DYNAMIC LOCAL SEARCH UNDER DISTANCE UPDATES FOR (k, z)-CLUSTERING

Anonymous authors
Paper under double-blind review

ABSTRACT

Machine learning systems often cluster data that keep changing, such as embeddings that drift during training or networks that keep gaining connections. We introduce a model of dynamic metric spaces in which the point set is fixed and distances only decrease, as happens for example when edges are inserted into a graph and shortest paths shrink. In this model we study a local search approach for (k, z)-clustering, a generalization of k-median and k-means objectives. The main computational cost of local search is finding an improving swap, a pair of one center and one non-center whose exchange lowers the value of the objective function. We give a deterministic dynamic algorithm that maintains the best improving swap in time $O(n \log n)$ per distance update, instead of $O(n^2)$ from scratch. Building on this data structure, we design an algorithm that maintains a constant factor approximate (k, z)-clustering with total update time $\tilde{O}(kn^2)$, matching (up to lower order terms) one single run of the fastest deterministic static local-search based approximation algorithm. On the metric space induced by two real evolving graphs with up to 55,000 vertices, our algorithm matches the quality of full recomputation to within 0.25 percent, while running far faster: it handles 1,000 edge insertions in about 10 s, compared to the roughly 5,100 s of a warm start approach which uses the previously computed solution, and to over a week through a naive full recomputation from scratch.

1 INTRODUCTION

Clustering is one of the most widely used tools in machine learning. It appears in representation learning, in the compression of large sets of vectors, and in data summarization before applying other methods (Gray & Neuhoff, 1998; Jain, 2010). In real systems, the data being clustered is rarely static: embeddings change as a model continues training, feature spaces are updated as models are retrained or fine-tuned, and graphs of users, documents, and products continuously gain new edges over time. Hence, a clustering that was good for yesterday's data can be a poor fit for today's, and running the algorithm from scratch after every small change is too slow at the scale of modern ML systems. This is one of the reasons continual learning is hard: a model has to keep using new information without starting over each time (Kirkpatrick et al., 2017); lifelong learning runs into the same problem on a larger scale (Parisi et al., 2019). The same need to stay up to date without recomputing everything from scratch motivates the long line of work on streaming and incremental clustering, where the solution quality is traded for efficiency (Guha et al., 2003; Charikar et al., 2004; Grilnberger & Skarlatos, 2026a; Blank et al., 2026). The natural question is then: *can we maintain a good clustering as the data evolve, updating only what needs to change instead of recomputing everything, without giving up much quality?*

Most existing dynamic clustering algorithms answer this question in a different model, where an adversary inserts or deletes points from a metric space (Fichtenberger et al., 2021; Bateni et al., 2023; Bhattacharya et al., 2024a;b; Forster & Skarlatos, 2025; Grilnberger & Skarlatos, 2026b). This model captures cases where the population being clustered grows or shrinks, but misses another case that often arises in machine learning: the set of items remains the same, while the distances between them change. One example is clustering the nodes of a graph that keeps gaining edges, as in a citation or social network. The nodes are known in advance, but shortest-path distances between them can only decrease as more edges are inserted (Cruciani et al., 2024; 2026), a setting

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

that has drawn considerable attention in graph representation learning (Kazemi et al., 2020). Another example is when clustering a fixed set of items inside a representation space that is still being trained or refined, so that some pairs of items move closer together as training continues, as is the case for methods that learn embeddings which keep changing over time (Kumar et al., 2019). To the best of our knowledge, the model where the point set is fixed and distances are updated has not been studied before.

We study this dynamic point-set model for arbitrary metrics in which the points are fixed and the distances between them are subject to updates, such that the triangle inequality (possibly relaxed by a factor ρ) holds after every update, where an update may consist of a batch of distance changes. This novel model is motivated by maintaining dynamic (k, z)-clustering on graphs under edge updates; we believe this model is of independent interest, whenever the relationships within a fixed set of items evolve over time. Note that in principle our distance-decrease model can be "simulated" by the point-update model: process each distance decrease between two points by first deleting one of the affected points and then re-inserting it (and reporting the new distance when queried in the future). However, in the most interesting parameter regime, this simulation comes at the cost of randomization as no deterministic algorithm for k-median (with sublinear k) can have both sublinear update time and constant-factor approximation guarantee (Costa & Farokhnejad, 2025). By developing algorithms directly tailored to the distance-decrease model such a loss of determinism can be avoided. In many scenarios for algorithmic decision making determinism is highly desirable to ensure reproducibility (Kroll, 2015; Pineau et al., 2021).

1.1 Terminology and Problem Statement

(k, z)-clustering. Let P be a set of $n := |P|$ points in a metric space $(\mathcal{X}, \mathrm{d})$. For any point $p \in P$ and any subset of points $C \subseteq P$, the *distance* between p and C is denoted by $\mathrm{d}(p, C) := \min_{c \in C} \mathrm{d}(p, c)$. Given an integer $k \geq 1$ and a constant $z \geq 1$, the goal of the (k, z)-*clustering problem* is to output a subset $C \subseteq P$ of at most k points, referred to as *centers*, that minimizes the cost $\mathrm{cost}^z(C) := \sum_{p \in P} \mathrm{d}(p, C)^z$, i.e., the sum of the z-th powers of the distances from each point to its closest center. Any subset $C \subseteq P$ with $|C| \leq k$ is referred to as a *solution*; the optimal cost is denoted by $\mathrm{OPT} := \min_{C \subseteq P : |C| \leq k} \mathrm{cost}^z(C)$, and for $\gamma \geq 1$, a solution C is γ-*approximate* if $\mathrm{cost}^z(C) \leq \gamma \cdot \mathrm{OPT}$. Setting $z = 1$ gives k-median and $z = 2$ gives k-means, the two clustering objectives used most often in machine learning. Studying (k, z)-clustering lets us treat both problems in a unified framework instead of building separate machinery for each.

Our results for point sets hold in the following more general setting, in which points are weighted and the triangle inequality may be violated by a factor ρ; this generality is needed for our application to graphs.

Definition 1.1 (ρ-metric space)**.** *A ρ-metric space $(\mathcal{X}, \mathrm{d})$ with $\rho \geq 1$ is a metric space in which the triangle inequality is replaced by the relaxed triangle inequality:*

$$\forall x, y, u \in \mathcal{X} : \ \mathrm{d}(x, y) \ \leq \ \rho \cdot (\mathrm{d}(x, u) + \mathrm{d}(u, y)).$$

Definition 1.2 (weighted (k, z)-clustering problem)**.** *Given a weighted point set P with weight function $\mathrm{wt}(\cdot)$ in a ρ-metric space $(\mathcal{X}, \mathrm{d})$, we denote the corresponding weighted (k, z)-clustering instance by $\mathcal{K} := (P, k, z, \mathrm{wt}, \mathrm{d}, \rho)$. If $z = 1$, or $\rho = 1$, or $\mathrm{wt}(\cdot)$ is uniform, we omit the corresponding parameters for simplicity; we write $\rho = \infty$ when d is not required to satisfy any form of the triangle inequality. The cost of a subset of points $C \subseteq P$ is defined as $\mathrm{cost}^z_{\mathrm{wt}}(C) := \sum_{p \in P} \mathrm{wt}(p) \cdot \mathrm{d}(p, C)^z$. The optimal (k, z)-clustering cost of $\mathcal{K}$ is denoted by $\mathrm{OPT}_{\mathcal{K}}$.*[1]

Point sets under distance updates. We study a dynamic model in which the point set P is fixed, while an adversary updates the distance function d such that $(\mathcal{X}, \mathrm{d})$ remains a ρ-metric space after every update. The goal is to maintain an approximate (k, z)-clustering solution after every update; the efficiency of a dynamic algorithm is measured by its preprocessing time and by its update time, either in the worst case per update or in total over the whole sequence of updates. We focus on the *distance-decrease setting*, in which the adversary is allowed only to decrease the distances between points in P. In the *single-update distance-decrease setting*, every update decreases the distance

[1] We assume that all non-zero distances and point weights are at least 1, and that the point weights are bounded by a polynomial in n.

between exactly one pair of points, whereas in the *batch-update distance-decrease setting*, every update is a batch of distance decreases between possibly many pairs of points.

Incremental graphs. Our main application is the *incremental graph setting*: given a weighted undirected graph $G = (V, E, w)$ with n vertices and integral edge weights in $[1, W]$ that undergoes adversarial edge insertions, the goal is to maintain an approximate (k, z)-clustering solution $C \subseteq V$ with respect to the shortest-path distances $\mathrm{dist}(\cdot, \cdot)$ in the current graph. We denote by m the number of edges after all insertions. Since edge insertions can only decrease shortest-path distances, this is a special case of the batch-update distance-decrease setting, in which a single edge insertion may decrease the distances between many pairs of vertices.

Local search. Given a solution C, a *swap* $(x, y) \in C \times (P \setminus C)$ exchanges the center x with the non-center y, yielding the solution $C_{xy} \coloneqq (C \setminus \{x\}) \cup \{y\}$. Its *gain* is $\Delta_{C,\mathrm{d}}(x, y) \coloneqq \mathrm{cost}^z_{\mathrm{wt}}(C) - \mathrm{cost}^z_{\mathrm{wt}}(C_{xy})$, where both costs are computed with respect to the distance function d, and the swap is *improving* if its gain is positive. The *swap table* $\Delta_{C,\mathrm{d}}$ is the two-dimensional table of size $|C| \times |P \setminus C|$ whose entries are the gains $\Delta_{C,\mathrm{d}}(x, y)$ of all swaps, and we denote its maximum entry by $\max(\Delta_{C,\mathrm{d}})$; we omit the subscript d when it is clear from the context. Starting from any solution, *local search* repeatedly performs an improving swap until none exists, i.e., until $\max(\Delta_C) \leq 0$, in which case the solution is a *local optimum*.

1.2 Our Contributions

Our main result is a dynamic version of local search, the classic method used to solve (k, z)-clustering well in practice, since local optima are known to give good approximations to the best possible solution (Arya et al., 2004; Gupta & Tangwongsan, 2008; Cohen-Addad et al., 2022). The dominant cost of local search is finding an improving swap. We show how to maintain the best improving swap in time $O(n \log n)$ per adversarial distance update, instead of the $O(n^2)$ time needed by the best known static method to find it from scratch (Schubert & Rousseeuw, 2021).

Theorem 1.3. *There is a deterministic algorithm in the single-update distance-decrease setting that given:*

- *a weighted (k, z)-clustering instance $\mathcal{K} = (P, k, z, \mathrm{wt}, \mathrm{d}, \infty)$ subject to distance decreases of the form (p, q, dst) (i.e., the adversarial update decreases the distance between p and q by dst),*
- *an initial (k, z)-clustering solution C,*
- *and the initial corresponding two-dimensional swap table Δ_C,*

maintains the value $\max(\Delta_C)$ and the pair of points $(x, y) \in C \times (P \setminus C)$ attaining it (i.e., $\Delta_{C,\mathrm{d}}(x, y) = \max(\Delta_{C,\mathrm{d}})$) with worst-case update time $O(k \log k + \log n)$ when $C \cap \{p, q\} = \emptyset$ and $O(n \log n)$ otherwise; the preprocessing time is $O(nk)$.

Utilizing Theorem 1.3 in the distance-decrease setting, we develop a dynamic local search algorithm that maintains a constant-factor approximate solution with a total update time of $\tilde{O}(kn^2)$ over all adversarial distance decreases (see Section A).[2] This matches—up to polylogarithmic factors—the running time of a single run of the static deterministic local search algorithm, so the dynamic version does not pay much more overall than one static run.[3]

Our dynamic algorithm is deterministic which implies robustness against an adaptive adversary who has access to the output of the algorithm and can adapt the updates accordingly. While Theorem 1.3 does not require the triangle inequality to be preserved over the updates, we need a weak version of the triangle inequality for our next results, where the distance update stem from shortest-path approximations. Thus we work with ρ-metric spaces (see Definition 1.1), where the triangle inequality may be violated by a factor of ρ.

[2]The notation $\tilde{O}(\cdot)$ hides polylogarithmic factors in n and other parameters.

[3]The original arguments in the static setting did not account for weighted point sets in ρ-metric spaces, but extending them to this setting is a mere technicality.

Theorem 1.4. *There is a deterministic algorithm in the batch-update distance-decrease setting that given:*

- *a weighted (k, z)-clustering instance $\mathcal{K} = (P, k, z, \mathrm{wt}, \mathrm{d}, \rho)$ subject to batches of distance decreases,*
- *an integer $k \geq 1$, constants $z \geq 1$ and $\rho = 1 + \frac{1}{16z}$, and an upper bound M on the distance function $\mathrm{d}(\cdot, \cdot)$,*

maintains an $O(1)$-approximate (k, z)-clustering solution with a total update time of $O\big(kn^2(\log nM)^3\big)$ and a preprocessing time of $O(kn^2 \log nM)$.

The hidden dependence on z in the $O(1)$-approximation ratio is $z^{O(z)}$. By combining the algorithm by Cruciani et al. (2026), the incremental reduction algorithm in Theorem B.4, and our dynamic local search algorithm in Theorem 1.4, we also obtain the following minor improvement in the incremental graph setting.

Theorem 1.5. *There is a randomized incremental algorithm for the (k, z)-clustering problem that, given a weighted undirected graph $G = (V, E, w)$ subject to edge insertions, an integer $k \geq 1$, and a constant $z \geq 1$, maintains with high probability an $O(1)$-approximate (k, z)-clustering solution with a total update time of $\tilde{O}(km^{1+o(1)} + k^3)$.*

This theorem improves the incremental algorithm of Cruciani et al. (2026), which trades off approximation ratio for running time: for any constant $\lambda \geq 1$, it maintains a $O(\lambda^{6z})$-approximation with total update time $\tilde{O}(km^{1+o(1)} + k^{1+1/\lambda}m)$. By essentially replacing their spanner-based recomputation with our dynamic local search, we obtain an approximation ratio that does not depend on λ and an improved total update time whenever $k \leq \sqrt{m}$.

Experiments. We back this up with experiments on two real graphs that grow over time, with up to about 55,000 nodes and 3.5 million edges, drawn from a public collection of temporal networks (Benson et al., 2018). Compared to recomputing the clustering from scratch after every batch of edge insertions, our dynamic algorithm gives almost the same solution quality, while being at the same time much faster. The cost differs by at most about 0.25 percent on the sparser graph, and matches exactly on the denser one; on our largest and densest graph, our algorithm handles 1,000 edge insertions in about 10 seconds, against roughly 5,100 seconds for a baseline that keeps the same solution but rebuilds its data structures at every step, and well over a week for a full recomputation from scratch.

2 Related Work

Local search for (k, z)-clustering. Besides LP rounding, primal-dual, and greedy methods (Charikar et al., 1999; Jain & Vazirani, 2001; Ahmadian et al., 2020; Cohen-Addad et al., 2025), local search is among the simplest approaches with provable guarantees for k-median and k-means. Arya et al. (2004) showed that local optima under p simultaneous swaps are $(3 + 2/p)$-approximate for k-median; Gupta & Tangwongsan (2008) simplified the analysis and extended it to (k, z)-clustering, showing a $\Theta(z)$-approximation for the z-th root of the cost. Local search also yields approximation schemes in Euclidean, doubling, and minor-free metrics (Cohen-Addad et al., 2019b; Friggstad et al., 2019) and, with a refined potential, improves on the factor 3 for k-median (Cohen-Addad et al., 2022). In machine learning, a few local search steps after k-means++ seeding (Arthur & Vassilvitskii, 2007) improve its $O(\log k)$ guarantee to a constant (Lattanzi & Sohler, 2019; Choo et al., 2020; Beretta et al., 2023), and swap-based k-medoids methods descending from PAM (Kaufman & Rousseeuw, 1990) are sped up by sampling (Ng & Han, 2002; Tiwari et al., 2020; 2023) or by sharing work across them (Schubert & Rousseeuw, 2019; 2021). The latter find the best swap in $O(n^2)$ time, giving the $\tilde{O}(kn^2)$ deterministic bound that we match; new randomized variants are even faster (Bhattacharya et al., 2024b), down to almost-linear time on graph metrics (Jiang et al., 2025).

Dynamic clustering under point updates. Most dynamic clustering algorithms maintain a solution while points are inserted into or deleted from a fixed metric space. For k-center, fully dynamic

algorithms (Chan et al., 2018; Schmidt & Sohler, 2019; Goranci et al., 2019) now achieve near-optimal update time (Bateni et al., 2023; Grilnberger & Skarlatos, 2026b); a parallel line of work minimizes the recourse, i.e., the number of center changes per update (Lattanzi & Vassilvitskii, 2017; Fichtenberger et al., 2021; Lacki et al., 2024; Forster & Skarlatos, 2025). For facility location, k-median, and k-means, a line of dynamic algorithms and coresets (Cohen-Addad et al., 2019a; Henzinger & Kale, 2020; Bhattacharya et al., 2022; 2023; la Tour et al., 2024) has culminated in near-optimal update time and recourse for k-median (Bhattacharya et al., 2024b;a); notably, both build on randomized local search but over a metric that never changes. In our model, instead, the points are fixed and the metric changes; these algorithms are not known to extend to this setting. The closest settings to ours are kinetic clustering, which maintains k-center for points moving along known trajectories (Gao et al., 2003; Har-Peled, 2004; Friedler & Mount, 2010), and evolutionary clustering, which trades off snapshot quality against temporal smoothness as similarities evolve (Chakrabarti et al., 2006; Chi et al., 2007); neither handles arbitrary distance updates with bounds on the update time.

Clustering on dynamic graphs. On static sparse graphs, where computing all pairwise distances is too costly, approximating k-median and k-center is well studied (Thorup, 2005; Eppstein et al., 2020; Abboud et al., 2023; Jin et al., 2025). Under edge updates, Cruciani et al. (2024) maintain a $(2+\varepsilon)$-approximate k-center in decremental and fully dynamic graphs and a $(4+\varepsilon)$-approximate one in incremental graphs, while Cruciani et al. (2026) maintain an $O(1)$-approximation for (k,z)-clustering problems on incremental graphs.

3 Technical Overview

In this section, we provide a high-level overview of our technical contributions. The main technical component is our swap table algorithm of Theorem 1.3, described in Section A.1. This algorithm serves as a subroutine in our local search algorithm of Theorem 1.4, described in Section A.3. In turn, the local search algorithm is used in Section B to establish Theorem 1.5 in the graph setting.

3.1 From Static Local Search to Dynamic Local Search

In the (k,z)-clustering problem, local search algorithms typically follow an "improving-swap" scheme. Consider an arbitrary (k,z)-clustering solution C whose cost is denoted by $\mathrm{cost}(C)$. It is natural to check whether this cost can be reduced by swapping one of the centers with a non-center, namely to check if there is $x \in C$ and $y \in P \setminus C$ such that $\mathrm{cost}(C \setminus \{x\} \cup \{y\}) < \mathrm{cost}(C)$. If such an *improving swap* exists, then the local search algorithm performs one of these swaps and improves the cost of the maintained solution. The values of the swaps (i.e., the gain of swapping x with y) can be visualized as being organized in a swap table with an entry

$$\Delta_C(x,y) \coloneqq \mathrm{cost}(C) - \mathrm{cost}(C \setminus \{x\} \cup \{y\})$$

for every pair of points $(x,y) \in C \times (P \setminus C)$. Observe that by finding the maximum such entry, the local search algorithm can decide whether an improving swap exists. By repeating this process until no improving swap exists, the local search algorithm reaches a local optimum.

Although local optimality is a weaker notion than global optimality, it has been shown for the k-median clustering problem (and more generally for (k,z)-clustering) that any local optimum is an $O(1)$-approximation to the global optimum (Arya et al., 2004; Gupta & Tangwongsan, 2008). More precisely, the following is known: if the current solution is not a constant-factor approximation to the global optimum, then there must exist a swap reducing the clustering cost by an $\Omega(\frac{1}{k})$ fraction of the current cost. Conversely: if no such *significantly* improving swap exists, the current solution is a constant-factor approximation. Therefore, to obtain an $O(1)$-approximation to the (k,z)-clustering problem, it suffices to check in each iteration whether a swap exists that reduces the cost by a *significant* factor.

Running time. With distances and weights ranging from 1 to $M = |P|^{O(1)}$, the cost of any initial solution is at most $|P|M^{z+1}$ and the cost of the final—not significantly improvable—solution is at least 1 (for non-zero costs). Hence, the total number of significant swaps from any initial solution to the final solution is $O(k \log(|P|M)) = \tilde{O}(k)$. Combined with an algorithm that computes the best

swap in $O(|P|^2)$ time (Schubert & Rousseeuw, 2021), the *static* deterministic local search algorithm computes an $O(1)$-approximate (k, z)-clustering solution on (P, d) in $\tilde{O}(k|P|^2)$ time.

Adversarial Distance Decreases. Consider an adversarial update that decreases one or several distances between pairs of points. After this adversarial distance decrease, both the global optimum and the (old) local optimum obtained by a prior local search have potentially decreased in cost, but have certainly not increased. To obtain an updated (k, z)-clustering solution, we can simply restart the local search algorithm using the previous local optimum as the initial solution, yielding eventually a new local optimum (in which no significantly improving swap exists).

By the monotonicity of the cost function under distance decreases, we can view the local search as a contiguous process spanning the computation of the initial solution as well as the computation of all updated solutions. Moreover, since every swap performed by the local search is significantly improving, the number of swaps performed over all adversarial distance decreases can be bounded by $\tilde{O}(k)$ using the same argument as in the static setting.

While this interplay between the monotonicity of the local search and the monotonicity of the cost function under distance decreases settles the bound on the total number of swaps, the claimed total update time of $\tilde{O}(k|P|^2)$ is not immediate. The reason is that maintaining the swap table to detect improving swaps may be costly when no significant swap exists.[4] This requires more machinery that we explain in the remainder of this technical overview.

