Incremental (k, z)-Clustering on Graphs

Emilio Cruciani
European University of Rome, Italy

Sebastian Forster
Department of Computer Science, University of Salzburg, Austria

Antonis Skarlatos
Department of Computer Science, University of Warwick, Coventry, UK

Abstract

Given a weighted undirected graph, a number of clusters k, and an exponent z, the goal in the (k, z)-clustering problem on graphs is to select k vertices as centers that minimize the sum of the distances raised to the power z of each vertex to its closest center. This problem includes the well-known k-median ($z = 1$) and k-means ($z = 2$) clustering problems. In the dynamic setting, the graph is subject to adversarial edge updates, and the goal is to maintain explicitly an exact (k, z)-clustering solution in the induced shortest-path metric.

Prior works by Bhattacharya, Costa, Garg, Lattanzi, and Parotsidis [FOCS 2024] and by Bhattacharya, Costa, and Farokhnejad [STOC 2025] consider the dynamic (k, z)-clustering problem for point sets in metric spaces. These algorithms support adversarial point insertions and deletions under a model with access to pairwise distances. This model differs significantly from the dynamic graph setting, where no oracle access is given to pairwise distances and a single edge update can affect many distances – making these approaches inefficient when applied to graphs. While efficient dynamic k-center approximation algorithms on graphs exist [Cruciani, Forster, Goranci, Nazari, and Skarlatos, SODA 2024], to the best of our knowledge, no prior work provides similar results for the dynamic (k, z)-clustering problem.

As the main result of this paper, we develop a randomized incremental (k, z)-clustering algorithm that maintains with high probability a constant-factor approximation in a graph undergoing edge insertions with a total update time of $\tilde{O}(km^{1+o(1)} + k^{1+\frac{1}{\lambda}}m)$, where $\lambda \geq 1$ is an arbitrary fixed constant. Our incremental algorithm also achieves an amortized update time of $\tilde{O}(kn^{o(1)} + k^{1+\frac{1}{\lambda}})$ and consists of two stages. In the first stage, we maintain a constant-factor bicriteria approximate solution of size $\tilde{O}(k)$ with a total update time of $m^{1+o(1)}$ (independent of the parameter k) over all adversarial edge insertions. This first stage is an intricate adaptation of the bicriteria approximation algorithm by Mettu and Plaxton [Machine Learning 2004] to incremental graphs. One of our key technical results is that the radii in their algorithm can be assumed to be non-decreasing while the approximation ratio remains constant – a property that may be of independent interest.

In the second stage, we maintain a constant-factor approximate (k, z)-clustering solution on a dynamic weighted instance induced by the bicriteria approximate solution. For this subproblem, we employ a dynamic spanner algorithm together with a static (k, z)-clustering algorithm.

2012 ACM Subject Classification Theory of computation → Dynamic graph algorithms

Keywords and phrases (k, z)-clustering, k-median, k-means, dynamic graph algorithms

Digital Object Identifier 10.4230/LIPIcs.ICALP.2026.70

Category Track A: Algorithms, Complexity and Games

Related Version *Full Version*: `https://arxiv.org/abs/2602.08542`

Funding *Sebastian Forster*: This work is supported by the Austrian Science Fund (FWF): P 32863-N. This project has received funding from the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme (grant agreement No 947702).
Antonis Skarlatos: Antonis Skarlatos is funded by the European Union (ERC grant, DYNALP, 101170133). Views and opinions expressed are however those of the author(s) only and do not necessarily reflect those of the European Union or the European Research Council Executive Agency. Neither the European Union nor the granting authority can be held responsible for them.

53rd International Colloquium on Automata, Languages, and Programming (ICALP 2026).
Editors: Sayan Bhattacharya, Danupon Nanongkai, Michael Benedikt, and Gabriele Puppis;
Article No. 70; pp. 70:1–70:24

Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

1 Introduction

Clustering is a fundamental and well-studied problem in computer science with numerous applications across various domains [34, 29, 52, 35, 17]. Conceptually, clustering involves partitioning data points into groups such that points within the same group are more similar to each other than to those in different groups, according to some measure of similarity. Clustering algorithms optimize a given objective function, and for k-clustering problems the goal is to output a set of k representative points that minimize a k-clustering objective.

While the graph clustering toolkit for network analysis tasks (e.g., community detection) encompasses a wide range of clustering objectives suited to different scientific communities [48, 50, 26, 2], researchers have in particular performed experimental studies with k-clustering objectives on graphs [51, 49]. It has been observed in these experiments that (a) high computational cost and (b) the sensitivity of distances to single edge updates pose a challenge when employing k-clustering in graphs [49] (see also [2, Section 2.3]), which motivates further algorithm design in this direction.

Among the most widely studied k-clustering objectives are *k-median* and *k-means*. The k-median objective is the sum of distances from each point to its closest center, while the k-means objective is the sum of squared distances from each point to its closest center. A generalization of both is the (k, z)-clustering objective, which is the sum of distances raised to the power z from each point to its closest center. Hence given a metric space $(\mathcal{X}, \mathrm{dist})$, a set of points $P \subseteq \mathcal{X}$, and a positive integer $k \leq |P|$, the goal in the (k, z)-clustering problem is to output a subset $S \subseteq P$ of at most k points, referred to as *centers*, such that the value $\sum_{p \in P} \mathrm{dist}(p, S)^z$ is minimized. Observe that the k-median clustering problem corresponds to $z = 1$, and the k-means clustering problem corresponds to $z = 2$.

The k-median clustering problem is known to be NP-hard in arbitrary metric spaces [40]. The first approximation algorithms leveraged tree embeddings and polynomial-time algorithms for the tree setting [53], achieving an approximation ratio of $O(\log n \log\log n)$ [6, 7]; later refined to $O(\log k \log\log k)$ [15]. The first constant-factor approximation was established in [16] using LP rounding and was later improved to $3 + \epsilon$ through local search [5, 28, 19]. Subsequently, approximation ratios below 3 were achieved [43, 13, 19], with the best-known approximation ratio currently standing at $2+\epsilon$ [18]. Meanwhile, the best known hardness result rules out an approximation factor less than $1 + 2/e \approx 1.74$ [36]. Many of the aforementioned algorithmic approaches carry over to k-means and (k, z)-clustering with worse approximation factors. For the k-means clustering problem, the best known approximation ratio is $5 + \epsilon$ [14] and the best known hardness result rules out an approximation factor less than $1 + 8/e \approx 3.94$ [36, 3].

Efficient k-clustering algorithms. Fast algorithms have been proposed for the k-median clustering problem in arbitrary metric spaces with oracle access to the distances between points. Jain and Vazirani [37] developed a deterministic 6-approximation algorithm that runs in $\tilde{O}(n^2)$ time. Subsequently, Mettu and Plaxton presented another approach for obtaining a constant-factor approximation in $\tilde{O}(n^2)$ time [46]. Later, they presented a randomized $O(1)$-approximation algorithm with a more efficient running time of $\tilde{O}(nk)$ [47]. As noted by Huang and Vishnoi [33], the algorithm in [47] can be easily generalized to the (k, z)-clustering problem.

Graph setting. In the *graph setting*, the (k, z)-clustering problem is defined as before, where the metric space is induced by the shortest-path distances in a weighted undirected graph with n vertices, m edges, and maximum edge weight W. Here, no oracle access is available,

and distance computations must be performed explicitly by the algorithm if needed. As a consequence, although algorithms designed for point sets can be applied in the graph setting, this approach leads to inefficient algorithms. Thorup [54] addressed this issue for the k-median clustering problem by developing a randomized $(12+o(1))$-approximation algorithm that runs in $\tilde{O}(m)$ time.[1] As noted in [42], it seems plausible that the approach of Thorup [54] generalizes to (k,z)-clustering using recent insights on the primal-dual method used as a subroutine. Very recently, alternative near-linear and almost-linear time constant-factor approximation algorithms based on a greedy approach [42] and local search [38] have been developed. It is worth noting that the fastest constant-factor approximation algorithms for Euclidean (k,z)-clustering can be obtained by running the state-of-the-art graph clustering algorithms on a sparse metric spanner [30]. The graph setting has also received notable attention for the k-center clustering problem [54, 23, 1, 12, 39].

Dynamic k-clustering algorithms. In recent years, the rapid growth and continuous change of data have led to strong interest in developing dynamic k-clustering algorithms [9, 11, 41, 10, 21, 20, 24, 25]. The related work most similar to ours is by Cruciani, Forster, Goranci, Nazari, and Skarlatos [21], who study the k-center clustering problem on dynamic graphs under adversarial edge updates; they achieve for example an amortized update time of $kn^{o(1)}$ in the incremental setting. Since the shortest-path metric of a graph induces a metric space, it may seem natural to apply algorithms designed for dynamic point sets in metric spaces [9, 11, 10]. However, as noted in [21], this approach leads to inefficient algorithms as discussed in the next paragraph.

Comparison of dynamic graphs and dynamic point sets. Let us briefly highlight the key differences and challenges between the model with dynamic graphs, which we also consider in this paper, and the model with dynamic point sets in metric spaces [9, 11, 10]. In the model with dynamic point sets, the adversarial updates are point insertions or point deletions. On the other hand, in our model with dynamic graphs, the adversarial updates are edge insertions or edge deletions. This distinction introduces two main challenges, as explained in [21]: (i) there is no oracle access to all-pairs shortest-path distances, and (ii) a single edge update can affect multiple distances simultaneously, whereas a point update does not affect distances between previously inserted points. As a result, directly applying black-box algorithms designed for dynamic point sets to dynamic graphs is very inefficient.

To the best of our knowledge, no algorithms have been developed so far for the k-median, k-means, or their generalization (k,z)-clustering on graphs undergoing edge updates. For this reason, a natural question arises about the efficiency of dynamic algorithms for the (k,z)-clustering problem on dynamic graphs:

> Are there efficient constant-factor approximation algorithms for the (k,z)-clustering problem on graphs undergoing edge updates?

1.1 Our Contributions

In this paper, we answer this question in the affirmative for the incremental setting, in which the weighted undirected graph undergoes adversarial edge insertions. We believe that the incremental setting is particularly interesting and well-motivated from a practical point

[1] The notation $\tilde{O}(\cdot)$ hides polylogarithmic factors in nW.

of view, as there exist inherently incremental graphs in practice. For example, real-world graphs such as co-authorship networks are incremental, since the fact that two scientists have co-authored a research paper (almost) never changes over time.

