The Open University of Israel
Department of Mathematics and Computer Science

Ramsey Partitions Based Approximate Distance Oracles

Thesis submitted as partial fulfillment of the requirements
toward an M.Sc. degree in Computer Science
The Open University Of Israel
Computer Science Division

By

Chaya Fredman (Schwob)

Prepared under the supervision of **Dr. Manor Mendel**

December, 2008

Acknowledgments

I would like to express my gratitude to Dr. Manor Mendel, my research advisor, for his guidance in my work. His advice and encouragement, and his patience and effort in proofreading many thesis drafts, made this work successful.

Special thanks to my husband, for his help during the entire time of my graduate studies.

My deepest gratitude is to my parents, for their love and dedication throughout my life. Especially, to my father, whose moral support has been essential for the completion of these studies.

Contents

1 Introduction **1**

1.1 Motivation 1

1.2 Known bounds and trade-offs 2

1.3 New results in this thesis 5

2 Notation and preliminaries **7**

2.1 The computational model 7

3 Ramsey partitions based ADOs **10**

3.1 The preprocessing algorithm 10

3.2 Correctness 12

4 Preprocessing Ramsey partitions based ADOs for weighted graphs **16**

4.1 Computing a CKR random partition 17

4.2 Picking $O(\alpha)$-padded vertices 20

4.3 Dispensing with the $\lg \phi$ factor 21

4.4 Preprocessing time / storage space trade-off 24

5 Source restricted ADOs **29**

6 Parallel ADOs preprocessing **32**

6.1 Thorup-Zwick ADOs 33

6.2 Ramsey partitions based ADOs 35

7 Open problems **40**

A Appendix: Parallel hop set construction parameters **41**

List of Tables

1 Available exact and approximate distance oracles 3

2 State of the art of known ADOs 6

3 Required storage space for S-source restricted ADOs 31

List of Figures

1 A schematic description of the behavior of the modified Dijkstra 18

BS"D

Abstract

This thesis studies constructions of *approximate distance oracles*. Approximate distance oracles are compact data structures that can answer approximate distance queries quickly. The three parameters by which their efficiency is evaluated are *preprocessing time*, *storage space requirement* and *query time*. The thesis presents a variety of algorithmic results related to approximate distance oracles: faster preprocessing time, source-restricted approximate distance oracles construction and parallel implementation.

1 Introduction

1.1 Motivation

Consider the following two common scenarios:

- When planning a journey we usually wish to travel the shortest route to our destination. Also, we like to know in advance the anticipated distance and the route achieving it.
- In every moment, an enormous amounts of data is sent over the Internet. It is reasonable to minimize the cost of routing them by minimizing the distance of the route.

We can describe both road map and the Internet as large graphs with n vertices and m edges. On the map vertices are addresses and edges are roads, and on the Internet vertices are multiple connected computers and the edges are wired or wireless medium that is used to link them. Then, both problems can be modeled by the well known *All Pairs Shortest Path* [APSP] problem. The input in this problem is a weighted graph with n vertices and m edges, which may be directed or undirected. The distance between a pair of vertices along a specific path is defined to be the sum of weights of the edges along it. The required output is the shortest paths between all pairs of vertices in the graph.

This problem is fundamental in many day to day scenarios. In most cases, one is not interested in the distance between all possible pairs of locations but want to retrieve the data quickly for those pairs he does need.

The first solution that comes to mind uses an APSP algorithm. We can produce an $n \times n$ distances matrix. Any distance query can then be answered in $O(1)$ time. There are, however, many cases where this solution is unacceptable. First, the performance time of any ASAP algorithm is bounded by $\Omega(mn)$ (see [31]), and that may be too long. Second the $n \times n$ matrix produced may be too large to store efficiently, as typically $m \ll n^2$, this table is much larger than the graph itself.

In situations where using approximate distances is acceptable, the preprocessing time and storage space requirement can be reduced, depending on the acceptable stretch. Such constructions that also support quick query time were coined *approximate distance oracles* [ADOs] by Thorup and Zwick [52]. The parameters by which we test their efficiency are: *preprocessing time, storage space requirement and* *query time*. In recent years much work has been done to achieve close to optimal trade-off between the various parameters. We review the known upper and lower bounds in Section 1.2.