3.2 Dynamic Local Search Algorithm

Our dynamic local search algorithm maintains the swap table dynamically under adversarial distance updates, in order to detect when a significantly improving swap becomes possible. However, we remark that the maintenance of the two-dimensional swap table Δ_C cannot be achieved in a straightforward way by maintaining each entry explicitly. The reason is that a single distance update can change $\Theta(k|P|)$ entries in the swap table.

To this end, we provide some structural lemmas (in Section A.1.1) to show that an efficient implicit maintenance of the swap table Δ_C—maintaining its maximum entry—under adversarial distance updates is possible. The crucial observation is that the only expensive adversarial updates occur when the distance update involves a center and a non-center. In this scenario, our main structural Lemma 3.1 demonstrates that each of the $\Theta(|P|-k)$ columns of $\Delta_C(\cdot, \cdot)$ is updated by adding a fixed value to a constant number of "ranges of entries". This means that a segment tree data structure (with lazy propagation) (Bentley, 1977) can be maintained for each column, as it guarantees $O(\log k)$ time per range update. As a consequence, in this scenario the swap table Δ_C can be updated implicitly with an efficient update time of $O(|P| \log k)$.

With this data structure at hand, the main amortized argument in the running time analysis is as follows. In the distance-decrease setting, the total number of points that ever belong to the set of centers over the course of all adversarial distance updates is $\tilde{O}(k)$ because the total number of swaps is bounded by $\tilde{O}(k)$. This implies that there are at most $\tilde{O}(k|P|)$ significant distance decreases involving a center and a non-center.[5] As a result, all of the adversarial distance decreases can be performed efficiently in a total update time of $\tilde{O}(k|P|^2)$.

3.2.1 Dynamic Swap Table Algorithm

In this subsection, we explain in more detail the structural observations from Section A.1.1 that are utilized in our dynamic swap table algorithm. Let $p, q \in P$ be a pair of points whose distance decreases by the adversary. First of all, the rows and columns of Δ_C corresponding to p and q (if they are defined) are treated independently by the algorithm, and thus it remains to analyze the entries $\Delta_C(x, y)$ for a pair of points $(x, y) \in (C \setminus \{p, q\}) \times ((P \setminus C) \setminus \{p, q\})$. In the scenario where both p

[4] The most naive approach of recomputing the swap table in $\Theta(|P|^2)$ time after each adversarial distance update is problematic when no significantly improving swap exists.

[5] We only need to consider *significant* distance decreases, in which the affected pairwise distances decrease by a constant factor relative to their respective values after their last significant decreases.

and q are either non-centers or centers, the aforementioned entries $\Delta_C(x, y)$ remain unchanged after the adversarial distance decrease between p and q (see Observation A.5 and Observation A.6).

Hence, let us focus on our main structural Lemma 3.1, which characterizes how $\Delta_C(x, y)$ is updated for every pair of points $(x, y) \in (C \setminus \{p, q\}) \times ((P \setminus C) \setminus \{p, q\})$, after the adversarial distance decrease between p and q when (without loss of generality) $p \in C$ is a center and $q \in P \setminus C$ is a non-center. We use the superscript (old) for the state before the adversarial distance decrease and omit it for the state after the distance decrease; for instance, $\mathrm{d}^{(\text{old})}$ is the old distance function and $\Delta_C^{(\text{old})} \coloneqq \Delta_{C,\mathrm{d}^{(\text{old})}}$ is the old swap table.

Lemma 3.1. *Let $p, q \in P$ be a pair of points whose distance decreases (i.e., $\mathrm{d}(p, q) < \mathrm{d}^{(old)}(p, q)$), where either $p \in C$ or $q \in C$, but not both (i.e., $|C \cap \{p, q\}| = 1$). Without loss of generality, assume that $p \in C$ and $q \notin C$. Then for every pair of points $(x, y) \in (C \setminus \{p, q\}) \times ((P \setminus C) \setminus \{p, q\})$, the gain of swapping x with y is calculated as follows:*

$$\Delta_C(x, y) \;=\; \Delta_C^{(old)}(x, y) + \mathrm{wt}(q) \cdot \big(\mathrm{d}(q, C) - \mathrm{d}(q, C_{xy}) - \mathrm{d}^{(old)}(q, C) + \mathrm{d}^{(old)}(q, C_{xy})\big).$$

Additionally, when $x \neq \bar{c}$ where $\bar{c} \coloneqq \operatorname{argmin}_{c \in C} \mathrm{d}^{(old)}(q, c)$, then:

- *If $\mathrm{d}^{(old)}(q, \bar{c}) \leq \mathrm{d}(p, q)$ then $\Delta_C(x, y) = \Delta_C^{(old)}(x, y)$.*
- *Otherwise if $\mathrm{d}^{(old)}(q, \bar{c}) > \mathrm{d}(p, q)$ and $\mathrm{d}(p, q) \leq \mathrm{d}^{(old)}(q, y)$ then:*

$$\Delta_C(x, y) \;=\; \Delta_C^{(old)}(x, y) + \mathrm{wt}(q) \cdot \Big(\min\big(\mathrm{d}^{(old)}(q, \bar{c}), \mathrm{d}^{(old)}(q, y)\big) - \mathrm{d}^{(old)}(q, \bar{c})\Big).$$

- *Otherwise if $\mathrm{d}^{(old)}(q, \bar{c}) > \mathrm{d}(p, q)$ and $\mathrm{d}(p, q) > \mathrm{d}^{(old)}(q, y)$ then:*

$$\Delta_C(x, y) \;=\; \Delta_C^{(old)}(x, y) + \mathrm{wt}(q) \cdot \big(\mathrm{d}(q, p) - \mathrm{d}^{(old)}(q, \bar{c})\big).$$

Let $\bar{c}$ be the nearest center to q. The primary goal of Lemma 3.1 is to establish that a fixed entry $\Delta_C(x, y)$ does not depend on x (for $x \neq \bar{c}$) and that the column $\Delta_C(\cdot, y)$ depends only on p, q, y, and $\bar{c}$. Once this has been established, ranges of entries in a column of Δ_C change by the same value.

To describe how this primary goal is proven in Lemma 3.1, assume for simplicity that $x \neq \bar{c}$. Moreover, after swapping x with y, only the point q needs to be analyzed; the justification is provided in the proof of Lemma 3.1 (we omit the argument here for simplicity). Since $\bar{c}$ is the closest center to q, if the adversarial distance decrease does not move p closer to q than $\bar{c}$ (i.e., $\mathrm{dist}(q, \bar{c}) \leq \mathrm{dist}(q, p)$), then the gain $\Delta_C(x, y)$ remains unaffected. Thus, assume that the adversary moves p closer to q than $\bar{c}$ (i.e., $\mathrm{dist}(q, \bar{c}) > \mathrm{dist}(q, p)$). In this case, it only matters whether the point y is closer to q than p. The main message is therefore that the change in the entry $\Delta_C(x, y)$ does not depend on x. In turn, this means that only a constant number of ranges of entries need to be updated by a fixed value per column.

3.3 (k, z)-Clustering on Incremental Graphs

Cruciani et al. (2026) developed an incremental bicriteria approximation algorithm that maintains an $(O(1), O(\log^3 n \, \log_{1+\epsilon} nW))$-bicriteria approximate solution S to the (k, z)-clustering problem on a graph undergoing edge insertions; namely, a set of vertices S of size $O(k \, \log^3 n \, \log_{1+\epsilon} nW)$ whose clustering cost is within a constant factor of the optimum. The authors use their incremental reduction algorithm to convert the bicriteria approximate solution into a constant-factor approximate (k, z)-clustering solution after every adversarial edge insertion. To achieve this, they maintain a (k, z)-clustering solution C over the bicriteria approximate solution S, using weights on S determined by the corresponding bicriteria approximate assignment σ.

Their incremental reduction algorithm employs a dynamic spanner algorithm together with a static (k, z)-clustering algorithm. In this work, we employ a different approach to obtain the desired running time by using our dynamic local search algorithm for point sets, which is deterministic. Since the changes to the point set P occur only in $\tilde{O}(1)$ many batches, we can restrict the distance-decrease local search algorithm to distance decreases by simply restarting it after every batch of point insertions. Note that the fully dynamic point set model is too powerful for the incremental graph setting.

Another natural approach. A natural alternative would be the state-of-the-art fully dynamic point-set algorithm of (Bhattacharya et al., 2024a) (amortized update time $\tilde{O}(k)$), simulated via reinsertion of both endpoints of each distance decrease, giving us $\tilde{O}(km^{1+o(1)} + k^3)$ total update time. However, this algorithm is not known to work for ρ-metric spaces with $\rho > 1$, which is required by our use of *approximate* shortest paths. Doing so would likely require significant modifications to both the algorithm and its analysis (Costa, 2025).

4 EXPERIMENTS

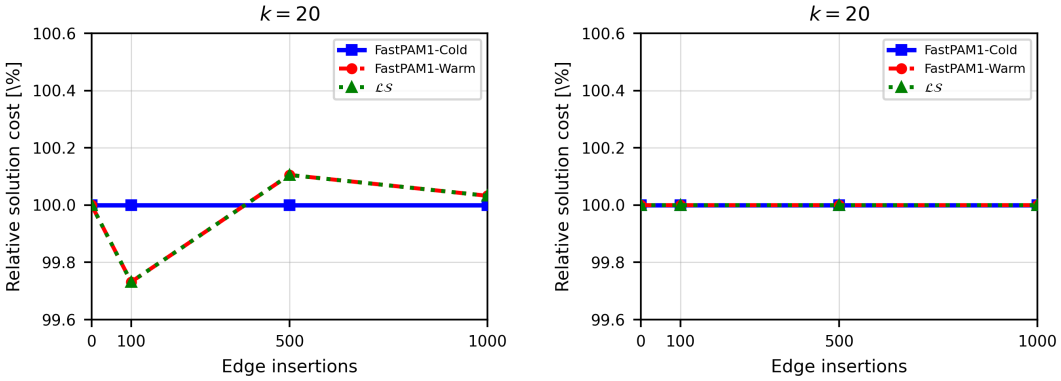
We complement our theoretical results with an experimental evaluation of our distance-decrease local search algorithm ($\mathcal{LS}$) against static local search on the k-median objective (i.e., $z = 1$), using an implementation of FastPAM1 (Schubert & Rousseeuw, 2021) with random k initialization. We realize distance decreases through edge insertions into two real, temporally evolving graphs, DBLP (co-authorship) and Stack Overflow (tag co-occurrence), both drawn from a public collection of temporal networks (Benson et al., 2018), with snapshots of up to 55,067 vertices and 3.5 million edges (Table 2). All competitors operate on an exact all-pairs shortest-path distance matrix that we maintain separately; we exclude the cost of maintaining this matrix from the reported running times, so as to isolate the cost of the local search subroutine itself. We compare three strategies: *FastPAM1-Cold*, which recomputes a locally optimal solution from scratch after every edge insertion; *FastPAM1-Warm*, a standard warm-start baseline (Yildirim & Wright, 2002) that reuses the previous center set but rebuilds the swap table from scratch; and our algorithm $\mathcal{LS}$, which updates the swap table incrementally. All experiments ran on a server with two AMD EPYC 7763 processors, with a 2.5 hour timeout per run; since running FastPAM1-Cold after every single update is infeasible, we execute it only at checkpoints after 100, 500, and 1,000 insertions and estimate its per-insertion cost from these runs. The complete set of experiments is reported in Sections C and D.

Table 1: Sizes of the snapshots used in Figures 1 and 2.

Dataset	# nodes n	# edges m	density $d = \frac{m}{n}$
DBLP 1983	16,134	33,413	2.07
DBLP 1990	55,067	122,282	2.22
Stack Overflow 40%	35,397	2,179,797	61.58

Solution quality. Across all $k \in \{10, 20, 30, 50, 100\}$ we test, $\mathcal{LS}$ matches the quality of full recomputation closely: on DBLP the cost differs by at most about 0.25 percent (Figure 1, left), and on the denser Stack Overflow graphs it matches exactly, since none of the 8000 edge insertions we process there ever trigger an improving swap (Figure 1, right). This reflects a broader pattern: across all our DBLP runs, about 98.4 percent of insertions do not trigger any improving swap, so the local optimum found early on stays stable under further updates.

Running time. $\mathcal{LS}$ is far faster than both baselines once the graph is large or dense. On our largest DBLP snapshot (1990), FastPAM1-Cold and FastPAM1-Warm both exceed the 2.5 hour timeout for $k \geq 50$ once 1000 insertions accumulate, while $\mathcal{LS}$ still finishes for $k \leq 30$ (Figure 2, left). On Stack Overflow 40%, the densest graph we test, the gap is the largest: after 1,000 insertions, $\mathcal{LS}$ takes about 1.2 to 10 seconds across all k, against roughly 5,100 to 5,200 seconds for FastPAM1-Warm and well over a week (estimated) for FastPAM1-Cold (Figure 2, right). The running time of $\mathcal{LS}$ grows with k, since it maintains center-dependent data structures, whereas FastPAM1-Warm's cost is dominated by rebuilding the swap table from scratch at every step and is largely insensitive to k; which strategy wins depends on how much of the shortest-path metric a typical insertion actually disturbs, not on the raw number of distance decreases alone. This gap does not simply track the number of distance decreases, as the update cost of $\mathcal{LS}$ depends on how many swap-table entries each update decreases, which grows when the changed pair involves a current center. Relative to the number of vertex pairs, insertions into DBLP 1990 disturb a substantially portion of the shortest-path metric than insertions into DBLP 1977, about 15 times fewer decreases per insertion, which is why the advantage of $\mathcal{LS}$ over FastPAM1-Warm persists to larger k there. Sparser, smaller snapshots and shorter update sequences show a smaller gap, or even favor FastPAM1-Warm at large k. On

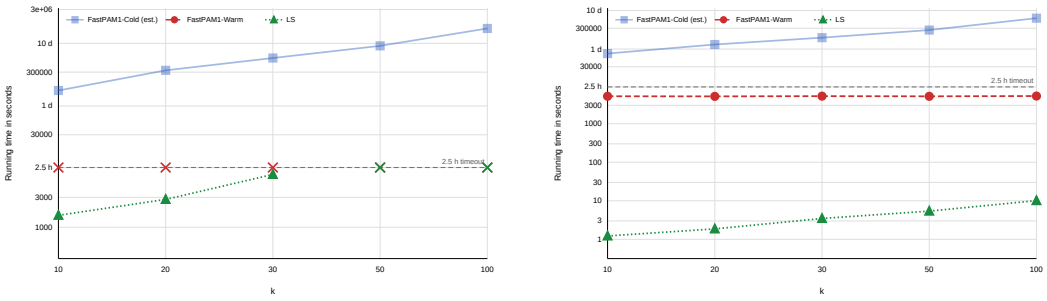


(a) DBLP 1983, $k = 20$

(b) Stack Overflow 40%, $k = 10$

Figure 1: Solution cost relative to full recomputation (FastPAM1-Cold, normalized to 100%) as edges are inserted. FastPAM1-Warm and $\mathcal{LS}$ always coincide here, since both start from the same center set and reach the same local optimum.

DBLP 1983, for example, $\mathcal{LS}$ is only faster up to around $k = 20$; beyond that, the per-step cost of FastPAM1-Warm, which does not grow with k, overtakes it.



(a) DBLP 1990, 1,000 insertions

(b) Stack Overflow 40%, 1,000 insertions

Figure 2: Running time (log scale) vs. k after 1000 edge insertions. FastPAM1-Cold is estimated from checkpoint runs; the dashed line marks the 2.5 hour timeout.

5 Conclusion

We introduced a model of dynamic metric spaces in which the point set is fixed and only distances decrease, and gave a deterministic $O(1)$-approximation dynamic local search algorithm for (k, z)-clustering in this model, with total update time matching a single run of the fastest deterministic static algorithm. Experimentally, it closely matches the quality of full recomputation while being substantially faster on large or dense graphs. We believe that the combination of local search with monotone dynamic updates is interesting on its own, and that our technique is of independent interest beyond clustering: whenever updates can only decrease the cost of every solution, as distance decreases do in our setting, local search can be run as a single process across the whole update sequence, with the number of significantly improving steps bounded by the same potential argument as in the static setting, so that the main challenge is to detect improving steps efficiently as the input changes. It remains open whether the factor k in our total update time can be shaved off, i.e., whether an $O(1)$-approximate (k, z)-clustering can be maintained under distance decreases with total update time $\tilde{O}(n^2)$.

AI USE STATEMENT

Generative AI tools were used for coding assistance during the implementation of the experimental algorithms, for generating plots in the experimental section, and for polishing the writing. The theoretical contributions, including all mathematical claims and proofs, were developed without AI assistance, as well as the original drafting of the paper. All AI-assisted text, code and figures were reviewed by the authors and verified for correctness.

REPRODUCIBILITY STATEMENT

We provide an implementation of all three evaluated algorithms, FastPAM1-Cold, FastPAM1-Warm, and $\mathcal{LS}$ as part of the Graph-Based Benchmark Suite (GBBS) (Dhulipala et al., 2020). The artifact includes build instructions, execution commands, experimental parameters, and instructions for reconstructing the temporal graph instances from publicly available DBLP and Stack Overflow datasets. The experimental setup, including the graph snapshots, update sequence lengths, center counts, hardware configuration, and timing methodology is documented in the paper and the artifact README.

The artifact is available publicly at: `https://github.com/graph-benchmarks2/dynamic-local-search`

REFERENCES

Amir Abboud, Vincent Cohen-Addad, Euiwoong Lee, and Pasin Manurangsi. On the fine-grained complexity of approximating k-center in sparse graphs. In Telikepalli Kavitha and Kurt Mehlhorn (eds.), *2023 Symposium on Simplicity in Algorithms, SOSA 2023, Florence, Italy, January 23-25, 2023*, pp. 145–155. SIAM, 2023. doi: 10.1137/1.9781611977585.CH14. URL https://doi.org/10.1137/1.9781611977585.ch14.

Sara Ahmadian, Ashkan Norouzi-Fard, Ola Svensson, and Justin Ward. Better guarantees for k-means and euclidean k-median by primal-dual algorithms. *SIAM Journal on Computing*, 49(4): FOCS17–97–FOCS17–156, 2020. doi: 10.1137/18M1171321. URL https://doi.org/10.1137/18M1171321.

David Arthur and Sergei Vassilvitskii. k-means++: The advantages of careful seeding. In *Proceedings of the Eighteenth Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pp. 1027–1035. SIAM, 2007.

Vijay Arya, Naveen Garg, Rohit Khandekar, Adam Meyerson, Kamesh Munagala, and Vinayaka Pandit. Local search heuristics for k-median and facility location problems. *SIAM Journal on Computing*, 33(3):544–562, 2004. doi: 10.1137/S0097539702416402. URL https://doi.org/10.1137/S0097539702416402.

MohammadHossein Bateni, Hossein Esfandiari, Hendrik Fichtenberger, Monika Henzinger, Rajesh Jayaram, Vahab Mirrokni, and Andreas Wiese. Optimal fully dynamic k-center clustering for adaptive and oblivious adversaries. In Nikhil Bansal and Viswanath Nagarajan (eds.), *Proceedings of the 2023 ACM-SIAM Symposium on Discrete Algorithms, SODA 2023, Florence, Italy, January 22-25, 2023*, pp. 2677–2727. SIAM, 2023. doi: 10.1137/1.9781611977554.CH101. URL https://doi.org/10.1137/1.9781611977554.ch101.

Austin R. Benson, Rediet Abebe, Michael T. Schaub, Ali Jadbabaie, and Jon M. Kleinberg. Simplicial closure and higher-order link prediction. *Proc. Natl. Acad. Sci. USA*, 115(48):E11221–E11230, 2018. doi: 10.1073/PNAS.1800683115. URL https://doi.org/10.1073/pnas.1800683115.

Jon Louis Bentley. Solutions to klee's rectangle problems. 1977.

Lorenzo Beretta, Vincent Cohen-Addad, Silvio Lattanzi, and Nikos Parotsidis. Multi-swap k-means++. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 36, pp. 26069–26091, 2023.

Sayan Bhattacharya, Silvio Lattanzi, and Nikos Parotsidis. Efficient and stable fully dynamic facility location. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 35, pp. 23358–23370, 2022.

Sayan Bhattacharya, Martín Costa, Silvio Lattanzi, and Nikos Parotsidis. Fully dynamic k-clustering in $\tilde{O}(k)$ update time. In *Proceedings of NeurIPS*, 2023.

Sayan Bhattacharya, Martín Costa, and Ermiya Farokhnejad. Fully dynamic k-median with near-optimal update time and recourse. In *Proceedings of the Symposium on Theory of Computing, STOC 2025*, 2024a.

Sayan Bhattacharya, Martin Costa, Naveen Garg, Silvio Lattanzi, and Nikos Parotsidis. Fully Dynamic k-Clustering with Fast Update Time and Small Recourse. In *2024 IEEE 65th Annual Symposium on Foundations of Computer Science (FOCS)*, pp. 216–227, Los Alamitos, CA, USA, October 2024b. IEEE Computer Society. doi: 10.1109/FOCS61266.2024.00023. URL https://doi.ieeecomputersociety.org/10.1109/FOCS61266.2024.00023.

Lotte Blank, Sergio Cabello, Mohammad Taghi Hajiaghayi, Robert Krauthgamer, Sepideh Mahabadi, André Nusser, Jeff M. Phillips, and Jonas Sauer. The expiration streaming model: Diameter, k-center, counting, sampling, and friends. In Sayan Bhattacharya, Danupon Nanongkai, Michael Benedikt, and Gabriele Puppis (eds.), *53rd International Colloquium on Automata, Languages, and Programming, ICALP 2026, Royal Holloway, University of London, Egham, United Kingdom, July 7-10, 2026*, volume 374 of *LIPIcs*, pp. 37:1–37:24. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2026. doi: 10.4230/LIPICS.ICALP.2026.37. URL https://doi.org/10.4230/LIPIcs.ICALP.2026.37.

