algorithms

Article

Optimal Clustering in Stable Instances Using Combinations of Exact and Noisy Ordinal Queries

Enrico Bianchi and Paolo Penna *

Department of Computer Science, ETH Zurich, 8092 Zurich, Switzerland; bianchie@student.ethz.ch
* Correspondence: paolo.penna@inf.ethz.ch

Abstract: This work studies clustering algorithms which operates with *ordinal* or *comparison-based* queries (operations), a situation that arises in many active-learning applications where "dissimilarities" between data points are evaluated by humans. Typically, *exact* answers are *costly* (or difficult to obtain in large amounts) while possibly *erroneous* answers have *low cost*. Motivated by these considerations, we study algorithms with non-trivial *trade-offs* between the number of exact (high-cost) operations and noisy (low-cost) operations with provable performance guarantees. Specifically, we study a class of polynomial-time *graph-based* clustering algorithms (termed Single-Linkage) which are widely used in practice and that guarantee *exact* solutions for *stable* instances in several clustering problems (these problems are NP-hard in the worst case). We provide several variants of these algorithms using *ordinal* operations and, in particular, non-trivial trade-offs between the number of *high-cost* and *low-cost* operations that are used. Our algorithms still guarantee *exact* solutions for *stable* instances of *k-medoids* clustering, and they use a rather small number of high-cost operations, without increasing the low-cost operations too much.

Keywords: clustering; stable instances; ordinal queries; minimum spanning tree; medoids; active learning; noisy queries

Citation: Bianchi, E.; Penna, P. Optimal Clustering in Stable Instances Using Combinations of Exact and Noisy Ordinal Queries. *Algorithms* **2021**, *14*, 55. https://doi.org/10.3390/a14020055

Academic Editors: Stefano Leucci and Marco Bressan
Received: 11 January 2021
Accepted: 4 February 2021
Published: 8 February 2021

1. Introduction

Clustering is a fundamental and widely studied problem in machine learning and in computational complexity as well. Intuitively, the goal is to partition a given set of points X (data set) into k clusters, each cluster corresponding to a group of "similar" data according to some underlying distance function $d(\cdot)$. Though most of the clustering problems are NP-hard in the worst case, for real data, certain (relatively simple) heuristics seem to perform quite well. One theoretical justification to support this empirical observation is that "real" instances have some "nice" distribution, and some *polynomial-time* algorithms come with the guarantee that, on these inputs, they return *optimal* solutions. Along these lines, one successful concept is that of *stability* of the input, meaning that a "small" perturbation of the input does not change the optimal solution (though it may change its value).

Quite surprisingly, for a rather general class of clustering problems, the following very *simple* algorithm guarantees *exact* solutions in stable instances in *polynomial-time* [1–3] (see also [4] for a nice introduction). Roughly speaking, this algorithm first computes a *(minimum) spanning tree* over the data pairwise distances or dissimilarities, and then removes a *suitable* subset of edges to obtain the optimal k-clustering. The latter step amounts to identify the $k-1$ edges whose removal partitions the spanning tree into k clusters (the nodes of the resulting forest) in order to minimize the underlying cost function of the clustering problem (the exact cost function depends on how we define the "center" of the clusters and on the distances of the points in the cluster to its center).

The implicit assumption is that the algorithm is provided with the *correct* pairwise dissimilarity/distances metric between all pairs of data points. In this work, we take a step further and consider the natural setting where we can only *compare* distances and these comparisons are also *noisy*. Furthermore, by accessing some type of expensive (expert) oracle, we can make sure that certain comparisons are *correctly* reported, though these operations are inherently more *expensive* and thus we would like to use *as few as possible*. In a nutshell, our work considers and combines the following three aspects that are often present in practical machine learning approaches:

- Using only *ordinal* information;
- Dealing with *noisy* data;
- Allowing *expensive* operations to remove errors.

This situation arises, for example, in semi-active learning approaches where the pairwise dissimilarity between objects (data points) is *evaluated* by *humans* via simple comparison queries (see, e.g., [5–7] and references therein). These are inherently subject to erroneous evaluations. Moreover, very often, the task is to *compare* objects and their pairwise dissimilarities. Such queries can also be of varying difficulty and thus more or less informative/costly. It is therefore natural to ask the following questions:

Which guarantees are still possible under such model?

What trade-offs between expensive and non-expensive (noisy) operations still allow for finding optimal solutions?

1.1. Our Contribution

In this work, we address these questions by (i) introducing a formal model and (ii) by considering a class of clustering problems/algorithms in this model. Specifically, we consider k-medoids clustering which applies to general dissimilarities of data (unlike, e.g., k-means), and for which the above "minimum-spanning-tree-based" optimal algorithm for stable instances can be implemented using (certain) ordinal queries. We detail our contributions and its relation to prior work in the following sections.

1.1.1. Our Model

This paper introduces a natural setting where we can only *compare* distances, and these comparisons are generally *noisy*; though use some sort of *expensive* (expert) oracle, we can make sure that certain comparisons are *correctly* reported. Our model (see Section 2 for formal definitions) captures the following aspects:

1. Using only *ordinal* information. Distances or dissimilarities can only be *compared* and not directly evaluated. One example of such queries is [5–7]

   Which one between y and y' is more dissimilar to x?

   Concretely, this means that we can compare $d(x, y)$ with $d(x, y')$ for some dissimilarity function $d(\cdot)$. In our model, we allow for comparing arbitrary groups (sum) of pairwise distances.
2. Dealing with *noisy* results. Comparisons are in general *noisy* and the resulting error is *persistent*. Due to measurements, we may report the wrong answer with probability bounded by some error probability $p < 1/2$, and repeating the same measurements would lead to the same answer [7–9]. (In order to import some results from the literature, we shall typically assume $p < 1/16$. All our results automatically extend to larger p if these prior results in the literature also do.)
3. Allowing *expensive* operations to remove errors. Errors can be fixed by performing some kind of *expensive* comparison. In this case, we know the answer is correct, but we would like to *limit* the total number of such expensive operations (see e.g., [10] for a survey on various practical methods).

This setting falls into the *dual-mode* framework in [11], where several optimization problems can be optimally solved using either only low-cost operations or a suitable combination with few high-cost ones. This framework suggests to evaluate the complexity of an algorithm by explicitly counting the low-cost and high-cost operations separately. Without errors, or ignoring that exact operations have high-cost, the problem falls into the class of problems that can be solved only with *ordinal* operations. Without the aid of high-cost operations that is using only noisy comparisons, the problem has been recently studied in the context of active learning under various query and error models (see, e.g., [12,13] and below for further discussions).

1.1.2. Algorithms and Bounds for k-Medoids

We provide new polynomial-time algorithms for k-medoids clustering. These algorithms achieve the following trade-offs between the number of high-cost and low-cost operations to compute optimal k-clustering in stable instances (with stability parameter $\gamma \geq 2$—see Definition 1):

- In Section 3, we investigate variants of the popular Single-Linkage algorithm, and its enhanced version Single-Linkage++ analyzed in [1–3]. (Following [4], we call Single-Linkage the simpler algorithm, which often used in practice, and Single-Linkage++ the enhanced version with provable guarantees [1–3]). This algorithm consists of two distinct phases (computing a minimum spanning tree and removing a suitable set of edges to obtain a clustering). A naive implementation, using only high-cost comparisons, would require $O(n^2)$ for such operations for the first phase and $O(n \log n)$ for the second one. The trade-offs are summarized in Table 1, where we also consider a *naive* (simpler) version of the algorithm with no approximation guarantee (this serves to illustrate some key ideas and develop some necessary tools). All other variants are either *exact* or guarantee *a* 2-*approximation* in the worst case. At this point, some comments are in order:
  - The overall algorithm consists of a combination of Phase 1 and Phase 2, and thus the overall complexity is given by the sum of these two. The total number of operations (also accounting for internal computations) is comparable with the sum of the high-cost and low-cost operations.
  - Phase 1 improves over the naive implementation (Remark 3) under some additional assumptions on the data (in particular, larger stability parameter $\gamma > 2$ helps, as well as a small radius—the ratio $d^{(X)}_{\max}/d^{(X)}_{\min}$ between the largest and the smallest distance between two points).
  - The naive algorithm (Single-Linkage) assumes that Phase 1 has been already solved and implements a simpler Phase 2 (the complexity bounds refer only to the latter). This algorithm is in fact a widely used heuristic with additional theoretical guarantees for hierarchical clustering (see, e.g., [14,15]).
  - Phase 2 can be implemented using *very few* high-cost operations, if we content with a 2-approximate solution. Exact solutions use larger number high-cost operations. Though the dynamic programming (DP) approach has a better theoretical performance for large k, the other algorithm is much simpler to implement (for example, it does not require memory $O(k^2 n^4 \log n)$ used to store the DP table).

  We remark that the best combination between Phase 1 and Phase 2 depends on k and, in some cases, on additional properties of the input data X.
- In Section 4, we show that, under additional assumptions on the input data, and a slightly more powerful comparison model, it is possible to implement exact or approximate *same-cluster* queries (see Section 4.1 and Lemma 10 therein).
- Since same-cluster queries may require some additional power, in Section 4.2, we provide algorithms which use *few* same-cluster queries in combination with our original (low-cost and high-cost) comparison operations. The obtained bounds are summarized in Theorem 5 and Theorem 6. Intuitively speaking, the ability to preform "few" exact same-cluster queries allows for us to reduce the number of high cost-operations significantly, at least for certain instances:
  - When the optimal solutions has approximately balanced clusters, $O(k \log k)$ same-cluster queries are enough, and the additional high-cost comparisons are $O(\log^2 n)$. Both bounds scale with the cluster "unbalance" n/n_1, where n_1 is

the smallest cluster size (slightly better bounds for the number of same-cluster queries hold).

- The additional assumption on the input data is only required to be able to implement these few same-cluster queries directly from our model. The result still applies in general if these same-cluster queries can be be implemented in a different—perhaps quite expensive—fashion.
- The aforementioned condition to simulate same-cluster queries, essentially requires that, in the optimal solution, points in the same cluster have a distance of at most $\gamma - 1$ times the minimum distance. For larger γ, this condition becomes less stringent, though of course we require a larger stability coefficient.

Table 1. Trade-off for various algorithms for γ-pertubation-stable instances ($\gamma \geq 2$) in this work. We distinguish the two-phases of the Single-Linkage++ algorithm (and a simpler naive version) and show the corresponding number of high-cost and low-cost comparisons depending on the size n of the dataset X, the number k of clusters; Parameters $d^{(X)}_{\min}$ and $d^{(X)}_{\max}$ denote the minimum and the maximum distance between points in X, respectively.

Algorithm	**Approx.**	**High-Cost**	**Low-Cost**
Single-Linkage (Phase 2 naive)	∞	$O(k + \log k \log n)$	$O(n \log n)$
Single-Linkage++ (Phase 1)	1	$O(\frac{d^{(X)}_{\max}}{(\gamma-2)d^{(X)}_{\min}} n \log^2 n)$	$O(n^2 \log n)$
Single-Linkage++ (Phase 2)	1	$O(\binom{n-1}{k-1} k \log(\frac{n}{k}))$	$O(\binom{n-1}{k-1} \log n)$
Single-Linkage++ (Phase 2 APX)	2	$O(k \log n)$	$O(\binom{n-1}{k-1} \log n)$
Single-Linkage++ (Phase 2 DP)	1	$O(kn^2 \log n)$	$O(k^2 n^4 \log n)$

All above mentioned algorithms (and results) hold with *high probability*, that is, with probability at least $1 - n^{-c}$, for some constant $c > 0$ (we actually prove success probability at least $1 - \frac{3}{n}$ or larger in all cases), where the probability is on the outcome of the distance comparisons (errors).

1.1.3. Techniques and Relation to Prior Work

This work focuses on the so-called *k-medoids* clustering problem, where the center of each cluster (its medoid) must be a point of the cluster (and thus of the data set) [16–19]. This is slightly different from k-means, where the centroid of a cluster may not be an element of the cluster/data set. It is well known that k-medoids clustering has several advantages. First, it can be applied to any dissimilarity/distance function between data points, while k-means requires the points to be embedded in some Eulidean space $\Re^{\ell}$. Moreover, it is well known that k-medoids is more stable to outliers (see, e.g., [20]). Despite k-medoids being NP-hard, it has been recently shown that, for *stable* instances, a relatively simple (Single-Linkage++) algorithm solves the problem *exactly* in *polynomial-time* [1–3] (see also [4] for a nice introduction).