In this thesis we contribute to the research in the area with new results and observations. We examine known algorithms and improve their efficiency. We also study other interesting related topics.

1.2 Known bounds and trade-offs

Fix a metric space (X, d_X). An estimate $\delta(x, y)$ of the distance $d_X(x, y)$ from $x \in X$ to $y \in X$ is called k-approximation if $d_X(x, y) \le \delta(x, y) \le k d_X(x, y)$.

Let $G = (V, E, w)$ be an undirected positively weighted graph with n vertices and m edges where $w : E \to (0, \infty)$ is a weight function on its edges. We assume the graph is connected. The distance $d_G(u, v)$ from u to v in the graph is the length of the shortest path between u and v in the graph, where the length of a path is the sum of the weights of the edges along it. The pair (V, d_G) define a graph metric space.

The input metric spaces are represented by either a distance matrix, or undirected positively weighted graph. We wish to preprocess the metric space efficiently, obtaining a data structure called *approximate distance oracle*, that supports quick approximate distance queries, and requires compact storage space.

We give a short review of known results. We present the best results in respect for all three parameters we mentioned. We focus on general results, *i.e.*, for integer k, approximation $O(k)$. For simplicity we use $\tilde{O}(\cdot)$ time bound to indicate the presence of logarithmic factors. Results for approximation of 3 or less are not reviewed here. Additionally, some results are described even though better results are known, since they contain interesting ideas. Table 1 contains a summary of known results, and in the rest of the section we review those constructions which are useful for this thesis. For a more detailed survey, see [56, 52].

Input	Approximation	Preprocessing Time	Storage Space	Query Time	Reference & Description
distance matrix	1	none	$O\left(n^2\right)$	$O(1)$	trivial
	$2k-1$	$O\left(n^2\right)$	$O\left(kn^{1+1/k}\right)$	$O(k)$	[52] Thorup-Zwick ADOs, Section 1.2
	$128k$ [1]	$\tilde{O}\left(n^{2+1/k}\right)$	$O\left(n^{1+1/k}\right)$	$O(1)$	[42] Ramsey partitions based ADOs, Section 1.2
	$768k$ [1]	$\tilde{O}\left(n^2\right)$	$O\left(n^{1+1/k}\right)$	$O(1)$	**this thesis**, Section 1.3
undirected positively weighted graph	1	none	$O(m)$	$O(m)$	trivial
		$O(mn)$	$O\left(n^2\right)$	$O(1)$	[19] Dijkstra APSP
	$2k-1$	$O(km)$	$O\left(kn^{1+1/k}\right)$	$O\left(kn^{1+1/k}\right)$	[6] Spanners, Section 1.2
		$\tilde{O}\left(kmn^{1/k}\right)$	$\tilde{O}\left(kn^{1+1/k}\right)$	$O\left(kn^{1/k}\right)$	[40] Matoušek, ℓ_∞ embedding
		$O\left(kmn^{1/k}\right)$	$O\left(kn^{1+1/k}\right)$	$O(k)$	[52] Thorup-Zwick ADOs, Section 1.2
		$O\left(\min\left(n^2, kmn^{1/k}\right)\right)$	$O\left(kn^{1+1/k}\right)$	$O(k)$	[5] Thorup-Zwick ADOs, Section 1.2
	$256k$ [1]	$\tilde{O}\left(kmn^{1/k}\right)$	$O\left(n^{1+1/k}\right)$	$O(1)$	**this thesis**, Section 1.3