Deepayan Chakrabarti, Ravi Kumar, and Andrew Tomkins. Evolutionary clustering. In *Proceedings of the 12th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD)*, pp. 554–560. ACM, 2006. doi: 10.1145/1150402.1150467.

T.-H. Hubert Chan, Arnaud Guerqin, and Mauro Sozio. Fully dynamic k-center clustering. In *Proceedings of the 2018 World Wide Web Conference (WWW)*, pp. 579–587. ACM, 2018. doi: 10.1145/3178876.3186124.

Moses Charikar, Sudipto Guha, Éva Tardos, and David B Shmoys. A constant-factor approximation algorithm for the k-median problem. In *Proceedings of the thirty-first annual ACM symposium on Theory of computing*, pp. 1–10, 1999.

Moses Charikar, Chandra Chekuri, Tomás Feder, and Rajeev Motwani. Incremental clustering and dynamic information retrieval. *SIAM J. Comput.*, 33(6):1417–1440, 2004. doi: 10.1137/S0097539702418498. URL https://doi.org/10.1137/S0097539702418498.

Yun Chi, Xiaodan Song, Dengyong Zhou, Koji Hino, and Belle L. Tseng. Evolutionary spectral clustering by incorporating temporal smoothness. In *Proceedings of the 13th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD)*, pp. 153–162. ACM, 2007. doi: 10.1145/1281192.1281212.

Davin Choo, Christoph Grunau, Julian Portmann, and Václav Rozhoň. k-means++: Few more steps yield constant approximation. In *Proceedings of the 37th International Conference on Machine Learning (ICML)*, volume 119 of *Proceedings of Machine Learning Research*, pp. 1909–1917. PMLR, 2020.

Vincent Cohen-Addad, Niklas Hjuler, Nikos Parotsidis, David Saulpic, and Chris Schwiegelshohn. Fully dynamic consistent facility location. In Hanna M. Wallach, Hugo Larochelle, Alina Beygelzimer, Florence d'Alché-Buc, Emily B. Fox, and Roman Garnett (eds.), *Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS 2019, December 8-14, 2019, Vancouver, BC, Canada*, pp. 3250–3260, 2019a. URL https://proceedings.neurips.cc/paper/2019/hash/fface8385abbf94b4593a0ed53a0c70f-Abstract.html.

Vincent Cohen-Addad, Philip N. Klein, and Claire Mathieu. Local search yields approximation schemes for k-means and k-median in Euclidean and minor-free metrics. *SIAM Journal on Computing*, 48(2):644–667, 2019b. doi: 10.1137/17M112717X.

Vincent Cohen-Addad, Anupam Gupta, Lunjia Hu, Hoon Oh, and David Saulpic. An improved local search algorithm for k-median. In *Proceedings of the 2022 Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pp. 1556–1612. SIAM, 2022.

Vincent Cohen-Addad, Fabrizio Grandoni, Euiwoong Lee, Chris Schwiegelshohn, and Ola Svensson. A $(2+\varepsilon)$-approximation algorithm for metric k-median. In *Proceedings of the Symposium on Theory of Computing, STOC 2025*, 2025.

Martin Costa. Personal Communication, 2025.

Martín Costa and Ermiya Farokhnejad. Deterministic k-median clustering in near-optimal time. In Keren Censor-Hillel, Fabrizio Grandoni, Joël Ouaknine, and Gabriele Puppis (eds.), *52nd International Colloquium on Automata, Languages, and Programming, ICALP 2025, Aarhus, Denmark, July 8-11, 2025*, LIPIcs, pp. 62:1–62:20. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2025. doi: 10.4230/LIPICS.ICALP.2025.62. URL https://doi.org/10.4230/LIPIcs.ICALP.2025.62.

Emilio Cruciani, Sebastian Forster, Gramoz Goranci, Yasamin Nazari, and Antonis Skarlatos. Dynamic algorithms for k-center on graphs. In David P. Woodruff (ed.), *Proceedings of the 2024 ACM-SIAM Symposium on Discrete Algorithms, SODA 2024, Alexandria, VA, USA, January 7-10, 2024*, pp. 3441–3462. SIAM, 2024. doi: 10.1137/1.9781611977912.123. URL https://doi.org/10.1137/1.9781611977912.123.

Emilio Cruciani, Sebastian Forster, and Antonis Skarlatos. Incremental (k, z)-clustering on graphs. In Sayan Bhattacharya, Danupon Nanongkai, Michael Benedikt, and Gabriele Puppis (eds.), *53rd International Colloquium on Automata, Languages, and Programming, ICALP 2026, Royal Holloway, University of London, Egham, United Kingdom, July 7-10, 2026*, volume 374 of *LIPIcs*, pp. 70:1–70:24. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2026. doi: 10.4230/LIPICS.ICALP.2026.70. URL https://doi.org/10.4230/LIPIcs.ICALP.2026.70.

Laxman Dhulipala, Jessica Shi, Tom Tseng, Guy E. Blelloch, and Julian Shun. The graph based benchmark suite (GBBS). In Akhil Arora, Semih Salihoglu, and Nikolay Yakovets (eds.), *GRADES-NDA'20: Proceedings of the 3rd Joint International Workshop on Graph Data Management Experiences & Systems (GRADES) and Network Data Analytics (NDA), Portland, OR, USA, June 14, 2020*, pp. 11:1–11:8. ACM, 2020. doi: 10.1145/3398682.3399168.

David Eppstein, Sariel Har-Peled, and Anastasios Sidiropoulos. Approximate greedy clustering and distance selection for graph metrics. *J. Comput. Geom.*, 11(1):629–652, 2020. doi: 10.20382/JOCG.V11I1A25. URL https://doi.org/10.20382/jocg.v11i1a25.

Hendrik Fichtenberger, Silvio Lattanzi, Ashkan Norouzi-Fard, and Ola Svensson. Consistent k-clustering for general metrics. In Dániel Marx (ed.), *Proceedings of the 2021 ACM-SIAM Symposium on Discrete Algorithms, SODA 2021, Virtual Conference, January 10 - 13, 2021*, pp. 2660–2678. SIAM, 2021. doi: 10.1137/1.9781611976465.158. URL https://doi.org/10.1137/1.9781611976465.158.

Sebastian Forster and Antonis Skarlatos. Dynamic consistent k-center clustering with optimal recourse. In Yossi Azar and Debmalya Panigrahi (eds.), *Proceedings of the 2025 Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2025, New Orleans, LA, USA, January 12-15, 2025*, pp. 212–254. SIAM, 2025. doi: 10.1137/1.9781611978322.7. URL https://doi.org/10.1137/1.9781611978322.7.

Sorelle A. Friedler and David M. Mount. Approximation algorithm for the kinetic robust K-center problem. *Computational Geometry*, 43(6–7):572–586, 2010. doi: 10.1016/j.comgeo.2010.01.001.

Zachary Friggstad, Mohsen Rezapour, and Mohammad R. Salavatipour. Local search yields a PTAS for k-means in doubling metrics. *SIAM Journal on Computing*, 48(2):452–480, 2019. doi: 10.1137/17M1127181.

Jie Gao, Leonidas J. Guibas, John Hershberger, Li Zhang, and An Zhu. Discrete mobile centers. *Discrete & Computational Geometry*, 30(1):45–63, 2003. doi: 10.1007/s00454-003-2925-6.

Gramoz Goranci, Monika Henzinger, Dariusz Leniowski, Christian Schulz, and Alexander Svozil. Fully dynamic k-center clustering in doubling metrics. *arXiv preprint arXiv:1908.03948*, 2019.

Robert M. Gray and David L. Neuhoff. Quantization. *IEEE transactions on information theory*, 44 (6):2325–2383, 1998.

Mara Grilnberger and Antonis Skarlatos. Incremental consistent k-center clustering. In Mohit Singh and Tom Gur (eds.), *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques, APPROX 2026, Boston University, Boston, Massachusetts, USA, and RANDOM 2026, Boston University, Boston, Massachusetts, USA, August 19-21, 2026, August 19-21, 2026*, volume 392 of *LIPIcs*, pp. 29:1–29:19. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2026a. doi: 10.4230/LIPICS.APPROX/RANDOM.2026.29. URL https://doi.org/10.4230/LIPIcs.APPROX/RANDOM.2026.29.

Mara Grilnberger and Antonis Skarlatos. Adaptive fully dynamic k-center clustering with (near-)optimal worst-case guarantees, 2026b. URL https://arxiv.org/abs/2604.01726.

Sudipto Guha, Adam Meyerson, Nina Mishra, Rajeev Motwani, and Liadan O'Callaghan. Clustering data streams: Theory and practice. *IEEE Transactions on Knowledge and Data Engineering*, 15(3):515–528, 2003. doi: 10.1109/TKDE.2003.1198387.

Anupam Gupta and Kanat Tangwongsan. Simpler analyses of local search algorithms for facility location. *CoRR*, abs/0809.2554, 2008. URL http://arxiv.org/abs/0809.2554.

Sariel Har-Peled. Clustering motion. *Discrete & Computational Geometry*, 31(4):545–565, 2004. doi: 10.1007/s00454-004-2822-7.

Monika Henzinger and Sagar Kale. Fully-dynamic coresets. In *28th Annual European Symposium on Algorithms (ESA 2020)*, pp. 57–1. Schloss Dagstuhl–Leibniz-Zentrum für Informatik, 2020.

Anil K. Jain. Data clustering: 50 years beyond k-means. *Pattern Recognition Letters*, 31(8):651–666, 2010. ISSN 0167-8655. doi: https://doi.org/10.1016/j.patrec.2009.09.011. URL https://www.sciencedirect.com/science/article/pii/S0167865509002323. Award winning papers from the 19th International Conference on Pattern Recognition (ICPR).

Kamal Jain and Vijay V. Vazirani. Approximation algorithms for metric facility location and k-median problems using the primal-dual schema and lagrangian relaxation. *J. ACM*, 48(2):274–296, 2001. doi: 10.1145/375827.375845. URL https://doi.org/10.1145/375827.375845.

Shaofeng H.-C. Jiang, Yaonan Jin, Jianing Lou, and Pinyan Lu. Local search for clustering in almost-linear time. *CoRR*, abs/2504.03513, 2025. doi: 10.48550/ARXIV.2504.03513.

Ce Jin, Yael Kirkpatrick, Virginia Vassilevska Williams, and Nicole Wein. Beyond 2-approximation for k-center in graphs. In *Proceedings of the 2025 Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pp. 175–211. SIAM, 2025.

Leonard Kaufman and Peter J. Rousseeuw. *Finding Groups in Data: An Introduction to Cluster Analysis*. Wiley Series in Probability and Statistics. Wiley, 1990. doi: 10.1002/9780470316801.

Seyed Mehran Kazemi, Rishab Goel, Kshitij Jain, Ivan Kobyzev, Akshay Sethi, Peter Forsyth, and Pascal Poupart. Representation learning for dynamic graphs: A survey. *Journal of Machine Learning Research*, 21(70):1–73, 2020.

James Kirkpatrick, Razvan Pascanu, Neil Rabinowitz, Joel Veness, Guillaume Desjardins, Andrei A. Rusu, Kieran Milan, John Quan, Tiago Ramalho, Agnieszka Grabska-Barwinska, Demis Hassabis, Claudia Clopath, Dharshan Kumaran, and Raia Hadsell. Overcoming catastrophic forgetting in neural networks. *Proceedings of the National Academy of Sciences*, 114(13):3521–3526, 2017. doi: 10.1073/pnas.1611835114.

Joshua A. Kroll. *Accountable Algorithms*. PhD thesis, Princeton University, USA, 2015. URL https://arks.princeton.edu/ark:/88435/dsp014b29b837r.

Srijan Kumar, Xikun Zhang, and Jure Leskovec. Predicting dynamic embedding trajectory in temporal interaction networks. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD)*. ACM, 2019. doi: 10.1145/3292500.3330895.

Max Dupré la Tour, Monika Henzinger, and David Saulpic. Fully dynamic k-means coreset in near-optimal update time. In *32nd Annual European Symposium on Algorithms, ESA 2024, September 2-4, 2024, Royal Holloway, London, United Kingdom*, 2024.

Jakub Lacki, Bernhard Haeupler, Christoph Grunau, Rajesh Jayaram, and Václav Rozhoň. Fully dynamic consistent k-center clustering. In *Proceedings of the 2024 Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pp. 3463–3484. SIAM, 2024. doi: 10.1137/1.9781611977912.124.

Silvio Lattanzi and Christian Sohler. A better k-means++ algorithm via local search. In *International Conference on Machine Learning*, pp. 3662–3671. PMLR, 2019.

Silvio Lattanzi and Sergei Vassilvitskii. Consistent k-clustering. In *Proceedings of the 34th International Conference on Machine Learning (ICML)*, volume 70 of *Proceedings of Machine Learning Research*, pp. 1975–1984. PMLR, 2017.

Yang P. Liu. Incremental shortest paths in almost linear time via a modified interior point method. *CoRR*, abs/2506.19207, 2025. doi: 10.48550/ARXIV.2506.19207.

Raymond T. Ng and Jiawei Han. CLARANS: A method for clustering objects for spatial data mining. *IEEE Transactions on Knowledge and Data Engineering*, 14(5):1003–1016, 2002. doi: 10.1109/TKDE.2002.1033770.

German I. Parisi, Ronald Kemker, Jose L. Part, Christopher Kanan, and Stefan Wermter. Continual lifelong learning with neural networks: A review. *Neural Networks*, 113:54–71, 2019. doi: 10.1016/j.neunet.2019.01.012.

Joelle Pineau, Philippe Vincent-Lamarre, Koustuv Sinha, Vincent Larivière, Alina Beygelzimer, Florence d'Alché-Buc, Emily B. Fox, and Hugo Larochelle. Improving reproducibility in machine learning research(a report from the neurips 2019 reproducibility program). *J. Mach. Learn. Res.*, 22:164:1–164:20, 2021. URL https://jmlr.org/papers/v22/20-303.html.

Melanie Schmidt and Christian Sohler. Fully dynamic hierarchical diameter k-clustering and k-center. *arXiv preprint arXiv:1908.02645*, 2019.

Erich Schubert and Peter J. Rousseeuw. Faster k-medoids clustering: Improving the PAM, CLARA, and CLARANS algorithms. In *Similarity Search and Applications (SISAP)*, Lecture Notes in Computer Science, pp. 171–187. Springer, 2019. doi: 10.1007/978-3-030-32047-8_16.

Erich Schubert and Peter J. Rousseeuw. Fast and eager k-medoids clustering: O(k) runtime improvement of the pam, clara, and CLARANS algorithms. *Inf. Syst.*, 101:101804, 2021. doi: 10.1016/J.IS.2021.101804. URL https://doi.org/10.1016/j.is.2021.101804.

Mikkel Thorup. Quick k-median, k-center, and facility location for sparse graphs. *SIAM Journal on Computing*, 34(2):405–432, 2005.

Mo Tiwari, Martin J. Zhang, James Mayclin, Sebastian Thrun, Chris Piech, and Ilan Shomorony. BanditPAM: Almost linear time k-medoids clustering via multi-armed bandits. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 33, pp. 10211–10222, 2020.

Mo Tiwari, Ryan Kang, Donghyun Lee, Sebastian Thrun, Ilan Shomorony, and Martin J. Zhang. BanditPAM++: Faster k-medoids clustering. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 36, pp. 73371–73382, 2023.

E. Alper Yildirim and Stephen J. Wright. Warm-start strategies in interior-point methods for linear programming. *SIAM J. Optim.*, 12(3):782–810, 2002. doi: 10.1137/S1052623400369235. URL https://doi.org/10.1137/S1052623400369235.

APPENDIX

A A MODEL FOR POINT SETS WITH DISTANCE UPDATES

In this model, given a weighted k-median instance $(P, k, \mathrm{wt}, \mathrm{d}, \rho)$ (see Definition 1.2), the adversary is allowed to update the distances between points in P. After the adversarial updates, it is ensured that the relaxed triangle inequality for ρ-metric spaces (see Definition 1.1) continues to be satisfied. Hence, a dynamic algorithm for this model receives adversarial updates that change the metric d, and the goal is to maintain an approximate k-median solution. We present our results for the k-median clustering problem; however, they can be generalized to (k, z)-clustering.

In this section we focus on the *distance-decrease setting* of this model, where the adversary is allowed only to decrease the distances between points in P. We consider a *batch-update distance-decrease setting* where the adversary is allowed to decrease the distances between multiple pairs of points in P per update, and a *single-update distance-decrease setting* where the adversary is allowed to decrease the distance between exactly two points in P per update.

Distance-decrease local search and swap table algorithms. In order to solve the weighted k-median clustering problem in the batch-update distance-decrease setting, we develop a *distance-decrease local search algorithm* that maintains a k-median solution C under a batch of distance decreases. Observe that after some decreases in the distances between points, both the optimal k-median cost and the cost of the maintained k-median solution C can either decrease or remain the same. To maintain proximity between $\mathrm{cost}_{\mathrm{wt}}(C)$ and OPT, our distance-decrease local search algorithm utilizes our *distance-decrease swap table algorithm* to determine whether a swap between a center and a point should be performed.

The swap table algorithm (described in Section A.1) operates in the single-update distance-decrease setting, where each adversarial update is a single distance decrease. Once the swap table algorithm signals that a swap should be performed, the local search algorithm (described in Section A.3) executes the swap and reinitializes the swap table algorithm. Notably, since the distance decreases from the batch are forwarded to the swap table algorithm sequentially rather than simultaneously, its input k-median instance may temporarily violate the triangle inequality. In fact, the (relaxed) triangle inequality is not required by the swap table algorithm; it is only needed to prove that the swap procedure guarantees the approximation ratio of the local search algorithm.

To distinguish between the states before and after an adversarial update, we use (old) as a superscript for the state before the update, and we omit the (old) superscript for the state after the update. In the next paragraph, we describe the notion of the swap table maintained by the distance-decrease swap table algorithm.

Swap table. Consider a weighted k-median instance $\mathcal{K} \coloneqq (P, k, \mathrm{wt}, \mathrm{d}, \infty)$, where d is a semi-metric (i.e., d may not satisfy the triangle inequality) and a k-median solution $C \subseteq P$. For a fixed pair of points $(x, y) \in C \times (P \setminus C)$, we denote by $C_{xy} \coloneqq C \cup \{y\} \setminus \{x\}$ the set of centers obtained by swapping x with y in the set of centers C. The *gain of swapping x by y in C* is denoted by $\Delta_{C,\mathrm{d}}(x, y) \coloneqq \mathrm{cost}_{\mathrm{wt}}(C) - \mathrm{cost}_{\mathrm{wt}}(C_{xy})$. By setting $\Delta_{C,\mathrm{d}}(x, y, p) \coloneqq \mathrm{wt}(p) \cdot \big(\mathrm{d}(p, C) - \mathrm{d}(p, C_{xy})\big)$ for every point $p \in P$, it follows that:

$$\Delta_{C,\mathrm{d}}(x, y) = \sum_{p \in P} \mathrm{wt}(p) \cdot \big(\mathrm{d}(p, C) - \mathrm{d}(p, C_{xy})\big) = \sum_{p \in P} \Delta_{C,\mathrm{d}}(x, y, p).$$

The definitions before the distance decrease are analogous, with the (old) superscript added. For instance, the *old gain of swapping x by y in C* is computed as follows:

$$\Delta_{C,\mathrm{d}^{(\mathrm{old})}}(x, y) = \sum_{p \in P} \mathrm{wt}(p) \cdot \big(\mathrm{d}^{(\mathrm{old})}(p, C) - \mathrm{d}^{(\mathrm{old})}(p, C_{xy})\big) = \sum_{p \in P} \Delta_{C,\mathrm{d}^{(\mathrm{old})}}(x, y, p).$$

The difference between the gains of swapping x with y before and after the adversarial distance decrease, is evaluated as follows:

$$\begin{aligned}\Delta_{C,\mathrm{d}^{(\text{old})}}(x,y) - \Delta_{C,\mathrm{d}}(x,y) &= \sum_{p\in P}\big(\Delta_{C,\mathrm{d}^{(\text{old})}}(x,y,p) - \Delta_{C,\mathrm{d}}(x,y,p)\big)\\ &= \sum_{p\in P}\mathrm{wt}(p)\cdot\big(\,\mathrm{d}^{(\text{old})}(p,C) - \mathrm{d}^{(\text{old})}(p,C_{xy}) - \mathrm{d}(p,C) + \mathrm{d}(p,C_{xy})\big).\end{aligned}$$

Definition A.1 (swap table)**.** *Given a weighted k-median instance $\mathcal{K} = (P,k,\mathrm{wt},\mathrm{d},\rho)$ and a k-median solution C, the swap table $\Delta_{C,\mathrm{d}}$ is the two-dimensional table of size $|C|\times|P\setminus C|$ whose entries are the values $\Delta_{C,\mathrm{d}}(x,y)$ for every pair of points $(x,y)\in C\times(P\setminus C)$.*

We denote by $\max(\Delta_{C,\mathrm{d}}) \coloneqq \max_{(x,y)\in C\times(P\setminus C)}\Delta_{C,\mathrm{d}}(x,y)$ the maximum value of the swap table $\Delta_{C,\mathrm{d}}(\cdot,\cdot)$. The distance function d in the swap table $\Delta_{C,\mathrm{d}}$ is omitted as a subscript whenever it is clear from the context. When the distance function d is omitted, we use the notation $\Delta_C^{(\text{old})}(\cdot,\cdot)$ to refer to $\Delta_{C,\mathrm{d}^{(\text{old})}}(\cdot,\cdot)$.

A.1 Distance-Decrease Swap Table Algorithm

Our *distance-decrease swap table algorithm* is used internally in our distance-decrease local search algorithm. The results are presented for the k-median clustering problem; since the analysis does not rely on the triangle inequality, generalizing them to (k,z)-clustering is straightforward.