The algorithms implementing Single-Linkage++ (Table 1) are based on two algorithmic results regarding, respectively, approximate sorting with noisy comparisons [21], and approximate matroid maximization with high-cost and low-cost operations [11]. The dynamic implementation of Phase 2 is instead an adaptation of the algorithm in [3] to work with our comparison-based model. Their algorithmic result is in fact more general, and it applies to the class of *center-based* clustering problems (intuitively, the solution and its cost are uniquely determined by a set of centers—in our case medoid).

Without allowing high-cost operations that are using only noisy comparisons, the problem has been studied in the context of (semi-supervised) *active learning* which involves (noisy) *comparison* queries of varying "complexity". Specifically, Refs. [12,13] consider comparisons between pairs of *classifiers* in some fixed hypothesis class (a noisy comparison between two candidate classifiers h and h^* consists of comparing the loss of these classifiers on a small set labelled data set). In [22], the authors consider queries of the form "$f(x) \geq$

$f(y)$" for specific functions $f(\cdot)$. Several works consider *ordinal* queries involving distances between (few) points: "triplets" are considered in [5,7] with queries of the form "$d(x,y) \geq d(x,z)$" (is x more similar to y or to z) and in [23] with queries giving the outlier among the three points "$d(x,y) \leq \min\{d(x,z), d(y,z)\}$". In [14], "quadruples" queries of the form "$d(x,y) \geq d(z,w)$" are used to simulate more complex queries (e.g., implementing the naive Single-Linkage algorithm in a noisy-free setting). In [8,9], queries involving some "scalar/multiplicative" factor $\alpha \geq 1$—similar to what we use to simulate same-cluster queries—are used; Their queries are of the form "$\alpha d(x,z) \geq d(y,z)$" and the answer is correct if the inequality holds, but the oracle may not answer whenever these distances are "close", i.e., $d(x,z) \leq d(y,z) < \alpha d(x,z)$; [9] considers the variant in which the answer is adversarially wrong in this case.

It widely believed that *same-cluster queries* may be difficult to implement in practice, though very powerful. Algorithms based on *same-cluster queries* both exact and with errors have largely been studied. The *error-free/exact* case is closely related to our "all at high cost" implementations (assuming one same-cluster query costs as one exact call in our model). Specifically, Ref. [24] considers k-means and provide an algorithm using $O(k^2 \log k + k \log n)$ same cluster queries, while Ref. [25] uses $O(\frac{k^{14} \log k \log n}{\epsilon^6})$ same-cluster queries for computing $(1+\epsilon)$-approximate correlation clustering; Ref. [26] provides an exact algorithm using $2C_{OPT}$ same-cluster queries, where C_{OPT} denotes the number of "disagreements" in the optimal solution. Same-cluster queries can also solve *non-center-based* clustering problems [27], where the corresponding algorithm uses $O(k^3 \log k \log n)$, with the hidden constant depending exponentially in the dimensionality of the clusters. Regarding *noisy* same-cluster queries, Ref. [28] uses $O(\frac{nk \log n}{(1-2p)^2})$ same-cluster queries to reconstruct the "latent" clusters. The closest result to ours is probably [29], proving that γ-pertubation-stable instances with $\gamma \geq 3$ can be solved using $O(n \log^2 n)$ noisy same-cluster queries [29], and with $O(n)$ queries in the noise-free case. Their result applies to a rather general class of center-based clustering problems (including ours). On the one hand, our algorithms use fewer low-cost noisy comparisons, namely $O(n \log n)$, though for a restricted class of inputs; for balanced clusters, the same-cluster queries are $O(k \log k)$, though we use exact queries. On the other hand, in some cases, same-cluster queries may be harder or more costly than comparisons of distances, and thus the costs may be incomparable in general. Finally, our algorithm using few same-cluster queries uses similar ideas to (coupon collector and the double Dixie cup extension) in [30] for k-mean instances that satisfy a γ-margin property (similar to γ-perturbation-stability, though not equivalent).

2. Model and Preliminary Definitions

An instance is a pair (X,d) where X is a dataset of $n = |X|$ points whose pairwise distances are specified by a non-negative function $d : X \times X \mapsto \mathbb{R}^+$ satisfying symmetry and triangle inequality: $d(x,x) = 0$, $d(x,y) = d(y,x)$, and $d(x,y) \leq d(x,z) + d(z,y)$ for any three points $x, y, z \in X$. The distance function extends naturally to *subsets* of pairwise distances $e = (x,y)$, i.e., subsets of edges. Specifically, for $E = X \times X$ being the set of all edges, and for any $S \subseteq E$ of pairwise distances, we let

$$d(S) := \sum_{e \in S} d(e) \ . \tag{1}$$

2.1. Stable Instances

For a given metric space (X,d) as above and a positive integer k, a clustering is a partition $C = \{C_1, \dots, C_k\}$ of X. The cost of a cluster C_i with respect to a point $x \in X$ is defined as

$$Cost(C_i, x) := \sum_{y \in C_i} d(y,x) \ .$$

The *medoid* (or *centroid*) of each cluster C_i is the point in that cluster minimizing this cost, i.e., $c_i := \arg\min_{x \in C_i} Cost(C_i, x)$, and the cost of a cluster is simply $Cost(C_i) := Cost(C_i, c_i)$. Then, the cost of the clustering is the sum of the cost of each cluster,

$$Cost(C) := \sum_{i=1}^{k} Cost(C_i) .$$

A clustering that minimizes this cost is called optimal k-clustering. (In the literature, this is sometimes called k-medoid. Since in this work we use centroid and medoid interchangeably, we simply use the term k-centroid.)

Definition 1 (γ-perturbation stability)**.** *A γ-perturbation of a metric space (X, d), for $\gamma \geq 1$, is another metric space (X, d') such that, for all $x, y \in X$,*

$$\frac{1}{\gamma} d(x, y) \leq d'(x, y) \leq d(x, y) .$$

For a given positive integer k, a metric space (X, d) is γ-perturbation-stable if there is a k-clustering $C_1^*, \ldots, C_k^*$, which is the unique optimal k-clustering in every γ-perturbation of (X, d).

Remark 1. *Observe first that the above definition restricts the perturbations from being metric too. Moreover, if a metric space is γ-perturbation-stable, then it has a unique optimal solution (regardless of the value of $\gamma \geq 1$).*

For $\gamma \geq 2$, there exists an exact polynomial-time algorithm for the k-clustering problem [3] (see also [2] for algorithms for $\gamma \geq 3$). The algorithm exploits the following key property of such stable instances. Intuitively, in γ-perturbation-stable instances, in the optimal clustering, every point is "much closer" to the centroid of its own cluster than to any other centroid:

Lemma 1 (γ-center proximity [2,3])**.** *Let (X, d) be γ-perturbation-stable and let $C_1^*, \ldots, C_k^*$ be its (unique) optimal solution. Then, for every $x \in X$, with $x \in C_i^*$, and every $j \neq i$, we have that*

$$d(x, c_j^*) > \gamma \cdot d(x, c_i^*)$$

where c_i^*, c_j^* are the respective centroids of C_i^*, C_j^*.

2.2. Comparisons and Errors

We consider the scenario in which the distances $d(x, y)$ between points, as well as those relative to subsets (1), are not directly measurable. Distances can only be *compared* either using a *cheap* but *erroneous* operation or an *expensive* but *always correct* operation. Specifically, we let $O_H(\cdot)$ denote the expensive and reliable operation (oracle) defined as

$$O_H(E_1, E_2) = \begin{cases} +1 & \text{if } d(E_1) > d(E_2) \\ -1 & \text{otherwise} \end{cases} \tag{2}$$

for any two subsets $E_1, E_2 \subseteq E$. The cheap but erroneous operations (oracle) is denoted by $O_L(\cdot)$, and its answers are wrong with a probability of at most p *independently* across all pairs E_1 and E_2. That is, for any two subsets $E_1, E_2 \subseteq E$ as above,

$$\Pr[O_L(E_1, E_2) \neq O_H(E_1, E_2)] \leq p ,$$

and these errors are *persistent*, that is, repeating the same comparison operation $O_L(E_1, E_2)$ multiple times always gives the same result (either always wrong or always correct). We shall sometimes assume that $p < 1/16$ as in prior works [21] in order to apply these results (though our approach/results are generic, in the sense that they can be parameterized by the dislocation of sorting with a generic p or even a generic error model).

Remark 2. *A weaker model would be to allow only comparisons between the weights/distances of two edges, analogously to pairwise comparisons in noisy sorting [11]. Unfortunately, this model seems too weak for the k-medoids clustering problem, since it is not possible to directly compute the optimal centroid of a cluster of more than four nodes. For this reason, we adopt the model that allows for comparing two* sets *of edges.*

Observation 1. *One might be tempted to simulate repeated comparisons between two distances, $d(x,y)$ and $d(x',y')$, via various comparisons of subsets, e.g., $d(\{(x,y),(a,b)\})$ and $d(\{(x',y'),(a,b)\})$. Though this is in principle possible, since comparisons are persistent, whenever the underlying algorithm will query the same subset, the answer is the same of that in our "simulated repeated comparison". This happens, for instance, if the algorithm during its execution needs to compare the following two candidate clustering solutions:*

$$\begin{array}{ccccc} C_1 = \{a,b\}, & C_2 = \{x,y\}, & C_3 = \{x'\} & \textit{and} & C_4 = \{y'\}; \\ C_1' = \{a,b\}, & C_2' = \{x',y'\}, & C_3 = \{x\} & \textit{and} & C_4 = \{y\}. \end{array}$$

In this work, we deliberately choose to not attempt any simulation of "repeated" comparison because of this issue. This has the additional advantage that some of our results might be in principle applicable to different error models where costs might be dependent on the sets, or the error probabilities might depend on the distance values involved.

2.3. Performance Evaluation

We evaluate the performance of our algorithms by distinguishing between the two types of queries we make: If $h(n)$ is the total number of query to O_H and $l(n)$ to O_L, where $n = |X|$, we say that the algorithm has $\langle h(n), l(n)\rangle$ high-low cost complexity [11]. Furthermore, we use the standard notation $O(\cdot)$ and write $O\langle h(n), l(n)\rangle$ to denote $\langle O(h(n)), O(l(n))\rangle$, while with $O(t(n))$ we denote only the total number of high-cost operations (this corresponds to the usual complexity notation where all operations are error free).

2.4. Two Algorithmic Tools

We will use two key results related to our error model, namely, sorting with only erroneous comparisons, and approximate solutions for matroids, respectively.

Lemma 2 (approximate sorting [21])**.** *Given a sequence S of n elements, and a comparison query that flips the answer with a small probability $p < 1/16$, there exists an algorithm with the following performance guarantee. For any confidence parameter $\Delta > 0$,*

1. *The algorithm uses $O(n \log n)$ low-cost queries only (and no high-cost query). Each query compares a pair of elements, and these low-cost queries have an error probability $p < 1/16$. These comparison errors are persistent.*
2. *The algorithm returns an almost sorted sequence $\hat{S}$, where the dislocation of each element is at most $O(\Delta \log n)$ with probability of at least $1 - \frac{1}{n^{\Delta}}$ (the probability depends only on the outcome of the comparisons).*

The dislocation of an element x in a sequence $\hat{S}$ is the absolute difference between the position of x in $\hat{S}$ and the position of x in the correctly sorted sequence S.

Lemma 3 (approximate matroid [11])**.** *Given a matroid (M, F) and two high-low cost oracles in order to compare the elements of M, it is possible to find an $(1+\epsilon)$-approximation of the optimal base using $O\left\langle \frac{1}{\epsilon}(\log n)^2, n \log n \right\rangle$ high-low queries.*

3. Clustering in Stable Instances

In order to describe the algorithms, it is convenient to think of the input (metric space) as the following (complete) induced weighted graph:

Definition 2 (induced graph, spanning forests, clustering)**.** *Given a metric space (X, d), the induced graph is a complete weighed undirected graph $G_X = (X, E_X, w)$ where $E_X = \binom{X}{2}$ and for every edge $e = (x, y)$ we have $w(e) = d(x, y)$. For any tree T spanning all nodes X, and for any subset of edges $F = T \setminus K$, let us denote by $C^{(F)}$ the connected components (nodes) of the forest F. These connected components is a k-clustering whenever we remove $k - 1$ edges from a spanning tree T.*

The known optimal polynomial time algorithms for stable clustering are based on the above tree/forest construction. The simplest of such algorithms is the following one.

Single-Linkage (Naive Algorithm):
- **Phase 1 (Minimum Spanning Tree):** Compute a tree T spanning all nodes X minimizing the overall cost $d(T)$.
- **Phase 2 (Remove Edges):** To obtain a k-clustering, remove the subset K of $k - 1$ edges having a maximum total weight $d(K)$.