Table 1: *Some available exact and approximate distance oracles*

Spanners

Let $G = (V, E, w)$ be an undirected graph with non-negative edge weights where $w : E \to [0, \infty)$ is a weight function on its edges. Often it is useful to obtain a subgraph $H = (V, F)$, $F \subseteq E$, which closely approximates the distances of G. For $k \geq 1$, H is said to be k-spanner of G if for every $u, v \in V$, $d_H(u, v) \leq k d_G(u, v)$. (Observe that $d_G(u, v) \leq d_H(u, v)$ is always true). The concept of spanners has been introduced by Peleg and Ullman [44], where they used spanners to synchronize asynchronous networks. In [1] a simple greedy algorithm for computing the k-spanner H of G is described. The algorithm is similar to Kruskal's algorithm for computing a minimum spanning tree: Initially set $F = \emptyset$. Consider the edges of G in nondecreasing order of weight. If $(u, v) \in E$ is the currently considered edge

[1]No effort was done to improve those constants. They can be significantly improved, although not as much as $2k-1$.

and $kw(u,v) < d_H(u,v)$ (using the convention that the distance between disconnected vertices is ∞), then add (u,v) to F. It is easily proved that, in the end of the process, $H = (V,F)$ is indeed a k-spanner of $G = (V,E,w)$. Also, let the *girth* of a graph be the smallest number of edges on a cycle in it. Then, $girth(H)$, the girth of the created subgraph H, is greater than $k+1$. A simple argument shows that for any integer t, any graph with $\Omega\left(n^{1+1/t}\right)$ edges contains a cycle with at most $2t$ edges, *i.e.*, its maximal girth is $2t$. Set $k = 2t-1$. As $girth(H) > k+1 = 2t$, it follows that any undirected weighted graph has a $2t-1$-spanner with $O\left(n^{1+1/t}\right)$ edges.

The fastest known implementation of algorithms computing such graphs is the randomized algorithm by Baswana and Sen in [6], having expected running time of $O(km)$.

Using spanners it is easy to achieve ADOs with optimal approximation / storage space trade-off(see Section 1.2). However the naïve query method for spanners is finding the length of the shortest path between the queried points, *i.e.*, full traversal of the graph. The running time of the distance query is $O(m)$ when m is the spanner size. There are several ideas to overcome this problem. ADOs constructions based on spanners are described in [1, 52].

Spanners based (Thorup-Zwick) ADOs

Thorup and Zwick [52] introduced a method for constructing in expected $O\left(kmn^{1/k}\right)$ time optimal storage space $2k-1$-spanners with the additional property of supporting distance queries with worst-case time $O(k)$. For a given undirected positively weighted graph $G = (V,E,w)$ with n vertices and m edges, the algorithm starts by defining an hierarchy $V = A_0 \supseteq A_1 \ldots \supseteq A_k = \emptyset$ of subsets where A_i is a random subset obtained by selecting each element of A_{i-1}, independently, with probability $n^{-1/k}$. For $v \in V$ and $1 \le i \le k$, define $p_i(v)$ as the closest vertex to v from A_i. Next, the algorithm defines, for each $v \in V$, $B(v)$ as the set of vertices $u \in V$ having the property that, for some i, $u \in A_i$ is strictly closer to v than all vertices of A_{i+1}. Finally, the algorithm constructs a hash table $H(v)$ of size $O(|B(v)|)$ that stores for each $u \in B(v)$, the distance $d_G(u,v)$. The expected size of $H(v)$ is proved to be $O\left(kn^{1/k}\right)$.

The query algorithm is defined as follows: for a pair $u,v \in V$, find the first index $1 \le i \le k$ having either $w = p_i(u) \in B(v)$ or $w = p_i(v) \in B(u)$. Return $d_G(u,w) + d_G(v,w)$. This sum is proved to be $2k-1$-approximation of $d_G(u,v)$.

Baswana and Kavitha [5] showed a faster preprocessing algorithm for Thorup-Zwick ADO, achieving an expected running time of $O\left(\min\left(n^2, kmn^{1/k}\right)\right)$.

Ramsey partitions based ADOs

Theorem 1.1. *[42] For any $k > 1$, every n-point metric space (X, d_X) admits a $O(k)$-ADO requiring storage space $O\left(n^{1+1/k}\right)$, and whose query time is a universal constant.*

The preprocessing algorithm defines iteratively a sequence of pairwise disjoint subsets $S_1, \ldots S_l \subset X$ such that $\cup_i S_i = X$. And, respectively, a sequence of ultrametrics $\rho_1, \ldots, \rho_l$ where ρ_i is defined on the point set $R_i = \cup_{j \ge i} S_i$, and $\rho_i(x,y)$ is $O(k)$-approximation of the distances $d_X(x,y)$ when $x \in S_i$, $y \in R_i$.