Our central result is a highly efficient incremental algorithm – independent of the parameter k – that maintains a constant-factor *bicriteria approximate solution* of size $\tilde{O}(k)$ (see Definition 5), which is optimal up to polylogarithmic factors for the (k, z)-clustering problem. Throughout the paper, we assume an oblivious adversary for our randomized results. An oblivious adversary fixes the entire sequence of updates before the algorithm begins. Namely, the adversary cannot adapt the updates based on the (randomized) outputs of the algorithm during the execution.

▶ **Theorem 1.** *There is a randomized incremental algorithm for the (k, z)-clustering problem that, given a weighted undirected graph $G = (V, E, w)$ with maximum edge weight W subject to edge insertions, an integer $k \geq 1$, and constants $z \geq 1, \epsilon \in (0, 1)$, maintains with high probability:*

- *a $(O(1), O(\log^3 n \log_{1+\epsilon} nW))$-bicriteria approximate solution and*
- *a $(O(1), O(\log^3 n \log_{1+\epsilon} nW))$-bicriteria approximate assignment,*

and with high probability has an amortized update time of $\tilde{O}(n^{o(1)})$.

For clarity of presentation, in Theorem 1 we hide the dependence on z in the approximation ratio – which is $O((1+\epsilon)^{2z})$ – and the dependence on ϵ and W in the update time. By combining Theorem 1 with our incremental reduction algorithm (in Theorem 15), we obtain our main result.

▶ **Theorem 2.** *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 constants $z \geq 1, \lambda \geq 1$, maintains with high probability an $O(1)$-approximate (k, z)-clustering solution with a total update time of $\tilde{O}(km^{1+o(1)} + k^{1+\frac{1}{\lambda}}m)$ and an amortized update time of $\tilde{O}(kn^{o(1)} + k^{1+\frac{1}{\lambda}})$.*

For clarity of presentation, in Theorem 2 (our incremental algorithm) we hide the dependence on W inside the notation $\tilde{O}(\cdot)$, which we use to suppress polylogarithmic factors in nW.

2 Technical Overview

In the following, we give a brief overview of our main technical contributions for the incremental setting, where the input graph $G = (V, E, w)$ undergoes edge insertions sent by the adversary. The goal is to develop an efficient incremental constant-factor approximation algorithm for the (k, z)-clustering problem on graphs. Our incremental (k, z)-clustering algorithm consists of two subroutines. The first subroutine described in Section 4, is an incremental $(O(1), O(\log^3 n \log_{1+\epsilon} nW))$-bicriteria approximation algorithm for the (k, z)-clustering problem on graphs. The second subroutine described in Section 5, efficiently converts the previous bicriteria approximate solution into a constant-factor approximate (k, z)-clustering solution after every adversarial edge insertion.

2.1 Basis of Our Incremental Bicriteria Approximation Algorithm

The basis of our incremental $(O(1), O(\log^3 n \log_{1+\epsilon} nW))$-bicriteria approximation algorithm for the (k, z)-clustering problem is the static $(O(1), O(\log^2 n))$-bicriteria approximation algorithm by Mettu and Plaxton [47], henceforth referred to as the MP-bi algorithm. A high-level overview of the static MP-bi algorithm is presented in the next paragraph.

MP-bi algorithm. Let $U_0 = V$ be the whole vertex set. The MP-bi algorithm performs at most $t := O(\log \frac{n}{k})$ iterations, where at the i-th iteration:

1. A candidate set S_i is constructed by sampling a small subset of U_i, where $|S_i| = \tilde{O}(k)$.
2. The smallest radius ν_i is computed such that the ball B_i of radius ν_i around S_i contains a constant fraction β of vertices from U_i (i.e., $|B_i \cap U_i| \geq \beta |U_i|$).
3. The ball B_i is removed from U_i, and the MP-bi algorithm recurses on the remaining vertices $U_{i+1} := U_i \setminus B_i$.

In the incremental setting, the adversary inserts new edges into the graph; our aim is to design an incremental version of the MP-bi algorithm under adversarial edge insertions. We refer to the i-th iteration of the MP-bi algorithm as the i-th level. Thus, our goal is to incrementally maintain the data structures of the MP-bi algorithm at each level i.

2.1.1 Challenges in the Incremental Setting

The central step of the MP-bi algorithm is its second step, where the smallest radius ν_i is computed. Under edge insertions the distances are non-increasing, and so for a fixed radius ν_i, the balls B_i can only "receive" new vertices. In this case, whenever a ball B_i contains more than a β fraction of vertices from U_i, the radius ν_i should be decreased accordingly. The core issue however, is that the subsequent set U_{i+1} of remaining vertices can change in a *fully dynamic* manner, as follows:

- Vertices and edges can be removed from the set U_{i+1} by entering the ball at a previous level. This occurs when a newly inserted adversarial edge decreases the distance between the candidate set S_j and a vertex in U_{i+1} to at most ν_j, for some previous level $j < i + 1$.
- Vertices and edges can be added to the set U_{i+1} by leaving the ball at a previous level. This occurs when the radius ν_j is decreased accordingly because many vertices enter the ball B_j due to a newly inserted adversarial edge, for some previous level $j < i + 1$.

As a consequence, the current candidate set S_{i+1} which was sampled from the old set $U_{i+1}^{(\text{old})}$, may no longer be a good representative in the new set U_{i+1}. For this reason, a new candidate set S_{i+1} should be sampled from the modified set U_{i+1}, and the new radius ν_{i+1} should be computed. In other words, the first two steps of the MP-bi algorithm should be repeated at level $i + 1$. The second step of the MP-bi algorithm in the incremental setting can be implemented straightforwardly by maintaining the ball B_j and the radius ν_j at each level j, as follows:

- either by using a fully dynamic SSSP (single-source shortest paths) algorithm with a super-source attached to the candidate set S_j,
- or by restarting an incremental SSSP algorithm with a super-source attached to the candidate set S_j, whenever the set U_j is modified.

The first option of employing a fully dynamic SSSP algorithm has a prohibitive update time. The second option of employing an incremental SSSP algorithm depends on the total number of times any set U_j is modified, which is naively proportional to the number of distinct sequences of the radii ν_j. Therefore, we can infer that the total update time of the second option depends on the number of distinct sequences of the radii ν_j.

Number of distinct sequences of the radii. Even when the radii are powers of $(1 + \epsilon)$ and the number of levels t is $O(\log n)$, the number of distinct sequences of the radii ν_j is naively at least:

$$\Omega\left(\min\left((\log_{1+\epsilon} nW)^{\log n}, m\right)\right),$$

where W is the maximum weight of an edge, and thus nW is the maximum possible distance between two vertices in the graph. The first term is because the new radius ν_j may differ arbitrarily from the old radius $\nu_j^{(\text{old})}$ at level j, after repeating the second step of the MP-bi algorithm. The second term is because the sequence of the radii could change after each adversarial edge insertion.

For the second straightforward option, an incremental SSSP algorithm is restarted at most $t = O(\log n)$ times (for the t sets U_j) per distinct sequence of the radii. As a result, the total update time required for the second straightforward option is at least:

$$\Omega\left(\min\left((\log_{1+\epsilon} nW)^{\log n}, m\right)\right) \cdot \log n \cdot \text{ the running time of an SSSP algorithm.}$$

In other words, the total update time required for the second straightforward option is at least $\Omega(m \cdot (\log_{1+\epsilon} nW)^{\log n})$, which is excessively high.

2.2 Incremental Bicriteria Approximation Algorithm

In principle our approach follows the second option, which is to restart an incremental SSSP algorithm as many times as the number of distinct sequences of the radii ν_i. However, to overcome the issue of exponentially many distinct sequences of ν_i while ensuring constant-factor approximation, we introduce a refined and elegant solution for managing the radii ν_i. Our solution is a variant of the MP-bi algorithm and relies on two properties that are enforced on the radii.

The first property which is referred to as the *non-increasing property*, is that the value of each radius is non-increasing over time (i.e., the value of a specific ν_i either decreases or remains the same). By the non-increasing property, along with the facts that each radius is a power of $(1 + \epsilon)$ and that $t = O(\log n)$, we argue that there are at most $O\left(\log_{1+\epsilon} nW \cdot \log n\right)$ distinct sequences of radii. In turn, the update time becomes:

$$O\left(\log_{1+\epsilon} nW \cdot \log n\right) \cdot \log n \cdot \text{ the update time of an incremental SSSP algorithm.}$$

Nonetheless, since the non-increasing property is enforced, there is no guarantee on the approximation ratio. In particular, simultaneously enforcing the non-increasing property and maintaining $t = O(\log n)$ levels may lead to decisions that do not guarantee a constant-factor approximation. To that end, in order to maintain a constant-factor approximation, we establish the following crucial structural property of the MP-bi algorithm:

> The order of the values of the radii ν_i can be enforced to be non-decreasing (i.e., $\nu_0 \leq \nu_1 \leq \cdots \leq \nu_t$) without affecting the approximation ratio by more than a constant.

We refer to this second property as the *monotonicity property* of the radii. Using the monotonicity property, along with the fact that the distances are non-increasing in the incremental setting (under adversarial edge insertions), we show that the approximation ratio is constant. As a warm-up, we introduce the monotonicity property also in the static setting. Our static MP-bi variant enforces the monotonicity property from level 0 up to level t, as follows:

1. If $\nu_i \geq \nu_{i-1}$, then our MP-bi variant retains the radius ν_i.
2. If $\nu_i < \nu_{i-1}$, then our MP-bi variant replaces the value of the radius ν_i with the (updated) value of the radius ν_{i-1} from the previous level.

Notably, we prove that this modification preserves the constant-factor approximation of the MP-bi algorithm. Therefore, the core idea of our approach is to enforce the non-increasing property to achieve efficient update time, while relying on the monotonicity property to guarantee a constant-factor approximation.

2.2.1 Combining Non-Increasing and Monotonicity Properties in the Incremental Setting

In the incremental setting, we demonstrate how both the non-increasing and monotonicity properties can be enforced simultaneously, and how the monotonicity property can be leveraged to argue about the approximation ratio. Let $\nu_i^{(\text{old})}$ be the old radius at level i, and ν_i be the new radius at level i maintained by the incremental algorithm after an adversarial edge insertion.