It is well-known that this naive algorithm does *not* achieve any (bounded) approximation guarantee even in our γ-pertubation-stable metric instances [2] (see also [15]). The reason is that the criteria for computing the removed edges K does not directly take into account the cost of the resulting clustering. The following algorithm indeed finds the optimum in several stable clustering problems [1–3].

Single-Linkage++ (Correct Algorithm):
- **Phase 1 (Minimum Spanning Tree):** Compute a tree T spanning all nodes X minimizing the overall cost $d(T)$.
- **Phase 2 (Remove Edges++):** To obtain a k-clustering, remove the subset K of $k - 1$ edges resulting in minimal cost $Cost(C^{(T \setminus K)})$.

Theorem 1 ([3])**.** *The Single-Linkage++ algorithm finds the unique optimal k-clustering in every γ-pertubation-stable instance, with $\gamma \geq 2$.*

Both algorithms above consists of two phases, each of them corresponding to solve *exactly* a matroid problem. Unfortunately, in order to have a small number of high-cost operations (Lemma 3), we have to content ourself with *approximate* solutions.

3.1. Warm-Up: Phase 2 of Single-Linkage Algorithm

In order to describe some difficulties and to convey some basic ideas, we shall first consider implementing Phase 2 of Single-Linkage algorithm only. We thus assume that the (exact) Minimum Spanning Tree (MST) has been already computed (or it is given). Observe that Phase 2 of Single-Linkage algorithm boils down to the problem of selecting the top $k - 1$ elements from a set of $n - 1$ edges of the MST, thus a simple matroid. We first observe the following three facts on this task (details below):

1. There is a naive approach using $O(k + (n - k) \log k) = O(n \log k)$ high-cost operations in total (and no low-cost operation).
2. Approximate sorting (Lemma 2) can reduce the total number of high-cost operations to $O(k + d \log k)$, where $d = O(\log n)$.
3. Approximate the matroid (Lemma 3) would directly further improve the above bound for some values of k. Unfortunately, this leads to a solution which is far more costly than the one returned by the algorithm with exact comparisons (or with the previous methods).

As we discuss below, though Single-Linkage has no approximation guarantee, we show that, in some stable instances where it would find an optimal solution, the matroids' approximation makes it compute a solution of much larger cost. This suggests that Phase 2 of this algorithm (removing the $k-1$ edges) is a "fragile" part of the algorithm (also for the more complex Single-Linkage++). Instead, we show in the next section that the first step (computing the MST) can be done approximately without "destroying" the clustering optimality.

In the remainder of this section, we discuss briefly some details on the three items above.

3.1.1. Naive Approach (all at High Cost)

We create a Min Heap with the first $k-1$ elements of E_T, and then iterate over the remaining ones: Every time an element is greater than the root, we replace the root with this element and we re-balance the tree. Finally, we return all the elements in the Min Heap. This strategy takes $O(k+(n-k)\log k)$ high-cost operations.

3.1.2. First Improvement (Combining Low and High Cost Operations)

Using Lemma 2, we can sort the elements in an array using only $O(n\log n)$ low-cost operations, and obtain a sequence with maximum dislocation $d = O(\log n)$. Having sorted the array in this way, we can restrict to the first $m = k-1+d$, elements (since we know that the $k-1$ heaviest elements are in this interval). By applying the previous Min Heap strategy to these m elements, we can find the true $k-1$ heaviest edges. This strategy leads to the following slightly more general result, which we shall use below as a subroutine:

Lemma 4 (find top-t elements)**.** *Given any set of n elements, and any confidence parameter Δ, with probability at least $1-\frac{1}{n^\Delta}$, we can extract the top-t (largest or smallest) elements using $O\langle t+\Delta\log t\log n, n\log n\rangle$ high-low cost operations.*

Proof. According to Lemma 2, with probability at least $1-\frac{1}{n^\Delta}$, we can obtain a sequence where every element has dislocation at most $d = O(\Delta\log n)$ using $O(n\log n)$ low-cost operations only. To find the t largest elements, we consider the first $m = t+d$ elements and apply the Min Heap strategy to these only. This takes $O(t+(m-t)\log t) = O(t+d\log t)$ high-cost operations only as observed above. The overall cost is thus

$$O\langle 0, n\log n\rangle + O\langle t+d\log t, 0\rangle = O\langle t+\Delta\log t\log n, n\log n\rangle\ .$$

□

The previous lemma yields the following result.

Theorem 2. *Phase 2 of Single-Linkage algorithm can be implemented using*

$$O\langle k+\log k\log n, n\log n\rangle$$

high-low cost operations, and its success probability is at least $1-\frac{1}{n}$.

3.1.3. Matroid Approximations Fail

We can use matroids in order to find the best forest F from which we can generate the k-clustering. In particular, we can consider the matroid $M = (E, I)$, where E is the set of all the edges in the MST T and $F = \{e \in 2^{E_X} \mid |e| \le n-k\}$ is the family of forests that can be obtained by removing exactly $k-1$ edges from T. Then, we have that the best forest F is simply the minimum base B of matroid M.

We could then apply Lemma 3 and obtain an $(1+\epsilon)$-approximation of the base using $O\left\langle \frac{(\log n)^2}{\epsilon}, n\log n\right\rangle$ high-low cost complexity. Note that this would improve significantly the previous bound for $k \gg (\log n)^2$. Unfortunately, the next example shows that this approach leads to a solution whose cost is *unbounded* compared to the solution returned by the algorithm.

Example 1 (unbounded error for the approximate matroid strategy)**.** *Let us consider instance X consisting of k groups, $G_1, \dots, G_k$, of m points each and one additional point v. The distance function d has the following values for an arbitrary small $\epsilon > 0$ and some very large L:*

1. *Distances inside the same group are 0, i.e, for any G_i and any two $x, y \in G_i$, we have $d(x,y) = 0$.*
2. *Distances between groups are $1+\epsilon$, i.e, for any two different groups G_i and G_j, and for $x \in G_i$ and $y \in G_j$, we have $d(x,y) = 1+\epsilon$.*
3. *Point v is at distance 1 from points in G_1 and distance L from all other points, i.e.,*

$$d(x,v) = \begin{cases} 1 & \text{if } x \in G_1 \\ L & \text{otherwise} \end{cases} .$$

The corresponding minimum spanning tree is composed by k spanning trees inside each group (each with cost 0), a spanning tree of the group composed of exactly $k-1$ edges each with cost $1+\epsilon$ and one single edge from G_1 to v. Now, if we use a matroid to find a minimum spanning forest, we will find a correct minimum base B^* in which we remove all the $k-1$ edges with cost $1+\epsilon$. This base has a weight of 1 and the cost of the cluster is 1. Now, supposing that we run the $(1+\epsilon)$-approximation algorithm in order to find a minimum base, this gives us a base $\hat{B}$ where we remove from the minimum spanning tree $k-2$ edges of cost $1+\epsilon$ and the edge with cost 1. This is possible since

$$d(\hat{B}) = 1+\epsilon \leq (1+\epsilon)d(B^*) .$$

However, the cost of the corresponding cluster is at least m, since we put two groups in the same cluster.

The above example shows that, even with an arbitrary small approximation, we can have cases where the error in the final solution (clustering) is unbounded.

3.2. Phase 1: Compute a "Good" Spanning Tree

In the previous section, we have assumed that we have given a MST T. This task can be solved with the standard MST algorithm using $O(n^2)$ high-cost operations. A natural question is thus whether there is a strategy that uses less than $o(n^2)$ high-cost operations. This seems difficult if we insist on computing a MST, since any edge in the MST can potentially appear in every position of the sorted sequence of edges.

A key observation is that we do not have to necessarily compute a MST, but we only need a spanning tree from which we can recover the optimal solution removing $k-1$ edges. This is captured by the following definition.

Definition 3 (k-spanning tree)**.** *Let (X,d) a metric space and let $C_1^*, \dots, C_k^*$ be an optimal k-clustering. A tree T of this metric space is k-spanning if every subtree $T[C_i^*]$ induced by C_i^* is connected.*

The definition essentially says that, by removing $k-1$ edges from a k-spanning tree, we obtain a forest whose connected components correspond to the optimal solution. Implicit in the proof of [3] (see also [4]), in order to successfully compute the optimum, it suffices to have a k-spanning tree (and not necessarily a MST one).

Theorem 3 (due to [3])**.** *The modification of Single-Linkage++ algorithm, where, in Phase 1, we compute a k-spanning tree, finds the unique optimal k-clustering in every γ-pertubation-stable instance, with $\gamma \geq 2$.*

We thus consider how to compute a k-spanning tree, instead of an exact MST. We first show the following key corollary of Lemma 1.

Corollary 1. *For any γ-perturbation-stable metric space (X, d), with $\gamma \geq 2$, and with optimal solution $C_1^*, \ldots, C_k^*$ the following holds. If $x \in C_i^*$ and $y \notin C_i^*$, for some cluster C_i^*, then it must hold that*

$$d(x, y) > (\gamma - 1)d(x, c_i^*)$$

where c_i^* is the centroid of C_i^*.

Proof. Since $y \in C_j^*$ for some j different from i, we have

$$d(x,y) \geq d(x,c_j^*) - d(y,c_j^*) > \gamma d(x,c_i^*) - \frac{d(y,c_i^*)}{\gamma} \geq \gamma d(x,c_i^*) - \frac{d(x,y) + d(x,c_i^*)}{\gamma}$$

where the first and last inequality are due to the triangle inequality, and the second inequality follows from Lemma 1. Then, by rearranging the terms, we obtain

$$1 + \frac{1}{\gamma} d(x,y) > (\gamma - \frac{1}{\gamma}) d(x, c_i^*)$$

which implies the corollary after some simplification. □

Now, we are ready to prove a sufficient condition for a tree to be k-spanning, which we shall use below to derive an algorithm good performance guarantee.

Lemma 5 (sufficient condition for k-spanning tree)**.** *Consider a γ-perturbation-stable metric space (X,d), with $\gamma > 2$. Any $(1+\epsilon)$-approximation $\hat{T}$ of the MST T, with $\epsilon \leq \frac{(\gamma-2)}{n-1} w$ and $w := \frac{d_{\min}^{(X)}}{d_{\max}^{(X)}}$, is a k-spanning tree.*

Proof. By contradiction, suppose $\hat{T}$ is not a k-spanning tree. We show that it cannot be a $(1+\epsilon)$-approximation of the MST T. Since T is not k-spanning, there is an optimal cluster C such that $\hat{T}[C]$ is not connected. Let C' be a connected component of $\hat{T}[C]$ that does not contain the centroid of C. Let $x \in C'$ be a node that is connected with some other $y \notin C$. Consider the tree

$$\bar{T} := (\hat{T} \setminus \{(x,y)\}) \cup \{(x,c)\}$$

and observe that, for $d := d(x,c)$, we have

$$\begin{aligned} d(\bar{T}) =& d(\hat{T}) - d(x,y) + d(x,c) \\ <& d(\hat{T}) - (\gamma-1)d(x,c) + d(x,c) = d(\hat{T}) - (\gamma-2)d \end{aligned}$$

where the inequality is due to Corollary 1. Since $d(T) \leq d(\bar{T})$, we have that

$$\begin{aligned} \frac{d(\hat{T})}{d(T)} \geq \frac{d(\hat{T})}{d(\bar{T})} >& \frac{d(\hat{T})}{d(\hat{T}) - (\gamma-2)d} = 1 + \frac{(\gamma-2)d}{d(\hat{T}) - (\gamma-2)d} \\ \geq& 1 + \frac{(\gamma-2)d}{d(\hat{T})} \geq 1 + \frac{(\gamma-2)d_{\min}^{(X)}}{(n-1)d_{\max}^{(X)}}. \end{aligned}$$

where the last inequality follows from $d \geq d_{\min}^{(X)}$ and $d(\hat{T}) \leq (n-1)d_{\max}^{(X)}$. This means that, for $\epsilon := \frac{(\gamma-2)d_{\min}^{(X)}}{(n-1)d_{\max}^{(X)}}$, our tree $\hat{T}$ is not a $(1+\epsilon)$-approximation of the MST T, which contradicts the hypothesis of this lemma. □

By combining Lemma 5 with Lemma 3 (recall that the MST problem is a matroid), we obtain the following result.