For every pair x, y, there is i such that $\rho_i(x, y)$ is $O(k)$-approximation of $d_X(x, y)$, and since query distances in ultrametrics can be implemented in constant time using an HST representation and the **lca** operator (see [4]), this gives an efficient query method.

Mendel and Naor [42] show that one can construct such a sequence where $l = n^{O(1/k)}$, so the storage space is $n^{1+O(1/k)}$. They use the notion of tight *Ramsey partitions* to prove that every X contains a subset S with $|S| \geq n^{1-O(1/k)}$, for which the metric d_X restricted to the set of pairs $x \in X$ and $y \in S$ is $O(k)$ equivalent to an ultrametric. Hence, the name Ramsey partitions based ADOs. For a definition and discussion of the term see Section 3.

Storage space lower bound

One of the important properties of ADOs is their compact storage space. Define $m_g(n)$ to be the maximal number of edges in an n-vertices graph with girth g. It is conjectured by many (e.g., [21]) that $m_{2k+2}(n) = \Omega\left(n^{1+1/k}\right)$. In [38] the bound $m_{2k} = \Omega\left(n^{1+2/3k}\right)$ is proved using a family of d-regular Ramanujan graphs having for n vertices graph, girth $\leq \frac{4}{3} \lg_{d-1} n$. This bound was slightly improved in [34, 35]. For a full survey see [52]. Since graphs with girth $2k + 2$, have no proper t-spanners for $t < 2k + 1$, the conjecture would imply a lower bound of $\Omega\left(n^{1+1/k}\right)$ on the storage space requirement of any $O(k)$-ADO. The proof uses counting methods and appeared in [52]. Similar arguments appeared in [40].

1.3 New results in this thesis

The following results appear in this thesis.

1. Faster preprocessing time for Ramsey partition based ADOs: As mentioned before the three efficiency parameters of the ADOs are: *preprocessing time, query time and storage space*. Looking in Table 1 we see several constructions achieving tight approximation / storage space trade-off. The best known preprocessing time — query time for this trade-off is either $O\left(\min\left(n^2, kmn^{1/k}\right)\right)$ — $O(k)$ for weighted graphs, or $\tilde{O}\left(n^{2+1/k}\right)$ — $O(1)$ for distance matrices. The later is achieved using Ramsey partitions based ADOs. In Section 4 we show preprocessing algorithm for those ADOs obtaining $\tilde{O}\left(kmn^{1/k}\right)$ running time for weighted graphs. Thus improving the currently known results.

2. Source restricted ADOs: Given a metric space, sometimes we are only interested in approximate distances from a specific set. Both the preprocessing time and the storage space requirement can be reduced in this case. This question was first raised by Roditty, Thorup and Zwick [47], where it was shown that a variant of the Thorup-Zwick ADOs [52] achieves, for a set of sources S with size s, expected preprocessing time of $\tilde{O}\left(ms^{1/k}\right)$ and storage space $O\left(kns^{1/k}\right)$. In Section 5 we first show a simple extension of any k-ADOs construction that gives S-source restricted $3k$-ADOs. Then we focus on S-source restricted Ramsey partitions based $O(k)$-ADOs and show how to obtain expected preprocessing time of $\tilde{O}\left(ms^{1/k}\right)$ and storage space $O\left(kns^{1/k}\right)$. We also prove lower bounds on the approximation /

storage space trade-off in this scenario, concluding that the storage space obtained is asymptotically tight.

3. Parallel implementation of Ramsey partition based ADOs: When working in parallel processing environment we would like to achieve maximum utilization of the parallel processors without increasing much the total work done. We are not aware of prior work on parallel preprocessing of ADOs. We present here the first results concerning Thorup-Zwick ADOs and Ramsey partitions based ADOs. For those constructions, the preprocessing time can be reduced to polylogaritmic with (almost) no increase in the total work.