Theorem 1.3. *There is a deterministic algorithm in the single-update distance-decrease setting that given:*

- *a weighted (k,z)-clustering instance $\mathcal{K} = (P,k,z,\mathrm{wt},\mathrm{d},\infty)$ subject to distance decreases of the form (p,q,dst) (i.e., the adversarial update decreases the distance between p and q by dst),*
- *an initial (k,z)-clustering solution C,*
- *and the initial corresponding two-dimensional swap table Δ_C,*

maintains the value $\max(\Delta_C)$ and the pair of points $(x,y)\in C\times(P\setminus C)$ attaining it (i.e., $\Delta_{C,\mathrm{d}}(x,y) = \max(\Delta_{C,\mathrm{d}})$) with worst-case update time $O(k\log k + \log n)$ when $C\cap\{p,q\} = \emptyset$ and $O(n\log n)$ otherwise; the preprocessing time is $O(nk)$.

Observe that the distance-decrease swap table algorithm operates in the single-update distance-decrease setting, and thus it can handle only a single distance decrease between exactly one pair of points in P per adversarial update. By setting $\rho = \infty$ in $(P,k,\mathrm{wt},\mathrm{d},\infty)$, we indicate that the swap table algorithm can operate on a semi-metric space where the triangle inequality is not required to hold.

Maintaining the two-dimensional swap table Δ_C explicitly is inefficient for our purposes. To that end, we maintain the swap table $\Delta_C(\cdot,\cdot)$ *implicitly* using one *dynamic range data structure* per column.

Definition A.2 (Dynamic range data structure (Bentley, 1977))**.** *Given a one-dimensional array L of numbers, a dynamic range data structure $\mathcal{RNG}$ supports the following query and update operations:*

- *Query the maximum number in L and the index where it is attained.*
- *Add a number c to all values in L with indices in the range $[i,j]$. Such an update is represented as $([i,j],c)$.*

Both query and update operations run in $O(\log|L|)$ worst-case time, and the preprocessing phase runs in $O(|L|)$ time.

State of the swap table algorithm. Recall that a k-median solution C and the corresponding swap table Δ_C are given in the input. The distance-decrease swap table algorithm maintains the swap table $\Delta_C(\cdot,\cdot)$ *implicitly* as follows: For each j-th point $y_j \in P \setminus C$ (representing the j-th column in Δ_C), the algorithm maintains a dynamic range data structure $\mathcal{RNG}_{y_j}$, which in turn encodes the values $\Delta_C(\cdot, y_j)$.

In addition, the swap table algorithm maintains a max-heap which stores, for each j-th column y_j, the maximum value of $\Delta_C(\cdot, y_j)$ along with the corresponding row x that attains it.[6] The maximum value of the swap table Δ_C and the pair of points $(x, y) \in C \times (P \setminus C)$ attaining it (i.e., $\Delta_{C,\mathrm{d}}(x, y) = \max(\Delta_{C,\mathrm{d}})$) are stored at the root of the max-heap $\mathcal{H}$. Further, the index ind_x of each center x in C is maintained.

Before we proceed with the distance-decrease swap table algorithm, we present in Section A.1.1 the necessary tools for its development. Then equipped with these tools, we describe in Section A.1.2 how the algorithm maintains the swap table after an adversarial distance decrease.

A.1.1 Tools Used in the Swap Table Algorithm

In this subsection, we demonstrate that a distance decrease affects the two-dimensional swap table Δ_C in a way that allows for efficient implicit maintenance of Δ_C. In particular, we provide the necessary tools (i.e., observations and lemmas) that help us update (implicitly) all entries $\Delta_C(\cdot,\cdot)$ in the desired worst-case update time. Recall that the entry $\Delta_C(x, y)$ represents the *gain* of swapping the center $x \in C$ with a point $y \in P \setminus C$ in the k-median solution C.

Observation A.3. *Let $p, q \in P$ be a pair of points whose distance decreases (i.e., $\mathrm{d}(p,q) < \mathrm{d}^{(old)}(p,q)$). Then for every point $a \in P \setminus \{p,q\}$, it holds that $\mathrm{d}(a,C) = \mathrm{d}^{(old)}(a,C)$ and $\mathrm{d}(a,C_{xy}) = \mathrm{d}^{(old)}(a,C_{xy})$.*

Proof. Since the point a is neither p nor q, its distances to any other point remain unaffected, and thus the claim follows. □

Lemma A.4. *Let $p, q \in P$ be a pair of points whose distance decreases (i.e., $\mathrm{d}(p,q) < \mathrm{d}^{(old)}(p,q)$). Then for every pair of points $(x, y) \in C \times (P \setminus C)$, the gain of swapping x with y is calculated as follows:*

$$\begin{aligned}\Delta_C(x,y) \;=\; \Delta_C^{(old)}(x,y) + \;& \mathrm{wt}(p) \cdot \big(\mathrm{d}(p,C) - \mathrm{d}(p,C_{xy}) - \mathrm{d}^{(old)}(p,C) + \mathrm{d}^{(old)}(p,C_{xy})\big) + \\ & \mathrm{wt}(q) \cdot \big(\mathrm{d}(q,C) - \mathrm{d}(q,C_{xy}) - \mathrm{d}^{(old)}(q,C) + \mathrm{d}^{(old)}(q,C_{xy})\big).\end{aligned}$$

Proof. Consider a pair of points $(x, y) \in C \times (P \setminus C)$. Based on Observation A.3, for every point $a \in P \setminus \{p,q\}$ we have $\mathrm{d}(a,C) = \mathrm{d}^{(\mathrm{old})}(a,C)$ and $\mathrm{d}(a,C_{xy}) = \mathrm{d}^{(\mathrm{old})}(a,C_{xy})$. In turn, for every point $a \in P \setminus \{p,q\}$ we obtain that:

$$\begin{aligned}\Delta_C^{(\mathrm{old})}(x,y,a) &= \mathrm{wt}(a) \cdot \big(\mathrm{d}^{(\mathrm{old})}(a,C) - \mathrm{d}^{(\mathrm{old})}(a,C_{xy})\big) \\ &= \mathrm{wt}(a) \cdot \big(\mathrm{d}(a,C) - \mathrm{d}(a,C_{xy})\big) \\ &= \Delta_C(x,y,a).\end{aligned}$$

The difference between the gains $\Delta_C(x,y)$ and $\Delta_C^{(\mathrm{old})}(x,y)$ is then evaluated as follows:

$$\begin{aligned}\Delta_C(x,y) - \Delta_C^{(\mathrm{old})}(x,y) \;&=\; \sum_{j \in P} \Delta_C(x,y,j) - \Delta_C^{(\mathrm{old})}(x,y,j) \\ &= \Delta_C(x,y,p) - \Delta_C^{(\mathrm{old})}(x,y,p) + \Delta_C(x,y,q) - \Delta_C^{(\mathrm{old})}(x,y,q) \\ &= \mathrm{wt}(p) \cdot \big(\mathrm{d}(p,C) - \mathrm{d}(p,C_{xy}) - \mathrm{d}^{(\mathrm{old})}(p,C) + \mathrm{d}^{(\mathrm{old})}(p,C_{xy})\big) + \\ &\quad \mathrm{wt}(q) \cdot \big(\mathrm{d}(q,C) - \mathrm{d}(q,C_{xy}) - \mathrm{d}^{(\mathrm{old})}(q,C) + \mathrm{d}^{(\mathrm{old})}(q,C_{xy})\big).\end{aligned}$$

□

[6] Each row x and each column y in $\Delta_C(x,y)$ are points such that $x \in C$ and $y \in P \setminus C$.

Note that Lemma A.4 describes how all entries are updated, but applying it to every entry explicitly requires $\Theta(nk)$ time. After a distance decrease between two points p and q, we apply Lemma A.4 only to the entries $\Delta_C(p,\cdot)$, $\Delta_C(\cdot,p)$, $\Delta_C(q,\cdot)$, and $\Delta_C(\cdot,q)$ (if they are defined). For the remaining entries $\Delta_C(x,y)$ where $x \in C \setminus \{p,q\}$ and $y \in (P \setminus C) \setminus \{p,q\}$, we show that these entries change smoothly and are efficiently manageable.

Specifically, Observation A.5 and Observation A.6 establish that whenever both p and q are either non-centers or centers, the aforementioned entries remain unchanged. The more challenging case arises when exactly one of p and q is a center. For this scenario, Lemma 3.1 provides (implicitly) an efficient way to update the aforementioned entries.

Observation A.5. *Let $p,q \in P \setminus C$ be a pair of points whose distance decreases (i.e., $\mathrm{d}(p,q) < \mathrm{d}^{(old)}(p,q)$), where neither p nor q is a center. Then for every pair of points $(x,y) \in (C \setminus \{p,q\}) \times ((P \setminus C) \setminus \{p,q\})$, it holds that $\Delta_C(x,y) = \Delta_C^{(old)}(x,y)$.*

Proof. Based on Lemma A.4, the difference $\Delta_C(x,y) - \Delta_C^{(\mathrm{old})}(x,y)$ equals:

$$\begin{aligned}\Delta_C(x,y) - \Delta_C^{(\mathrm{old})}(x,y) \;=\; & \mathrm{wt}(p) \cdot \big(\mathrm{d}(p,C) - \mathrm{d}(p,C_{xy}) - \mathrm{d}^{(\mathrm{old})}(p,C) + \mathrm{d}^{(\mathrm{old})}(p,C_{xy})\big) + \\ & \mathrm{wt}(q) \cdot \big(\mathrm{d}(q,C) - \mathrm{d}(q,C_{xy}) - \mathrm{d}^{(\mathrm{old})}(q,C) + \mathrm{d}^{(\mathrm{old})}(q,C_{xy})\big).\end{aligned}$$

The fact that $p \notin C$ and $q \notin C$ implies that each center $c \in C$ belongs to $C \setminus \{p,q\}$. Hence, it follows that $\mathrm{d}(p,C) = \mathrm{d}^{(\mathrm{old})}(p,C)$ and $\mathrm{d}(q,C) = \mathrm{d}^{(\mathrm{old})}(q,C)$. Furthermore, since $y \neq p$ and $y \neq q$, each $c \in C_{xy}$ belongs to $C_{xy} \setminus \{p,q\}$. Thus, it follows that $\mathrm{d}(p,C_{xy}) = \mathrm{d}^{(\mathrm{old})}(p,C_{xy})$ and $\mathrm{d}(q,C_{xy}) = \mathrm{d}^{(\mathrm{old})}(q,C_{xy})$. As a consequence, $\Delta_C(x,y)$ is equal to $\Delta_C^{(\mathrm{old})}(x,y)$. □

Observation A.6. *Let $p,q \in C$ be a pair of points whose distance decreases (i.e., $\mathrm{d}(p,q) < \mathrm{d}^{(old)}(p,q)$), where both p and q are centers. Then for every pair of points $(x,y) \in (C \setminus \{p,q\}) \times ((P \setminus C) \setminus \{p,q\})$, it holds that $\Delta_C(x,y) = \Delta_C^{(old)}(x,y)$.*

Proof. Based on Lemma A.4, the difference $\Delta_C(x,y) - \Delta_C^{(\mathrm{old})}(x,y)$ equals:

$$\begin{aligned}\Delta_C(x,y) - \Delta_C^{(\mathrm{old})}(x,y) \;=\; & \mathrm{wt}(p) \cdot \big(\mathrm{d}(p,C) - \mathrm{d}(p,C_{xy}) - \mathrm{d}^{(\mathrm{old})}(p,C) + \mathrm{d}^{(\mathrm{old})}(p,C_{xy})\big) + \\ & \mathrm{wt}(q) \cdot \big(\mathrm{d}(q,C) - \mathrm{d}(q,C_{xy}) - \mathrm{d}^{(\mathrm{old})}(q,C) + \mathrm{d}^{(\mathrm{old})}(q,C_{xy})\big).\end{aligned}$$

The fact that $p \in C$ and $q \in C$ implies that $\mathrm{d}(p,C) = \mathrm{d}^{(\mathrm{old})}(p,C) = 0$ and $\mathrm{d}(q,C) = \mathrm{d}^{(\mathrm{old})}(q,C) = 0$. Furthermore, since $x \neq p$ and $x \neq q$ then $p \in C_{xy}$ and $q \in C_{xy}$. Thus, it follows that $\mathrm{d}(p,C_{xy}) = \mathrm{d}^{(\mathrm{old})}(p,C_{xy}) = 0$ and $\mathrm{d}(q,C_{xy}) = \mathrm{d}^{(\mathrm{old})}(q,C_{xy}) = 0$. As a consequence, $\Delta_C(x,y)$ is equal to $\Delta_C^{(\mathrm{old})}(x,y)$. □

Lemma 3.1. *Let $p,q \in P$ be a pair of points whose distance decreases (i.e., $\mathrm{d}(p,q) < \mathrm{d}^{(old)}(p,q)$), where either $p \in C$ or $q \in C$, but not both (i.e., $|C \cap \{p,q\}| = 1$). Without loss of generality, assume that $p \in C$ and $q \notin C$. Then for every pair of points $(x,y) \in (C \setminus \{p,q\}) \times ((P \setminus C) \setminus \{p,q\})$, the gain of swapping x with y is calculated as follows:*

$$\Delta_C(x,y) \;=\; \Delta_C^{(old)}(x,y) + \mathrm{wt}(q) \cdot \big(\mathrm{d}(q,C) - \mathrm{d}(q,C_{xy}) - \mathrm{d}^{(old)}(q,C) + \mathrm{d}^{(old)}(q,C_{xy})\big).$$

Additionally, when $x \neq \bar{c}$ where $\bar{c} \coloneqq \mathrm{argmin}_{c \in C}\, \mathrm{d}^{(old)}(q,c)$, then:

- *If $\mathrm{d}^{(old)}(q,\bar{c}) \leq \mathrm{d}(p,q)$ then $\Delta_C(x,y) = \Delta_C^{(old)}(x,y)$.*
- *Otherwise if $\mathrm{d}^{(old)}(q,\bar{c}) > \mathrm{d}(p,q)$ and $\mathrm{d}(p,q) \leq \mathrm{d}^{(old)}(q,y)$ then:*

$$\Delta_C(x,y) \;=\; \Delta_C^{(old)}(x,y) + \mathrm{wt}(q) \cdot \Big(\min\big(\mathrm{d}^{(old)}(q,\bar{c}), \mathrm{d}^{(old)}(q,y)\big) - \mathrm{d}^{(old)}(q,\bar{c})\Big).$$

- *Otherwise if $\mathrm{d}^{(old)}(q,\bar{c}) > \mathrm{d}(p,q)$ and $\mathrm{d}(p,q) > \mathrm{d}^{(old)}(q,y)$ then:*

$$\Delta_C(x,y) \;=\; \Delta_C^{(old)}(x,y) + \mathrm{wt}(q) \cdot \big(\mathrm{d}(q,p) - \mathrm{d}^{(old)}(q,\bar{c})\big).$$

Proof. Consider a pair of points $(x, y) \in (C \setminus \{p, q\}) \times ((P \setminus C) \setminus \{p, q\})$. Without loss of generality, we assume that $p \in C$ and $q \notin C$. Based on Lemma A.4, the difference between the gains $\Delta_C(x, y)$ and $\Delta_C^{(\text{old})}(x, y)$ is evaluated as follows:

$$\begin{aligned}\Delta_C(x, y) - \Delta_C^{(\text{old})}(x, y) \;=\; & \operatorname{wt}(p) \cdot \big(\mathrm{d}(p, C) - \mathrm{d}(p, C_{xy}) - \mathrm{d}^{(\text{old})}(p, C) + \mathrm{d}^{(\text{old})}(p, C_{xy}) \big) + \\ & \operatorname{wt}(q) \cdot \big(\mathrm{d}(q, C) - \mathrm{d}(q, C_{xy}) - \mathrm{d}^{(\text{old})}(q, C) + \mathrm{d}^{(\text{old})}(q, C_{xy}) \big).\end{aligned}$$

Since $p \in C$ and $x \neq p$, the center p remains in the set of centers after swapping x with y, which means that $p \in C_{xy}$. This implies that $\mathrm{d}^{(\text{old})}(p, C) = \mathrm{d}(p, C) = 0$ and $\mathrm{d}^{(\text{old})}(p, C_{xy}) = \mathrm{d}(p, C_{xy}) = 0$, and thus the difference between the gains $\Delta_C(x, y)$ and $\Delta_C^{(\text{old})}(x, y)$ simplifies to:

$$\Delta_C(x, y) - \Delta_C^{(\text{old})}(x, y) \;=\; \operatorname{wt}(q) \cdot \big(\mathrm{d}(q, C) - \mathrm{d}(q, C_{xy}) - \mathrm{d}^{(\text{old})}(q, C) + \mathrm{d}^{(\text{old})}(q, C_{xy}) \big).$$

Hence, the claim holds in the case where $x = \operatorname{argmin}_{c \in C} \mathrm{d}^{(\text{old})}(q, c)$. It remains to consider the case where $x \neq \operatorname{argmin}_{c \in C} \mathrm{d}^{(\text{old})}(q, c)$. Let $\overline{c} \coloneqq \operatorname{argmin}_{c \in C} \mathrm{d}^{(\text{old})}(q, c)$, which means that $\mathrm{d}^{(\text{old})}(q, C) = \mathrm{d}^{(\text{old})}(q, \overline{c})$. Since $x \neq \overline{c}$, the center $\overline{c}$ remains in the set of centers after swapping x with y, which means that $\overline{c} \in C$ and $\overline{c} \in C_{xy}$. We separate the analysis based on whether $\mathrm{d}^{(\text{old})}(q, \overline{c})$ is less than $\mathrm{d}(p, q)$:

- Assume that $\mathrm{d}^{(\text{old})}(q, \overline{c}) \leq \mathrm{d}(p, q)$. In this case, it holds that $\mathrm{d}(q, C) = \mathrm{d}(q, \overline{c})$. Moreover, notice that $\overline{c} \neq p$; otherwise, we would have $\mathrm{d}(q, p) = \mathrm{d}(q, \overline{c}) < \mathrm{d}^{(\text{old})}(q, \overline{c})$, which contradicts the assumption. In turn, it follows that $\mathrm{d}(q, \overline{c}) = \mathrm{d}^{(\text{old})}(q, \overline{c})$, and thus $\mathrm{d}(q, C) = \mathrm{d}^{(\text{old})}(q, C)$. Regarding the difference between $\mathrm{d}^{(\text{old})}(q, C_{xy})$ and $\mathrm{d}(q, C_{xy})$, consider the two subcases depending on whether $\mathrm{d}^{(\text{old})}(q, \overline{c})$ is less than $\mathrm{d}^{(\text{old})}(q, y)$:
  - Assume that $\mathrm{d}^{(\text{old})}(q, \overline{c}) \leq \mathrm{d}^{(\text{old})}(q, y)$. In this subcase, it holds that $\mathrm{d}^{(\text{old})}(q, C_{xy}) = \mathrm{d}^{(\text{old})}(q, \overline{c})$ and $\mathrm{d}(q, C_{xy}) = \mathrm{d}(q, \overline{c})$, and thus $\mathrm{d}^{(\text{old})}(q, C_{xy}) = \mathrm{d}(q, C_{xy})$. Hence, we infer that $\Delta_C(x, y) - \Delta_C^{(\text{old})}(x, y) = 0$, as needed.
  - Assume that $\mathrm{d}^{(\text{old})}(q, y) < \mathrm{d}^{(\text{old})}(q, \overline{c})$. In this subcase, it holds that $\mathrm{d}^{(\text{old})}(q, C_{xy}) = \mathrm{d}^{(\text{old})}(q, y)$ and $\mathrm{d}(q, C_{xy}) = \mathrm{d}(q, y)$. Since $y \notin \{p, q\}$, we have $\mathrm{d}(q, y) = \mathrm{d}^{(\text{old})}(q, y)$, and thus $\mathrm{d}^{(\text{old})}(q, C_{xy}) = \mathrm{d}(q, C_{xy})$. Hence, we conclude that $\Delta_C(x, y) - \Delta_C^{(\text{old})}(x, y) = 0$, as required.
- Assume that $\mathrm{d}^{(\text{old})}(q, \overline{c}) > \mathrm{d}(p, q)$. In this case, it holds that $\mathrm{d}(q, p) \leq \mathrm{d}(q, \overline{c})$; otherwise, the strict inequality $\mathrm{d}(q, p) > \mathrm{d}(q, \overline{c})$ would imply that $p \neq \overline{c}$, which in turn would imply that $\mathrm{d}^{(\text{old})}(q, \overline{c}) = \mathrm{d}(q, \overline{c}) < \mathrm{d}(p, q)$, contradicting the assumption. In turn, we can infer that $\mathrm{d}(q, C) = \mathrm{d}(q, p)$. Since $y \notin \{p, q\}$, we have $\mathrm{d}(q, y) = \mathrm{d}^{(\text{old})}(q, y)$. Consider the two subcases depending on whether $\mathrm{d}(q, p)$ is less than $\mathrm{d}^{(\text{old})}(q, y)$:
  - Assume that $\mathrm{d}(q, p) \leq \mathrm{d}^{(\text{old})}(q, y)$. In this subcase, since $\mathrm{d}(q, y) = \mathrm{d}^{(\text{old})}(q, y)$ then $\mathrm{d}(q, p) \leq \mathrm{d}(q, y)$, and thus it holds that $\mathrm{d}(q, C_{xy}) = \min\big(\mathrm{d}(q, p), \mathrm{d}(q, \overline{c}) \big)$. Since $\mathrm{d}(q, p) \leq \mathrm{d}(q, \overline{c})$, it follows that $\mathrm{d}(q, C_{xy}) = \mathrm{d}(q, p) = \mathrm{d}(q, C)$. Moreover, it holds that $\mathrm{d}^{(\text{old})}(q, C_{xy}) = \min\big(\mathrm{d}^{(\text{old})}(q, \overline{c}), \mathrm{d}^{(\text{old})}(q, y) \big)$. Hence, we deduce that:
    $$\Delta_C(x, y) - \Delta_C^{(\text{old})}(x, y) \;=\; \operatorname{wt}(q) \cdot \Big(\min\big(\mathrm{d}^{(\text{old})}(q, \overline{c}), \mathrm{d}^{(\text{old})}(q, y) \big) - \mathrm{d}^{(\text{old})}(q, \overline{c}) \Big).$$
  - Assume that $\mathrm{d}(q, p) > \mathrm{d}^{(\text{old})}(q, y)$. In this subcase, it holds that $\mathrm{d}^{(\text{old})}(q, C_{xy}) = \mathrm{d}^{(\text{old})}(q, y)$ and $\mathrm{d}(q, C_{xy}) = \min\big(\mathrm{d}(q, y), \mathrm{d}(q, \overline{c}) \big)$. Since $\mathrm{d}(q, y) = \mathrm{d}^{(\text{old})}(q, y)$ then $\mathrm{d}(q, y) < \mathrm{d}(p, q)$, and since $\mathrm{d}(p, q) \leq \mathrm{d}(q, \overline{c})$ then $\mathrm{d}(q, y) < \mathrm{d}(q, \overline{c})$. In turn, we deduce that $\mathrm{d}(q, C_{xy}) = \mathrm{d}(q, y)$, and thus $\mathrm{d}^{(\text{old})}(q, C_{xy}) = \mathrm{d}(q, C_{xy})$. Hence, we conclude that:
    $$\Delta_C(x, y) - \Delta_C^{(\text{old})}(x, y) \;=\; \operatorname{wt}(q) \cdot \big(\mathrm{d}(q, p) - \mathrm{d}^{(\text{old})}(q, \overline{c}) \big).$$

□

Consider a fixed point $y \in (P \setminus C) \setminus \{p, q\}$ and the corresponding column $\Delta_C(\cdot, y)$ in the swap table. The crucial structural observation in Lemma 3.1 is that all the entries $\Delta_C(x, y)$ where $x \in C \setminus \{p, q\}$, depend only on p, q, y, and the closest center $\overline{c}$ to q. In other words, a fixed column $\Delta_C(\cdot, y)$ does not depend on $x \neq \overline{c}$. Therefore, the entries in the fixed column $\Delta_C(\cdot, y)$ change smoothly, enabling efficient maintenance of $\Delta_C(\cdot, y)$ in $O(\log k)$ update time using a dynamic range data structure. Consequently, after an adversarial distance decrease, the two-dimensional swap table Δ_C can be maintained implicitly with an efficient worst-case update time of $O(n \log k)$.