Corollary 2. *For any γ-perturbation-stable metric space (X, d), with $\gamma > 2$, a k-spanning tree can be computed using $O\left\langle \frac{n \log^2 n}{(\gamma-2)w}, n \log n \right\rangle$ high-low cost operations with $w = \frac{d_{\min}^{(X)}}{d_{\max}^{(X)}}$.*

Proof. By Lemma 5, it is enough to compute a $(1+\epsilon)$-approximate MST with $\epsilon = \frac{(\gamma-2)}{n} w$. Since the MST is a matroid, Lemma 3 implies the result. □

Remark 3. *Note that the result above depends the values of w and γ. This result provides an alternative method for computing a k-spanning tree, which, in some cases, can be more efficient than using $O(n^2)$ high-cost operations for computing a MST. In particular, the bound in Corollary 2 is better for $\frac{n \log^2 n}{(\gamma-2)w} \ll n^2$, that is, for $\frac{d_{\max}^{(X)}}{d_{\min}^{(X)}} \ll \frac{n}{\log^2 n}(\gamma - 2)$.*

We conclude this section by observing that, in a sense, the above approach cannot be easily improved. In particular, it might be desirable to extend Corollary 2 to some small but *constant* $\epsilon > 0$, which would improve the bound and also do not require any knowledge about w and γ. Unfortunately, the following lemma provides a negative answer. Intuitively, this is because the cost of the MST and the cost of the optimal clustering may be very different.

Lemma 6 (limitations of approximate MST)**.** *For every $\epsilon > 0$, there exists a metric space (X, d) such that, even if we find a spanning tree $\hat{T}$, which is an $(1+\epsilon)$-approximation of the MST T, this tree $\hat{T}$ is not k-spanning. In particular, for every subset K of $k-1$ edges of $\hat{T}$, the corresponding clustering has an unbounded error.*

Proof. Letting $\epsilon > 0$ be arbitrary, we consider the following set of points located on the 1-dimensional Euclidean space:

$$X = \{0, 1, L_2 + 1, L_2 + 2, -L_1, -L_1 - 1\},$$

where L_1 is an arbitrary positive number and $L_2 = \frac{1}{\epsilon} L_1$. We take d as the Euclidean distance (i.e., the absolute difference between these numbers) and consider $k = 3$ clusters.

Note that the cost of the MST T is $d(T) = 3 + L_1 + L_2$ and the cost of the optimal cluster C is $d(C) = 3$. Now, consider the tree $\hat{T}$ obtained by replacing in T edge $(-L_1, -L_1 - 1)$ with edge $(0, -L_1 - 1)$. The cost of this new tree $\hat{T}$ is

$$d(\hat{T}) = 3 + 2L_1 + L_2 = 3 + L_1 + L_2 + \epsilon L_2 \leq (1+\epsilon) d(T),$$

and we can see that $\hat{T}$ is not k-spanning since $T[\{L_1, -L_1 - 1\}]$ is not connected. Since $\hat{T}$ contains three edges of cost L_1, $L_1 + 1$ and $L_2 \geq L_1$, removing any two edges from $\hat{T}$ has a cost $d(\hat{C}) \geq L_1$. Therefore, the approximation guarantee is at least $\frac{d(\hat{C})}{d(C)} \geq \frac{L_1}{3}$, which can be made arbitrarily large by increasing L_1 in this construction. □

3.3. Phase 2 of Single-Linkage++ (Removing the Edges)

In this section, we focus on Phase 2 of the algorithm Single-Linkage++, namely, computing the subset set K^* of $k-1$ edges whose removal (from the tree T computed in Phase 1) yields the (unique) optimal clustering. This phase can also be seen as a matroid, though it is more complex than the one of the Single-Linkage (naive) algorithm because of these two issues:

1. In order to evaluate the cost of a candidate solution (set K of edges to remove), we need to compute the centroids of each cluster;
2. The number of elements in the associated matroid is $N = \binom{n-1}{k-1}$, though we are interested in extracting only one (the optimal K^*).

Another caveat is that, since we shall try all $N = \binom{n-1}{k-1}$ subsets, in order to ensure a correct answer with high probability, we need for the subroutine determining the "correct" centroids to succeed with sufficiently high probability.

Definition 4. *An (α, q)-centroids procedure is an algorithm, which, on inputting any optimal k-clustering $C_1, \dots, C_k$, with probability at least $1 - q$ returns a tuple*

$$(\tilde{c}_1, \dots, \tilde{c}_k) = \mathsf{centroids}(C_1, \dots, C_k)$$

of α-approximate centroids,

$$Cost(C_i, \tilde{c}_i) \leq \alpha Cost(C_i, c_i) \qquad \text{where } c_i = \arg\min_{x \in C_i} Cost(C_i, x)\ .$$

For $\alpha = 1$, we have an exact centroid procedure that returns the optimal centroids c_i of each cluster C_i.

Definition 5. *Given a k-spanning tree T, an α-approximate removal is a subset of $k - 1$ edges such that the k-clustering obtained by removing these edges from T is α-approximate.*

Theorem 4. *Suppose there exists an (α, q)-centroids procedure* centroids *using $O\langle h_{CEN}, l_{CEN}\rangle$ high-low operations. Then, there exists an algorithm, which, with probability at least $1 - q - \frac{1}{N}$, finds an α-approximate removal using $O\langle Nh_{CEN} + \log N, Nl_{CEN} + N \log N\rangle$ high-low operations in total.*

Proof. Given any k-spanning tree T, we simply try all possible candidate subsets K of $k - 1$ edges of T, and find the one that corresponds to the optimum as follows. For each candidate subset $K^{(a)}$, with $a = 1, \dots, N$, we proceed as follows:

1. Consider the corresponding clustering $C_1^{(a)}, \dots, C_k^{(a)}$. This can be done by simply inspecting the nodes of each connected component when removing edges $K^{(a)}$ from T.
2. Run the procedure centroids to compute (some) centroids for this candidate clustering, $(c_1^{(a)}, \dots, c_k^{(a)}) = \mathsf{centroids}(C_1^{(a)}, \dots, C_k^{(a)})$. This gives a set of edges from centroid $c_i^{(a)}$ to the other elements of the cluster $C_i^{(a)}$, for all clusters:

$$E^{(a)} = \{(x, y) \in E \mid x \in C_i^{(a)} \text{ and } y = c_i^{(a)} \text{ for some } i\}\ .$$

We now observe two key properties of these (candidate) centroids. For each cluster $C_i^{(a)}$, consider the sum of the distances to its candidate centroid $c_i^{(a)}$, and observe that

$$Cost(C^{(a)}) \leq d(E^{(a)}) = \sum_{i=1}^{k} \sum_{x \in C_i^{(a)}} d(x, c_i^{(a)})\ ,$$

since $c_i^{(a)}$ may not be the optimal centroid for $C_i^{(a)}$. Moreover, for the optimal $K^{(opt)}$ and the corresponding optimal k-clustering $C_1^{(opt)}, \dots, C_k^{(opt)}$, with a probability of at least $1 - q$, centroids returns an α-approximate centroid $c_i^{(opt)}$ for each cluster $C_i^{(opt)}$, for all k clusters (Definition 4). That is, the truly optimal centroid $c_i^* = \arg\min_{x \in C_i^{(opt)}} Cost(C_i^{(opt)}, x)$ satisfies

$$Cost(C_i^{(opt)}, c_i^{(opt)}) \leq \alpha Cost(C_i^{(opt)})$$

and therefore

$$d(E^{(opt)}) \leq \alpha Cost(C^{(opt)}) \ .$$

This implies that the candidate $K^{(min)} = \arg\min_{K^{(a)}} d(E^{(a)})$ which minimizes $d(E^{(a)})$ among the N candidates is indeed an α-approximate removal:

$$Cost(C^{(min)}) \leq d(E^{(min)}) \leq d(E^{(opt)}) \leq \alpha Cost(C^{(opt)}) \ .$$

The total high-low cost to create the list of all N elements is $O\langle Nh_{CEN}, Nl_{CEN}\rangle$. As observed above, with a probability of at least $1-q$, the minimum in this list yields an α-approximate removal. Each pairwise comparison between two elements, say $d(E^{(a)})$ and $d(E^{(b)})$, can be done with a *single* call to our oracles, either $O_L(E^{(a)}, E^{(b)})$ or $O_H(E^{(a)}, E^{(b)})$. We can thus apply Lemma 4 and, with a probability of at least $1 - \frac{1}{N}$, find the minimum element using $\langle \log N, N \log N\rangle$ high-low cost operations. The overall high-low cost to find an α-approximate removal is thus

$$O\langle Nh_{CEN} + \log N, Nl_{CEN} + N \log N\rangle \ .$$

Finally, by the union bound, the probability that we actually find an α-approximate removal is at least $1 - q - \frac{1}{N}$. □

From the previous result, we can focus on constructing a suitable procedure centroids.

3.3.1. Exact Centroids and Exact Solutions

We begin by considering procedures to compute *exact* centroids ($\alpha = 1$), which automatically lead to *optimal* k-clustering. The first (naive) approach is simply to use only high-cost (exact) operations.

All at High-Cost.

We want to find an implementation of centroids using only high-cost operations. For each cluster C_i, and two candidate points $x, y \in C_i$, we can compute

$$E_x = \{(x,z) \in E | z \in C_i\} \qquad \text{and} \qquad E_y = \{(y,z) \in E | z \in C_i\} \tag{3}$$

and, by calling $O_H(E_x, E_y)$, we can decide which of these two points is a better centroid. By repeating this for all the points in one cluster, and for all the clusters, we can find the optimal k centroids using only $O(n)$ high-cost operations in total. This shows the following.

Observation 2. *The best subset K^* can be found using $O\langle Nn, 0\rangle$ high-low operations in total, where $N = \binom{n-1}{k-1}$.*

We next investigate how to reduce the number of high-cost operations by introducing low-cost ones (still aiming for exact centroids).

Using low cost operations

Similarly to what we discussed above, our low-high cost operations are calls to our oracles $O_L(E_x, E_y)$ and $O_H(E_x, E_y)$, where $x, y \in C_i$ and E_x, E_y are as in (3). According to Lemma 4, for each cluster C_i consisting of $n_i = |C_i|$ elements, we can find its optimal centroid (the minimum) using $O\langle \Delta_i \log n_i, n_i \log n_i\rangle$ high-low cost operations with a probability of at least $1 - \left(\frac{1}{n_i}\right)^{\Delta_i}$. In order to optimize the overall success probability, we set

$$\Delta_i = \Delta \frac{\log n}{\log n_i} \tag{4}$$

for a suitable Δ independent of i. The resulting procedure centroids has the following performance guarantees:

1. *Success probability at least* $1 - k/n^{\Delta}$. For each cluster C_i, the probability of finding the optimal centroid is at least $1 - (1/n)^{\Delta}$ since $n_i^{\Delta_i} = 2^{\Delta_i \log n_i} = 2^{\Delta_i \log n} = n^{\Delta}$. By the union bound over all k clusters, the success probability is at least $1 - k/n^{\Delta}$.
2. *High-cost operations* $O\big(k \log \frac{n}{k}\big)$. The total high-cost of centroids for computing all centroids of any given partition $C_1, \dots, C_k$ of X is
$$O\left(\sum_{i=1}^{k} \Delta_i \log n_i\right) = O(k\Delta \log n)\ .$$
3. *Low-cost operations*. We observe that this is
$$O\left(\sum_{i=1}^{k} n_i \log n_i\right) = O\Big(n \log \frac{n}{k}\Big)\ ,$$
where these bounds can be easily obtained using the Lagrange Multiplier.

We have thus obtained an $(1, q)$-centroids procedure with $q \le k/n^{\Delta} \le 1/n^{\Delta - 1}$, since $k \le n$. Taking $\Delta = 3 \log N = O(k \log n)$, we get the following.

Observation 3. *The best subset K^* can be found using $O\langle N(k \log n)^2, Nn \log \frac{n}{k}\rangle$ high-low operations in total, where $N = \binom{n-1}{k-1}$.*

Remark 4. *Note that the bottleneck in the number of high-cost operations is in the repeated application of procedure* centroids *for all N possible candidate subsets of edges. Even an improved implementation of* centroids *using $O(1)$ high-cost operations would still result in an overall $O(N)$ high-cost. On the other hand, once all possible $N = \binom{n-1}{k-1}$ candidate subsets E_Q have been created, one could find the best one using only $O\langle \log N, N \log N\rangle$ high-low cost via Lemma 4. Thus, a much smaller $O(\log N)$ high-cost is sufficient, as compared to the $O(N)$ high-cost.*

In the next section, we shall reduce significantly the high-cost by introducing an approximate computation of the clusters which uses *zero* high-cost operation. We shall see that this results in a (small) factor approximation in the solution.