Consider the set U_i at level i, and note that after an adversarial edge insertion some vertices may move closer to the corresponding candidate set S_i. In other words, the ball B_i may receive new vertices, causing $|B_i \cap U_i|$ to exceed $\beta|U_i|$ and the radius ν_i to be decreased accordingly (i.e., $\nu_i < \nu_i^{(\text{old})}$). Consider a vertex $v \in U_i$ that is within a distance of $\nu_i^{(\text{old})}$ from the candidate set S_i. Since under edge insertions the distances are non-increasing, the distance between v and S_i remains at most $\nu_i^{(\text{old})}$. However, it is possible that the distance between v and S_i is greater than the new radius ν_i, which implies that the vertex v does not belong to the ball B_i anymore. In this case, we add such a vertex v in a set Z, which we call the *leaking set*. Specifically, the leaking set Z contains all vertices that have exited a ball from a previous level and have *leaked* to a subsequent level.

Since the set U_{i+1} may be modified, we repeat the first two steps of the MP-bi algorithm at level $i+1$: we resample a fresh candidate set $\tilde{S}_{i+1}$, add it to the old candidate set S_{i+1}, and compute a *suggested* radius $\tilde{\nu}_{i+1}$ based on the second step of the MP-bi algorithm. The difference between our incremental algorithm and the straightforward option lies in the fact that the new radius ν_{i+1} is updated according to our MP-bi variant, while enforcing both the non-increasing and monotonicity properties:

1. If $\tilde{\nu}_{i+1} \geq \nu_{i+1}^{(\text{old})}$, then our incremental algorithm ignores the suggested radius $\tilde{\nu}_{i+1}$ and retains the old radius $\nu_{i+1}^{(\text{old})}$ (i.e., $\nu_{i+1} = \nu_{i+1}^{(\text{old})}$).
2. Else if $\tilde{\nu}_{i+1} < \nu_i$, then our incremental algorithm sets the new radius ν_{i+1} to the current radius ν_i of the previous level.
3. Otherwise, our incremental algorithm sets the new radius ν_{i+1} to the suggested radius $\tilde{\nu}_{i+1}$.

The first condition enforces the non-increasing property, while the first and second conditions enforce the monotonicity property. As a consequence of the non-increasing property, the number of distinct sequences of the radii ν_j is reduced to $O(\log n \cdot \log_{1+\epsilon} nW)$. Although we prove that the monotonicity property preserves the constant-factor approximation of the MP-bi algorithm in both the static and incremental settings, in the incremental setting we must also handle the leaking set, which poses an additional challenge. Recall that the leaking set contains vertices that have exited a ball at a previous level and have leaked into a subsequent level.

Role of the leaking set. Since the non-increasing property is enforced for efficiency reasons, in order to identify a β fraction of vertices from U_{i+1} (as dictated by the second step of the MP-bi algorithm), we may need to use some vertices from the leaking set Z. In the analysis, we prove that a suitable subset of the leaking set Z exists. Finding a ball B_{i+1} containing a

constant fraction β of vertices from U_{i+1} (second step of the MP-bi algorithm), and removing B_{i+1} from U_{i+1} (third step of the MP-bi algorithm) is important for maintaining at most $t = O(\log n)$ levels.

To appropriately provide an upper bound on the assigned cost of leaked vertices, we leverage the monotonicity property, as described in the next paragraph. The monotonicity property acts as a glue, ensuring both the non-increasing property (i.e., efficiency) and the constant-factor approximation are maintained.

Assigned cost of the leaking set. Roughly speaking, subsets of the leaking set Z are distributed among the sets U_i for which the corresponding suggested radius $\tilde{\nu}_i$ is greater than the current radius $\nu_i^{(\text{old})} = \nu_i$. For a fixed vertex $v \in U_i$ that belongs to the leaking set Z, there is a crucial structural observation to be made: The vertex v must have entered the set U_i from a previous level $j \leq i$. Thus, the vertex v is within a distance of at most $\nu_j^{(\text{old})}$ from the candidate set S_j. Observe also that the vertex v may be far from the candidate set S_i. Since under edge insertions the distance between v and S_j remains at most $\nu_j^{(\text{old})}$ and $\nu_j^{(\text{old})} \leq \nu_i^{(\text{old})} = \nu_i$ by the monotonicity property, we can charge for the vertex v a cost of ν_i.

Therefore using the monotonicity property, we can argue that each relevant vertex of any set U_i can be charged a cost of ν_i. This in turn helps us to upper bound the approximation ratio of our incremental algorithm, while ensuring that the value of the last level t remains $O(\log n)$ throughout the algorithm.

2.3 Maintaining (k, z)-Clustering on Bicriteria Approximate Solution

So far, we have described how our incremental algorithm maintains a constant-factor bicriteria approximate solution $S \subseteq V$ of size $\tilde{O}(k)$. The bicriteria approximate solution S induces a dynamic $(1+\epsilon)$-metric space (P, d)[2] by maintaining incremental $(1+\epsilon)$-approximate SSSP algorithms for each vertex in S (i.e., for each node in $P = S$). We "dynamize" a known argument to show that the optimal (k, z)-clustering solution on the $(1+\epsilon)$-metric space (P, d), with appropriate weights on the nodes in P, is an $O(1)$-approximation to the optimal (k, z)-clustering solution on the graph G.

Over the course of the adversarial edge insertions to the incremental graph G, both the set P and the weights of nodes in P change at most $\tilde{O}(1)$ times. The set P can change because the bicriteria approximate solution S may receive at most $\tilde{O}(k)$ new vertices due to an adversarial edge insertion into G. We emphasize that, for the efficiency of all the subroutines of our incremental (k, z)-clustering algorithm, it is important that the total number of vertices that ever belong to S (and in turn in P) is at most $\tilde{O}(k)$. Additionally, the distance between any pair of nodes in P may decrease at most $\tilde{O}(1)$ times, resulting in a total of $\tilde{O}(k^2)$ distance decreases.

The straightforward approach for maintaining an approximate (k, z)-clustering on (P, d) would be to recompute it from scratch in time $\tilde{O}(|P|k) = \tilde{O}(k^2)$ after every change to the set P or to the distance function d. This straightforward approach would result in a total update time of $\tilde{O}(km^{1+o(1)} + k^4)$ for our overall incremental (k, z)-clustering algorithm, which implies an amortized update time of $kn^{o(1)}$ for any $k \leq m^{1/3+o(1)}$. To obtain an update time close to k for the full parameter range, we instead employ an approach involving a dynamic

[2] Recall that in a ρ-metric space, the triangle inequality may be violated by a factor of ρ.

spanner algorithm. At a high level, since we achieve vertex sparsification via the bicriteria approximate solution S, we exploit it further by performing edge sparsification on top of the vertex-sparsified graph $H = (P, P \times P)$ whose edges are weighted by approximate distances in G.

Our approach. In more detail, we view (P, d) as a complete graph H with $\tilde{O}(k^2)$ edges, where the weight of an edge (x, y) in H corresponds to the approximate distance between x and y in G. We further bin the edges of H into weight classes and maintain a fully dynamic $O(\lambda)$-spanner [8] of size $\tilde{O}(k^{1+1/\lambda})$ on the edges of each weight class. The union $\tilde{H}$ of these spanners is a $O(\lambda)$-spanner for H of size $\tilde{O}(k^{1+1/\lambda})$; that is, $\tilde{H}$ is a subgraph of H (in terms of edges) in which the distance between every pair of vertices in H is preserved up to a multiplicative factor of $O(\lambda)$. This dynamic spanner algorithm [8] can deal with adversarial edge updates in polylogarithmic time per update.

Since the changes to the vertex set of H (and of $\tilde{H}$) occur only in $\tilde{O}(1)$ many batches, we can restrict the dynamic spanner algorithm to adversarial edge insertions by simply restarting it after every batch of vertex insertions. Since distances in G are non-increasing, each edge appears at most once in each weight class in between restarts and thus the total update time for maintaining the spanner $\tilde{H}$ is $\tilde{O}(k^2)$. As the final step of handling each of the m adversarial edge insertions to G, we compute a (k, z)-clustering solution on $\tilde{H}$ in time $\tilde{O}(k^{1+1/\lambda})$ from scratch using one of the state-of-the-art static (k, z)-clustering algorithms [54, 42, 38]. Overall, all these steps result in an incremental constant-factor approximation (k, z)-clustering algorithm with a total update time of $\tilde{O}(km^{1+o(1)} + k^{1+\frac{1}{\lambda}}m)$ and an amortized update time of $\tilde{O}(kn^{o(1)} + k^{1+\frac{1}{\lambda}})$, where $\lambda \geq 1$ is an arbitrary fixed constant.

3 Preliminaries

Consider a weighted undirected graph $G = (V, E, w)$ with nonnegative weights. We denote by $n := |V|$ the number of vertices, by $m := |E|$ the number of edges, and by W the maximum weight of an edge.[3] For a vertex $v \in V$ and a subset of vertices $S \subseteq V$, let $\mathrm{dist}(v, S) := \min_{s \in S} \mathrm{dist}(v, s)$ be the *distance* between the vertex v and the set S. We write $\mathrm{dist}_H(\cdot, \cdot)$ to explicitly denote the corresponding distance in a graph H; when the subscript is omitted, it refers to $\mathrm{dist}_G(\cdot, \cdot)$.

▶ **Definition 3** ((k, z)-clustering problem on graphs). *Given a weighted undirected graph $G = (V, E, w)$, an integer $k \geq 1$, and a constant $z \geq 1$, the goal in the (k, z)-clustering problem is to output a subset of vertices $S \subseteq V$ with size at most k (i.e., $|S| \leq k$) such that the value of $\mathrm{cost}^z(S) := \sum_{v \in V} \mathrm{dist}(v, S)^z$ is minimized.*

The (k, z)-clustering problem includes the problems of k-median ($z = 1$) and k-means ($z = 2$). A *(k, z)-clustering instance* is the triple consisting of the given input object, the integer parameter k, and the constant z. Any subset of vertices $S \subseteq V$ with size at most k is referred to as a *(feasible) solution*, and its cost is referred to as its *(k, z)-clustering cost*. Hence, the goal of the (k, z)-clustering problem can be rephrased as finding a solution S with minimum (k, z)-clustering cost $\mathrm{OPT} := \min_{S \subseteq V : |S| \leq k} \mathrm{cost}^z(S)$. The value OPT is called *optimal (k, z)-clustering cost*, and a solution that minimizes the cost is called an *optimal (k, z)-clustering solution*.

[3] We assume that the weights are at least 1 and are upper bounded by a polynomial in n. This can be achieved by rescaling all the weights.