A.1.2 Adversarial Distance Decreases

When the adversary decreases the distance between two points $p, q \in P$ (i.e., $\mathrm{d}(p, q) < \mathrm{d}^{(\text{old})}(p, q)$), the distance-decrease swap table algorithm proceeds according to Section A.1.1. Initially, for every point $y \in \{p, q\} \cap (P \setminus C)$ and for every center $x \in C$, the swap table algorithm performs two steps:

1. First, the value of *diffGain* is calculated as indicated (see Lemma A.4):

$$\begin{aligned} \textit{diffGain} \;\coloneqq\; & \mathrm{wt}(p) \cdot \big(\mathrm{d}(p, C) - \mathrm{d}(p, C_{xy}) - \mathrm{d}^{(\text{old})}(p, C) + \mathrm{d}^{(\text{old})}(p, C_{xy}) \big) + \\ & \mathrm{wt}(q) \cdot \big(\mathrm{d}(q, C) - \mathrm{d}(q, C_{xy}) - \mathrm{d}^{(\text{old})}(q, C) + \mathrm{d}^{(\text{old})}(q, C_{xy}) \big). \end{aligned}$$

2. Second, the swap table algorithm passes to $\mathcal{RNG}_y$ the single-index update of the form $([\textit{ind}_x, \textit{ind}_x], \textit{diffGain})$.

Similarly, for every center $x \in \{p, q\} \cap C$ and for every point $y \in (P \setminus C) \setminus \{p, q\}$, the swap table algorithm performs the same two steps:

1. First, the value of *diffGain* is calculated as indicated (see Lemma A.4):

$$\begin{aligned} \textit{diffGain} \;\coloneqq\; & \mathrm{wt}(p) \cdot \big(\mathrm{d}(p, C) - \mathrm{d}(p, C_{xy}) - \mathrm{d}^{(\text{old})}(p, C) + \mathrm{d}^{(\text{old})}(p, C_{xy}) \big) + \\ & \mathrm{wt}(q) \cdot \big(\mathrm{d}(q, C) - \mathrm{d}(q, C_{xy}) - \mathrm{d}^{(\text{old})}(q, C) + \mathrm{d}^{(\text{old})}(q, C_{xy}) \big). \end{aligned}$$

2. Second, the swap table algorithm passes to $\mathcal{RNG}_y$ the single-index update of the form $([\textit{ind}_x, \textit{ind}_x], \textit{diffGain})$.

Next, there are two mutually exclusive scenarios for the distance-decrease swap table algorithm, depending on whether the points p and q belong to the set of centers C.

Scenario 1**.** In the first scenario, either $p, q \notin C$ or $p, q \in C$, and the swap table algorithm does nothing further. The justification is the following: the first subscenario is when neither p nor q is a center and the second subscenario is when both p and q are centers. By Observation A.5 for the first subscenario and Observation A.6 for the second subscenario, the rest of the entries in $\Delta_C(\cdot, \cdot)$ remain unchanged, and so the algorithm does not need to perform further steps.

Scenario 2**.** In the second scenario, without loss of generality we have $p \in C$ and $q \notin C$, since otherwise the swap table algorithm exchanges the labels of p and q. Let $\overline{c} \coloneqq \operatorname{argmin}_{c \in C} \mathrm{d}^{(\text{old})}(q, c)$ be the closest center to q before the adversarial distance decrease. Then for every point $y \in (P \setminus C) \setminus \{p, q\}$, there are four steps:

1. First, the value of *diffGain* is calculated as indicated (see Lemma 3.1):
   - If $\mathrm{d}^{(\text{old})}(q, \overline{c}) \leq \mathrm{d}(p, q)$ then $\textit{diffGain} \;\coloneqq\; 0$.
   - Otherwise if $\mathrm{d}^{(\text{old})}(q, \overline{c}) > \mathrm{d}(p, q)$ and $\mathrm{d}(p, q) \leq \mathrm{d}^{(\text{old})}(q, y)$ then:

$$\textit{diffGain} \;\coloneqq\; \mathrm{wt}(q) \cdot \Big(\min \big(\mathrm{d}^{(\text{old})}(q, \overline{c}), \mathrm{d}^{(\text{old})}(q, y) \big) - \mathrm{d}^{(\text{old})}(q, \overline{c}) \Big).$$

   - Otherwise if $\mathrm{d}^{(\text{old})}(q, \overline{c}) > \mathrm{d}(p, q)$ and $\mathrm{d}(p, q) > \mathrm{d}^{(\text{old})}(q, y)$ then:

$$\textit{diffGain} \;\coloneqq\; \mathrm{wt}(q) \cdot \big(\mathrm{d}(q, p) - \mathrm{d}^{(\text{old})}(q, \overline{c}) \big).$$

2. Second, let I_1, I_2, I_3 be the three segments partitioning the range $[1, |C|]$ by excluding the indices of p and $\overline{c}$. Assuming without loss of generality that $\text{ind}_p \leq \text{ind}_{\overline{c}}$, the segments I_1, I_2, I_3 are defined as specified:
$$I_1 \coloneqq [1, \mathit{ind}_p - 1], \quad I_2 \coloneqq [\text{ind}_p + 1, \text{ind}_{\overline{c}} - 1], \quad I_3 \coloneqq [\text{ind}_{\overline{c}} + 1, |C|].$$
Subsequently, the swap table algorithm passes to $\mathcal{RNG}_y$ three updates of the form $(I_1, \mathit{diffGain})$, $(I_2, \mathit{diffGain})$, and $(I_3, \mathit{diffGain})$.
3. Third, for the center $\overline{c} \in C$ (if $\overline{c} \neq p$), the value of *diffGain* is calculated as indicated with $x = \overline{c}$ (see Lemma 3.1):
$$\mathit{diffGain} \coloneqq \text{wt}(q) \cdot \big(\text{d}(q, C) - \text{d}(q, C_{xy}) - \text{d}^{(\text{old})}(q, C) + \text{d}^{(\text{old})}(q, C_{xy}) \big).$$
4. Fourth, the swap table algorithm passes to $\mathcal{RNG}_y$ the single-index update of the form $([\mathit{ind}_{\overline{c}}, \mathit{ind}_{\overline{c}}], \mathit{diffGain})$.

Update the max-heap. In the end of both scenarios, for every *affected* point $y \in P \setminus C$ (i.e., a point whose $\mathcal{RNG}_y$ received an update), the distance-decrease swap table algorithm performs two steps:

- First, the swap table algorithm queries $\mathcal{RNG}_y$, which returns a value *maxColGain* and the corresponding row index x.
- Second, the tuple $(\mathit{maxColGain}, x)$ replaces in the max-heap $\mathcal{H}$ the old tuple representing the point y.

The value $\max(\Delta_C)$ and the pair of points $(x, y) \in C \times (P \setminus C)$ attaining it (i.e., $\Delta_{C,\text{d}}(x, y) = \max(\Delta_{C,\text{d}})$) are then maintained via the root of the max-heap $\mathcal{H}$.

A.2 Analysis of the Distance-Decrease Swap Table Algorithm

Using the lemmas and observations from Section A.1.1, together with the description of the distance-decrease swap table algorithm in Section A.1.2, we can now finish the proof of Theorem 1.3 which we restate for convenience.

Theorem 1.3. *There is a deterministic algorithm in the single-update distance-decrease setting that given:*

- *a weighted (k, z)-clustering instance $\mathcal{K} = (P, k, z, \text{wt}, \text{d}, \infty)$ subject to distance decreases of the form (p, q, dst) (i.e., the adversarial update decreases the distance between p and q by dst),*
- *an initial (k, z)-clustering solution C,*
- *and the initial corresponding two-dimensional swap table Δ_C,*

maintains the value $\max(\Delta_C)$ and the pair of points $(x, y) \in C \times (P \setminus C)$ attaining it (i.e., $\Delta_{C,\text{d}}(x, y) = \max(\Delta_{C,\text{d}})$) with worst-case update time $O(k \log k + \log n)$ when $C \cap \{p, q\} = \emptyset$ and $O(n \log n)$ otherwise; the preprocessing time is $O(nk)$.

Proof. For every point $y \in P \setminus C$, the distance-decrease swap table algorithm maintains a dynamic range data structure $\mathcal{RNG}_y$. Since there are $n - k$ such points y and the preprocessing time for each $\mathcal{RNG}_y$ is $O(|C|) = O(k)$ (see Definition A.2), the total preprocessing time of the swap table algorithm is $O(nk)$. Note that the preprocessing time of the max-heap $\mathcal{H}$ is $O(|P| - |C|) = O(n)$.

Computation of the distances $\text{d}(\cdot, C)$ and $\text{d}(\cdot, C_{xy})$. For every point $a \in P$ and swap $(x, y) \in C \times (P \setminus C)$, the distances $\text{d}(a, C)$ and $\text{d}(a, C_{xy})$ can be computed in constant time, as follows. The distance $\text{d}(a, C)$ and the closest center to a can be maintained in constant time, using information from the preprocessing phase and updating it upon each adversarial distance decrease.

Regarding the distance $\text{d}(a, C_{xy})$, let c_1 be the closest center to a in C (i.e., $\text{d}(a, C) = \text{d}(a, c_1)$), and c_2 be the second closest center to a in C (which can be maintained in a similar way). Notice that if $x \neq c_1$, then $\text{d}(a, C_{xy}) = \min(\text{d}(a, c_1), \text{d}(a, y))$; otherwise if $x = c_1$ then $\text{d}(a, C_{xy}) = \min(\text{d}(a, c_2), \text{d}(a, y))$. All of this relevant information can be maintained in constant update time.

Correctness. Regarding the correctness of the distance-decrease swap table algorithm, consider a distance decrease between two points p and q (i.e., $\mathrm{d}(p,q) < \mathrm{d}^{(\text{old})}(p,q)$). Initially, the columns and rows of the two-dimensional swap table Δ_C corresponding to p and q are (implicitly) updated correctly, according to Lemma A.4.

Next, the distance-decrease swap table algorithm continues based on two mutually exclusive scenarios, in order to implicitly update the remaining columns of Δ_C. The correctness of the swap table algorithm follows then from Observation A.5, Observation A.6, and Lemma 3.1, and the fact that the root of the max-heap $\mathcal{H}$ stores the maximum value $\max(\Delta_C)$.

Update time. Regarding the update time of the distance-decrease swap table, we analyze the number of queries and updates performed per column to the dynamic range data structure and the max-heap. Notice that the algorithm performs $|C| = k$ such updates only for the columns corresponding to p and q; for each of the remaining columns, it performs a constant number of such queries and updates.

There are $n-k$ columns in $\Delta_C(\cdot,\cdot)$ (i.e., the number of non-centers), and the worst-case update and query times of these data structures (dynamic range data structure and max-heap) are $O(\log |L|)$, where $|L|$ denotes the number of elements stored (see Definition A.2). As a result, the worst-case update time of the swap table algorithm is $O(n \log n)$, because for the max-heap we have $|L| = n - k$.[7] Moreover, when neither p nor q is a center, the algorithm updates only the corresponding two columns. Therefore, in this scenario the worst-case update time becomes $O(k \log k + \log n)$. Observe that the max-heap receives updates only from the *affected* columns. □

A.3 Distance-Decrease Local Search Algorithm

We remind the reader that the model studied involves a (k,z)-clustering instance consisting of points in a ρ-metric space (the relaxed triangle inequality is satisfied), where the adversary can decrease multiple distances between points per update. Our *distance-decrease local search algorithm* for the weighted (k,z)-clustering problem maintains a constant-factor approximate solution in the batch-update distance-decrease setting, as stated in the following theorem.

Theorem 1.4. *There is a deterministic algorithm in the batch-update distance-decrease setting that given:*

- *a weighted (k,z)-clustering instance $\mathcal{K} = (P,k,z,\mathrm{wt},\mathrm{d},\rho)$ subject to batches of distance decreases,*
- *an integer $k \geq 1$, constants $z \geq 1$ and $\rho = 1 + \frac{1}{16z}$, and an upper bound M on the distance function $\mathrm{d}(\cdot,\cdot)$,*

maintains an $O(1)$-approximate (k,z)-clustering solution with a total update time of $O\big(kn^2(\log nM)^3\big)$ and a preprocessing time of $O(kn^2 \log nM)$.

Maintaining distances as powers of $(1+\epsilon)$. For efficiency reasons, the distances between points are maintained as powers of $(1+\epsilon)$ with $\epsilon \in (0, \frac{1}{16z})$, and are upper bounded by M. As a consequence, the total number of distance decreases is at most $O\big(n^2 \log_{1+\epsilon} M\big)$, at the cost of an extra factor of $(1+\epsilon)^z$ in the approximation ratio. The local search algorithm internally rounds distances up to the nearest power of $(1+\epsilon)$ by performing lazy updates as Algorithm 1 in Section B; the analysis of the correctness follows by Lemmas B.7 and A.7.

In Section B, our distance-decrease local search algorithm is applied in $(1+\epsilon)^2$-metric spaces, which arises due to the use of $(1+\epsilon)$-approximate SSSP algorithms combined with rounding distances up to the nearest power of $(1+\epsilon)$. Hence in that case, the parameter ρ is set to $(1+\epsilon)^2$, where ϵ is chosen appropriately.

Lemma A.7. *Given a weighted (k,z)-clustering instance $\mathcal{K}_1 = (P,k,z,\mathrm{wt},\mathrm{d},\rho)$, let $\mathcal{K}_2 = (P,k,z,\mathrm{wt},\tilde{\mathrm{d}},\rho)$ be a weighted (k,z)-clustering instance such that for every pair of points $p_1,p_2 \in P$:*

$$\mathrm{d}(p_1,p_2) \;\leq\; \tilde{\mathrm{d}}(p_1,p_2) \;\leq\; \gamma \cdot \mathrm{d}(p_1,p_2).$$

[7] Note that an $O(n \log k)$ update time is possible by rebuilding the max-heap, which takes $O(n)$ time.

Then it holds that $\mathrm{OPT}_{\mathcal{K}_2} \leq \gamma^z \, \mathrm{OPT}_{\mathcal{K}_1}$, where $\mathrm{OPT}_{\mathcal{K}_1}$ and $\mathrm{OPT}_{\mathcal{K}_2}$ are the optimal (k,z)-clustering costs of $\mathcal{K}_1$ and $\mathcal{K}_2$ respectively.

Proof. Let C^* be an optimal (k,z)-clustering solution to $\mathcal{K}_1$ with cost $\mathrm{OPT}_{\mathcal{K}_1}$. The optimal (k,z)-clustering cost of $\mathcal{K}_2$ can be upper bounded as follows:

$$\mathrm{OPT}_{\mathcal{K}_2} \;\leq\; \sum_{p \in P} \mathrm{wt}(p) \cdot \tilde{\mathrm{d}}(p, C^*)^z \;\leq\; \gamma^z \cdot \sum_{p \in P} \mathrm{wt}(p) \cdot \mathrm{d}(p, C^*)^z \;=\; \gamma^z \cdot \mathrm{OPT}_{\mathcal{K}_1}.$$

□

Our distance-decrease local search algorithm utilizes the static swap table algorithm by Schubert & Rousseeuw (2021).

Lemma A.8 (static swap table algorithm (Schubert & Rousseeuw, 2021))**.** *There is a deterministic algorithm that, given a weighted (k,z)-clustering instance $\mathcal{K} = (P, k, z, \mathrm{wt}, \mathrm{d}, \rho)$ and a k-median solution C, computes the two-dimensional swap table Δ_C in $O(n^2)$ time.*

Our distance-decrease local search algorithm internally maintains our distance-decrease swap table algorithm $\mathcal{SW}_{\mathrm{decr}}$, the static swap table algorithm $\mathcal{SW}_{\mathrm{stat}}$ from Lemma A.8, and the (k,z)-clustering solution C.[8]

A.3.1 Adversarial Batches of Distance Decreases

Let $tf \coloneqq \frac{\delta}{k}$ be the threshold factor for the maximum value $\max(\Delta_C)$ of the two-dimensional swap table, where $\delta \in (0, 0.1)$ is a fixed constant (see Lemmas A.14 and A.16 for details). When the adversary sends a batch B of distance decreases, the local search algorithm updates the cost of the maintained (k,z)-clustering solution C, and processes each distance decrease in the batch B separately as follows: For every distance decrease $(p, q, dst) \in B$ where $\mathrm{d}(p,q) = \mathrm{d}^{(\mathrm{old})}(p,q) - dst < \mathrm{d}^{(\mathrm{old})}(p,q)$, the distance-decrease local search algorithm forwards (p, q, dst) to the distance-decrease swap table algorithm $\mathcal{SW}_{\mathrm{decr}}$.

Once the entire batch has been processed by the swap table algorithm $\mathcal{SW}_{\mathrm{decr}}$, the value $\max(\Delta_C)$ is retrieved using $\mathcal{SW}_{\mathrm{decr}}$. If $\max(\Delta_C) < tf \cdot \mathrm{cost}^z_{\mathrm{wt}}(C)$, then the local search algorithm does nothing further. Otherwise, if $\max(\Delta_C) \geq tf \cdot \mathrm{cost}^z_{\mathrm{wt}}(C)$ then the local search algorithm continues with the following two phases:

1. **First phase**: The pair of points $(x,y) \in C \times (P \setminus C)$ such that $\Delta_C(x,y) = \max(\Delta_C)$ is retrieved using $\mathcal{SW}_{\mathrm{decr}}$, and the swap between x and y is performed, producing $C_{xy} = C \cup \{y\} \setminus \{x\}$. Subsequently, the (k,z)-clustering solution is updated to C_{xy}, namely $C \coloneqq C_{xy}$. The static swap table algorithm $\mathcal{SW}_{\mathrm{stat}}$ is then executed with the updated C included in the input, returning the corresponding updated swap table Δ_C.
   Next, if $\max(\Delta_C) < tf \cdot \mathrm{cost}^z_{\mathrm{wt}}(C)$ then the first phase ends. Otherwise, if $\max(\Delta_C) \geq tf \cdot \mathrm{cost}^z_{\mathrm{wt}}(C)$ then the first phase is repeated. However, from the second repetition onward, the relevant pair of points (x,y) is retrieved using the explicit swap table computed by $\mathcal{SW}_{\mathrm{stat}}$, since $\mathcal{SW}_{\mathrm{decr}}$ still contains outdated information.
2. **Second phase**: Let C be the final (k,z)-clustering solution produced after the end of the first phase. Additionally, let Δ_C be the corresponding swap table after the end of the first phase, and observe that $\max(\Delta_C) < tf \cdot \mathrm{cost}^z_{\mathrm{wt}}(C)$. Finally, the distance-decrease swap table algorithm $\mathcal{SW}_{\mathrm{decr}}$ is restarted with the updated C and Δ_C included in the input.

A.4 Analysis of the Distance-Decrease Local Search Algorithm

In this section, we prove Theorem 1.4 by analyzing our distance-decrease local search algorithm. The analysis is divided into three subsections, as outlined below. First, in Section A.4.1 we analyze the total update time of the local search algorithm. Next, in Section A.4.2 we derive a useful relationship between the approximation ratio of the maintained (k,z)-clustering solution C and the two-dimensional swap table Δ_C. Finally, in Section A.4.3 we conclude the proof of Theorem 1.4.

[8] The subscripts `decr` and `stat` indicate that the algorithms are distance-decrease and static, respectively.