3.3.2. Approximate Centroids and Approximate Solutions

As we have seen in the previous section (Remark 4), it is quite easy to reduce the number of high-cost queries used to find the best subset once we have computed the function centroids for every partition. The problem is that, even if we improve the high-cost cost of the function centroids to be a constant, since we apply it to $O(N)$ different partitions, we still have in total an expensive high-cost. For this reason, we now consider a method to compute *approximate* centroids without using any high-cost operation, that is, an approximate implementation of centroids with only low-cost operations.

Though this may not give us an exact solution, we shall prove that this shall lead to a 2-approximate clustering. The next result is central for this section.

Lemma 7 (centroid approximation)**.** *Given a metric space (X, d), for any cluster $C_i \subseteq X$, and for any $x \in C_i$, it holds that*
$$Cost(C_i, x) \le \left(1 + \frac{n_i - 2}{n_i - D_i(x)}\right) Cost(C_i)$$
where $n_i = |C_i|$ and $D_i(x)$ is the number of points y in C_i which are a better centroid than x,
$$D_i(x) := |\{y \in C_i \mid Cost(C_i, y) < Cost(C_i, x)\}|\ .$$

Proof. Let $c_i^* = \arg\min_{z \in C_i} Cost(C_i, z)$ be the optimal centroid of C_i and the corresponding optimal cost is $O = Cost(C_i) = Cost(C_i, c_i^*)$. Observe that, for every $c_i' \in C_i$, we have

$$\begin{aligned} Cost(C_i, c_i') &= \sum_{y \in C_i} d(y, c_i') = \sum_{\substack{y \in C_i \\ c_i' \neq y}} d(y, c_i') \\ &\leq \sum_{\substack{y \in C_i \\ c_i' \neq y}} (d(y, c_i^*) + d(c_i^*, c_i')) \\ &= (\sum_{\substack{y \in C_i \\ c_i' \neq y}} d(y, c_i^*)) + (n-1) d(c_i^*, c_i') \\ &= \sum_{y \in C_i} d(y, c_i^*) + (n-2) d(c_i', c_i^*) \\ &= O + (n-2) d(c_i', c_i^*)\ . \end{aligned}$$

Now, let $\hat{O} = Cost(C_i, x)$, and let us partition C_i into the following two sets:

$$\begin{aligned} H_- &:= \{y \in C_i \mid Cost(C_i, y) < Cost(C_i, x)\}\ , \\ H_+ &:= \{y \in C_i \mid Cost(C_i, y) \geq Cost(C_i, x)\}\ . \end{aligned}$$

By definition $|H_-| = D_i(x)$ and $|H_+| = n_i - D_i(x)$. Moreover, for any $y \in H_+$, we have that

$$\hat{O} = Cost(C_i, x) \leq Cost(C_i, y) \leq O + (n_i - 2) d(y, c_i^*)$$

that is

$$d(y, c^*) \geq \frac{\hat{O} - O}{n_i - 2}\ .$$

Using the fact that the distances between two points are nonnegative, we get the following lower bound on the optimal cost of the cluster,

$$\begin{aligned} O = \sum_{y \in C_i} d(y, c_i^*) &= \sum_{y \in H_-} d(y, c_i^*) + \sum_{y \in H_+} d(y, c_i^*) \geq \sum_{y \in H_+} d(y, c^*) \\ &\geq \sum_{y \in H_+} \frac{\hat{O} - O}{n_i - 2} = |H_+| \frac{\hat{O} - O}{n_i - 2} = \frac{n_i - D}{n_i - 2} (\hat{O} - O) \end{aligned}$$

and, by rearranging the terms, we obtain

$$\hat{O} \leq \left(1 + \frac{n_i - 2}{n_i - D_i(x)}\right) O,$$

which completes the proof. □

Intuitively, by combining the lemma above with Lemma 2, our approximation depends on the dislocation $D_i(x)$ that the sorting procedure (using only low-cost operations) guarantees when applied to the $n_i = |C_i|$ elements of cluster C_i. This leads to the following result.

Lemma 8. *For any parameter $\Delta \geq 0$, there exists an (α, q)-centroids procedure with using $O\langle 0, n \log n \rangle$ high-low cost operations, with*

$$q \leq k \cdot n^{-\Delta} \qquad \text{and} \qquad \alpha \leq 1 + \frac{n_{\min} - 2}{n_{\min} - O(\Delta \log n)}$$

where $n_{\min}$ is the size of the smallest cluster in the optimal k-clustering $C_1, \ldots, C_k$, that is, $n_{\min} = \min_i\{n_i\}$ for $n_i = |C_i|$.

Proof. We choose Δ_i as in the previous subsection, eq. (4), so that $\Delta_i \log n_i = \Delta \log n$ for a fixed $\Delta \geq 0$ independent of i. By Lemma 2, for each cluster C_i, with probability at least $1 - n_i^{-\Delta_i} = 1 - n^{-\Delta}$, we can find an element $x \in C_i$ such that $D_i(x) \leq O(\Delta_i \log n_i) = O(\Delta \log n)$. By the union bound, the overall probability of computing such points for all k clusters is at least $1 - k \cdot n^{-\Delta}$, hence $q = k \cdot n^{-\Delta}$. Note also that we use only low-cost operations, in particular, the total low-cost operations are $\sum_{i=1}^{k} O(n_i \log n_i) = O(n \log n)$.

We next argue about the approximation guarantee. By Lemma 7, for each cluster C_i, the corresponding centroid x of our procedure satisfies

$$\frac{Cost(C_i, x)}{Cost(C_i)} \leq 1 + \frac{n_i - 2}{n_i - D_i(x)} = 1 + \frac{n_i - 2}{n_i - O(\Delta \log n)} .$$

The latter is maximized when n_i is the smallest, that is, for $n_i = n_{\min} = \min_i\{n_i\}$. □

3.4. Dynamic Programming

A dynamic programming to find the best clustering giving a k-spanning tree can be found in [3]. In this section, we shall simplify their dynamic programming (which works for a more general class of problems) in order to adapt it to our oracles model.

In general, it is rather difficult to work with "approximate" solutions with dynamic programming, since the errors accumulate during the execution of the algorithm, and we do not have a clear way to control this (for a rare example, a dynamic programming algorithm working with errors, see [31]). Therefore, our approach will be to always produce correct intermediate results, while still trying to limit the number of high-cost operations.

3.4.1. Basic Notation and Adaptations

First of all, observe that the algorithm in [3] applies to a more general problem in which we have a generic function $g(u, d)$ applied to the distance function d, and a function $f(c)$ that indicates the cost of having c as a centroid. Therefore, we instantiate these two functions according to our problem, that is, we consider $g(u, d) \equiv d$ and $f(c) \equiv 0$.

Before starting with the actual algorithm, we have to transform our tree $T_X = (X, E_X)$ in a binary tree $B_X = (V_B, E_B)$ using the procedure described in [3] (select an arbitrary node as the root and then add nodes in a top-down fashion to get a binary tree). In the following, we use X as the set of the original points and U as the set of the points that we add so that $V_B = X \cup U$. One can easily check that, for a node x that has $c(x)$ children, we have to add $c(x) - 2$ nodes if x has more than two children and one if $c(x)$ equals one. Note that the number of nodes is $|V_B| = |X \cup U| = O(n)$, and thus we can still denote with n the input size and count the number of queries as a function of n.

We define L the set that contains all the leaves of B_X, and we create two sequences S and S' of the nodes in reverse breadth first order from the root, where S' contains only the non leaf nodes.

Then, we build a data structure $Cost$ where, at each key (u, j, c), we encode information to evaluate the optimal solution for partitioning the subtree B_u rooted at u into $j \leq k$ clusters with c being the centroid of node u (here B_u is the set of nodes in subtree rooted at u). In particular, for every key (u, j, c), the data structure $Cost$ contains a set $E(u, j, c)$ of edges of the tree that represents the connections of nodes in subtree B_u to their clusters in that particular solution, and an integer variable $s(u, j, c)$ that represents how many of these centers are from U, and thus how "illegal" is the solution (we have to add this variable since we set f to be always equal to zero). This is in contrast with the original dynamic programming in [3], where we associate only a single integer to every key.

The last thing we have to do is to modify the output of the algorithm. In [3], it how to compute the total cost of the perfect clustering is the only thing mentioned. This of course can not be done with our model since we have no information about the absolute distance between the points. Thus, we have to further modify it in order to obtain the cluster centroid as output of the algorithm, so that we can easily compute the cluster by associating each point to the closest centroid. In order to achieve that, we create a second data structure *Centroid* indexed by keys (u, k, c), where each entry contains a set of the best centroids for partitioning the sub tree rooted at u in k clusters.

3.4.2. The Actual Algorithm

We are now in a position to describe the dynamic programming algorithm based on the binary tree B_X and the two data structures described above.

Initialization

Similarly to [3], we have to initialize the leaves. In particular, we associate in the data structure *Cost*, for each key $(u, 1, c)$, where $u \in L$ and $c \in X$, a list that contains only the edge (u, c) and the variable s is set to 1 if $c \in U$, and zero otherwise. For all the key (u, j, c), with $u \in L$, $c \in X$ and $j > 1$, we associate an empty set and 1 to the variable s. Furthermore, in the second data structure *Centroid*, we associate an empty set to all the previous keys.

The Algorithm

We traverse the key in the order for j from 1 to k, for node $u \in S'$ and $c \in S$, and we compute the optimum for each key (u, j, c). In the original dynamic programming [3], this is done by searching for the couple $(l, j_1, c_1) + (r, j_2, c_2)$ that corresponds to the best value, where l and r are the two children of u in the binary tree. Since we do *not* have access to the individual value for a couple of keys, we create a list containing all possible couples $(l, j_1, c_1) + (r, j_2, c_2)$, and then we extract the best solution. In order to perform the latter task, we have to be able to compare two candidate couples for the same left and right child l and r, say

$$(l, j_1', c_1') + (r, j_2', c_2') \qquad \text{and} \qquad (l, j_1'', c_1'') + (r, j_2'', c_2'')\,. \tag{5}$$

We show that this can be done by using a single call of our oracle in the following way. For a generic couple, $(l, j_1, c_1) + (r, j_2, c_2)$, the data structure associated with the two elements of this couple consists of

$$\begin{aligned}(E(l, j_1, c_1), s(l, j_1, c_1)) &= Cost(l, j_1, c_1)\\ (E(r, j_2, c_2), s(r, j_2, c_2)) &= Cost(r, j_2, c_2)\end{aligned}$$

To express the dynamic programming, it is useful to introduce the following auxiliary functions depending on (u, c):

$$\begin{aligned}E((l, j_1, c_1) + (r, j_2, c_2)) &:= E(l, j_1, c_1) \cup E(r, j_2, c_2)\\ s((l, j_1, c_1) + (r, j_2, c_2)) &:= s(l, j_1, c_1) + s(r, j_2, c_2)\\ f_{c,\hat{c}} &:= \begin{cases}1 & \text{if } c = \hat{c} \text{ and } c \in U\\ 0 & \text{else}\end{cases}\end{aligned}$$

We are now in a position to express how to choose between the two couples in (5). Consider

$$\begin{aligned}E' &= E((l, j_1', c_1') + (r, j_2', c_2'))\\ s' &= s((l, j_1', c_1') + (r, j_2', c_2')) - f_{c,c_1'} - f_{c,c_2'}\\ E' &= E((l, j_1', c_1') + (r, j_2', c_2'))\\ s'' &= s((l, j_1'', c_1'') + (r, j_2'', c_2'')) - f_{c,c_1''} - f_{c,c_2''}\end{aligned}$$

If $s' \neq s''$, then we choose the couple corresponding to the minimum between these two values. If instead $s' = s''$, then we evaluate the corresponding cost by performing an oracle query to E and E' after removing all edges incident to some point in U. In this case, we return the couple with a minimum cost.

The above criterion extends naturally to subsets of couples, and it will replace the basic 'min' operation in the original dynamic programming formulation [3].

Having determined the optimal couple $(l, j_1^*, c_1^*) + (r, j_2^*, c_2^*)$ for the key (u, j, c), we set the corresponding content of data structures *Cost* and *Centroid* as follows:

$$\begin{aligned} E(u,j,c) =& E((l,j_1^*,c_1^*) + (r,j_2^*,c_2^*)) \cup \{(u,c)\} \\ s(u,j,c) =& s((l,j_1^*,c_1^*) + (r,j_2^*,c_2^*)) + f_{c,c} \\ Centroid(u,j,c) =& Centroid(l,j_1^*,c_1^*) \cup Centroid(r,j_2^*,c_2^*) \cup \{c_1^*, c_2^*\} \,. \end{aligned}$$

After we have filled all possible keys (u, j, c), we simply have to find the best value of c^* among all keys $(root, k, c)$ using the same method, and return the set associated with $Centroid(root, k, c^*)$.