▶ **Definition 4** (ρ-approximate (k,z)-clustering problem). *Given a (k,z)-clustering instance, the goal of the ρ-approximate (k,z)-clustering problem is to output a subset of vertices $S \subseteq V$ such that $|S| \leq k$ and $cost^z(S) \leq \rho \cdot OPT$.*

▶ **Definition 5** ((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:*

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

▶ **Definition 6** ((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 $\sigma : V \to V$ such that:*

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

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

Our incremental reduction algorithm in Section 5 deals with *weighted* (k,z)-clustering instances on a subgraph H, where H contains both edge and vertex weights.

▶ **Definition 7** (weighted (k,z)-clustering problem on graphs). *Given a weighted undirected graph $\hat{G} = (\hat{V},\hat{E},\hat{w})$ with vertex weights $\mathrm{wt}(\cdot)$, the corresponding weighted (k,z)-clustering instance is denoted by $\mathcal{K} \coloneqq (\hat{G},k,z,\mathrm{wt})$. The cost of a subset of vertices $C \subseteq \hat{V}$ is defined as $\mathrm{cost}^z_{\mathrm{wt}}(C) \coloneqq \sum_{v \in \hat{V}} \mathrm{wt}(v) \cdot \mathrm{dist}_{\hat{G}}(v,C)^z$. The optimal (k,z)-clustering cost of $\mathcal{K}$ is denoted by $OPT_{\mathcal{K}}$.*

Our incremental algorithms utilize the following incremental $(1+\epsilon)$-approximate single-source shortest paths (SSSP) algorithm. We note that similar results have been known implicitly for undirected graphs; these were randomized and assumed an oblivious adversary [32, 4, 45].

▶ **Lemma 8** (incremental $(1+\epsilon)$-approximate SSSP [44]). *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 the 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.

For a fixed a subset of vertices $S \subseteq V$ and a positive real number r, the set $\mathrm{Ball}(S,r) \coloneqq \{v \in V \mid \mathrm{dist}(v,S) < r\}$ denotes the *open ball* of radius r around S, and the set $\mathrm{Ball}[S,r] \coloneqq \{v \in V \mid \mathrm{dist}(v,S) \leq r\}$ denotes the *closed ball* of radius r around S. Throughout our algorithms, we use incremental $(1+\epsilon)$-approximate SSSP algorithms $\mathcal{A}$ (Lemma 8) with a super-source being a fixed set S, providing distance estimates $\delta_S(\cdot)$. Hence, we extend the definition of balls as follows:

- $\mathrm{Ball}(S,r,\delta_S) \coloneqq \{v \in V \mid \delta_S(v) < r\}$ to denote the *open ball* of radius r around S using the distance estimates $\delta_S(\cdot)$.
- $\mathrm{Ball}[S,r,\delta_S] \coloneqq \{v \in V \mid \delta_S(v) \leq r\}$ to denote the *closed ball* of radius r around S using the distance estimates $\delta_S(\cdot)$.

A dynamic algorithm has *amortized update time* $u(n,m)$ if its total time spent for processing any sequence of ℓ adversarial updates is bounded by $\ell \cdot u(n,m)$. Throughout this paper, whenever we write that an event holds *with high probability*, we mean that it holds with probability at least $1 - O(1/n^c)$ for some positive constant c. We also make use of the following concentration bounds in our proofs.

▶ **Lemma 9** ([22]). *Let $X_1, X_2, \cdots, X_i$ be i independent random variables with $0 \le X_j \le 1$, and let $X = \sum_{j=1}^{i} X_j$. Then for any $\delta \ge 0$ it holds that:*

$$\Pr\left(X \ge (1+\delta)\cdot \mathbb{E}[X]\right) \le \exp\left(-\frac{\delta^2}{2+\delta}\mathbb{E}[X]\right).$$

Furthermore, for any $\delta \in [0,1]$ it holds that:

$$\Pr\left(X \le (1-\delta)\cdot \mathbb{E}[X]\right) \le \exp\left(-\frac{\delta^2}{2}\mathbb{E}[X]\right).$$

4 Incremental Bicriteria Approximation for (k,z)-Clustering on Graphs

In this section, we develop an incremental bicriteria approximation algorithm for the (k,z)-clustering problem on graphs undergoing adversarial edge insertions. Our incremental bicriteria approximation algorithm, with high probability, achieves an amortized update time of $n^{o(1)}$ and maintains a constant-factor bicriteria approximate solution of size at most $O(k \log^3 n \log_{1+\epsilon} nW)$. A pseudocode of our incremental bicriteria approximation algorithm is provided in Algorithm 2, and the result is demonstrated in the following theorem.

▶ **Theorem 1.** *There is a randomized incremental algorithm for the (k,z)-clustering problem that, given a weighted undirected graph $G = (V,E,w)$ with maximum edge weight W subject to edge insertions, an integer $k \ge 1$, and constants $z \ge 1, \epsilon \in (0,1)$, maintains with high probability:*

- *a $(O(1), O(\log^3 n \log_{1+\epsilon} nW))$-bicriteria approximate solution and*
- *a $(O(1), O(\log^3 n \log_{1+\epsilon} nW))$-bicriteria approximate assignment,*

and with high probability has an amortized update time of $\tilde{O}(n^{o(1)})$.

Throughout the description and analysis of the incremental bicriteria approximation algorithm in Section 4, we fix a sufficiently large constant $\alpha \ge 1$ and a small constant $\beta \in (0,1)$. In the next two paragraphs, we first describe the information maintained by our incremental bicriteria approximation algorithm, and then outline the properties satisfied by the radii ν_i.

State of the incremental bicriteria approximation algorithm. Our incremental bicriteria approximation Algorithm 2 is structured into multiple *levels* starting from 0. The last level of the algorithm is denoted by t, and the value of t is upper bounded by $O(\log n)$ throughout the algorithm. At each level $i \in [0,t]$, the bicriteria approximation algorithm maintains an *execution set* U_i, a *candidate set* S_i, a *radius* ν_i, an incremental $(1+\epsilon)$-approximate SSSP algorithm $\mathcal{A}_i$ with a super-source attached to S_i, an *approximate ball* B_i, and a *leaking set* Z_i.

The incremental algorithm then maintains the bicriteria approximate solution $S := \bigcup_{i=0}^{t} S_i$, and an *assignment* $\sigma : V \to S$ that maps each vertex to a *candidate center* in S at a specified distance. The algorithm also uses the temporary leaking set Z, an auxiliary set that essentially stores the *pending leaked* vertices to be assigned to some leaking sets Z_i (see Lines 7, 42, 43, and 44 in Algorithm 2).

Properties of the radii. For efficiency reasons, the values of the radii ν_i (where $0 \leq i \leq t$) satisfy three properties over the course of the algorithm. (1) The first property is that the value of each radius ν_i is a power of $(1+\epsilon)$. Hence, the number of possible values of a radius ν_i is reduced to $\log_{1+\epsilon} nW$,[4] at the cost of an extra factor of $(1+\epsilon)$ in the approximation ratio. (2) The second property is that the value of each radius ν_i is non-increasing over time, which means that after an adversarial edge insertion, the value of a specific ν_i either decreases or remains the same. As a result, the number of distinct sequences of ν_i is reduced from $(\log_{1+\epsilon} nW)^t$ to $\log_{1+\epsilon} nW \cdot t$. We note that the value of the last level t may increase due to adversarial edge insertions, resulting in more radii ν_i. However, the incremental algorithm ensures that $t = O(\log n)$ throughout its execution.

Furthermore, in order to upper bound the cost of the bicriteria approximate solution, the values of the radii ν_i (where $0 \leq i \leq t$) satisfy also a third property over the course of the algorithm. (3) The third property is that the order of the values of ν_i is non-decreasing (i.e., $\nu_0 \leq \nu_1 \leq \cdots \leq \nu_t$). Let us provide a high-level justification for the third property: Consider a vertex $v \in U_i$ that is within a distance of ν_i from the candidate set S_i. Assume that after an adversarial edge insertion, the vertex v moves to the next execution set U_{i+1} and is not within a distance of ν_{i+1} from the next candidate set S_{i+1}. Since under edge insertions the distance between v and S_i remains at most ν_i and $\nu_i \leq \nu_{i+1}$ by the third property, the vertex v can be charged a cost of ν_{i+1}. Therefore using the third property, we can argue that a (leaked) vertex of any execution set U_j can be charged a cost of ν_j, which in turn helps us to upper bound the approximation ratio of our incremental algorithm. To sum up, our incremental bicriteria approximation algorithm ensures that the following three properties are satisfied throughout its execution:

Property 1 : The value of each radius ν_i is a power of $(1+\epsilon)$.

Property 2 : The value of each radius ν_i is non-increasing over time.

Property 3 : The order of the values of the radii ν_i is non-decreasing (i.e., $\nu_0 \leq \nu_1 \leq \cdots \leq \nu_t$).

4.1 Incremental Bicriteria Approximation Algorithm on Graphs

First we describe the preprocessing phase, and afterwards we describe how to handle adversarial edge insertions. A pseudocode of our incremental bicriteria approximation algorithm is provided in Algorithms 1 and 2.

▶ **Definition 10** (Retain). *Given a set A and a nonnegative integer λ, the set* $\text{Retain}(A, \lambda)$ *denotes an arbitrary subset of A with size* $\min(\lambda, |A|)$.

4.1.1 Preprocessing Phase

During the preprocessing phase, dynamic data structures are also employed, as described in Algorithm 1.

[4] Recall that W is the maximum weight of an edge, and thus nW is the maximum possible distance between two vertices in G.

Algorithm 1 Initialization of Incremental Bicriteria Approximation Algorithm.