Table 2 summarizes the state of the art of the efficiency parameters of Thorup-Zwick ADOs and Ramsey partitions based ADOs.

	Thorup-Zwick ADOs	Ramsey partitions based ADOs
Approximation Preprocessing time — Storage space — Query time	$2k-1$ $O\left(\min\left(n^2, kmn^{1/k}\right)\right)$ — $O\left(kn^{1+1/k}\right)$ — $O(k)$ [52], [5].	$256k$ $\tilde{O}\left(kmn^{1/k}\right)$ — $O\left(n^{1+1/k}\right)$ — $O(1)$ [42], **this thesis**.
S-source restricted	$O\left(kns^{1/k}\right)$ storage space [47], $O\left(ks^{1+3/(k+1)}+n\right)$ storage space **this thesis**.	$O\left(n^{1/k}s+n\right)$ storage space **this thesis**.
Derandomization	Preprocessing time is multiplied by a $O(\lg n)$ factor [47].	*Poly* (n) time. [Bartal (unpublished), Tao and Naor (unpublished)].
Parallelization	*Polylog* (n) time. Total work does not increase significantly. **this thesis**.	*Polylog* (n) time. Total work and storage space does not increase significantly. **this thesis**.

Table 2: *State of the art of Thorup-Zwick ADOs and Ramsey partitions based ADOs for weighted graphs with n vertices and m edges*

2 Notation and preliminaries

Let (X, d_X) be a metric space, *i.e.*, X is a set, and $d_X : X \times X \to [0, \infty)$ satisfy the following: $d_X(x, y) = 0$ if and only if $x = y$, $d_X(x, y) = d_X(y, x)$ and $d_X(x, y) + d_X(y, z) \geq d_X(x, z)$. Given $Y \subseteq X$, define the metric space (Y, d_X) as the metric d_X restricted to pairs of points from Y. Define $\operatorname{diam}(Y, d_X) = \max\{d_X(x, y) \mid x, y \in Y\}$, when d_X is clear from the context we may omit it. For $x \in X$ and $R \in \mathbb{R}$, $B_X(x, R)$ is defined as the closed ball around x including all points in X at distance at most R from x, *i.e.*, $B_X(x, R) = \{y \in X \mid d_X(x, y) \leq R\}$.

Let $G = (V, E, w)$ be an undirected positively weighted graph where the weights are $w : E \to (0, \infty)$. Let $d_G : (V \times V) \to [0, \infty)$ be the shortest path metric on G. We denote by n the number of vertices, and by m the number of edges. We assume an adjacency list representation of graphs.

Given a weighted graph, the aspect ratio is defined to be the largest distance/smallest non-zero distance in the graph.

Given $\alpha \geq 1$, α-ADO is defined as an approximate distance oracle with approximation factor α.

An ultrametric is a metric space (X, d_X) such that for every $x, y, z \in X$, $d_X(x, z) \leq \max\{d_X(x, y), d_X(y, z)\}$. A more restricted class of finite metrics with an inherently hierarchical structure is that of k-hierarchically well-separated trees, defined as follows:

Definition 2.1. [3] For $k \geq 1$, a k-hierarchically well-separated tree (T, Δ) (known as k-HST) is a metric space whose elements are the leaves of a rooted finite tree T. To each vertex $u \in T$ there is associated a label $\Delta(u) \geq 0$ such that $\Delta(u) = 0$ iff u is a leaf of T. It is required that if a vertex u is a child of a vertex v then $\Delta(u) \leq \Delta(v)/k$. The distance between two leaves $x, y \in T$ is defined as $\Delta(\mathbf{lca}(x, y))$, where $\mathbf{lca}(x, y)$ is the least common ancestor of x and y in T.

The notion of 1-HST, which we hereafter refer simply as HST, coincides with that of a finite ultrametric. Query distances for HST can be implemented in constant time using the following result [4]: There exists a simple scheme that takes a tree and preprocess it in linear time so that it is possible to compute the least common ancestor of two given nodes in constant time.