Analysis of High-Low Cost

The total running time is given by the central part of the algorithm. Let $\ell_{u,j,c}$ be the length of the list that contains the candidate couples associated with the key (u, j, c). We have to extract the minimum every time, which can be done using $O\langle \log \ell, \ell \log \ell \rangle$ the high-low cost, for $\ell = \ell_{u,j,c}$ (Lemma 4). Since the length $\ell_{u,j,c}$ is $O(j|B_{u_l}||B_{u_r}|) = O(kn^2)$, and the overall high-low cost is

$$\begin{aligned} &\sum_{j=1}^{k} \sum_{u \in S'} \sum_{c \in S} O\langle \log \ell_{u,j,c}, \ell_{u,j,c} \log \ell_{u,j,c} \rangle = \\ &\sum_{j=1}^{k} \sum_{u \in S'} \sum_{c \in S} O\langle \log(kn^2), kn^2 \log(kn^2) \rangle = \\ &\qquad O\langle kn^2 \log n, k^2 n^4 \log n \rangle \,. \end{aligned}$$

Though the high-cost is worse than the previous result, for k large, this bound is exponentially better in terms of low-cost. Finally, observe that we can invoke Lemma 4 with parameter $\Delta = \Theta(\log n)$ so that each of these operations fails with a probability of at most n^{-a} for sufficiently large a, so that the overall procedure succeeds with high probability (union bound), and the overall complexity is as above (again using $\ell_{u,j,c} = O(kn^2)$).

4. Same Cluster Queries

In this section, we investigate so-called *same-cluster queries*: Given two points in our metric space (X, d), the corresponding same-cluster query returns "yes" if and only if the two points are in the same cluster in the optimal solution. Of course, this type of query is extremely powerful, though rather difficult to implement in general. In Section 4.1, we first show that, under certain conditions, we can simulate same-cluster queries in our model. In Section 4.2, we present algorithms which find the optimal k-clustering using "few" same-cluster queries—these use "black-box" and thus the algorithm can be applied to any setting where such queries are available but perhaps even more expensive than comparisons in our model.

4.1. Small-Radius and Same-Cluster Queries

Observe that Corollary 1 says that, if two points are *not* in the same cluster, then their distance must be larger than $(\gamma - 1)$ times the distance of one point to its optimal centroid. This condition does not automatically give a same-cluster query because it requires the knowledge of the optimal centroids, and it does not give an 'if and only if" characterization. As we show below, under some additional assumptions, we can obtain a *characterization* for two points being in the same cluster which does not involve the optimal centroids (see the next definition and lemma). This in turn implies that we can build a same-cluster query.

Definition 6 (γ-radius optimal clusters)**.** *Let $d_{\max}^{(centroid)}$ be the maximum radius of the clusters in the optimal solution, that is, $d_{\max}^{(centroid)} = \max_{x \in X} d(x, c^*_{i(x)})$ where $c^*_{i(x)}$ is the centroid, the cluster containing x in the optimal solution. We say that a γ-perturbation-stable metric space (X, d) has γ-radius optimal clusters if*

$$2d_{\max}^{(centroid)} \leq (\gamma - 1)d_{\min}^{(X)}$$

where $d_{\min}^{(X)} = \min_{x \neq y} d(x, y)$ is the minimum distance between two points.

Lemma 9 (same cluster condition)**.** *For any γ-perturbation-stable metric space (X, d) which has γ-radius optimal clusters, with $\gamma \geq 2$, the following holds. For every $x, y \in X$,*

$$d(x, y) \leq (\gamma - 1)d_{\min}^{(X)} \qquad \Leftrightarrow \qquad x, y \in C^*_i \text{ for some } i\,,$$

where $C^*_1, \ldots, C^*_k$ is the optimal solution.

Proof. We first prove the ($\Rightarrow$) direction:

$$d(x, y) \leq (\gamma - 1)d_{\min}^{(X)} \Rightarrow d(x, y) \leq (\gamma - 1)d(x, c^*_{i(x)}) \Rightarrow y \in C^*_{i(x)}\,,$$

where $C^*_{i(x)}$ and $c^*_{i(x)}$ are the cluster and the centroid of x in the optimal solution, respectively, and the second implication is given by the contrapositive of Corollary 1.

In order to prove the other direction ($\Leftarrow$), consider arbitrary x and y in the same cluster with centroid c^* and observe that

$$d(x, y) \leq d(x, c^*) + d(y, c^*) \leq \frac{d_{\min}^{(X)}(\gamma - 1)}{2} + \frac{d_{\min}^{(X)}(\gamma - 1)}{2} = d_{\min}^{(X)}(\gamma - 1)\,,$$

where the first inequality is by triangle inequality (metric space) and the second by the γ-radius optimal clusters assumption (Definition 6). □

Implementing a Same-Cluster Query

According to Lemma 9, our same-cluster query, upon inputting two points x and y, is simply checking whether $d(x, y) \leq (\gamma - 1)d_{\min}^{(X)}$ holds. We are now in a position to construct our same cluster query. In order to do so, we shall allow more general queries to compare pairwise distances multiplied by *scalars*, that is,

$$O_H^{(\alpha)}(e_1, e_2) = \begin{cases} +1 & \text{if } d(e_1) > \alpha d(e_2) \\ -1 & \text{otherwise} \end{cases} \tag{6}$$

for any two pairs of points $e_1 = (x_1, y_1)$ and $e_1 = (x_2, y_2)$. The cheap but erroneous counterpart $O_L^{(\alpha)}(\cdot)$ is defined similarly, and its answers are wrong with the same probability $p \leq 1/16$ as in our oracle $O_L(\cdot)$. Observe that we can check whether $d(x, y) \leq (\gamma - 1)d_{\min}^{(X)}$ by one call to the above oracle with $e_1 = (x, y)$ and $e_{\min} = \arg\min_{x' \neq y'} d(x', y')$ being from the pair of points at a minimal distance, $d_{\min}^{(X)} = d(e_{\min})$. Lemma 9 thus implies the following:

Lemma 10. *For every γ-perturbation-stable metric space (X, d) which has γ-radius optimal clusters, $\gamma \geq 2$, the following query*

$$SCQ_H^{(\gamma)}(x, y) := O_H^{(\gamma-1)}((x, y), e_{\min})$$

is an exact same-cluster query, where $e_{\min} = \arg\min_{x' \neq y'} d(x', y')$. The analogous cheap but erroneous query $SCQ_L^{(\gamma)}(x, y) := O_L^{(\gamma-1)}((x, y), e_{\min})$ is a same-cluster query that is correct with probability at least $1 - p$.

Note that these queries are similar to those used in [8,9], though these works consider "triples" of points, $e_1 = (x, y)$ and $e_2 = (x, z)$, and the error model is different. We next discuss how expensive is to build such queries and whether they can be built based on our initial query model.

Observation 4. *Observe that the pair of points $e_{\min} = \arg\min_{x' \neq y'} d(x', y')$ can be determined* once and for all *using $O\langle \log n, n^2 \log n \rangle$ high-low cost (Lemma 4). Therefore, any algorithm using SCQs only requires this additional high-low cost to be implemented.*

Observation 5. *For $\gamma = 2$, the above query is a simple query between distances. Moreover, for any integer $\gamma \geq 2$, the query can be simulated by considering a multiset of pairs:*

$$SCQ_{type}(x, y) := O_{type}(\{(x, y)\}, E_\gamma) \qquad\qquad type \in \{L, H\}$$

where E_γ consists of $\gamma - 1$ copies of $e_{\min}$.

Assuming an oracle like in (6) is available, and each exact/erroneous SCQ corresponds to one high/low cost operation. Alternatively, if we allow multiset comparisons in our original oracle model, the same also holds true (without directly appealing on the oracle in (6)). Finally, for $\gamma = 2$, we simply can use the original oracle.

Remark 5. *Though every γ-perturbation-stable instance with $\gamma \geq 2$ is also 2-perturbation-stable instance, since we require the additional 'γ-radius optimal clusters' condition there might be instances that satisfy the hypothesis to obtain SCQs above only for some $\gamma > 2$.*

In the sequel, we shall devise algorithms that attempt to use as few as possible SCQs. In their analysis, we shall then account explicitly for the overall number of such queries and the rest of high-low cost operations derived from queries to our original oracle model. Whenever the original oracle model suffices to simulate the SCQ_H and/or SCQ_L, each SCQ corresponds to a single operation (high or low cost).

4.2. An Algorithm Using Few SCQs

We next describe a simple algorithm which uses much less SCQs, though a comparable number of high-low cost operations. The algorithm works for certain instances, and its performance is summarized in the following theorem.

Theorem 5. *For any $\gamma \geq 2$ and for any positive integer n_1, there exists an algorithm with the following performance guarantees. The algorithm computes the optimal k-clustering, with a probability of at least $1 - \frac{3}{n}$, on input any γ-perturbation-stable metric space (X, d) which has γ-radius optimal clusters, and such that every cluster in the optimal solution has a size of at least n_1. Moreover, the algorithm uses only the following number of exact same-cluster queries and high-low cost operations parameterized in $p_1 = \frac{n_1}{n}$:*

1. *The algorithm uses $O(\frac{k \log k}{p_1})$ exact same-cluster queries, which is independent of n for approximately balanced clusters, that is, $n_1 = \Theta(n/k)$.*

2. *The additional high-low cost operations used by the algorithm is*

$$O\left\langle \frac{(\log n)^2}{p_1}, (\frac{\log k}{p_1} + n) k \log n \right\rangle .$$

4.2.1. The Algorithm

We first observe that Corollary 1, in addition to the condition used to simulate same-cluster queries in Lemma 9, implies another useful property that we can use to reduce the number of high-cost queries. Intuitively, under the same conditions of Lemma 9, we have that every point is closer to all points in its own cluster than to any of the points outside.

Corollary 3. *For any γ-perturbation-stable metric space (X, d) that has γ-radius optimal clusters, with $\gamma \geq 2$, the following holds. For every optimal cluster C^*, every $x, y \in C^*$ and $z \notin C^*$, it holds that $d(x, y) < d(x, z)$.*

Proof. Let x, y, z be arbitrary points of X, with x, y in the same cluster C^* with centroid c^*. Then, we have that

$$d(x, y) \leq (\gamma - 1) d_{\min}^{(X)} \leq (\gamma - 1) d(x, c^*) < d(x, z)$$

where the last inequality is given by Corollary 1. □

Remark 6 (main intuition)**.** *Corollary 3 means that, in order to identify a cluster C_i^*, instead of its optimal centroid, we can simply determine an arbitrary point x_i and assign this point as a (representative) centroid of that cluster C_i^*, for all k clusters. Then, we can map every other points to the closest (representative) centroid.*

The above result suggests the following natural algorithm (the implementation of the single steps is described below together with the analysis):

Coupon–Collector Clustering:

1. Extract random points from X until we obtain one centroid for each cluster;
2. Having one centroid per cluster, extract an additional m random points to get $C = \Theta(\log n)$ centroids per cluster;
3. Map all the remaining points in X to the closest centroid.

The idea is to pick random points from X until we have a different centroid for each cluster, and then simply map all the remaining points to the closest centroid. In the first step, for every new point we extract, we use the (exact) same-cluster query SCQ_H to determine whether this point belongs to one of the clusters for which we already have found one point.

Analysis of the First Step

The analysis is based on the well-known Coupon Collector's Problem (CCP) [32,33]. In this step, we can safely extract points *with* a replacement as in the CCP problem (see Lemma 11 below) since extracting the same point more than once has no effect on the stopping condition (all clusters have at least one point, and having more than one point or the same point several times does not affect this step).

Lemma 11. *Consider the process in which we repeatedly draw a coupon from a finite collection of k coupons according to a probability distribution $\mathbf{p} = (p_1, p_2, \ldots, p_k)$, meaning that, at every step, $p_h \in (0, 1)$ is the probability of picking the h-th coupon. Then, the following bounds on the expected number of steps $\mathcal{C}(\mathbf{p})$ required to collect at least c copies of each of the k coupons are known (for $p_1 \leq p_2 \leq \cdots \leq p_k$). For the (approximately) uniform case, that is, $p_i = \Theta(\frac{1}{k})$,*

$$\mathcal{C}(\mathbf{p}) = \Theta(k \log k)$$

and, more in general, the following upper bounds hold (see [32])

$$\mathcal{C}(\mathbf{p}) \le \frac{H_k}{p_1} = O(\frac{\log k}{p_1}); \qquad \mathcal{C}(\mathbf{p}) \le \sum_{j=1}^{k} \frac{1}{j \cdot p_j}; \qquad \mathcal{C}(\mathbf{p}) \le \sum_{j=1}^{k} 1 / \left(\sum_{\ell=1}^{j} \frac{1}{p_\ell} \right)$$

where H_k is the k-th harmonic number.