A.4.1 TOTAL UPDATE TIME

Lemma A.9. *The total number of swaps performed by the distance-decrease local search algorithm is at most $O(k \log nM)$.*

Proof. Observe that a swap between a center $x \in C$ and a point $y \in P \setminus C$ is performed only when $\Delta_C(x, y) = \max(\Delta_C) \geq tf \cdot \mathrm{cost}^z_{\mathrm{wt}}(C)$. By the definition of $\Delta_C(x, y)$, we have $\mathrm{cost}^z_{\mathrm{wt}}(C_{xy}) = \mathrm{cost}^z_{\mathrm{wt}}(C) - \Delta_C(x, y)$, where $C_{xy} = C \cup \{y\} \setminus \{x\}$. Hence, it follows that:

$$\begin{aligned} \mathrm{cost}^z_{\mathrm{wt}}(C_{xy}) &= \mathrm{cost}^z_{\mathrm{wt}}(C) - \Delta_C(x, y) \\ &\leq \mathrm{cost}^z_{\mathrm{wt}}(C) - tf \cdot \mathrm{cost}^z_{\mathrm{wt}}(C) \\ &= \mathrm{cost}^z_{\mathrm{wt}}(C)\big(1 - tf\big). \end{aligned}$$

Since the local search algorithm updates the (k, z)-clustering solution C to C_{xy}, the cost of C decreases by a multiplicative factor of $\big(1 - tf\big)$ per swap. Moreover, the cost of the solution is non-increasing under adversarial distance decreases, which means that the cost of C can only decrease. The maximum initial cost of C is at most nM,[9] and thus the total number of swaps is upper bounded by:

$$O\Big(\frac{\log nM}{tf}\Big) = O\Big(\frac{k \cdot \log nM}{\delta}\Big) = O(k \cdot \log nM),$$

where δ is a fixed positive value. □

Lemma A.10. *The total recourse of the (k, z)-clustering solution C is at most $O(k \log nM)$.*

Proof. Whenever a swap between a center and a point is performed, the (k, z)-clustering solution changes by one center. Based on Lemma A.9, the total number of swaps is at most $O(k \cdot \log nM)$. Consequently, the total recourse of C is also $O(k \cdot \log nM)$. □

Corollary A.11. *The total number of distinct points ever included in the (k, z)-clustering solution C is at most $O(k \log nM)$.*

Proof. The claim follows from the fact that the initial size of C is k, together with Lemma A.10. □

Lemma A.12. *Let $p, q \in P$ denote a pair of points whose distance decreases (i.e., $\mathrm{d}(p, q) < \mathrm{d}^{(old)}(p, q)$). Throughout the execution of the distance-decrease local search algorithm:*

- *The event that at least one of p and q is a center (i.e., $|C \cap \{p, q\}| \geq 1$) occurs at most $O\big(kn(\log nM)^2\big)$ times.*
- *The event that neither p nor q is a center occurs at most $O\big(n^2 \log nM\big)$ times.*

Proof. Based on Corollary A.11, the number of distinct points ever included in the (k, z)-clustering solution C is upper bounded by $O(k \log nM)$. Hence regarding the first event, the total number of pairs (x, y) where either $x \in C$ or $y \in C$, is upper bounded by $O\big(kn \log nM\big)$.

Regarding the second event, the total number of pairs (x, y) where $x \in P \setminus C$ and $y \in P \setminus C$, is upper bounded by $O(n^2)$. Since the distances are maintained as powers of $(1 + \epsilon)$, the distance between any fixed pair of points $(x, y) \in P \times P$ can decrease at most $O(\log_{1+\epsilon} M)$ times, and thus both claims follow. □

Lemma A.13. *The total update time of the distance-decrease local search algorithm is:*

$$O\big(kn^2(\log nM)^3\big).$$

Proof. The static swap table algorithm $\mathcal{SW}_{\mathrm{stat}}$ is executed only when the local search algorithm performs a swap. Based on Lemma A.9, the total number of swaps is upper bounded by $O(k \log nM)$. Since the running time of $\mathcal{SW}_{\mathrm{stat}}$ is $O(n^2)$ by Lemma A.8, the total time due to $\mathcal{SW}_{\mathrm{stat}}$ is $O(kn^2 \log nM)$. Similarly, the distance-decrease swap table algorithm $\mathcal{SW}_{\mathrm{decr}}$ is restarted only

[9] For simplicity, we assume that $z = 1$ and $\mathrm{wt}(\cdot) = 1$; the proof is easily generalized.

when a swap is performed, and its preprocessing time is $O(nk)$ according to Theorem 1.3. Hence, it remains to analyze the total update time due to $\mathcal{SW}_{\mathrm{decr}}$ under adversarial distance decreases.

Consider a distance decrease between two points $p, q \in P$ (i.e., $\mathrm{d}(p,q) < \mathrm{d}^{(\mathrm{old})}(p,q)$). Based on Theorem 1.3, the worst-case update time of the swap table algorithm $\mathcal{SW}_{\mathrm{decr}}$ is $O(k \log n)$ when $p, q \in P \setminus C$ and $O(n \log n)$ otherwise. Therefore, based on Lemma A.12 we deduce that:

- the total time charged to the first event is $O\big(kn(\log nM)^2\big) \cdot O(n \log n) = O\big(kn^2(\log nM)^3\big)$, and
- the total time charged to the second event is $O\big(n^2 \log nM\big) \cdot O(k \log n) = O\big(kn^2(\log nM)^2\big)$.

Combining all these together, the total update time of the local search algorithm is upper bounded by $O\big(kn^2(\log nM)^3\big)$. □

A.4.2 Approximation Ratio

The analyses by Gupta & Tangwongsan (2008), and by Bhattacharya et al. (2024b), reveal a useful relationship between the approximation ratio of the (k,z)-clustering solution C and the two-dimensional swap table Δ_C (similar to Lemma A.14). In the papers by Gupta & Tangwongsan (2008) and Bhattacharya et al. (2024b), an entry $\Delta_C(x,y)$ in the swap table is defined as the difference $\mathrm{cost}^z_{\mathrm{wt}}(C_{xy}) - \mathrm{cost}^z_{\mathrm{wt}}(C)$, whereas in our work, we define it as the gain $\mathrm{cost}^z_{\mathrm{wt}}(C) - \mathrm{cost}^z_{\mathrm{wt}}(C_{xy})$. As a result, negative values in the papers by Gupta & Tangwongsan (2008) and Bhattacharya et al. (2024b) correspond to positive values in our context. Moreover, Gupta & Tangwongsan (2008) and Bhattacharya et al. (2024b) allow a center to be swapped with another center, while we only allow a center to be swapped with a non-center.

For the purposes of Lemma A.14, we define $\hat{\Delta}_C(x,y)$ analogously to $\Delta_C(\cdot,\cdot)$ for all pairs of points $(x,y) \in C \times P$, whereas the swap table $\Delta_C(x,y)$ is defined only for pairs of points $(x,y) \in C \times (P \setminus C)$ (see Definition A.1). The following statement follows by modifying the corresponding proofs in the papers by Gupta & Tangwongsan (2008) and Bhattacharya et al. (2024b), replacing the triangle inequality with the ρ-approximate triangle inequality. The switch to the ℓ_z-objective is for technical reasons, but any α-approximation for the ℓ_z-objective is also an α^z-approximation for our (k,z)-clustering objective.

Lemma A.14. *For any constants $z \geq 1$ and $0 < \delta < 0.1$, there exist constants $\rho \geq 1$ and $\alpha \geq 1$ such that the following holds:*

- *Let $C^* \coloneqq \arg\min_{C \subseteq P : |C| = k} (\mathrm{cost}^z_{\mathrm{wt}}(C))^{1/z}$ be an optimal solution with respect to the ℓ_z-objective, and*
- *let $C \subseteq P$ with $|C| = k$ be a set of centers such that $(\mathrm{cost}^z_{\mathrm{wt}}(C))^{1/z} > \alpha \cdot (\mathrm{cost}^z_{\mathrm{wt}}(C^*))^{1/z}$.*

Then there is a set $T = \{(x_i, y_i)\}_{i=1}^k \subseteq C \times P$ of k swaps such that:

$$\sum_{i=1}^{k} \hat{\Delta}_C(x_i, y_i) \;\geq\; \delta \cdot \mathrm{cost}^z_{\mathrm{wt}}(C).$$

Proof. Consider the assignment $\sigma^* : P \mapsto C^*$ that maps each point to its nearest optimal center in C^*. For every $c^* \in C^*$, the set $N^*(c^*) \coloneqq \{p \in P \mid \mathrm{d}(p, C^*) = \mathrm{d}(p, c^*)\} = \{p \in P \mid \sigma^*(p) = c^*\}$ denotes the corresponding optimal cluster, where $\bigcup_{c^* \in C^*} N^*(c^*)$ forms a partition of the point set P. Similarly, consider the assignment $\sigma : P \mapsto C$ that maps each point to its nearest center in C. For every center $c \in C$, the set $N(c) \coloneqq \{p \in P \mid \mathrm{d}(p, C) = \mathrm{d}(p, c)\} = \{p \in P \mid \sigma(p) = c\}$ denotes the cluster associated with c in C, where $\bigcup_{c \in C} N(c)$ forms a partition of P.

Consider the swaps $(x,y) \in T$ and the assignment $\pi : C^* \mapsto C$ which are defined in the paper by Gupta & Tangwongsan (2008) as follows:[10] The assignment π maps each optimal center $c^* \in C^*$

[10] In Gupta & Tangwongsan (2008), the set T is denoted by S and the assignment π is denoted by η.

to its closest center in C. Hence, the set C is partitioned into three subsets:

$$C_0 \coloneqq \{c \in C \mid |\pi^{-1}(c)| = 0\},\ C_1 \coloneqq \{c \in C \mid |\pi^{-1}(c)| = 1\},\ \text{and } C_{\geq 2} \coloneqq \{c \in C \mid |\pi^{-1}(c)| \geq 2\}.$$

The k pairs of T are selected such that:

- Each $c^* \in C^*$ appears in exactly one pair (c, c^*), where $c \in C_0 \cup C_1$.
- Each $c \in C_1$ appears only in the pair $(c, \pi^{-1}(c))$.
- Each $c \in C_0$ appears in at most two pairs.

Such a set T exists by an averaging argument (Gupta & Tangwongsan, 2008). The sum of the gains in T is then evaluated as follows:

$$\begin{aligned}
\sum_{(x,y)\in T} \hat{\Delta}_C(x,y) \;&=\; \sum_{(x,y)\in T} \Big(\mathrm{cost}^z_{\mathrm{wt}}(C) - \mathrm{cost}^z_{\mathrm{wt}}(C_{xy}) \Big) \\
&\geq \sum_{(x,y)\in T} \left(\sum_{p\in N^*(y)} \mathrm{wt}(p) \cdot \Big(\mathrm{d}(p,C)^z - \mathrm{d}(p,y)^z \Big) + \sum_{p\in N(x)} \mathrm{wt}(p) \cdot \Big(\mathrm{d}(p,C)^z - \mathrm{d}(p,\pi(\sigma^*(p)))^z \Big) \right) \\
&\geq\; -\sum_{p\in P} \mathrm{wt}(p)\, \mathrm{d}(p,C^*)^z \;+\; 3\sum_{p\in P} \mathrm{wt}(p)\, \mathrm{d}(p,C)^z \;-\; 2\sum_{p\in P} \mathrm{wt}(p)\, \mathrm{d}(p,\pi(\sigma^*(p)))^z.
\end{aligned} \tag{1}$$

The first inequality holds because once a swap (x, y) is performed, any candidate assignment of the points in P to a center in C_{xy} is an upper bound on $\mathrm{cost}^z_{\mathrm{wt}}(C_{xy})$. Specifically for a fixed swap $(x, y) \in T$, the chosen candidate assignment is:

- Every point $p \in N^*(y)$ is assigned to y.
- Every point $p \in P \setminus N^*(y)$ whose closest center was not x is still assigned to the same center in $C \cap C_{xy}$.
- Every point $p \in P \setminus N^*(y)$ whose closest center was x is assigned to $\pi(\sigma^*(p))$.

In order to continue with (1), we need the next claim which addresses the relaxed triangle inequality of ρ-metric spaces.

Claim A.15. *For every point $p \in P$, it holds that* $\mathrm{d}(p,\pi(\sigma^*(p)))^z \leq \rho^{2z} \cdot \big(2\,\mathrm{d}(p,C^*) + \mathrm{d}(p,C)\big)^z$.

Proof. By the relaxed triangle inequality in ρ-metric spaces where $\rho \geq 1$, we obtain:

$$\begin{aligned}
\mathrm{d}(p,\pi(\sigma^*(p)))^z \;&\leq\; \rho^z \cdot \big(\mathrm{d}(p,\sigma^*(p)) + \mathrm{d}(\sigma^*(p),\pi(\sigma^*(p))) \big)^z \\
&=\; \rho^z \cdot \big(\mathrm{d}(p,C^*) + \mathrm{d}(\sigma^*(p),C) \big)^z \\
&\leq\; \rho^z \cdot \Big(\mathrm{d}(p,C^*) + \rho \cdot \big(\mathrm{d}(p,C) + \mathrm{d}(p,C^*) \big) \Big)^z \qquad (2) \\
&\leq\; \rho^{2z} \cdot \big(2\,\mathrm{d}(p,C^*) + \mathrm{d}(p,C) \big)^z.
\end{aligned}$$

The second inequality (2) holds because $\mathrm{d}(\sigma^*(p),C) \leq \mathrm{d}(\sigma^*(p),\sigma(p)) \leq \rho \cdot \big(\mathrm{d}(\sigma^*(p),p) + \mathrm{d}(p,\sigma(p)) \big) = \rho \cdot \big(\mathrm{d}(C^*,p) + \mathrm{d}(p,C) \big)$. □

Using Claim A.15, we apply Minkowski's inequality (see Lemma 5.3 in Bhattacharya et al. (2024b)) to get the bound $\sum_{p\in P} \mathrm{wt}(p) \big(2\,\mathrm{d}(p,C^*) + \mathrm{d}(p,C)\big)^z \leq \big(2(\mathrm{cost}^z_{\mathrm{wt}}(C^*))^{1/z} + (\mathrm{cost}^z_{\mathrm{wt}}(C))^{1/z}\big)^z$. The analysis for the sum of gains in T continues from (1) as follows:

$$\begin{aligned}
\sum_{(x,y)\in T} \hat{\Delta}_C(x,y) \;&\geq\; -\sum_{p\in P} \mathrm{wt}(p)\, \mathrm{d}(p,C^*)^z \;+\; 3\sum_{p\in P} \mathrm{wt}(p)\, \mathrm{d}(p,C)^z \;-\; 2\rho^{2z} \cdot \sum_{p\in P} \mathrm{wt}(p) \cdot \big(2\,\mathrm{d}(p,C^*) + \mathrm{d}(p,C)\big)^z \\
&\geq\; -\mathrm{cost}^z_{\mathrm{wt}}(C^*) + 3\,\mathrm{cost}^z_{\mathrm{wt}}(C) - 2\rho^{2z} \cdot \big(2(\mathrm{cost}^z_{\mathrm{wt}}(C^*))^{1/z} + (\mathrm{cost}^z_{\mathrm{wt}}(C))^{1/z}\big)^z.
\end{aligned}$$

Thus, to prove that $\sum_{(x,y)\in T} \hat{\Delta}_C(x,y) \geq \delta \operatorname{cost}^z_{\mathrm{wt}}(C)$, it suffices to show that:

$$\frac{\operatorname{cost}^z_{\mathrm{wt}}(C^*) + 2\rho^{2z} \cdot \left(2(\operatorname{cost}^z_{\mathrm{wt}}(C^*))^{1/z} + (\operatorname{cost}^z_{\mathrm{wt}}(C))^{1/z}\right)^z}{\operatorname{cost}^z_{\mathrm{wt}}(C)} \;\leq\; 3-\delta,$$

or equivalently:

$$\left(\frac{(\operatorname{cost}^z_{\mathrm{wt}}(C^*))^{1/z}}{(\operatorname{cost}^z_{\mathrm{wt}}(C))^{1/z}}\right)^z + 2\rho^{2z} \cdot \left(\frac{2(\operatorname{cost}^z_{\mathrm{wt}}(C^*))^{1/z} + (\operatorname{cost}^z_{\mathrm{wt}}(C))^{1/z}}{(\operatorname{cost}^z_{\mathrm{wt}}(C))^{1/z}}\right)^z \;\leq\; 3-\delta.$$

By the assumption that $(\operatorname{cost}^z_{\mathrm{wt}}(C))^{1/z} > \alpha\,(\operatorname{cost}^z_{\mathrm{wt}}(C^*))^{1/z}$, it suffices to show that:

$$\left(\frac{1}{\alpha}\right)^z + 2\rho^{2z} \cdot \left(\frac{2}{\alpha}+1\right)^z \;\leq\; 3-\delta.$$

Since the inequality holds for $\rho = 1 + \frac{1}{16z}$ and $\alpha = \frac{z}{\delta}$, the claim follows. □

The values of α and ρ depend on z and δ; a suitable choice is $\alpha = \frac{z}{\delta}$ and $\rho = 1 + \frac{1}{16z}$. Note also that for $z = 1$ and $\rho = 1$, the analysis can be adapted to yield an approximation factor close to 5.

Lemma A.16. *For any constants $z \geq 1$ and $0 < \delta < 0.1$, there exist constants $\rho \geq 1$ and $\alpha \geq 1$ such that the following holds:*

- *Let $C^* := \arg\min_{C\subseteq P:|C|=k}(\operatorname{cost}^z_{\mathrm{wt}}(C))^{1/z}$ be an optimal solution with respect to the ℓ_z-objective, and*
- *let $C \subseteq P$ with $|C| = k$ be a set of centers such that $(\operatorname{cost}^z_{\mathrm{wt}}(C))^{1/z} > \alpha\cdot(\operatorname{cost}^z_{\mathrm{wt}}(C^*))^{1/z}$.*

Then for $tf := \frac{\delta}{k}$, it holds that:

$$\max(\Delta_C) \;\geq\; tf \cdot \operatorname{cost}^z_{\mathrm{wt}}(C) \;>\; 0.$$

Proof. Based on Lemma A.14, there is a set $T = \{(x_i, y_i)\}_{i=1}^k \subseteq C \times P$ of k swaps such that:

$$\sum_{i=1}^{k} \hat{\Delta}_C(x_i, y_i) \;\geq\; \delta \operatorname{cost}^z_{\mathrm{wt}}(C).$$

By an average argument, there must be a pair of points $(x, y) \in C \times P$ such that:

$$\hat{\Delta}_C(x,y) \;\geq\; \frac{\delta}{k} \operatorname{cost}^z_{\mathrm{wt}}(C) \;=\; tf \cdot \operatorname{cost}^z_{\mathrm{wt}}(C).$$

For any two distinct centers $x_1, x_2 \in C$ it holds that $\hat{\Delta}_C(x_1, x_2) \leq 0$, and for every center $x_1 \in C$ it holds that $\hat{\Delta}_C(x_1, x_1) = 0$. Since $\delta > 0$ we have $\hat{\Delta}_C(x, y) > 0$, and thus the point y must belong to $P \setminus C$. Namely, there must be a pair of points $(x, y) \in C \times (P \setminus C)$ such that:

$$\Delta_C(x,y) \;\geq\; tf \cdot \operatorname{cost}^z_{\mathrm{wt}}(C) \;>\; 0.$$

Therefore, it follows that $\max(\Delta_C) \;\geq\; tf \cdot \operatorname{cost}^z_{\mathrm{wt}}(C) \;>\; 0$, as desired. □

A.4.3 Proof Completion of Theorem 1.4

Using all the tools we have developed, we can now complete the proof of Theorem 1.4 which we restate for convenience.

Theorem 1.4. *There is a deterministic algorithm in the batch-update distance-decrease setting that given:*

- *a weighted (k, z)-clustering instance $\mathcal{K} = (P, k, z, \mathrm{wt}, \mathrm{d}, \rho)$ subject to batches of distance decreases,*
- *an integer $k \geq 1$, constants $z \geq 1$ and $\rho = 1 + \frac{1}{16z}$, and an upper bound M on the distance function $\mathrm{d}(\cdot,\cdot)$,*

maintains an $O(1)$-approximate (k,z)-clustering solution with a total update time of $O\big(kn^2(\log nM)^3\big)$ and a preprocessing time of $O(kn^2 \log nM)$.

Proof. Consider the distance-decrease local search algorithm (described in Section A.3) and its analysis. The preprocessing time follows from Lemmas A.8 and A.9; the total update time follows from Lemma A.13. After a batch of distance decreases, the local search algorithm proceeds in two phases. Based on Lemma A.16, whenever $(\mathrm{cost}^z_{\mathrm{wt}}(C))^{1/z} > \alpha \cdot (\mathrm{cost}^z_{\mathrm{wt}}(C^*))^{1/z}$ (for constant α), we have $\max(\Delta_C) \geq tf \cdot \mathrm{cost}^z_{\mathrm{wt}}(C)$.

During the first phase, a swap is performed as long as $\max(\Delta_C) \geq tf \cdot \mathrm{cost}^z_{\mathrm{wt}}(C)$. Hence in the second phase, it holds that $\max(\Delta_C) < tf \cdot \mathrm{cost}^z_{\mathrm{wt}}(C)$. Therefore by the contrapositive of Lemma A.16, we can deduce that $(\mathrm{cost}^z_{\mathrm{wt}}(C))^{1/z} \leq \alpha \cdot (\mathrm{cost}^z_{\mathrm{wt}}(C^*))^{1/z}$, and thus $\mathrm{cost}^z_{\mathrm{wt}}(C) \leq \alpha^z \cdot \mathrm{OPT}_{\mathcal{K}} = O(\mathrm{OPT}_{\mathcal{K}})$. □

B Application to (k,z)-Clustering on Incremental Graphs

In this section, we present an application of our distance-decrease local search algorithm (from Theorem 1.4) to dynamic graphs. Hence, we demonstrate that our model for point sets under distance updates can be utilized in the graph model under edge updates.