Solving the undirected single source shortest paths [USSSP] problem is an important step in some of the algorithms in this thesis. Given a weighted graph with n vertices and m edges, Dijkstra's classical SSSP algorithm with source w maintains for each vertex v an upper bound on the distance between w and v, $\delta(w, v)$. If $\delta(w, v)$ has not been assigned yet, it is interpreted as infinite. Initially, we just set $\delta(w, w) = 0$, and we have no visited vertices. At each iteration, we select an unvisited vertex u with the smallest finite $\delta(w, u)$, visit it, and relax all its edges. That is, for each incident edge $(u, v) \in E$, we set $\delta(w, v) \leftarrow \min\{\delta(w, v), \delta(w, u) + w(u, v)\}$. We continue in this way until no vertex is left unvisited. Using Fibonacci heaps [25] the algorithm is implemented in $O(m + n \lg n)$.

2.1 The computational model

In this section we discuss the computational model we assume for the algorithms in this thesis. Ideally, we would have use the addition-comparison model. This model is a restricted version of the algebraic decision tree model [7] where the only operations allowed are addition and comparison of real numbers. It is widely used in graphs algorithms, specifically, shortest path algorithms (see, *e.g.*, [57, 13, 55]) and it is reasonable to consider it here. However, because of the lack of division and floor operations, the enumerating of integer scales in a given real range done in Section 4.3 becomes non-trivial. Simulating ,*e.g.*, the floor operation $floor(U)$ for real $U \geq 0$ costs $O\left(\lg \lg^{O(1)} |U|\right)$ using a binary search. In our case $U = O(\Phi)$ where Φ is the graph diameter. (Observe that a factor of $O(\lg \Phi)$ in the storage space is unavoidable in metrics with maximal diameter Φ when the input is given in exact way.) We therefore consider other candidate models.

Another natural model for real numbers applications is the real-RAM model. This is a powerful random access machine that can perform exact arithmetic operations with arbitrary real numbers. It is popular in algebraic complexity [45] and computational geometry [39]. Specifically, this is an infinite precision floating point RAM supporting unit-cost arithmetics, floor, ceil, logarithm and exponent operations. However, this model is too good to be true as it actually allows any problem in PSPACE to be solved in polynomial time [48]. See also [54].

As the result of those observations, there are now two standard RAM definitions avoiding this problem [2].

1. Log-cost RAM: Each memory location is arbitrary size. The cost of each operation is proportional to the total number of bits involved.

2. Unit-cost RAM: Each memory location is word with $\Theta(\lg n)$ size where n is the total of numbers in the input. Operations on words take constant time, presumably because of hardware parallelism. Operations on larger-size numbers must be simulated.

We describe a unit-cost RAM model with the additional convenience that the size of the words is big enough to store the input numbers. This model is also described in [29, Section 2.2].

The given input consists of n numbers at the range $\left[-\Phi, -\Phi^{-1}\right]$ and $\left[\Phi^{-1}, \Phi\right]$ and an accuracy parameters $t \in \mathbb{N}$. The machine have words of size $O(\lg n + \lg \lg \Phi + t)$. These words accommodates floating points number of the set,

$$\left\{ \pm (1+x) 2^{y} \;\middle|\; x \in [0,1],\ x2^{-t} \in \mathbb{N},\ y \in \left[-n^{O(1)} \lg^{O(1)} \Phi, n^{O(1)} \lg^{O(1)} \Phi\right] \cap \mathbb{Z} \right\}$$

This is an extension of the IEEE standard representation for binary floating-point numbers (IEEE 754) [49].

These words accommodate integers in the range

$$\left[-\left(2^{t} n \lg \Phi\right)^{O(1)}, \left(2^{t} n \lg \Phi\right)^{O(1)}\right].$$

The basic instructions are conditional jumps, direct and indirect addressing for loading and storing words in registers and arithmetic operations such as addition, subtraction, comparisons, flooring and logical bits operations for numbers in the registers.