Require: Graph G and positive integer k (problem input); positive constants α, β (MP-bi parameters); positive constant ϵ (MP-bi variant parameter, used also as SSSP approximation parameter)
Ensure: Bicriteria approximate solution S and bicriteria approximate assignment σ
▷ The algorithm has global access to all $U_i, S_i, \nu_i, \mathcal{A}_i, B_i, Z_i, Z, \sigma(\cdot)$
function INITIALIZE(G, k)
  $\nu_{-1} \leftarrow 0$
  $i \leftarrow 0$
  $U_0 \leftarrow V$
  while $|U_i| > \alpha k \log n$ **do**
    Construct S_i by sampling each $v \in U_i$ independently with probability $\min\left(\frac{\alpha k \log n}{|U_i|}, 1\right)$
    $\mathcal{A}_i.initialize(G, S_i)$ ▷ $\mathcal{A}_i$ is an incremental $(1+\epsilon)$-approximate SSSP algorithm, providing distance estimates $\delta_{S_i}(\cdot)$
    $\tilde{\nu}_i \leftarrow \min_{j \in \mathbb{Z}^{\geq 0}} \{(1+\epsilon)^j \mid |\text{Ball}[S_i, (1+\epsilon)^j, \delta_{S_i}] \cap U_i| \geq \beta |U_i|\}$
    $\nu_i \leftarrow \max(\tilde{\nu}_i, \nu_{i-1})$
    $B_i \leftarrow \text{Retain}(\text{Ball}[S_i, \nu_i, \delta_{S_i}] \cap U_i, \lceil \beta |U_i| \rceil)$
    For every $v \in B_i : \sigma(v) \leftarrow \text{argmin}_{u \in S_i} \delta_u(v)$
    $Z_i \leftarrow \emptyset$
    $U_{i+1} \leftarrow U_i \setminus B_i$
    $i \leftarrow i + 1$
  $t \leftarrow i$
  $S_t \leftarrow U_t$, $B_t \leftarrow U_t$, $Z_t \leftarrow \emptyset$, $Z \leftarrow \emptyset$, $\nu_t \leftarrow \nu_{t-1}$, $\forall v \in U_t : \sigma(v) \leftarrow v$
  $S \leftarrow \bigcup_{i=0}^{t} S_i$

Since we have to deal with adversarial edge insertions, an incremental $(1+\epsilon)$-approximate SSSP algorithm $\mathcal{A}_i$ (from Lemma 8) with a super-source attached to S_i is initialized for every level $i \in [0, t)$.[5] In Algorithm 1 of Algorithm 1, we denote by $\delta_u(v)$ the distance estimate from a candidate center $u \in S_i$ to a vertex $v \in V$. We remark that even though we cannot access all $\delta_u(v)$ directly, we can efficiently compute the closest candidate center $u \in S_i$ to a vertex $v \in V$ with respect to $\delta_{S_i}(\cdot)$. This follows from the fact that the incremental $(1+\epsilon)$-approximate SSSP algorithm $\mathcal{A}_i$ has access to the first edge of the corresponding approximate shortest path in $\tilde{O}(1)$ time; such an incremental SSSP $\mathcal{A}_i$ can be either the deterministic algorithm from Lemma 8 or a randomized algorithm against an oblivious adversary (implicitly given in [32, 4, 45]). Thus, the assignment in Algorithm 1 can be computed properly and efficiently.

▶ **Lemma 11.** *Immediately after the preprocessing phase, it holds that* $|U_{i+1}| = |U_i| - \lceil \beta |U_i| \rceil$ *for every level* $i \in [0, t)$.

4.1.2 Adversarial Edge Insertions

Before describing how our incremental bicriteria approximation algorithm for the (k, z)-clustering problem on graphs handles adversarial edge insertions, we define the *i-th valid radius* in order to guarantee that **Property 3** is satisfied. Recall that $\text{Ball}[S_i, r, \delta_{S_i}] := \{v \in V \mid \delta_{S_i}(v) \leq r\}$ denotes the closed ball of radius r around S_i using the distance estimates $\delta_{S_i}(\cdot)$.

▶ **Definition 12** (i-th valid radius). *For a fixed level $i \in [0, t]$, consider the i-th execution set U_i, the i-th candidate set S_i, the distance estimates $\delta_{S_i}(\cdot)$, and the $(i-1)$-th radius ν_{i-1}.*[6] *Let $\tilde{\nu}_i$ be the smallest power of $(1+\epsilon)$ such that $|\text{Ball}[S_i, \tilde{\nu}_i, \delta_{S_i}] \cap U_i| \geq \beta |U_i|$. Then $\hat{\nu}_i := \max(\tilde{\nu}_i, \nu_{i-1})$ denotes the i-th valid radius.*

[5] Namely, we introduce a fake root s and add an edge (s, v) of zero weight, for every candidate center $v \in S_i$. Then, the incremental $(1+\epsilon)$-approximate SSSP algorithm is executed with source s.
[6] For the definition of the 0-th valid radius, we set $\nu_{-1} = 0$.

When an edge (u^+, v^+) is inserted into the graph $G = (V, E, w)$ from the adversary, the incremental bicriteria approximation algorithm proceeds as follows. Initially, this adversarial edge insertion (u^+, v^+) is forwarded to all the incremental $(1+\epsilon)$-approximate SSSP algorithms $\mathcal{A}_i$, where $0 \leq i < t$. The algorithm then finds the smallest level $\ell \in [0, t]$ for which the (updated) ℓ-th valid radius $\hat{\nu}_\ell$ is less than the ℓ-th radius ν_ℓ (i.e., $\hat{\nu}_\ell < \nu_\ell$).[7] This step is implemented through the function First-Level-Decrease() (see Lines 9 and 26 in Algorithm 2).

Next, the function Update-DS-Rad-Decr(ℓ) is called within the function First-Level-Decrease(), which updates the data structures at level ℓ. In the function Update-DS-Rad-Decr(ℓ), the incremental algorithm updates the ℓ-th radius ν_ℓ to $\hat{\nu}_\ell$ and the ℓ-th approximate ball B_ℓ to an arbitrary subset of $\text{Ball}[S_\ell, \hat{\nu}_\ell, \delta_{S_\ell}] \cap U_\ell$ with size $\lceil \beta |U_\ell| \rceil$ (in Lines 3 and 5 of Algorithm 2). For each vertex $v \in B_\ell$, the algorithm updates its assignment $\sigma(v)$ to its nearest candidate center in S_ℓ with respect to $\delta_{S_\ell}(\cdot)$ (in Line 6 of Algorithm 2). The incremental algorithm then assigns the temporary leaking set Z as the union of the ℓ-th leaking set Z_ℓ and the vertices that were in the old ℓ-th approximate ball but are not in the updated ℓ-th approximate ball B_ℓ (in Line 7 of Algorithm 2). Eventually, the algorithm empties the ℓ-th leaking set Z_ℓ (in Line 8 of Algorithm 2).

Once the function First-Level-Decrease() has finished, the algorithm updates the $(\ell+1)$-th execution set $U_{\ell+1}$ to $U_\ell \setminus B_\ell$. In turn, the incremental algorithm must verify whether the value of the $(\ell+1)$-th radius $\nu_{\ell+1}$ should also decrease. Note that as the $(\ell+1)$-th execution set $U_{\ell+1}$ may change, the current $(\ell+1)$-th candidate set $S_{\ell+1}$ which was sampled from the old execution set $U_{\ell+1}$, may no longer be a good representative in the new $(\ell+1)$-th execution set $U_{\ell+1}$. To that end, the algorithm proceeds with the *Resampling Phase* in the updated $(\ell+1)$-th execution set $U_{\ell+1}$, as follows. Since ℓ denotes the smallest level at which the corresponding ℓ-th radius ν_ℓ is decreased, let $i \coloneqq \ell + 1$ represent the subsequent levels starting at level $\ell + 1$.

Resampling Phase. First, a supporting candidate set $\tilde{S}_i$ is constructed by sampling each vertex of the new i-th execution set U_i independently with probability $\min\left(\frac{\alpha k \log n}{|U_i|}, 1\right)$. Second, the incremental algorithm adds the i-th supporting candidate set $\tilde{S}_i$ to the i-th candidate set S_i, and restarts the i-th incremental $(1+\epsilon)$-approximate SSSP algorithm $\mathcal{A}_i$ with a super-source attached to the updated i-th candidate set S_i.[8] Next, the algorithm computes the i-th valid radius $\hat{\nu}_i$, using the updated i-th candidate set S_i and the updated distance estimates $\delta_{S_i}(\cdot)$ (see Lines 36 and 37 in Algorithm 2). The incremental algorithm then proceeds based on whether the (updated) i-th valid radius $\hat{\nu}_i$ is less than the i-th radius ν_i, as follows:

- If $\hat{\nu}_i < \nu_i$, then the algorithm updates the data structures as described previously with further modifications (see the function Update-DS-Rad-Decr(i) in Lines 2 and 39 of Algorithm 2). In particular, the incremental algorithm updates the i-th radius ν_i to $\hat{\nu}_i$ and the i-th approximate ball B_i to an arbitrary subset of $\text{Ball}[S_i, \hat{\nu}_i, \delta_{S_i}] \cap U_i$ with size $\lceil \beta |U_i| \rceil$ (in Lines 3 and 5 of Algorithm 2). For each vertex $v \in B_i$, the algorithm updates its assignment $\sigma(v)$ to its nearest candidate center in S_i with respect to $\delta_{S_i}(\cdot)$. Subsequently, the incremental algorithm adds to the temporary leaking set Z both the i-th leaking set Z_i and the vertices that were in the old i-th approximate ball but are not in the updated i-th approximate ball B_i (see Figure 1). The algorithm then empties the i-th leaking set Z_i and updates the $(i+1)$-th execution set U_{i+1} to $U_i \setminus B_i$.

[7] If no such level ℓ exists, then the incremental bicriteria approximation algorithm does nothing.

[8] When we say in Line 35 of Algorithm 2 "$\mathcal{A}_i$ provides distance estimates $\delta_u(\cdot)$ for all $u \in S_i$", we mean the nearest candidate center $u \in S_i$ to a vertex $v \in V$ with respect to $\delta_{S_i}(v)$.

- Otherwise if $\hat{v}_i \geq v_i$, then the algorithm removes from the i-th approximate ball B_i all vertices that no longer belong to the i-th execution set U_i.[9] Observe also that for a level $j : 0 \leq j < i$, some vertices that belonged to the old j-th approximate ball before the adversarial edge insertion, may no longer belong to the new j-th approximate ball B_j after the adversarial edge insertion. Note that some of these vertices have been added to the temporary leaking set Z. Subsequently, the incremental algorithm adds to the i-th leaking set Z_i an arbitrary subset of the temporary leaking set Z, ensuring that $Z_i \subseteq U_i$ and $|Z_i| = \lceil \beta |U_i| \rceil - |B_i|$ (see Lines 41, 42, and 43 in Algorithm 2). The algorithm then removes from the temporary leaking set Z the subset of Z that entered the i-th leaking set Z_i (see Line 44 in Algorithm 2) and updates the $(i+1)$-th execution set U_{i+1} to $U_i \setminus (B_i \cup Z_i)$.

Afterwards, the incremental algorithm increases i by one (i.e., moves to the next level), and continues with the *Resampling Phase* in the updated i-th execution set U_i (for the updated level i) as described earlier. The Resampling Phase terminates when the size of the current i-th execution set U_i is at most $\alpha k \log n$. Then, the algorithm updates the value of the last level t to the final value of i (see Line 48 in Algorithm 2). Finally, as a concluding step for the adversarial edge insertion (u^+, v^+) into the graph G, the incremental bicriteria approximation algorithm trivially updates the data structures at level t (in Line 49 of Algorithm 2) and assigns the maintained bicriteria approximate solution S as $\bigcup_{i=0}^{t} S_i$ (in Line 50 of Algorithm 2).