Lemma 12. *The expected number of exact SCQs in Step 1 of the Coupon–Collector Clustering algorithm is $O(k\mathcal{C}(\mathbf{p}))$, where $\mathbf{p} = (p_1, p_2, \ldots, p_k)$ is given by $p_i = n_i/n$ and $n_i = |C_i^*|$, and $C_1^*, \ldots C_k^*$ being the optimal k-clustering,*

Proof. We simply observe that, for each point, we extract from X that the probability that it belongs to cluster C_i^* is $n_i/n = p_i$. Moreover, for every newly picked point, we have to perform a same-cluster query SCQ_H with the centroid of each cluster that we have already found in order to be sure that the new point belongs really to a new cluster. This costs at most the number of clusters k. □

Observation 6. *The expected number of exact SCQs in Step 1 is*

$$O(k\mathcal{C}(\mathbf{p})) = O\left(k \min\left\{ \frac{\log k}{p_1}, \sum_{j=1}^{k} \frac{1}{j \cdot p_j}, \sum_{j=1}^{k} 1 / \left(\sum_{\ell=1}^{j} \frac{1}{p_\ell} \right) \right\} \right) \le O\left(\frac{k \log k}{p_1} \right).$$

Analysis of the Second Step

In this step, we exact points *without replacements*. The analysis is essentially based on the generalization of the CCP problem in which we want to obtain $C \ge 1$ copies of each of the k coupons. This problem is known as the *double Dixie cup problem* [33,34]. The variant in which points are chosen *without* replacement has been recently used for a similar clustering problem in [30]. We borrow their concentration result:

Lemma 13 (Theorem 3.3 in [30])**.** *For any subset of k clusters of size $n_1 \le n_2 \le \cdots \le n_k$, the following holds. Suppose m points are sampled uniformly at random without replacement from X. Denote S_i as the number of samples filled in cluster i after this process. Then, the probability that each cluster is filled with at least $C = \frac{mp_1}{2}$ points is bounded as*

$$\Pr\{\min S_i \ge C\} \ge 1 - k \exp(-\frac{C}{4}), \tag{7}$$

where $p_1 = \frac{n_1}{n}$ and n_1 is the size of the smallest cluster among the clusters under consideration.

Corollary 4. *For any $C \ge 8 \log n$, by sampling at least $m = 2C/p_1 = 2Cn/n_1$, the probability that in every cluster we have selected at least C points is at least $1 - \frac{k}{n^2} \ge 1 - \frac{1}{n}$.*

Given that in each each cluster we select at least C points, the only thing that we have to do is to map all selected points to their correct clusters. For an arbitrary point x, we have to find the closest centroid. By Lemma 4, this requires $O\langle \Delta \log k, k \log k \rangle$ high-low cost for each point x, and the probability of success is at least $1 - \frac{1}{k^\Delta}$. By the union bound, all $m = \Theta(\frac{n \log n}{n_1})$ points are then mapped to the correct cluster with probability at least $1 - \frac{m}{k^\Delta}$. Therefore, by Corollary 4, the union bound implies that Phase 2 terminates with at least C points in each cluster of the optimal solution with a probability at least $1 - \frac{1}{n} - \frac{m}{k^\Delta}$. For $\Delta = \frac{\log(nm)}{\log k}$, we have $\frac{m}{k^\Delta} = \frac{1}{n}$ and the previous probability is at least $1 - \frac{2}{n}$. Moreover, the total high-low cost of this step is

$$m \cdot O\langle \Delta \log k, k \log k \rangle = O\langle m \log(mn), mk \log k \rangle \tag{8}$$

and since $m = \Theta(\frac{n \log n}{n_1}) = \Theta(\frac{\log n}{p_1})$ we have $\log(mn) = O(\log n)$ that is

$$= O\left\langle \frac{(\log n)^2}{p_1}, \frac{\log n}{p_1} k \log k \right\rangle . \tag{9}$$

Remark 7. *Similarly to what was done in the previous sections, one might think of using no high-cost operations at all, by introducing some approximation. In this case, however, this would mean to assign a point to an incorrect cluster. This might be a problem since assigning a point even to the second closest centroid may result in an unbounded error.*

Analysis of the Third Step

Recall that every point is closer to *all* the points in its optimal cluster than to any point outside that cluster (Corollary 3). Since we have found C centroids for each cluster, for every remaining point x, we can assign it to a cluster using a simple "majority voting" using only low-cost operations. Specifically, for any two clusters and two corresponding centroids, say $y \in C_i^*$ and $z \in C_j^*$, we use a low-cost comparison to check whether $d(x,y) < d(x,z)$. Fix a subset of $C' \leq C$ of centroids in each of the k clusters. Let W_{ij} denote the number of "wins" of C_i^* vs C_j^*, that is, the number of pairs of fixed centroids (y,z) with $y \in C_i^*$ and $z \in C_j^*$ such that the low-cost comparison '$d(x,z) > d(x,y)$' is correct (i.e., the strict inequality holds according to Corollary 3). Since W_{ij} is the sum of $N = \binom{C'}{2}$ Bernoulli random variables such that $\Pr[X_e = 1] \geq 1 - p$, standard Chernoff bounds imply the following:

Lemma 14. *For C_i^* being the correct cluster of x and any other cluster C_j^*,*

$$\Pr[W_{ij} \leq N/2] \leq e^{-N f_p}, \tag{10}$$

where $N = \binom{C'}{2}$, and f_p is a constant depending on $p \in (0, 1/2)$ only.

Proof. For $\mu = E[W_{ij}] \geq (1-p)N$ and for $\delta = 1 - \frac{1}{2(1-p)}$, we have (note that $\delta > 0$ since $p < 1/2$)

$$\begin{aligned} \Pr\left[W_{ij} \leq N/2\right] &= \Pr[W_{ij} \leq (1-\delta)(1-p)N] \\ &\leq \Pr[W_{ij} \leq (1-\delta)\mu] \\ &\leq e^{-\frac{\mu\delta^2}{2}} \leq e^{-\frac{(1-p)N\delta^2}{2}} \end{aligned}$$

where the first and third inequalities are from $\mu \geq (1-p)N$ and the second inequality is the Chernoff bound. Finally, note that

$$(1-p)\delta^2 = (1-p)\left(1 - \frac{1}{2(1-p)}\right)^2 = \frac{(1-2p)^2}{4(1-p)} = \frac{(2\epsilon)^2}{4(1/2+\epsilon)} = \frac{\epsilon^2}{1/2+\epsilon}$$

for $p = 1/2 - \epsilon$. Thus, $f_p = \frac{\epsilon^2}{1+2\epsilon}$. □

By the union bound, the probability that the correct cluster C_i^* of x wins against all other clusters is at least $1 - ke^{-Nf_p}$. In particular, since in the previous step we have chosen $C = \Omega(\log n)$, we can choose a suitable $C' = \Theta(\sqrt{C}) = \Theta(\sqrt{\log n})$ such that $N = \binom{C'}{2} \geq \frac{3 \log n}{f_p} = \Omega(\log n)$ and the fail probability is bounded as $ke^{-Nf_p} \leq 1/n^2$. In order to find the optimal cluster for x, we use a simple "tournament" where we compare the current best cluster with the next one that we did not yet consider. Comparing two clusters costs $O(N) = O(C)$ comparisons, and thus overall we use $O(kC^2)$ low-cost operations only for every x. Putting things together, we have the following:

Observation 7. *In Step 3, with a probability of at least $1 - \frac{1}{n}$, all points are assigned to their optimal clusters using $O(knC) = O(kn \log n)$ low-cost operations only.*

Putting Things Together (Proof of Theorem 5)

The number of SCQs and the high-low cost complexity of the three steps are as follows:

$$\text{Step 1: } O\left(\frac{k \log k}{p_1}\right) \text{ exact SCQ only}$$
$$\text{Step 2: } O\left\langle \frac{(\log n)^2}{p_1}, \frac{\log n}{p_1} k \log k \right\rangle$$
$$\text{Step 3: } O\langle 0, kn \log n \rangle$$

The overall high-low cost is thus due to the last two steps only, and it is as in (7). Finally, each of the three steps succeeds with a probability of at least $1 - \frac{1}{n}$, conditioned to the previous step being successful. By the union bound, the overall success probability is at most $1 - \frac{3}{n}$.

4.2.2. Extensions of Theorem 5

In the algorithm presented above, we need to set a parameter n_1 and then the algorithm works for all inputs where the smallest cluster has a size of at least n_1. We next observe that this prior knowledge is *not* needed in Step 1 of the algorithm, and that Step 2 can be modified to work without this assumption on the input. Specifically:

1. Lemma 12 still holds since in Step 1 we simply continue extracting random points until every cluster has at least one point. As already observed, we can safely extract points *with* replacement as in the CCP problem since extracting the same point more than once has no effect on the stopping condition. Therefore, all upper bounds on the CCP problem also apply (see Lemma 11).
2. In Step 2, we can use *high-cost* comparisons to assign every point to its correct cluster. Since these operations are always correct, we no longer need $C = \Omega(\log n)$ points per cluster, and can in fact reduce them to $C' = \Theta(\sqrt{\log n})$, which is the number we need in Step 3 to have a high-probability of success. Selecting $m' \geq 2C'/p_1$ points guarantees that the probability that not all clusters have at least C' points is at most $q(m') = k \exp(-\frac{C'}{4}) = k \exp(-\frac{p_1 m''}{8})$. For a suitable $m' = \Theta(\frac{\log k}{p_1})$, this probability is constant, say $q(m') < 1/3$. Therefore, in expectation, we have to wait a constant number of "rounds" of m' extractions each, meaning that the number of extractions needed before reaching C' points in each cluster is $O(m') = O(\frac{C' + \log k}{p_1})$. Since for every point we have to check all clusters, the expected high-low cost operations of Phase 2 is $O\left\langle k \frac{\sqrt{\log n} + \log k}{p_1}, 0 \right\rangle$.
3. Step 3 is as before and it takes $O\langle 0, kn \log n \rangle$ high-low cost operations. Its success probability is at least $1 - \frac{1}{n}$, while the two previous steps are always successful since we use high-cost operations.

This leads to the following extension of Theorem 5:

Theorem 6. *For any γ-perturbation-stable metric space (X, d) which has γ-radius optimal clusters, $\gamma \geq 2$, the optimal k-clustering can be computed using only the following number of exact same-cluster queries and high-low cost operations parameterized in $p_1 = \frac{n_1}{n}$, where n_1 is the size of the smallest optimal cluster:*

1. *The algorithm uses $O(k\mathcal{C}(\mathbf{p})) \leq O(\frac{k \log k}{p_1})$ exact same-cluster queries in expectation, where $\mathbf{p}$ depends on the size of the optimal clusters (Lemma 11), and $\mathcal{C}(\mathbf{p})$ is the corresponding coupon collector bound (Lemma 12 and Observation 6).*

2. *The additional high-low cost operations used by the algorithm is*

$$O\left\langle k\frac{\sqrt{\log n}+\log k}{p_1}, kn\log n\right\rangle . \tag{11}$$

If the size of the smallest cluster is known, then the high-low cost upper bound in (7) *also applies.*

5. Conclusions, Extensions, and Open Questions

In this work, we have shown algorithms for clustering that use a suitable combination of high-cost and low-cost comparison (ordinal) operations on the underline distance (dissimilarity) function of the data points. Some of the results are based on a popular graph-based Single-Linkage algorithm at its enhanced version Single-Linkage++ which has provable optimal performance in γ-pertubation-stable instances [1–3]. Our results apply to k-medoids clustering, and use comparisons between groups (sum) of distances according to (1). Though this is a rather rich model, and some of these queries may be difficult to realize in practice, some of the results can be extended to simpler queries as we explain below (in fact, the simpler Single-Linkage algorithm uses very simple queries, and the two algorithms differ only in the edge-removal step—Phase 2). The second set of results we provide are algorithms that make use of a rather limited number of exact *same-cluster queries*, in addition to the comparison high-low cost operations of our model. For example, for instances where the smallest optimal cluster has size $n_1 = \Omega(\frac{n}{k})$, there is an algorithm using $O(k \log k)$ same-cluster queries, $O(k \log^2 n)$ high-cost operations, and $O(kn \log n)$ low-cost operations (Theorem 5 and Theorem 6 provide general trade-offs and also deal with the case n_1 not being known, respectively).