Preliminaries on graphs. We consider the incremental graph setting defined in section 1.1, and use the following notions of bicriteria approximation.

Definition B.1 ((α,β)-bicriteria approximation)**.** *Given a (k,z)-clustering instance, an (α,β)-bicriteria approximate solution is a subset of vertices $S \subseteq V$ such that:*

$$\mathrm{cost}^z(S) \leq \alpha \cdot \mathit{OPT} \;\; \mathit{and} \;\; |S| \leq \beta \cdot k.$$

Definition B.2 ((α,β)-bicriteria approximate assignment)**.** *Given a weighted undirected graph $G = (V,E,w)$ and an integer $k \geq 1$, an (α,β)-bicriteria approximate assignment is a function $\sigma : V \mapsto V$ such that:*

$$\mathrm{cost}^z(\sigma) \coloneqq \sum_{v \in V} \mathrm{dist}(v,\sigma(v))^z \leq \alpha \cdot \mathit{OPT} \;\; \mathit{and} \;\; |\sigma(V)| \leq \beta \cdot k.$$

Conversely, the preimage of every vertex $u \in V$ is $\sigma^{-1}(u) \coloneqq \{v \in V \mid \sigma(v) = u\}$.

In the model with dynamic graphs, our incremental algorithms utilize the following incremental $(1+\epsilon)$-approximate single-source shortest paths (SSSP) algorithm.

Lemma B.3 (incremental $(1+\epsilon)$-approximate SSSP Liu (2025))**.** *Let $m \geq 1$ and $\epsilon \geq \exp(-(\log m)^{0.99})$. Let $G = (V,E,w)$ be an incremental graph undergoing m edge insertions with integral edge weights in the range $[1,W]$ with a source $s \in V$. There is a deterministic data structure that explicitly maintains distance estimates $\delta_s : V \to \mathbb{R}_{\geq 0}$ such that:*

$$\mathrm{dist}(s,v) \;\leq\; \delta_s(v) \;\leq\; (1+\epsilon)\,\mathrm{dist}(s,v),$$

and the ability to report corresponding approximate shortest paths $\pi_{s,v}$ from s to v in time $O(|\pi_{s,v}|)$. The total update time of the data structure is $m^{1+o(1)} \log W$. Moreover, the algorithm detects and reports the distance estimate changes explicitly.

Cruciani, Forster, and Skarlatos Cruciani et al. (2026) developed an incremental bicriteria approximation algorithm that maintains an $(O(1), O(\log^3 n \log_{1+\epsilon} nW))$-bicriteria approximate solution to the (k,z)-clustering problem on a graph undergoing edge insertions (Theorem 1.1 in Cruciani et al. (2026)). Afterwards, the authors use their incremental reduction algorithm (Theorem 6.3 in Cruciani et al. (2026)) to convert the bicriteria approximate solution into a constant-factor approximate (k,z)-clustering solution after every adversarial edge insertion. To achieve this, they maintain a (k,z)-clustering solution C over the bicriteria approximate solution S, using weights on S determined by the corresponding bicriteria approximate assignment σ.

Their incremental reduction algorithm employs a dynamic spanner algorithm together with a static (k,z)-clustering algorithm. In this section, we explain how to use instead our distance-decrease local search algorithm $\mathcal{LS}$ (described in Section A.3), in order to derive the following theorem. Notice that since edge insertions in the incremental graph setting can only decrease distances, the local search algorithm $\mathcal{LS}$ operates under non-increasing distances, as required.

Theorem B.4. *There is a deterministic dynamic algorithm for the (k, z)-clustering problem that given:*

- *a weighted undirected graph $G = (V, E, w)$ subject to edge insertions,*
- *an integer $k \geq 1$, a constant $z \geq 1$, and*
- *a dynamic (α, β)-bicriteria approximate assignment σ, for any $\alpha \geq 1, \beta \geq 1$,*

maintains an $O(\alpha)$-approximate (k, z)-clustering solution with a total update time of:

$$|\sigma_{max}| \cdot m^{1+o(1)} \log W \ + \ O\left(\sigma_{chg} \cdot k|\sigma_{max}|^2 \left(\log nW\right)^3\right).$$

A pseudocode of our incremental reduction algorithm is provided in Algorithm 1. Our incremental reduction algorithm in Theorem B.4 is similar to Theorem 6.3 in Cruciani et al. (2026); however, we emphasize that we need to define another value in Definition B.6 due to the use of our local search algorithm $\mathcal{LS}$. Both of the following two definitions are used to analyze the total update time in Theorem B.4.

Definition B.5 (Definition 6.1 in Cruciani et al. (2026))**.** *Given a dynamic (α, β)-bicriteria approximate assignment σ, the dynamic set $\sigma_{max} \coloneqq \{s \in V \mid s \in \sigma(V)$ at some moment$\}$ contains all vertices that have ever belonged to the image of σ.*

Definition B.6. *Given a dynamic (α, β)-bicriteria approximate assignment σ, the value σ_{chg} denotes the total number of times that $\sigma(\cdot)$ changes. Formally, $\sigma_{chg} \coloneqq |\{\tau \in \mathbb{N} \mid \exists v \in V :$ $\sigma(v)$ is modified at the τ-th update$\}|$.*

Adversarial updates. Similarly to Cruciani et al. (2026), we combine the incremental bicriteria approximation algorithm from Theorem 1.1 of Cruciani et al. (2026) with our own incremental reduction Algorithm 1. Since the reduction algorithm is given as input a dynamic bicriteria approximate assignment σ, one type of adversarial update consists of modifications to the assignment σ. The other type involves edge insertions into the input graph G, where each adversarial edge insertion can trigger a batch of distance decreases between vertices in G.

State of the reduction algorithm. Our incremental reduction Algorithm 1 maintains a dynamic (α, β)-bicriteria approximate solution $S \coloneqq \sigma_{\max}$, which depends on the image of the dynamic (α, β)-bicriteria approximate assignment σ from the input. For every vertex $s \in S$, the reduction algorithm maintains an incremental $(1+\epsilon)$-approximate SSSP algorithm $\mathcal{A}_s$ providing distance estimates $\delta_s(\cdot)$. In turn, the reduction algorithm maintains a weighted point set P in a $(1+\epsilon)^2$-metric space $(\mathcal{X}, \mathrm{d})$ where $\epsilon \in (0, \frac{1}{64z})$, specified as follows:

- Each vertex $s \in S$ is represented by a unique point $\tau(s) \in P$, and its weight $\mathrm{wt}(\tau(s))$ is equal to $|\sigma^{-1}(s)|$. In other words, the weight of $\tau(s)$ is the number of vertices mapped to s under the assignment $\sigma(\cdot)$.
- For every pair of vertices $x, y \in S$, the distance $\mathrm{d}(\tau(x), \tau(y))$ between the two corresponding points $\tau(x)$ and $\tau(y)$ is equal to $\min(\delta_x(y), \delta_y(x))$, rounded up to the nearest power of $(1+\epsilon)$.

The incremental reduction algorithm then applies the distance-decrease local search algorithm $\mathcal{LS}$ of Theorem 1.4 on the dynamic weighted (k, z)-clustering instance $(P, k, z, \mathrm{wt}, \mathrm{d}, (1+\epsilon)^2)$ (see Definition 1.2); the maintained (k, z)-clustering solution C is the output of $\mathcal{LS}$.

Efficiency. For efficiency reasons, both (1) the total number of vertices that have ever belonged to the image of the dynamic assignment σ (i.e., $|\sigma_{\max}|$) and (2) the total number of times that σ is modified (i.e., σ_{chg}) should remain bounded. The usage of approximate distance estimates provided by the incremental $(1+\epsilon)$-approximate SSSP algorithms (see Lemma B.3) and the fact that the distances $\mathrm{d}(\cdot, \cdot)$ are rounded up to the nearest power of $(1+\epsilon)$, imply that the point set P in the reduction algorithm lies in a $(1+\epsilon)^2$-metric space (see Definition 1.1). Since the parameter ρ from Theorem 1.4 is set to $(1+\epsilon)^2$, the parameter ϵ is chosen appropriately.

B.1 Edge Insertions and Assignment Modifications

The next two paragraphs, describe how the incremental reduction algorithm processes an adversarial edge insertion into the graph G, and how it processes a batch of modifications to the dynamic assignment σ. Notice that in fact, each edge insertion into the graph G can trigger a batch of distance decreases between vertices in G alongside a batch of modifications to σ. To ensure that the maintained distances satisfy the relaxed triangle inequality, the reduction algorithm first processes the adversarial edge insertion and then processes the resulting modifications to σ.

Edge insertion into G. When an edge (u, v) is inserted into the graph $G = (V, E, w)$ from the adversary, the edge insertion (u, v) is forwarded to all the incremental $(1 + \epsilon)$-approximate SSSP algorithms $\mathcal{A}_s$, where $s \in S$. Let Δ_S be the set containing all the pairs of vertices $(x, y) \in S \times S$ such that the value of the distance estimate $\delta_x(y)$ becomes less than $\frac{\mathrm{d}(\tau(x),\tau(y))}{1+\epsilon}$.

For every pair of vertices $(x, y) \in \Delta_S$, the reduction algorithm updates the metric by setting the distances $\mathrm{d}(\tau(x), \tau(y))$ and $\mathrm{d}(\tau(y), \tau(x))$ to $\min(\delta_x(y), \delta_y(x))$, rounding them up to the nearest power of $(1 + \epsilon)$. Let $B \coloneqq \big\{\big(\tau(x), \tau(y), \mathrm{d}(\tau(y), \tau(x))\big) \mid (x, y) \in \Delta_S \text{ or } (y, x) \in \Delta_S\big\}$ be the batch of distance decreases. The reduction algorithm then forwards the batch B of distance decreases to the distance-decrease local search algorithm $\mathcal{LS}$.

Modifications to the assignment σ. Let R_S be the set containing all vertices $s \in V$ for which the preimage $\sigma^{-1}(s)$ is modified. For every vertex $s \in R_S$ that already belongs to the dynamic (α, β)-bicriteria approximate solution S, the reduction algorithm updates only its corresponding weight $\mathrm{wt}(\tau(s))$ to $|\sigma^{-1}(s)|$.

For every vertex $s \in R_S$ that does not belong to S, the incremental reduction algorithm proceeds as follows:

1. First, the reduction algorithm adds s to S and its representative point $\tau(s)$ to P.[11] The corresponding weight $\mathrm{wt}(\tau(s))$ is set to $|\sigma^{-1}(s)|$.

2. Second, the reduction algorithm initializes an incremental $(1 + \epsilon)$-approximate SSSP algorithm $\mathcal{A}_s$ providing distance estimates $\delta_s(\cdot)$. For every vertex $x \in S$, the distances $\mathrm{d}(\tau(s), \tau(x))$ and $\mathrm{d}(\tau(x), \tau(s))$ are set to $\min(\delta_s(x), \delta_x(s))$, rounded up to the nearest power of $(1 + \epsilon)$.

The reduction algorithm then restarts the distance-decrease local search algorithm $\mathcal{LS}$ on the new weighted (k, z)-clustering instance $(P, k, z, \mathrm{wt}, \mathrm{d}, (1 + \epsilon)^2)$.

At the end of each adversarial update, the incremental reduction algorithm assigns the maintained (k, z)-clustering solution C as the output of the local search algorithm $\mathcal{LS}$.

[11] Recall that $\tau(s)$ belongs to the point set P if and only if the vertex s belongs to the set S.

Algorithm 1 Incremental Reduction Algorithm

Require: Graph G, integer k, and bicriteria approximate assignment σ
Ensure: (k, z)-Clustering solution C

function EDGE-INSERTION(u, v)
  Insert (u, v) into G
  for $s \in S$ **do**
    $\mathcal{A}_s$.*insert*(u, v)
  $\Delta_S \leftarrow \{(x, y) \in S \times S \mid \delta_x(y) < \frac{\mathrm{d}(\tau(x),\tau(y))}{1+\epsilon}\}$ ▷ Set of affected distances
  for $(x, y) \in \Delta_S$ **do**
    $\mathrm{d}(\tau(x), \tau(y)) \leftarrow \min(\delta_x(y), \delta_y(x))$
    $\mathrm{d}(\tau(y), \tau(x)) \leftarrow \min(\delta_x(y), \delta_y(x))$
    Round $\mathrm{d}(\tau(y), \tau(x))$ and $\mathrm{d}(\tau(x), \tau(y))$ up to the nearest power of $(1 + \epsilon)$
  $B \leftarrow \big\{\big(\tau(x), \tau(y), \mathrm{d}(\tau(y), \tau(x))\big) \mid (x, y) \in \Delta_S \text{ or } (y, x) \in \Delta_S\big\}$
  $\mathcal{LS}$.*update*(B)
  $C \leftarrow \mathcal{LS}$.*output*$()$

function ASSIGNMENT-MODIFICATIONS$(\hat{\sigma})$
  $R_S \leftarrow \{s \in V \mid \sigma^{-1}(s) \neq \hat{\sigma}^{-1}(s)\}$
  $\sigma \leftarrow \hat{\sigma}$
  for $s \in R_S$ **do**
    $\mathrm{wt}(\tau(s)) \leftarrow |\sigma^{-1}(s)|$
    if $s \notin S$ **then**
      Add s to S and $\tau(s)$ to P
      $\mathcal{A}_s$.*initialize*(G, s) ▷ $\mathcal{A}_s$ provides distance estimates $\delta_s(\cdot)$
      for $x \in S$ **do**
        $\mathrm{d}(\tau(s), \tau(x)) \leftarrow \min(\delta_s(x), \delta_x(s))$
        $\mathrm{d}(\tau(x), \tau(s)) \leftarrow \min(\delta_s(x), \delta_x(s))$
        Round $\mathrm{d}(\tau(s), \tau(x))$ and $\mathrm{d}(\tau(x), \tau(s))$ up to the nearest power of $(1 + \epsilon)$
  $\mathcal{LS}$.*initialize*$(P, k, z, \mathrm{wt}, \mathrm{d}, (1 + \epsilon)^2)$
  $C \leftarrow \mathcal{LS}$.*output*$()$

B.2 Analysis of the Incremental Reduction Algorithm

In this section, we prove Theorem B.4 by analyzing the incremental reduction Algorithm 1. The reduction algorithm maintains a weighted (k, z)-clustering instance $\mathcal{K} \coloneqq (P, k, z, \mathrm{wt}, \mathrm{d}, (1 + \epsilon)^2)$ induced by the dynamic bicriteria approximate solution S in the graph G. To distinguish the cost of a (k, z)-clustering solution C in $\mathcal{K}$ from its cost in G, we use the notation $\mathrm{cost}^z_{\mathrm{wt}}(C, \mathcal{K})$ and $\mathrm{cost}^z(C, G)$ respectively.

Lemma B.7. *The point set P and the distance function d in Algorithm 1 define a $(1 + \epsilon)^2$-metric space.*

Proof. Since the distance function $\mathrm{d}(\cdot, \cdot)$ is zero at a point and itself, nonnegative, and symmetric, it remains to verify that it satisfies Definition 1.1. Consider three points $\tau(x), \tau(y), \tau(z) \in P$. The distance $\mathrm{d}(\tau(x), \tau(y))$ is set and maintained as $\min(\delta_x(y), \delta_y(x))$, rounded up to the nearest power of $(1 + \epsilon)$, and so $\mathrm{d}(\tau(x), \tau(y)) \leq (1 + \epsilon) \min(\delta_x(y), \delta_y(x))$. Hence, the distance $\mathrm{d}(\tau(x), \tau(y))$ is upper bounded as follows:

$$\begin{aligned}
\mathrm{d}(\tau(x), \tau(y)) &\leq (1 + \epsilon) \min(\delta_x(y), \delta_y(x)) \leq (1 + \epsilon)^2 \operatorname{dist}(x, y) \\
&\leq (1 + \epsilon)^2 \big(\operatorname{dist}(x, z) + \operatorname{dist}(z, y)\big) \\
&\leq (1 + \epsilon)^2 \big(\min(\delta_x(z), \delta_z(x)) + \min(\delta_z(y), \delta_y(z))\big) \\
&\leq (1 + \epsilon)^2 \big(\mathrm{d}(\tau(x), \tau(z)) + \mathrm{d}(\tau(z), \tau(y))\big).
\end{aligned}$$

The second inequality is due to Lemma B.3, and the third inequality follows because $\operatorname{dist}(\cdot, \cdot)$ satisfies the triangle inequality in the graph G. The fourth inequality holds since the approximate distance

estimates from Lemma B.3 never underestimate the actual distances in G. The final inequality follows from the fact that the distances $\mathrm{d}(\cdot,\cdot)$ are rounded upwards. Therefore, the maintained space (P, d) satisfies Definition 1.1 with $\rho = (1+\epsilon)^2$, which concludes the claim. □

Lemma B.8. *For every vertex $s \in S$, its corresponding weight $\mathrm{wt}(\tau(s))$ in P is always equal to $|\sigma^{-1}(s)|$.*

Proof. Whenever the preimage $\sigma^{-1}(s)$ of a vertex $s \in V$ is modified, the reduction algorithm adds the vertex s to the set R_S in Line 14 of Algorithm 1. In turn, the weight $\mathrm{wt}(\tau(s))$ is updated to $|\sigma^{-1}(s)|$ in Line 17. □

Lemma B.9. *For every pair of vertices $x, y \in S$, it always holds that:*

$$\mathrm{dist}(x,y) \;\le\; \mathrm{d}(\tau(x),\tau(y)) \;\le\; (1+\epsilon)^2\,\mathrm{dist}(x,y).$$

Proof. The proof follows from that of Lemma B.7. □

B.2.1 Approximation Ratio

The approximation ratio analysis is similar to that in Section 6.2.1 of Cruciani et al. (2026), using the auxiliary Lemmas B.8 and B.9.

Lemma B.10. *Consider a γ-approximate solution C to the weighted (k,z)-clustering instance $\mathcal{K} = (P,k,z,\mathrm{wt},\mathrm{d},(1+\epsilon)^2)$. Then it holds that:*

$$\mathrm{cost}^z(C,G) \;\le\; 2^{z-1}\cdot(\alpha \mathit{OPT} + \gamma \mathit{OPT}_{\mathcal{K}}).$$

Proof. The proof follows from that of Lemma 6.10 in Cruciani et al. (2026). □

Lemma B.11. *For the optimal (k,z)-clustering cost of $\mathcal{K} = (P,k,z,\mathrm{wt},\mathrm{d},(1+\epsilon)^2)$ it holds that:*

$$\mathit{OPT}_{\mathcal{K}} \;\le\; \left(4(1+\epsilon)^2\right)^z \cdot (\alpha \mathit{OPT} + \mathit{OPT}).$$

Proof. The proof follows from that of Lemma 6.11 in Cruciani et al. (2026). □

Lemma B.12 (Lemma 6.8 in Cruciani et al. (2026))**.** *Any γ-approximate solution to the weighted (k,z)-clustering instance $\mathcal{K} = (P,k,z,\mathrm{wt},\mathrm{d},(1+\epsilon)^2)$ is also an $O(a\gamma)$-approximate solution to the (k,z)-clustering instance (G,k,z).*

Proof. The proof follows from Lemmas B.10 and B.11. □

Corollary B.13. *After an adversarial update, the cost of the (k,z)-clustering solution C is at most $O(a \cdot \mathit{OPT})$.*

Proof. The maintained (k,z)-clustering solution C is the output of the distance-decrease local search algorithm $\mathcal{LS}$ of Theorem 1.4 on the weighted (k,z)-clustering instance $\mathcal{K} = (P,k,z,\mathrm{wt},\mathrm{d},(1+\epsilon)^2)$. Since by Theorem 1.4 we have $\mathrm{cost}^z_{\mathrm{wt}}(C,\mathcal{K}) = O(1)\cdot \mathrm{OPT}_{\mathcal{K}}$, by Lemma B.12 it follows that $\mathrm{cost}^z(C,G) = O(\alpha)\cdot \mathrm{OPT}$. □

B.2.2 Total Update Time and Proof Completion of Theorem B.4

Lemma B.14. *The total update time of the incremental reduction Algorithm 1 is:*

$$|\sigma_{max}|\cdot m^{1+o(1)}\log W \;+\; O\left(\sigma_{chg}\cdot k|\sigma_{max}|^2\,(\log nW)^3\right).$$

Proof. Regarding the update time of the reduction Algorithm 1, we provide an upper bound on the total update time due to all the incremental $(1+\epsilon)$-approximate SSSP algorithms and the distance-decrease local search algorithm $\mathcal{LS}$ of Theorem 1.4.

Based on Lemma B.3, the total update time of a single incremental $(1+\epsilon)$-approximate SSSP algorithm is $m^{1+o(1)}\log W$. Algorithm 1 initializes a new incremental $(1+\epsilon)$-approximate SSSP algorithm only when a new vertex is added to the (α,β)-bicriteria approximate solution S. Hence,

the total update time of all incremental $(1+\epsilon)$-approximate SSSP algorithms is bounded by $|\sigma_{\max}| \cdot m^{1+o(1)} \log W$. Regarding the total update time of the local search algorithm $\mathcal{LS}$, consider the two types of adversarial updates:

- Consider an adversarial edge insertion into the graph G. For every pair of vertices $x, y \in S$ such that the distance estimate $\delta_x(y)$ decreases by a factor of at least $(1+\epsilon)$, the reduction Algorithm 1 adds the pair (x, y) to the set Δ_S. In turn, a batch B of distance decreases is passed to the distance-decrease local search algorithm $\mathcal{LS}$. Based on Theorem 1.4, the total update time incurred by $\mathcal{LS}$ under adversarial edge insertions is:

$$O\left(k|P|^2 \left(\log |P|M\right)^3\right) \;=\; O\left(k|S|^2 \left(\log |S|M\right)^3\right) \;=\; O\left(k|\sigma_{\max}|^2 \left(\log nW\right)^3\right),$$

  where $M = O(nW)$, since W is the maximum weight of an edge and nW is the maximum possible distance between any two vertices in the graph.