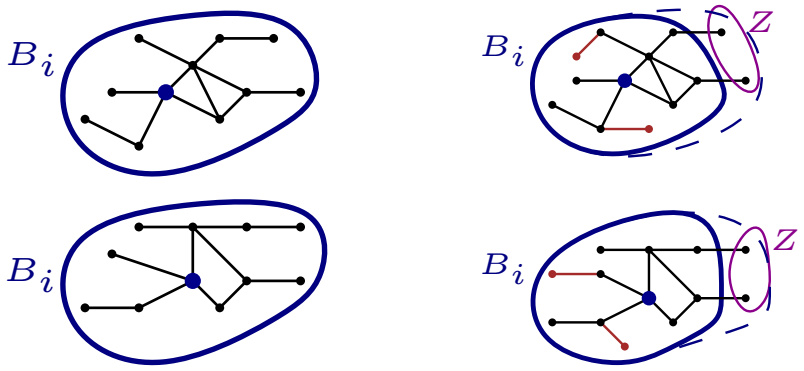

Figure 1 The blue thick regions depict the i-th approximate ball B_i for a level $i \in [0, t]$, and the larger blue vertices represent the i-th candidate set S_i. The black lines indicate distances. In the right figure, the brown lines and dots depict new shorter distances of vertices that enter B_i due to the adversarial edge insertion.

Left figure: The i-th approximate ball before the adversarial edge insertion.

Right figure: After the adversarial edge insertion, some vertices enter the i-th approximate ball B_i (brown vertices). Since the i-th radius v_i is decreased (the dashed blue region indicates the old radius), some vertices enter the temporary leaking set Z.

[9] A vertex that belonged to the old i-th execution set is removed from the current i-th execution set U_i when it enters a previous j-th approximate ball B_j at some level $j < i$.

Algorithm 2 Incremental Bicriteria Approximation Algorithm.

Require: Graph G and positive integer k (problem input); positive constants α, β (MP-bi parameters); positive constant ϵ (MP-bi variant parameter, used also as SSSP approximation parameter)
Ensure: Bicriteria approximate solution S and bicriteria approximate assignment σ

▷ The algorithm has global access to all $U_i, S_i, \nu_i, \mathcal{A}_i, B_i, Z_i, Z, \sigma(\cdot)$

function Update-DS-Rad-Decr(i)
  $\nu_i \leftarrow \hat{\nu}_i$
  $B'_i \leftarrow B_i$
  $B_i \leftarrow \text{Retain}\big(\text{Ball}[S_i, \nu_i, \delta_{S_i}] \cap U_i, \lceil \beta |U_i| \rceil\big)$
  For every $v \in B_i : \sigma(v) \leftarrow \text{argmin}_{u \in S_i} \delta_u(v)$
  $Z \leftarrow Z \cup Z_i \cup (B'_i \setminus B_i)$
  $Z_i \leftarrow \emptyset$

function First-Level-Decrease()
  $i \leftarrow -1$
  do
    $i \leftarrow i + 1$
    $\tilde{\nu}_i \leftarrow \min_{j \in \mathbb{Z}^{\geq 0}} \{(1+\epsilon)^j \mid |\text{Ball}[S_i, (1+\epsilon)^j, \delta_{S_i}] \cap U_i| \geq \beta |U_i|\}$
    $\hat{\nu}_i \leftarrow \max(\tilde{\nu}_i, \nu_{i-1})$
  while $i < t$ **and** $\hat{\nu}_i \geq \nu_i$
  if $i < t$ **then**
    $Z \leftarrow \emptyset$
    Update-DS-Rad-Decr(i)
  return i

function Edge-Insertion(u^+, v^+)
  ▷ If a ν_i is undefined, then assume that $\nu_i = nW + 1$
  ▷ If a set is undefined, then assume that it is the empty set
  Insert (u^+, v^+) into G
  for $i : 0 \leq i < t$ **do**
    $\mathcal{A}_i.insert(u^+, v^+)$
  $i \leftarrow$ First-Level-Decrease()
  $U_{i+1} \leftarrow U_i \setminus B_i$
  $i \leftarrow i + 1$
  if $i > t$ **then**
    exit from Edge-Insertion()
  ▷ Entering the Resampling Phase
  while $|U_i| > \alpha k \log n$ **do**
    Construct $\tilde{S}_i$ by sampling each $v \in U_i$ independently with probability $\min\left(\frac{\alpha k \log n}{|U_i|}, 1\right)$
    $S_i \leftarrow S_i \cup \tilde{S}_i$
    $\mathcal{A}_i.initialize(G, S_i)$ ▷ $\mathcal{A}_i$ provides distance estimates $\delta_{S_i}(\cdot)$ and $\delta_u(\cdot)$ for all $u \in S_i$
    $\tilde{\nu}_i \leftarrow \min_{j \in \mathbb{Z}^{\geq 0}} \{(1+\epsilon)^j \mid |\text{Ball}[S_i, (1+\epsilon)^j, \delta_{S_i}] \cap U_i| \geq \beta |U_i|\}$
    $\hat{\nu}_i \leftarrow \max(\tilde{\nu}_i, \nu_{i-1})$
    if $\hat{\nu}_i < \nu_i$ **then**
      Update-DS-Rad-Decr(i)
    else
      $B_i \leftarrow B_i \cap U_i$, $Z_i \leftarrow Z_i \cap U_i$
      $Z' \leftarrow \text{Retain}\big(Z \cap U_i, \lceil \beta |U_i| \rceil - |B_i| - |Z_i|\big)$
      $Z_i \leftarrow (Z_i \cup Z') \cap U_i$
      $Z \leftarrow Z \setminus Z'$
    $U_{i+1} \leftarrow U_i \setminus (B_i \cup Z_i)$
    $Z \leftarrow Z \cap U_{i+1}$
    $i \leftarrow i + 1$
  $t \leftarrow i$
  $S_t \leftarrow U_t$, $B_t \leftarrow U_t$, $Z_t \leftarrow \emptyset$, $Z \leftarrow \emptyset$, $\nu_t \leftarrow \nu_{t-1}$, $\forall v \in U_t : \sigma(v) \leftarrow v$
  $S \leftarrow \bigcup_{i=0}^{t} S_i$

5 From Bicriteria Approximation to (k,z)-Clustering on Incremental Graphs

Thus far, our incremental bicriteria approximation Algorithm 2 of Theorem 1 efficiently 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. Recall that by Definition 5, this means that Algorithm 2 maintains a subset of vertices $S \subseteq V$ such that $\text{cost}^z(S) \leq O(\text{OPT})$ and $|S| \leq O(k \log^3 n \log_{1+\epsilon} nW)$. Our goal though is to compute a constant-factor approximate solution C for the (k,z)-clustering problem (see Definition 3), and thus the output C must consist of at most k vertices. In other words, we want to maintain a subset of vertices $C \subseteq V$ such that $\text{cost}^z(C) \leq O(\text{OPT})$ and $|C| \leq k$.

In this section, we describe an efficient *incremental reduction algorithm* that reduces the size of an $(O(1), O(\log^3 n \log_{1+\epsilon} nW))$-bicriteria approximate solution S by converting it into a constant-factor approximate (k,z)-clustering solution C in a graph that undergoes edge insertions. To achieve this, we 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.

We remark that our Theorem 15 (the incremental reduction algorithm) is an adaptation of a theorem in [47, 27] in the incremental graph setting. The following definitions are used to analyze the total update time in Theorem 15. Notice that in Definition 14, the value of $\sigma_{\text{inc}}(V)$ is equal to the total number of batches of new vertices entering the image of σ. Hence, whenever $|\sigma_{\max}(V)|$ increases in a single update due to multiple new vertices, $\sigma_{\text{inc}}(V)$ increases by one.

▶ **Definition 13.** *Given a dynamic (α,β)-bicriteria approximate assignment σ, the dynamic set $\sigma_{max}(V) \coloneqq \{s \in V \mid s \in \sigma(V)$ at some moment$\}$ contains all vertices that have ever belonged to the image of σ.*

▶ **Definition 14.** *Given a dynamic (α,β)-bicriteria approximate assignment σ, the value $\sigma_{inc}(V)$ denotes the total number of times that $|\sigma_{max}(V)|$ increases.*
Formally, $\sigma_{inc}(V) \coloneqq |\{\tau \in \mathbb{N} \mid |\sigma_{max}(V)|$ increases at the τ-th update$\}|$.

▶ **Theorem 15.** *There is a (randomized) dynamic algorithm for the (k,z)-clustering problem that given:*

- *a weighted undirected graph $G = (V,E,w)$ subject to edge insertions,*
- *an integer $k \geq 1$, constants $z \geq 1, \epsilon \in (0, \frac{1}{2})$, 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:

$$|\sigma_{max}(V)| \cdot m^{1+o(1)} \; + \; \tilde{O}\big(|\sigma_{max}(V)|^2 \cdot \sigma_{inc}(V) \; + \; m \cdot |\sigma_{max}(V)|^{1+\epsilon}\big).$$

A pseudocode of our incremental reduction algorithm is provided in Algorithm 3. Our incremental reduction algorithm for the (k,z)-clustering problem (in Theorem 15) uses the fully dynamic spanner algorithm of Baswana, Khurana, and Sarkar [8]. Their initial result is for unweighted graphs, but as the authors mention in their Remark 1.2 in [8], it can be extended to weighted graphs.

▶ **Lemma 16** (Theorem 4.12/Theorem 5.7 in [8]). *There is a randomized fully dynamic algorithm that, given a weighted undirected graph $G = (V,E,w)$ subject to edge updates and a constant $\lambda \geq 1$, maintains a $O(2\lambda - 1)$-spanner of expected size $\tilde{O}(n^{1+\frac{1}{\lambda}})$ and has an expected amortized update time of $\tilde{O}(1)$. The preprocessing time of the algorithm is $\tilde{O}(m)$.*[10]

[10] For unweighted graphs, the stretch of the spanner is $2\lambda - 1$; the $O(2\lambda - 1)$ arises for weighted graphs.

Note that an incremental spanner (i.e., non-increasing distances) would be sufficient; however, the fully dynamic spanner algorithm of Lemma 16 is already suitable in our context. The *expected* guarantees can be turned into *high probability* guarantees by running $\Theta(\log n)$ copies of the algorithm in parallel; see [31] for more details. The incremental reduction algorithm also utilizes the static (k, z)-clustering algorithm of Dupre la Tour and Saulpic [42].

