemann, J., Li, Y., Parekh, O., Sinha, A.: An approximation algorithm for the edge-dilation k-center problem. Operations Research Letters **32**(5), 491–495 (2004)

45. Li, A., Stuckey, P., Koenig, S., Kumar, T.K.S.: A FastMap-based algorithm for block modeling. In: Proceedings of the International Conference on the Integration of Constraint Programming, Artificial Intelligence, and Operations Research (2022)
46. Li, J., Felner, A., Koenig, S., Kumar, T.K.S.: Using FastMap to solve graph problems in a Euclidean space. In: Proceedings of the International Conference on Automated Planning and Scheduling (2019)
47. Li, S., Svensson, O.: Approximating k-median via pseudo-approximation. SIAM Journal on Computing **45**(2), 530–547 (2016)
48. Mladenović, N., Labbé, M., Hansen, P.: Solving the p-center problem with tabu search and variable neighborhood search. Networks: An International Journal **42**(1), 48–64 (2003)
49. Özsoy, F.A., Pınar, M.Ç.: An exact algorithm for the capacitated vertex p-center problem. Computers & Operations Research **33**(5), 1420–1436 (2006)
50. Pacheco, J.A., Casado, S.: Solving two location models with few facilities by using a hybrid heuristic: a real health resources case. Computers & operations research **32**(12), 3075–3091 (2005)
51. Perozzi, B., Al-Rfou, R., Skiena, S.: Deepwalk: Online learning of social representations. In: Proceedings of the 20th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (2014)
52. Plesník, J.: A heuristic for the p-center problems in graphs. Discrete Applied Mathematics **17**(3), 263–268 (1987)
53. Pullan, W.: A memetic genetic algorithm for the vertex p-center problem. Evolutionary computation **16**(3), 417–436 (2008)
54. Rana, R., Garg, D.: The analytical study of k-center problem solving techniques. International journal of information technology and knowledge management **1**(2), 527–535 (2008)
55. Rdusseeun, L., Kaufman, P.: Clustering by means of medoids. In: Proceedings of the Statistical Data Analysis Based on the L1 Norm Conference, Neuchatel, Switzerland. pp. 405–416 (1987)
56. Robič, B., Mihelič, J.: Solving the k-center problem efficiently with a dominating set algorithm. Journal of computing and information technology **13**(3), 225–234 (2005)
57. Schubert, E., Rousseeuw, P.J.: Fast and eager k-medoids clustering: O(k) runtime improvement of the pam, clara, and clarans algorithms. Information Systems **101**, 101804 (2021)
58. Seidel, R.: On the all-pairs-shortest-path problem in unweighted undirected graphs. Journal of computer and system sciences **51**(3), 400–403 (1995)
59. Shmoys, D.B.: Computing near-optimal solutions to combinatorial optimization problems. Combinatorial Optimization **20**, 355–397 (1995)
60. Takaoka, T.: A new upper bound on the complexity of the all pairs shortest path problem. Information Processing Letters **43**(4), 195–199 (1992)
61. Thorup, M.: Undirected single-source shortest paths with positive integer weights in linear time. Journal of the ACM (JACM) **46**(3), 362–394 (1999)
62. Thorup, M.: Floats, integers, and single source shortest paths. Journal of algorithms **35**(2), 189–201 (2000)
63. Whitaker, R.: A fast algorithm for the greedy interchange for large-scale clustering and median location problems. INFOR: Information Systems and Operational Research **21**(2), 95–108 (1983)