In the remainder of this section, we shall discuss further (possible) extensions of our results, including different *query models* and different *error models* that can be considered. We conclude with a discussion on interesting *open questions* and their relation to the above aspects.

5.1. Query Model

Some of our algorithms use only certain types of queries. In particular, we have the following:

- While Phase 1 of the algorithm seems already quite expensive in terms of high-cost operations, we note that it makes use of much simpler queries "$d(x,y) > d(z,w)$?" like in [14]. This is also the case for the Single-Linkage algorithm (Phase 2 naive), which uses very few such high-cost queries (Table 1). While our implementation combines exact (high-cost) and noisy (low-cost) queries, the algorithms in [14] use only noisy queries, though under a different noisy model related to certain *planted* instances.
- The first exact implementation of Single-Linkage++ (Phase 2) requires computing/estimating the cost of the clusters given their centroids (3). Instead, the dynamic programming version makes use of the full power of the model. Indeed, our approximation (Phase 2 APX in Table 1) is based on the idea that queries/algorithms that approximately compute the centers (medois) are enough. Definition 4 and Theorem 4 suggest that a query model/algorithm which allows for approximating the centers, and their costs (3), yields an approximate implementation of Phase 2 of Single-Linkage++.
- The Coupon–Collector Clustering algorithm using same-cluster queries (Section 4) uses very simple comparison-queries, namely "$d(x,y) < d(x,z)$" (second and third steps), in addition to the same-cluster queries (first step). The latter can either be available or can be simulated by a richer class of "scalar" queries in (6). As already observed, these are slightly more complex than those in [8,9].
- The model can be refined by introducing a cost dependent on the *set* of distances involved, e.g., how many they are. Whether comparing just *two* pairwise distances is easier then

comparing *groups* of distances seems to depend on the application at hand; sometimes comparing groups is easier because they provide a richer "context" which helps.

5.2. Error Model

- Our error model assumes constant (and independent) error probability across all comparisons. Other error models are also possible and they typically incorporate the "distance values" in the error probabilities (very different values are easier to detect correctly than very close ones). Examples of such models can be found in, e.g., [6,9].
- Different error models may result in different (perhaps better) dislocation bounds on the approximate sorting problem (Lemma 2). This may directly improve some of our bounds, where the maximum dislocation is the bottleneck for finding the minimum (or top-k-elements) with high probability (if the maximum dislocation becomes $D' \ll \log n$, then we need $O(D') \ll \log n$ high-cost operations for the latter problem, which is used as a subroutine in most of our algorithms).

5.3. Open Questions

The results and the above considerations suggest a number of natural (still open) questions.

- Reduce the high-cost complexity of Phase 1 of Single-Linkage++. We notice that, in the analysis, the same cluster is considered several times when trying all $N = \binom{n-1}{k-1}$ edge-removals. A possible direction might be to estimate the number of partitions of a given (spanning) tree, though this problem does not seem to have a general closed form solution [35].
- Our counterexample shows that a direct improvement of Phase 2 solely based on approximate minimum-spanning tree is not possible. We feel that a finer analysis based on some combinatorial structure of the problem might help.
- Extend the results to other center-based clustering problems for which the Single-Linkage++ algorithm is optimal [3]. Our scheme based on approximate centers (Definition 4 and Theorem 4) suggests a natural approach in which we simply need queries that approximate the costs.

Author Contributions: Conceptualization, E.B. and P.P.; Formal analysis, E.B. and P. P.; Investigation, E.B. and P.P. Both the authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Balcan, M.F.; Blum, A.; Vempala, S. A Discriminative Framework for Clustering via Similarity Functions. In Proceedings of the Fortieth Annual ACM Symposium on Theory of Computing, Victoria, BC, Canada, 17–20 May 2008; Association for Computing Machinery: New York, NY, USA, 2008; STOC '08, pp. 671–680. [CrossRef]
2. Awasthi, P.; Blum, A.; Sheffet, O. Center-based clustering under perturbation stability. *Inf. Process. Lett.* **2012**, *112*, 49–54. [CrossRef]
3. Angelidakis, H.; Makarychev, K.; Makarychev, Y. Algorithms for Stable and Perturbation-Resilient Problems. In Proceedings of the 49th Annual ACM SIGACT Symposium on Theory of Computing (STOC), Montreal, PQ, Canada, 19 June–23 June 2017; pp. 438–451. [CrossRef]
4. Roughgarden, T. CS264: Beyond Worst-Case Analysis Lecture# 6: Perturbation-Stable Clustering. 2017. Available online: http://theory.stanford.edu/~tim/w17/l/l6.pdf (accessed on 8 February 2021).
5. Tamuz, O.; Liu, C.; Belongie, S.J.; Shamir, O.; Kalai, A. Adaptively Learning the Crowd Kernel. In Proceedings of the 28th International Conference on Machine Learning, ICML 2011, Bellevue, DC, USA, 28 June 28–2 July 2011; pp. 673–680.
6. Jain, L.; Jamieson, K.G.; Nowak, R. Finite Sample Prediction and Recovery Bounds for Ordinal Embedding. In *Advances in Neural Information Processing Systems*; Lee, D.; Sugiyama, M., Luxburg, U., Guyon, I., Garnett, R., Eds.; Curran Associates, Inc.: Red Hook, NY, USA, 2016 ; Volume 29, pp. 2711–2719.
7. Emamjomeh-Zadeh, E.; Kempe, D. Adaptive hierarchical clustering using ordinal queries. In Proceedings of the Twenty-Ninth Annual ACM-SIAM Symposium on Discrete Algorithms, New Orleans, LA, USA, 7–10 January 2018; pp. 415–429.
8. Perrot, M.; Esser, P.; Ghoshdastidar, D. Near-optimal comparison based clustering. *arXiV* **2020**, arXiv:2010.03918.

9. Addanki, R.; Galhotra, S.; Saha, B. How to Design Robust Algorithms Using Noisy Comparison Oracles. Available online: https://people.cs.umass.edu/_sainyam/comparison_fullversion.pdf (accessed on 8 February 2021.)
10. Gilyazev, R.; Turdakov, D.Y. Active Learning and Crowdsourcing: A Survey of Optimization Methods for Data Labeling. *Program. Comput. Softw.* **2018**, *44*, 476–491. [CrossRef]
11. Geissmann, B.; Leucci, S.; Liu, C.; Penna, P.; Proietti, G. Dual-Mode Greedy Algorithms Can Save Energy. In Proceedings of the 30th International Symposium on Algorithms and Computation (ISAAC), Shanghai, China, 8–11 December 2019; Volume 149, pp. 1–18. [CrossRef]
12. Xu, Y.; Zhang, H.; Miller, K.; Singh, A.; Dubrawski, A. Noise-tolerant interactive learning using pairwise comparisons. In *Advances in Neural Information Processing Systems*; Curran Associates, Inc.: Red Hook, NY, USA, 2017; pp. 2431–2440.
13. Hopkins, M.; Kane, D.; Lovett, S.; Mahajan, G. Noise-tolerant, reliable active classification with comparison queries. *arXiv* **2020**, arXiv:2001.05497.
14. Ghoshdastidar, D.; Perrot, M.; von Luxburg, U. Foundations of Comparison-Based Hierarchical Clustering. In *Advances in Neural Information Processing Systems*; Curran Associates, Inc.: Red Hook, NY, USA, 2019 ; Volume 32, pp. 7456–7466.
15. Cohen-addad, V.; Kanade, V.; Mallmann-trenn, F.; Mathieu, C. Hierarchical Clustering: Objective Functions and Algorithms. *J. ACM* **2019**, *66*. [CrossRef]
16. Ng, R.; Han, J. CLARANS: A method for clustering objects for spatial data mining. *Knowl. Data Eng. IEEE Trans.* **2002**, *14*, 1003–1016. [CrossRef]
17. Park, H.S.; Jun, C.H. A simple and fast algorithm for K-medoids clustering. *Expert Syst. Appl.* **2009**, *36*, 3336–3341. [CrossRef]
18. Schubert, E.; Rousseeuw, P.J. Faster k-medoids clustering: Improving the PAM, CLARA, and CLARANS algorithms. In *International Conference on Similarity Search and Applications*; Springer: Cham, Switzerland, 2019; pp. 171–187.
19. Wang, X.; Wang, X.; Wilkes, D.M. An Efficient K-Medoids Clustering Algorithm for Large Scale Data. In *Machine Learning-based Natural Scene Recognition for Mobile Robot Localization in An Unknown Environment*; Springer: Cham, Switzerland, 2020; pp. 85–108.
20. Jin, X.; Han, J., K-Medoids Clustering. In *Encyclopedia of Machine Learning*; Sammut, C., Webb, G.I., Eds.; Springer: Boston, MA, USA, 2010; pp. 564–565. [CrossRef]
21. Geissmann, B.; Leucci, S.; Liu, C.; Penna, P. Optimal Sorting with Persistent Comparison Errors. In Proceedings of the 27th Annual European Symposium on Algorithms (ESA), Munich/Garching, Germany, 9–11 September 2019; [CrossRef]
22. Kane, D.M.; Lovett, S.; Moran, S.; Zhang, J. Active Classification with Comparison Queries. In Proceedings of the 2017 IEEE 58th Annual Symposium on Foundations of Computer Science (FOCS), Berkeley, CA, USA, 15–17 October 2017; pp. 355–366. [CrossRef]
23. Ukkonen, A. Crowdsourced Correlation Clustering with Relative Distance Comparisons. In Proceedings of the 2017 IEEE International Conference on Data Mining (ICDM), New Orleans, LA, USA, 18–21 November 2017; pp. 1117–1122. [CrossRef]
24. Ashtiani, H.; Kushagra, S.; Ben-David, S. Clustering with Same-Cluster Queries. In *Advances in Neural Information Processing Systems*; Lee, D., Sugiyama, M., Luxburg, U., Guyon, I., Garnett, R., Eds.; Curran Associates, Inc.: Red Hook, NY, USA, 2016; Volume 29, pp. 3216–3224.
25. Ailon, N.; Bhattacharya, A.; Jaiswal, R. Approximate correlation clustering using same-cluster queries. In *Latin American Symposium on Theoretical Informatics (LATIN)*; Springer: Berlin/Heidelberg, Germany, 2018; pp. 14–27.
26. Saha, B.; Subramanian, S. Correlation Clustering with Same-Cluster Queries Bounded by Optimal Cost. In Proceedings of the 27th Annual European Symposium on Algorithms (ESA), Munich/Garching, Germany, 9–11 September 2019; Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik: Wadern, Germany, 2019.
27. Bressan, M.; Cesa-Bianchi, N.; Lattanzi, S.; Paudice, A. Exact Recovery of Mangled Clusters with Same-Cluster Queries. *arXiV* **2020**, arXiv:2006.04675.
28. Mazumdar, A.; Saha, B. Clustering with Noisy Queries. In *Advances in Neural Information Processing Systems*; Guyon, I., Luxburg, U.V., Bengio, S., Wallach, H., Fergus, R., Vishwanathan, S., Garnett, R., Eds.; Curran Associates, Inc.: Red Hook, NY, USA,2017; Volume 30, pp. 5788–5799.
29. Sanyal, D.; Das, S. On semi-supervised active clustering of stable instances with oracles. *Inf. Process. Lett.* **2019**, *151*, 105833. [CrossRef]
30. Chien, I.; Pan, C.; Milenkovic, O. Query k-means clustering and the double dixie cup problem. *Adv. Neural Inf. Process. Syst.* **2018**, *31*, 6649–6658.
31. Geissmann, B. Longest Increasing Subsequence Under Persistent Comparison Errors. In *Approximation and Online Algorithms*; Epstein, L., Erlebach, T., Eds.; Springer International Publishing: Cham, Switzerland, 2018; pp. 259–276.
32. Berenbrink, P.; Sauerwald, T. The Weighted Coupon Collector's Problem and Applications. In *Computing and Combinatorics*; Ngo, H.Q., Ed.; Springer: Berlin/Heidelberg, Germany, 2009; pp. 449–458.
33. Doumas, A.V.; Papanicolaou, V.G. The coupon collector's problem revisited: generalizing the double Dixie cup problem of Newman and Shepp. *ESAIM Probab. Stat.* **2016**, *20*, 367–399. [CrossRef]
34. Newman, D.J.; Shepp, L. The Double Dixie Cup Problem. *Am. Math. Mon.* **1960**, *67*, 58–61. [CrossRef]
35. Székely, L.; Wang, H. On subtrees of trees. *Adv. Appl. Math.—Advan Appl Math* **2005**, *34*, 138–155. [CrossRef]