- Consider a batch of modifications to the (α, β)-bicriteria approximate assignment σ. Every vertex $s \in S$ for which the preimage $\sigma^{-1}(s)$ is modified is added to the set R_S. By Definition B.6, the local search algorithm $\mathcal{LS}$ is restarted at most σ_{chg} times during all assignment modifications. Therefore by Theorem 1.4, the total time incurred by $\mathcal{LS}$ under assignment modifications is:

$$O\left(\sigma_{\text{chg}} \cdot k|S|^2 \log |S|M\right) \;=\; O\left(\sigma_{\text{chg}} \cdot k|\sigma_{\max}|^2 \log nW\right).$$

Consequently, since $\mathcal{LS}$ is restarted σ_{chg} times and the analysis for adversarial edge insertions applies to a single execution of $\mathcal{LS}$, the total update time due to the distance-decrease local-search algorithm $\mathcal{LS}$ is:

$$O\left(\sigma_{\text{chg}} \cdot k|\sigma_{\max}|^2 \left(\log nW\right)^3\right).$$

□

Proof of Theorem B.4. Consider the incremental reduction Algorithm 1 and its analysis. The result then follows by Corollary B.13 and lemma B.14. □

B.3 Putting Everything Together

Recall that we combine our incremental reduction algorithm from Theorem B.4 with the incremental bicriteria approximation algorithm by Cruciani, Forster, and Skarlatos Cruciani et al. (2026). The next lemma shows that the distance-decrease local search algorithm $\mathcal{LS}$ is restarted at most $\tilde{O}(1)$ times in total.

Lemma B.15. *Consider the (α, β)-bicriteria approximate assignment σ which is the output of the algorithm from Theorem 1.1 in Cruciani et al. (2026) and the input of Algorithm 1. Then it holds that $\sigma_{chg} = O(\log n \cdot \log_{1+\epsilon} nW)$.*

Proof. In Lemma 6.13 from Cruciani et al. (2026), the authors upper bound a value $\sigma_{\text{inc}}(V)$ which is always at most σ_{chg}. However, the proof of Lemma 6.13 already bounds the number of times that $\sigma(\cdot)$ changes; therefore, the claim follows from the same proof in Cruciani et al. (2026). □

Theorem 1.5. *There is a randomized incremental algorithm for the (k, z)-clustering problem that, given a weighted undirected graph $G = (V, E, w)$ subject to edge insertions, an integer $k \geq 1$, and a constant $z \geq 1$, maintains with high probability an $O(1)$-approximate (k, z)-clustering solution with a total update time of $\tilde{O}(km^{1+o(1)} + k^3)$.*

Proof. Based on the proof of Theorem 1.2 in Cruciani et al. (2026), the total update time required to maintain a $(O(1), O(\log^3 n \log_{1+\epsilon} nW))$-bicriteria approximate assignment σ under adversarial edge insertions is $\tilde{O}(m^{1+o(1)})$, and additionally $|\sigma_{\max}| = O(k \log^3 n \log_{1+\epsilon} nW)$. Furthermore by Lemma B.15, we have $\sigma_{\text{chg}} = O(\log n \log_{1+\epsilon} nW)$. Therefore by Theorem B.4, the total update time to maintain the $O(1)$-approximate (k, z)-clustering solution is $\tilde{O}(km^{1+o(1)} + k^3)$, as desired. □

C Experimental Evaluation

We complement our theoretical results with an experimental evaluation of our distance-decrease local search algorithm $\mathcal{LS}$ against static local search. As the static algorithm, we use an implementation of FastPAM1 by Schubert and Rousseeuw Schubert & Rousseeuw (2021) with random k initialization, which performs the best improving swap in the entire swap table at every step. We focus our evaluation on the k-median objective and perform every strictly improving swap, i.e., we do not employ a swap-gain threshold. The dynamic range data structures used to implement the dynamic swap table of $\mathcal{LS}$ are segment trees with lazy propagation, as described in our theoretical analysis.

C.1 Distance-decrease model

We realize distance decreases through edge insertions in an incremental graph, where vertices represent points and shortest-path distances induce the metric between them. An edge insertion may decrease the distances between many pairs of vertices simultaneously. Our evaluation focuses on the local-search component only rather than on the computation of shortest-path distances. We therefore exclude all graph-specific shortest-path computations from the reported running times. All local search strategies have access to an exact all-pairs shortest-path (APSP) distance matrix. After each edge insertion, we first fully update this matrix and provide all resulting distance decreases to the local search algorithm. The reported running times thus measure only the work required to construct or maintain the swap table, identify improving swaps, perform them, and restore local optimality. For all experiments, shortest-path distances and distance decreases are computed exactly. Additionally, we completely repair the APSP distance matrix before updating the dynamic swap table of $\mathcal{LS}$.

C.2 Local Search Strategies

As mentioned before, we compare our incremental algorithm $\mathcal{LS}$ against a static approach. However, there is another differentiation to make: In optimization, a common approach for solving a modified instance is to *warm-start* the static algorithm using the solution obtained for the previous instance Yildirim & Wright (2002). A warm start preserves the previously computed solution, but not the algorithmic state used to obtain it. In contrast, a dynamic algorithm also maintains auxiliary data structures across updates and updates them incrementally. This means we are looking at the following local search strategies for our implementations:

1. **FastPAM1-Cold**: Recompute a locally optimal solution from scratch after every edge insertion.
2. **FastPAM1-Warm**: Use the previously computed set of centers as the initial solution after an insertion, but reconstruct auxiliary data structures such as the swap table.
3. $\mathcal{LS}$: Maintain the solution and dynamic swap table across insertions and continue local search as described in Section A.3.

C.3 Research Questions

The main research questions for this evaluation are:

1. How does the solution quality obtained by $\mathcal{LS}$ compare to that of FastPAM1-Cold when recomputed from scratch after a sequence of updates?
2. How does the update time of $\mathcal{LS}$ compare to FastPAM1-Warm?

FastPAM1-Warm and $\mathcal{LS}$ differ only in how the swap table is obtained immediately after an edge insertion. FastPAM1-Warm reconstructs the table from scratch, whereas $\mathcal{LS}$ updates it dynamically. Both approaches start from the same previously computed center set and operate on the same swap table. After the first accepted swap, $\mathcal{LS}$ also reconstructs the swap table statically. Consequently, our solution quality comparison focuses on FastPAM1-Cold, while FastPAM1-Warm serves as the static baseline for update time.

C.4 General Setup

All algorithms presented in this evaluation were implemented in C++20 as part of the Graph-Based Benchmark Suite Dhulipala et al. (2020). The code was compiled using g++ 12 through Bazel, using optimization level `-O2` and architecture-specific optimization enabled through `-march=native`. All experiments were conducted under Debian 12 bookworm on a server equipped with two AMD EPYC 7763 64-core processors and approximately 4 TB of RAM available, accessed through a virtual machine. Unless stated otherwise, each experiment was repeated three times and the median running time is reported. The timeout for the experimental sweep across all instances was set to 2.5 hours per run.

C.5 Datasets

We evaluate our algorithms on two real-world temporal networks, *DBLP* and *Stack Overflow*, obtained from the network collection of Benson et al. Benson et al. (2018). DBLP is a co-authorship graph, where vertices represent authors and edges represent collaborations. Stack Overflow is a co-occurrence graph, where vertices correspond to tags and edges correspond to the appearance of two tags in the same thread. For both graphs, since they are unweighted, we ignore multi-edge insertions. Consequently, the resulting update sequences consist exclusively of genuine edge insertions. To study these networks at different stages of their evolution, we construct several snapshots of each dataset at different points in time. For a specific cut-off time T, we first construct the graph containing all edges observed up to T and restrict it to its largest connected component. We then fix this vertex set for the remainder of the experiment. Starting from this initial graph, we process subsequent first-time co-occurrences in chronological order and retain an insertion only if both endpoints belong to the fixed vertex set. This construction avoids introducing isolated vertices corresponding to entities that have not yet appeared, while preserving the temporal order of the evolution of the selected network. For DBLP, we use snapshots corresponding to the years 1977, 1983, 1984, and 1990. For Stack Overflow, we use snapshots after approximately 5%, 40%, and 80% of its temporal evolution. Table 2 summarizes the resulting initial graphs:

Table 2: Initial graph sizes of the temporal snapshots used in our experiments. The density d is $\frac{m}{n}$.

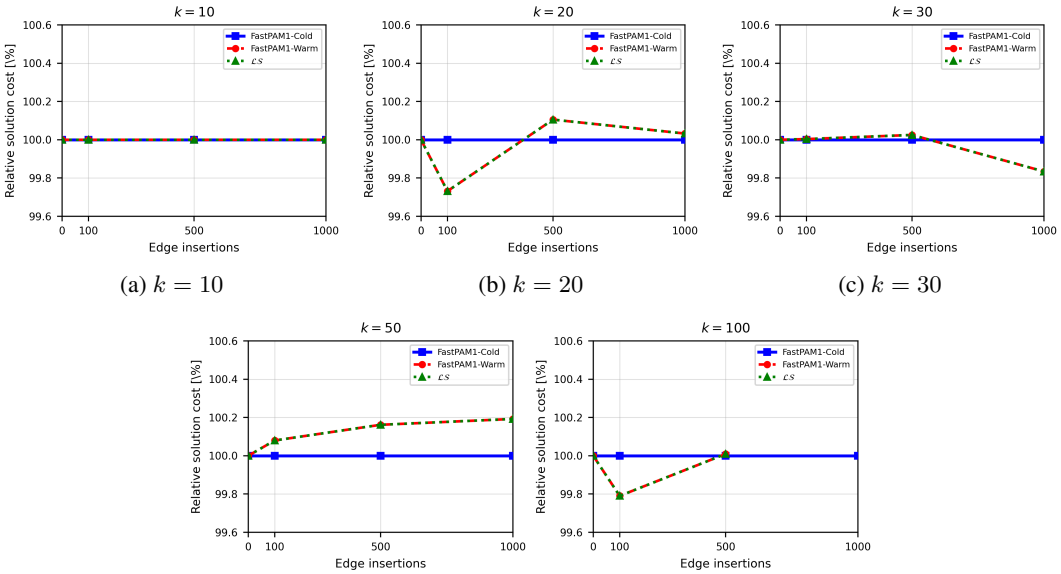
Dataset	Cut-Off	n	m	d
DBLP	1977	5,263	10,319	1.96
DBLP	1983	16,134	33,413	2.07
DBLP	1984	19,432	40,425	2.08
DBLP	1990	55,067	122,282	2.22
Stack Overflow	5%	22,126	520,820	23.54
Stack Overflow	40%	35,397	2,179,797	61.58
Stack Overflow	80%	45,497	3,505,795	77.06

C.6 Update Sequences and Number of Centers

For each snapshot, we evaluate prefixes of the subsequent update sequence containing 100, 500, and 1000 edge insertions. Thus, all algorithms evaluated for a given snapshot and update count operate on exactly the same sequence of graph states. Since it is not feasible to execute a fresh Cold Start after every individual update, we execute FastPAM1-Cold only at checkpoints after 100, 500 and 1000 updates respectively. We consider $k \in \{10, 20, 30, 50, 100\}$ for the number of centers. Since our graph snapshot contain up to 55000 vertices, $k = 100$ is already close to $\sqrt{n}$ which is typically used as an upper bound in k-clustering applications.

D EXPERIMENTAL RESULTS

D.1 SOLUTION QUALITY

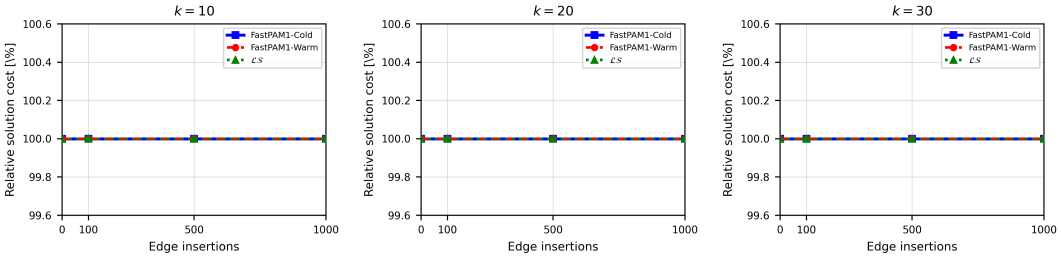


(a) $k = 10$ (b) $k = 20$ (c) $k = 30$

(d) $k = 50$ (e) $k = 100$

Figure 3: Solution quality on DBLP 1983 relative to FastPAM1-Cold. FastPAM1-Cold is normalized to 100%; lower values correspond to better solutions. If a data point is missing, it means that the corresponding algorithm timed out.

We compared the solution quality of $\mathcal{LS}$ relative to a fresh cold start of FastPAM1. As mentioned before, FastPAM1-Warm is expected to yield identical solutions as $\mathcal{LS}$. Figure 3 shows the relative solution qualities on the DBLP 1983 snapshot. We observe that for all values of k, the solution qualities are very similar, differing by at most about 0.25%. Sometimes, FastPAM1-Warm and $\mathcal{LS}$ produce slightly better solutions than FastPAM1-Cold and sometimes their solutions are slightly worse. In terms of absolute values, the reported k-median objective values on DBLP 1983 range in the tens of thousands. For example, for the data points in Figure 3 subfigure (b) at 100 insertions, FastPAM1-Cold reported 61087 as the solution while FastPAM1-Warm and $\mathcal{LS}$ reported 60923. For $k = 100$, both FastPAM1-Warm and $\mathcal{LS}$ timed out for the 1000 insertions run. One thing to highlight is that the initially found local optima have shown to be very robust: Across all runs on DBLP 1983, which amount to 7000 insertions (1600 each for $k = 10, 20, 30, 50$) and 600 for $k = 100$), about 98.4% of insertions did not trigger any improving swap.



(a) $k = 10$ (b) $k = 20$ (c) $k = 30$

Figure 4: Solution quality on Stack Overflow 40% relative to FastPAM1-Cold. FastPAM1-Cold is normalized to 100%; lower values correspond to better solutions.

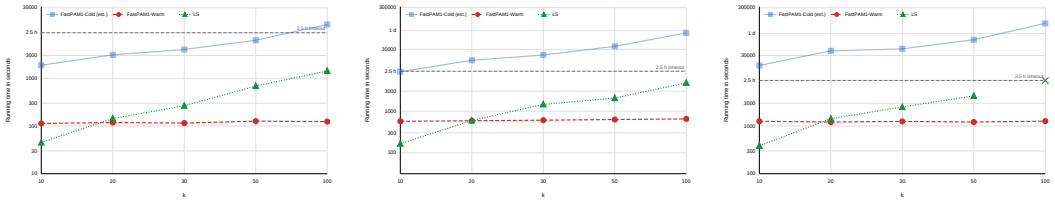
For the Stack Overflow snapshot graphs, the solution quality was identical across all runs for any k and number of updates. Figure 4 shows an example for $k = 10, 20, 30$ of the 40% snapshot. For this instance, the ratio of edge insertions to improving swaps is even more one-sided: Not a single one of the 8000 insertions processed across all runs triggered a single solution changing swap.

D.2 Running Times

We also compared the running times of our algorithms on the same instances. As mentioned before, it was not feasible to execute FastPAM1-Cold after every update due to time constraints. It was therefore executed only at certain checkpoints after 100, 500 and 1000 updates. Since FastPAM1-Cold starts from a fresh random initial solution and performs the complete local search from scratch, we estimate the running time of applying FastPAM1-Cold after every insertion from the measured cost of individual Cold Start computations. The blue curves representing FastPAM1-Cold in the running time plots therefore show estimated rather than measured running times. The horizontal dashed line indicates the time limit of 2.5 hours used in our experiments.

Across the different graph instances, we observe several general trends. The running time of $\mathcal{LS}$ increases considerably with k. This is expected since $\mathcal{LS}$ incrementally maintains center-dependent data structures and therefore has to update more state as the number of centers increases. In contrast, the running time of FastPAM1-Warm is largely insensitive to k in our experiments. Its dominant cost is rebuilding the local search data structures after every insertion, which has to be performed regardless of whether the update significantly changes the current solution. The estimated running time of FastPAM1-Cold also increases with k. Unlike FastPAM1-Warm, it does not start from the previously computed local optimum but instead starts from a new random solution and performs the complete local search. As a result, the k-dependent costs of the local search procedure have to be paid for every update, affecting the running time.

A second important factor for the performance of $\mathcal{LS}$ is how strongly an edge insertion affects the maintained local search data structures. An insertion may decrease one or more shortest-path distances, which in turn can require updates to the swap information maintained by $\mathcal{LS}$. The resulting amount of work does not depend solely on the number of distance decreases. A single changed distance may affect many swap computations, whereas other distance changes may have little effect on the maintained state. Nevertheless, a large number of distance decreases increases the potential for such changes to propagate into the swap table. FastPAM1-Warm, on the other hand, rebuilds its local search data structures after every insertion and therefore cannot exploit updates that affect only a small part of the maintained state.

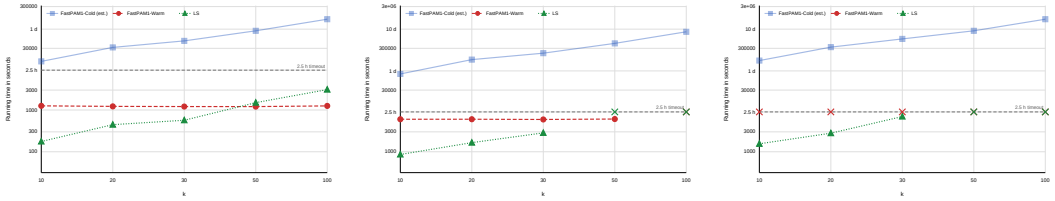


(a) 100 insertions (b) 500 insertions (c) 1000 insertions

Figure 5: Running times on DBLP 1983. FastPAM1-Cold is an estimate. The horizontal dashed line marks the 2.5 hour timeout. The y-axis uses a logarithmic scale.

Figure 5 shows the running time comparison on the DBLP 1983 snapshot. Here, we are starting to see an increase in relative performance of $\mathcal{LS}$ compared to FastPAM1-Warm. For $k = 10$, $\mathcal{LS}$ is clearly faster while for $k = 20$ the running times of both approaches are relatively close. For larger values of k, FastPAM1-Warm again becomes considerably faster and for $k = 100$ and 1000 updates $\mathcal{LS}$ even reaches the timeout limit. Interestingly, the absolute number of distance decreases is higher than on DBLP 1977, with approximately 45000 to 55000 decreases per insertion. However, DBLP 1983 is also substantially larger, containing 16134 vertices compared to 5263 vertices in DBLP 1977. Relative to the number of vertex pairs, an insertion therefore affects a considerably smaller fraction of the shortest-path metric. This is consistent with our intuition that the performance of

$\mathcal{LS}$ does not depend on the absolute number of distance decreases alone, but rather on how strongly these changes affect the maintained local search data structures.

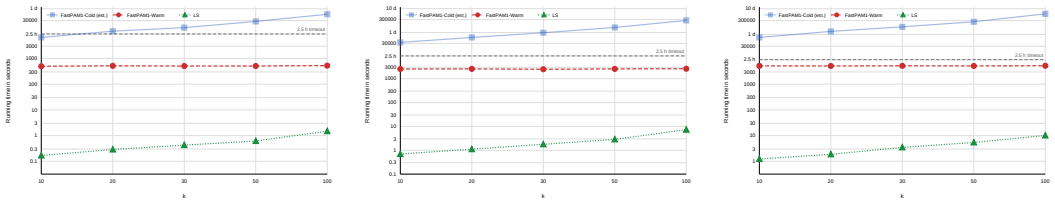


(a) 100 insertions (b) 500 insertions (c) 1000 insertions

Figure 6: Running times on DBLP 1990. FastPAM1-Cold is an estimate. The horizontal dashed line marks the 2.5 hour timeout. The y-axis uses a logarithmic scale.

The behavior changes more noticeably for DBLP 1990, shown in Figure 6. On this considerably larger snapshot, $\mathcal{LS}$ remains faster than FastPAM1-Warm for $k = 10$, 20 and 30, and the crossover between the two approaches is shifted towards larger values of k. This coincides with updates affecting a substantially smaller fraction of the shortest-path metric than on the earlier DBLP snapshots. Relative to the number of vertex pairs, the average number of distance decreases per insertion is approximately 15 times smaller than on DBLP 1977. Nevertheless, the advantage eventually disappears as k increases. For larger update sequences, we are starting to see timeouts for both approaches. At 1000 insertions, only $\mathcal{LS}$ manages to finish in time for $k = 10, 20, 30$ and times out for $k = 50, 100$ while FastPAM1-Warm could not finish in time for any value of k.

On the much denser Stack Overflow snapshots, the behavior changes substantially.



(a) 100 insertions (b) 500 insertions (c) 1000 insertions

Figure 7: Running times on Stack Overflow 40%. FastPAM1-Cold is an estimate. The horizontal dashed line marks the 2.5 hour timeout. The y-axis uses a logarithmic scale.

The different behavior is clearly visible for Stack Overflow 40%, shown in Figure 7. Again, $\mathcal{LS}$ is substantially faster than FastPAM1-Warm for every value of k. For 1000 updates, FastPAM1-Warm requires approximately 5100 to 5200 seconds, while $\mathcal{LS}$ requires only between approximately 1.2 and 10 seconds. As observed in the solution-quality experiments, none of the 8000 edge insertions processed across all runs on this snapshot triggered an improving swap. Hence, although updates do change shortest-path distances, the maintained local optimum and its associated local search state appear to be particularly robust on this instance. Figure 7 also demonstrates the infeasibility of executing FastPAM1-Cold for every update: Its estimated running time for $k = 100$ and 1000 insertions is more than a week.