▶ **Lemma 17** ([42]). *There is randomized algorithm for the (k, z)-clustering problem that, given a weighted undirected graph $G = (V, E, w)$, an integer $k \geq 1$, and constants $z \geq 1, \lambda \geq 5$, computes with high probability a $O(\lambda)$-approximate (k, z)-clustering solution in $\tilde{O}(m^{1+\frac{1}{\lambda}})$ time.*

At a high level, we maintain the (k, z)-clustering solution C over the bicriteria approximate solution S, using weights on S derived from the bicriteria approximate assignment σ. Hence, the input graph instance contains both edge and vertex weights, and the static (k, z)-clustering algorithm should be able to work with vertex weights in our context. The algorithm by Dupre la Tour and Saulpic [42] is a simplification of the algorithm by Mettu and Plaxton [46].[11] Even though the authors in [42] present a version without vertex weights, the authors in [46] present a version with weights on points. Thus, it is not surprising that the static (k, z)-clustering algorithm of Lemma 17 can be generalized to handle vertex weights.

▶ **Definition 7** (weighted (k, z)-clustering problem on graphs). *Given a weighted undirected graph $\hat{G} = (\hat{V}, \hat{E}, \hat{w})$ with vertex weights $\mathrm{wt}(\cdot)$, the corresponding weighted (k, z)-clustering instance is denoted by $\mathcal{K} := (\hat{G}, k, z, \mathrm{wt})$. The cost of a subset of vertices $C \subseteq \hat{V}$ is defined as $\mathrm{cost}^z_{\mathrm{wt}}(C) := \sum_{v \in \hat{V}} \mathrm{wt}(v) \cdot \mathrm{dist}_{\hat{G}}(v, C)^z$. The optimal (k, z)-clustering cost of $\mathcal{K}$ is denoted by $OPT_{\mathcal{K}}$.*

The randomization in Theorem 15 arises from the randomization in Lemmas 16 and 17; replacing the randomized subroutines with deterministic counterparts would yield a deterministic version of Theorem 15.

Adversarial updates. Our aim is to combine the incremental bicriteria approximation Algorithm 2 with the incremental reduction Algorithm 3. Hence, the output (α, β)-bicriteria approximate assignment σ of Algorithm 2 becomes the input (α, β)-bicriteria approximate assignment σ of Algorithm 3. The assignment σ in Algorithm 2 can be modified due to an adversarial edge insertion into the graph G, making it a *dynamic assignment* in Theorem 15.

Since our incremental reduction algorithm is given as input a dynamic bicriteria approximate assignment, one type of adversarial update consists of modifications to the assignment σ. The other type of adversarial update involves edge insertions into the input graph G, where each adversarial edge insertion can trigger a batch of distance decreases between vertices in G.

Note that an efficient combination of Algorithm 2 and Algorithm 3, requires that both (1) the total number of vertices that have ever belonged to the image of the dynamic assignment σ (i.e., $|\sigma_{\max}(V)|$) and (2) the total number of batches of new vertices entering $\sigma_{\max}(V)$ (i.e., $\sigma_{\mathrm{inc}}(V)$) remain bounded. For efficiency reasons, we employ the approximate distance estimates from incremental $(1+\epsilon)$-approximate SSSP algorithms and perform lazy updates by maintaining distances as powers of $(1+\epsilon)$.

[11] The two algorithms by Mettu and Plaxton in [46] and [47] (which includes the MP-bi algorithm) are different.

State of the reduction algorithm. Our incremental reduction Algorithm 3 maintains a dynamic (α, β)-bicriteria approximate solution $S \coloneqq \sigma(V)$, which is the image of the dynamic (α, β)-bicriteria approximate assignment σ from the input.[12] 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 complete subgraph $H \coloneqq (S, S \times S, w_H)$ with edge and vertex weights, specified as follows:

- Each vertex $s \in S$ belongs to the vertex set of H, and the vertex weight $\mathrm{wt}(s)$ is equal to $|\sigma^{-1}(s)|$. In other words, the vertex weight of s in H is the number of vertices mapped to s under the assignment $\sigma(\cdot)$.
- For every pair of vertices $x, y \in S$, the edge (x, y) belongs to the edge set of H with an edge weight $w_H(x, y)$ 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 dynamic spanner algorithm $\mathcal{SP}$ of Lemma 16 on the subgraph H to obtain an edge-sparsified graph $\tilde{H}$. Finally, the reduction algorithm runs the static (k, z)-clustering algorithm $\mathcal{CL}$ of Lemma 17 on the weighted (k, z)-clustering instance $(\tilde{H}, k, z, \mathrm{wt})$, producing a (k, z)-clustering solution C.

5.1 Adversarial Edge Insertions and Assignment Modifications

In the next two paragraphs, we first describe how our incremental reduction algorithm processes an adversarial edge insertion into the graph G, and then how it processes a batch of modifications to the (α, β)-bicriteria approximate assignment σ. We remark that in fact, each adversarial edge insertion into the graph G can trigger a batch of distance decreases between vertices in G alongside a batch of 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 incremental reduction algorithm proceeds as follows. Initially, this edge insertion (u, v) is forwarded to all the incremental $(1+\epsilon)$-approximate SSSP algorithms $\mathcal{A}_s$, where $s \in S$. Based on Lemma 8, the incremental $(1+\epsilon)$-approximate SSSP algorithm $\mathcal{A}_s$ reports a change whenever the distance estimate $\delta_s(x)$ is updated for a vertex $x \in V$. After the adversarial insertion of the edge (u, v) into the graph G, 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 $\frac{w_H(x,y)}{1+\epsilon}$.

Next, for every pair of vertices $(x, y) \in \Delta_S$, the incremental reduction algorithm updates the edge weights $w_H(x, y)$ and $w_H(y, x)$ to $\min(\delta_x(y), \delta_y(x))$, rounding them up to the nearest power of $(1+\epsilon)$. Let $B \coloneqq \{(x, y, w_H(x, y)) \mid (x, y) \in \Delta_S \text{ or } (y, x) \in \Delta_S\}$ be the batch of distance decreases. The reduction algorithm then forwards the batch B of distance decreases to the dynamic spanner algorithm $\mathcal{SP}$ from Lemma 16,[13] which runs on H and maintains the edge-sparsified graph $\tilde{H}$.

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 vertex weight $\mathrm{wt}(s)$ to $|\sigma^{-1}(s)|$.

For every vertex $s \in R_S$ that does not belong to the dynamic (α, β)-bicriteria approximate solution S, the incremental reduction algorithm proceeds as follows:

1. First, the reduction algorithm adds s to S and to the vertex set of H, and sets its vertex weight $\mathrm{wt}(s)$ to $|\sigma^{-1}(s)|$.

[12] Since the algorithm never removes vertices from S, we actually have $S = \sigma_{\max}(V)$.
[13] The dynamic spanner algorithm $\mathcal{SP}$ processes each distance decrease in the batch B separately.

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

Next, if at least one new vertex is added to S (i.e., $|\sigma_{\max}(V)|$ of Definition 13 increases) then the reduction algorithm restarts the dynamic spanner algorithm $\mathcal{SP}$ on the subgraph H, producing a new edge-sparsified graph $\tilde{H}$. Otherwise all vertices in R_S already belong to S, and the dynamic spanner algorithm $\mathcal{SP}$ maintains the same edge-sparsified graph $\tilde{H}$.

At the end of each adversarial update, the incremental reduction algorithm runs the static (k, z)-clustering algorithm $\mathcal{CL}$ of Lemma 17 on the updated weighted (k, z)-clustering instance $(\tilde{H}, k, z, \mathrm{wt})$, producing the maintained (k, z)-clustering solution C.

Algorithm 3 Incremental Reduction Algorithm.

Require: Graph G and positive integer k (problem input); Bicriteria approximate solution S 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**
    $\mathcal{A}_s.insert(u, v)$
  $\Delta_S \leftarrow \{(x, y) \in S \times S \mid \delta_x(y) < \frac{w_H(x,y)}{1+\epsilon}\}$ ▷ Set of affected distances
  for $(x, y) \in \Delta_S$ **do**
    $w_H(x, y) \leftarrow \min(\delta_x(y), \delta_y(x))$
    $w_H(y, x) \leftarrow \min(\delta_x(y), \delta_y(x))$
    Round $w_H(y, x)$ and $w_H(x, y)$ up to the nearest power of $(1+\epsilon)$
  $B \leftarrow \{(x, y, w_H(x, y)) \mid (x, y) \in \Delta_S \text{ or } (y, x) \in \Delta_S\}$
  $\mathcal{SP}.update(B)$
  $\tilde{H} \leftarrow \mathcal{SP}.output(H)$ ▷ $\mathcal{SP}$ is applied on H
  $C \leftarrow \mathcal{CL}.output(\tilde{H})$ ▷ $\mathcal{CL}$ is applied on $\tilde{H}$

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**
    $\mathrm{wt}(s) \leftarrow |\sigma^{-1}(s)|$
    if $s \notin S$ **then**
      Add s to S and to the vertex set of H
      $\mathcal{A}_s.initialize(G, s)$ ▷ $\mathcal{A}_s$ provides distance estimates $\delta_s(\cdot)$
      for $x \in S$ **do**
        $w_H(s, x) \leftarrow \min(\delta_s(x), \delta_x(s))$
        $w_H(x, s) \leftarrow \min(\delta_s(x), \delta_x(s))$
        Round $w_H(s, x)$ and $w_H(x, s)$ up to the nearest power of $(1+\epsilon)$
  if $|\sigma_{\max}(V)|$ increases **then** ▷ Some new vertices have been added to S (see Definition 13)
    $\mathcal{SP}.initialize(H)$
  $\tilde{H} \leftarrow \mathcal{SP}.output(H)$ ▷ $\mathcal{SP}$ is applied on H
  $C \leftarrow \mathcal{CL}.output(\tilde{H})$ ▷ $\mathcal{CL}$ is applied on $\tilde{H}$

References

1 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, editors, *2023 Symposium on Simplicity in Algorithms, SOSA 2023, Florence, Italy, January 23-25, 2023*, pages 145–155. SIAM, 2023. doi:10.1137/1.9781611977585.CH14.

2 Charu C. Aggarwal and Haixun Wang. *A Survey of Clustering Algorithms for Graph Data*, pages 275–301. Springer US, Boston, MA, 2010. doi:10.1007/978-1-4419-6045-0_9.

3 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.