Proving correctness of the algorithms in this model requires taking into account precision caused robustness problems. Rounding errors can sometimes be tolerated and interpreted as small perturbations in inputs, especially as our algorithms deal with approximated results. However, serious problems can arise from the built up of such errors and their complicated interaction withing the logic of the computation. See also [36, 53, 24]. We compromise by describing and proving correctness of our algorithms in the real-RAM model and pointing to the fact that the described floating-point unit-cost RAM model can be used after proper changes are done in the algorithms and the result has slightly larger approximation factor.

Finally, we assume our machine is equipped with a random number generator.

Parallel computational model: Section 6 deals with parallel computation. Parallel RAM models [PRAM] are essentially RAM models with the added force of multiple processors. Still, we need to define the way the processors access the common memory.

The priority CRCW PRAM model supports concurrent read as well as write into any memory location by multiple processors. In case of simultaneous write operations by two or more processors into a memory location, the processor with highest priority succeeds.

It is a very strong model comparing to the weaker and more practical EREW PRAM (exclusive read, exclusive write). However, priority CRCW PRAM algorithms can be implemented in the EREW PRAM model.

Lemma 2.2. *[30] Single step of p processors in priority CRCW PRAM model with $O(m)$ memory can be executed in $O(\lg p)$ steps by p processors in EREW PRAM model with $O(m + p)$ memory.*

The algorithms we present and all known parallel algorithms we use are implemented in the priority CRCW PRAM model or weaker PRAM models. Using Lemma 2.2 the algorithms can also be implemented in the EREW PRAM model, with running time and work multiplied by $O(\lg p)$ where p is the number of processors they use. Observe that p is trivially bound by the total work.

3 Ramsey partitions based ADOs

In Section 1.2 we briefly described Ramsey partitions based ADOs. This ADO achieves, for a given approximation factor $\alpha \geq 1$, $O\left(n^{2+1/\alpha} \lg n\right)$ expected preprocessing time, (universal) constant query time, and $O\left(n^{1+1/\alpha}\right)$ storage space.

We begin with reviewing Ramsey partitions based ADOs, giving a detailed proof of Theorem 1.1.

As implied by the term *Ramsey partitions based ADO*, Ramsey partitions are instrumental in the ADO's construction. We begin by defining them.

Definition 3.1. [42] Let (X, d_X) be a metric space. Given a partition $\mathscr{P}$ of X and $x \in X$ we denote by $\mathscr{P}(x)$ the unique element of $\mathscr{P}$ containing x. For $\Delta > 0$ we say that $\mathscr{P}$ is Δ-bounded if for every $C \in \mathscr{P}$, $\operatorname{diam}(C) \leq \Delta$. A *partition tree* of X is a sequence of partitions $\{\mathscr{P}_k\}_{k=0}^{\infty}$ of X such that $\mathscr{P}_0 = \{X\}$, for all $k \geq 0$ the partition $\mathscr{P}_k$ is $c^{-k} \cdot \operatorname{diam}(X)$-bounded (for a fixed $c > 1$), and $\mathscr{P}_{k+1}$ is a refinement of $\mathscr{P}_k$. For $\beta, \gamma > 0$ we shall say that a probability distribution Pr over partition trees $\{\mathscr{P}_k\}_{k=0}^{\infty}$ of X is completely β-padded with exponent γ if for every $x \in X$,

$$\Pr\left[\forall\, k \in \mathbb{N},\ B_X\left(x, \beta \cdot c^{-k} \operatorname{diam}(X)\right) \subseteq \mathscr{P}_k(x)\right] \geq |X|^{-\gamma}.$$

Such probability distributions over partition trees are called *Ramsey partitions*.

The definition is based on Bartal's definition of probabilistic HST [3]. Gupta, Hajiaghayi and Räcke [26] give the same definition with fixed $c = 2$, and present a construction of Ramsey partitions where $\gamma = \beta = \Omega\left(\frac{1}{\lg n}\right)$. A construction of Ramsey partitions with $\beta = \Omega\left(\frac{\gamma}{\lg(1/\gamma)}\right)$ can be deduced from the metric Ramsey theorem of Bartal, Linial, Mendel and Naor [4] (see [42, Appendix B]). In [42] Mendel and Naor show an asymptotically tight construction of Ramsey partition , *i.e.*, $\beta = \Omega(\gamma)$. Their construction is based on a random partition due to Calinescu, Karloff and Rabani [14] and on improving the analysis of Fakcharoenphol, Rao and Talwar [23] to that random construction.