4 Bertie Ancona, Monika Henzinger, Liam Roditty, Virginia Vassilevska Williams, and Nicole Wein. Algorithms and hardness for diameter in dynamic graphs. In Christel Baier, Ioannis Chatzigiannakis, Paola Flocchini, and Stefano Leonardi, editors, *46th International Colloquium on Automata, Languages, and Programming, ICALP 2019, July 9-12, 2019, Patras, Greece*, volume 132 of *LIPIcs*, pages 13:1–13:14. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2019. doi:10.4230/LIPIcs.ICALP.2019.13.

5 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.

6 Yair Bartal. Probabilistic approximation of metric spaces and its algorithmic applications. In *Proceedings of 37th Conference on Foundations of Computer Science*, pages 184–193. IEEE, 1996. doi:10.1109/SFCS.1996.548477.

7 Yair Bartal. On approximating arbitrary metrices by tree metrics. In *Proceedings of the thirtieth annual ACM symposium on Theory of computing*, pages 161–168, 1998. doi:10.1145/276698.276725.

8 Surender Baswana, Sumeet Khurana, and Soumojit Sarkar. Fully dynamic randomized algorithms for graph spanners. *ACM Trans. Algorithms*, 8(4):35:1–35:51, 2012. doi:10.1145/2344422.2344425.

9 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, editors, *Proceedings of the 2023 ACM-SIAM Symposium on Discrete Algorithms, SODA 2023, Florence, Italy, January 22-25, 2023*, pages 2677–2727. SIAM, 2023. doi:10.1137/1.9781611977554.CH101.

10 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*, 2024.

11 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)*, pages 216–227, Los Alamitos, CA, USA, October 2024. IEEE Computer Society. doi:10.1109/FOCS61266.2024.00023.

12 Leyla Biabani and Ami Paz. k-center clustering in distributed models. In *International Colloquium on Structural Information and Communication Complexity*, pages 83–100. Springer, 2024. doi:10.1007/978-3-031-60603-8_5.

13 Jarosław Byrka, Thomas Pensyl, Bartosz Rybicki, Aravind Srinivasan, and Khoa Trinh. An improved approximation for k-median and positive correlation in budgeted optimization. *ACM Trans. Algorithms*, 13(2), March 2017. doi:10.1145/2981561.

14 Jarosław Byrka, Yuhao Guo, Yang Hu, Shi Li, Chengzhang Wan, and Zaixuan Wang. k-clustering via iterative randomized rounding, 2026. doi:10.48550/arXiv.2604.06046.

15 Moses Charikar, Chandra Chekuri, Ashish Goel, and Sudipto Guha. Rounding via trees: deterministic approximation algorithms for group steiner trees and k-median. In *Proceedings of the thirtieth annual ACM symposium on Theory of computing*, pages 114–123, 1998. doi:10.1145/276698.276719.

16 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*, pages 1–10, 1999.

17 Adam Coates and Andrew Y. Ng. Learning feature representations with k-means. In *Neural Networks: Tricks of the Trade*, pages 561–580. Springer, 2 edition, 2012. doi:10.1007/978-3-642-35289-8_30.

18 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.

19 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)*, pages 1556–1612. SIAM, 2022. doi:10.1137/1.9781611977073.65.

20 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, editors, *Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS 2019, December 8-14, 2019, Vancouver, BC, Canada*, pages 3250–3260, 2019. URL: https://proceedings.neurips.cc/paper/2019/hash/fface8385abbf94b4593a0ed53a0c70f-Abstract.html.

21 Emilio Cruciani, Sebastian Forster, Gramoz Goranci, Yasamin Nazari, and Antonis Skarlatos. Dynamic algorithms for k-center on graphs. In *Proceedings of the 2024 Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 3441–3462. SIAM, 2024. doi:10.1137/1.9781611977912.123.

22 Devdatt P Dubhashi and Alessandro Panconesi. *Concentration of measure for the analysis of randomized algorithms*. Cambridge University Press, 2009.

23 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.

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

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

26 Santo Fortunato. Community detection in graphs. *Physics Reports*, 486(3):75–174, 2010. doi:10.1016/j.physrep.2009.11.002.

27 S Guha, N Mishra, R Motwani, and L O'Callaghan. Clustering data streams. In *Proceedings 41st Annual Symposium on Foundations of Computer Science*, pages 359–366. IEEE, 2000.

28 Anupam Gupta and Kanat Tangwongsan. Simpler analyses of local search algorithms for facility location. *arXiv preprint arXiv:0809.2554*, 2008. arXiv:0809.2554.

29 Pierre Hansen and Brigitte Jaumard. Cluster analysis and mathematical programming. *Math. Program.*, 79:191–215, 1997. doi:10.1007/BF02614317.

30 Sariel Har-Peled, Piotr Indyk, and Anastasios Sidiropoulos. Euclidean spanners in high dimensions. In Sanjeev Khanna, editor, *Proceedings of the Twenty-Fourth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2013, New Orleans, Louisiana, USA, January 6-8, 2013*, pages 804–809. SIAM, 2013. doi:10.1137/1.9781611973105.57.

31 Monika Henzinger, Sebastian Krinninger, and Danupon Nanongkai. A subquadratic-time algorithm for decremental single-source shortest paths. In Chandra Chekuri, editor, *Proceedings of the Twenty-Fifth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2014, Portland, Oregon, USA, January 5-7, 2014*, pages 1053–1072. SIAM, 2014. doi:10.1137/1.9781611973402.79.

32 Monika Henzinger, Sebastian Krinninger, and Danupon Nanongkai. Decremental single-source shortest paths on undirected graphs in near-linear total update time. *Journal of the ACM*, 65(6):36:1–36:40, 2018. Announced at FOCS 2014. doi:10.1145/3218657.

33 Lingxiao Huang and Nisheeth K. Vishnoi. Coresets for clustering in euclidean spaces: importance sampling is nearly optimal. In Konstantin Makarychev, Yury Makarychev, Madhur Tulsiani, Gautam Kamath, and Julia Chuzhoy, editors, *Proceedings of the 52nd Annual ACM SIGACT Symposium on Theory of Computing, STOC 2020, Chicago, IL, USA, June 22-26, 2020*, pages 1416–1429. ACM, 2020. doi:10.1145/3357713.3384296.

34 A. K. Jain, M. N. Murty, and P. J. Flynn. Data clustering: A review. *ACM Comput. Surv.*, 31(3):264–323, 1999. doi:10.1145/331499.331504.

35 Anil K. Jain. Data clustering: 50 years beyond k-means. *Pattern Recognition Letters*, 31(8):651–666, 2010. Award winning papers from the 19th International Conference on Pattern Recognition (ICPR). doi:10.1016/j.patrec.2009.09.011.

36 Kamal Jain, Mohammad Mahdian, and Amin Saberi. A new greedy approach for facility location problems. In *Proceedings of the Thiry-Fourth Annual ACM Symposium on Theory of Computing*, STOC '02, pages 731–740, New York, NY, USA, 2002. Association for Computing Machinery. doi:10.1145/509907.510012.

37 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.

38 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.

39 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)*, pages 175–211. SIAM, 2025. doi:10.1137/1.9781611978322.6.

40 Oded Kariv and S Louis Hakimi. An algorithmic approach to network location problems. i: The p-centers. *SIAM journal on applied mathematics*, 37(3):513–538, 1979.

41 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.

42 Max Dupré la Tour and David Saulpic. Faster and simpler greedy algorithm for k-median and k-means, 2025. arXiv:2407.11217.

43 Shi Li and Ola Svensson. Approximating k-median via pseudo-approximation. In Dan Boneh, Tim Roughgarden, and Joan Feigenbaum, editors, *Symposium on Theory of Computing Conference, STOC'13, Palo Alto, CA, USA, June 1-4, 2013*, pages 901–910. ACM, 2013. doi:10.1145/2488608.2488723.

44 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.

45 Jakub Łącki and Yasamin Nazari. Near-Optimal Decremental Hopsets with Applications. In *49th International Colloquium on Automata, Languages, and Programming (ICALP 2022)*, volume 229 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 86:1–86:20. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2022. doi:10.4230/LIPIcs.ICALP.2022.86.

46 Ramgopal R. Mettu and C. Greg Plaxton. The online median problem. *SIAM J. Comput.*, 32(3):816–832, 2003. doi:10.1137/S0097539701383443.

47 Ramgopal R Mettu and C Greg Plaxton. Optimal time bounds for approximate clustering. *Machine Learning*, 56:35–60, 2004. doi:10.1023/B:MACH.0000033114.18632.E0.

48 M. E. J Newman. Detecting community structure in networks. *The European Physical Journal B*, 38, 2004. doi:10.1140/epjb/e2004-00124-y.

49 Matthew J. Rattigan, Marc E. Maier, and David D. Jensen. Graph clustering with network structure indices. In Zoubin Ghahramani, editor, *Machine Learning, Proceedings of the Twenty-Fourth International Conference (ICML 2007), Corvallis, Oregon, USA, June 20-24, 2007*, volume 227 of *ACM International Conference Proceeding Series*, pages 783–790. ACM, 2007. doi:10.1145/1273496.1273595.

50 Satu Elisa Schaeffer. Graph clustering. *Comput. Sci. Rev.*, 1(1):27–64, 2007. doi:10.1016/J.COSREV.2007.05.001.

51 Adam Schenker, Mark Last, Horst Bunke, and Abraham Kandel. Comparison of algorithms for web document clustering using graph representations of data. In Ana L. N. Fred, Terry Caelli, Robert P. W. Duin, Aurélio C. Campilho, and Dick de Ridder, editors, *Structural, Syntactic, and Statistical Pattern Recognition, Joint IAPR International Workshops, SSPR 2004 and SPR 2004, Lisbon, Portugal, August 18-20, 2004 Proceedings*, volume 3138 of *Lecture Notes in Computer Science*, pages 190–197. Springer, 2004. doi:10.1007/978-3-540-27868-9_19.

52 Jianbo Shi and Jitendra Malik. Normalized cuts and image segmentation. *IEEE Trans. Pattern Anal. Mach. Intell.*, 22(8):888–905, 2000. doi:10.1109/34.868688.

53 Arie Tamir. An $o(pn^2)$ algorithm for the p-median and related problems on tree graphs. *Operations research letters*, 19(2):59–64, 1996. doi:10.1016/0167-6377(96)00021-1.

54 Mikkel Thorup. Quick k-median, k-center, and facility location for sparse graphs. *SIAM Journal on Computing*, 34(2):405–432, 2005. doi:10.1137/S0097539701388884.