Using asymptotically tight Ramsey partitions we can find for a set X a subset S with $|S| \geq n^{1-O(1/k)}$, for which the metric d_X restricted to pairs $x \in X$ and $y \in S$ is $O(k)$ equivalent to an ultrametric. This is the main step in the algorithm, as described in the introduction, Section 1.2.

In Section 3.1 we describe in detail the preprocessing stage of the Ramsey partitions based ADO, and in Section 3.2 we prove its correctness.

3.1 The preprocessing algorithm

Given a metric space (X, d_X). Calinescu, Karloff and Rabani in [14] introduced a randomized algorithm for creating a partition $\mathscr{P}$ of X. We call such a partition *a CKR random partition*.

Algorithm 1 CKR-Partition

```
Input: An n-point metric space $(X, d_X)$, scale $\Delta > 0$
Output: Partition $\mathscr{P}$ of $X$
 1: $\pi :=$ random permutation of $X$
 2: $R :=$ real random in $\left[\frac{\Delta}{4}, \frac{\Delta}{2}\right]$
 3: for $i = 1$ to $n$ do
 4:     $C_i := B_X(x_{\pi(i)}, R) \setminus \bigcup_{j=1}^{i-1} C_j$
 5: end for
 6: $\mathscr{P} := \{C_1, \ldots, C_n\} \setminus \{\emptyset\}$
```

Given an n-point metric space (X, d_X) and approximation factor $\alpha \geq 1$. Algorithm 2 (**RamseyPartitions-ADO**) outline the construction of a set of ultrametrics in HST representation. This set is used for approximating the distances in X in constant time.

Algorithm 2 RamseyPartitions-ADO

```
Input: An $n$-point metric space $(X, d_X)$, approximation factor $\alpha \geq 1$
Output: Set of HSTs $\bigcup (T_i, \Delta_i)$
 1: $X_0 := X$
 2: $i := 1$
 3: repeat
 4:     $\phi := \operatorname{diam}(X_{i-1})$
 5:     $\mathscr{E}_0 := \mathscr{P}_0 := \{X_{i-1}\}$
 6:     $k := 1$
 7:     repeat // Sample a partition tree from Ramsey partitions
 8:         $\mathscr{P}_k :=$ CKR-Partition $(X_{i-1}, d_X)$, $8^{-k}\phi$
 9:         $\mathscr{E}_k := \{C \cap C' \mid C \in \mathscr{E}_{k-1},\ C' \in \mathscr{P}_k\}$ // Refinement
10:         $k := k + 1$
11:     until $|\mathscr{E}_{k-1}| = |X_{i-1}|$
12:     $Y_i := \left\{x \in X_{i-1} \mid \forall\, k \in \mathbb{N},\ B_X\left(x, \frac{8^{-k}\phi}{16\alpha}\right) \subseteq \mathscr{E}_k(x)\right\}$ // Picking $O(\alpha)$-padded vertices
13:     Construct $(T_i, \Delta_i)$ to be the HST representation of the ultrametric $(X_{i-1}, \rho_i)$ defined as
        $\rho_i(x, y) = 8^{-k}\phi$ where $k := \max\left\{j \mid \mathscr{E}_j(x) = \mathscr{E}_j(y)\right\}$ // $O(\alpha)$-approximate ultrametric
14:     $X_i := X_{i-1} \setminus Y_i$
15:     $i := i + 1$
16: until $X_{i-1} = \emptyset$
```

The set of ultrametrics $\bigcup (X_{i-1}, \rho_i)$ can be used as $O(\alpha)$-ADO with constant query time. We describe the query method. Given $\left(X_{j-1}, \rho_j\right)$, let $\left(T_j, \Delta_j\right)$ be its HST representation. For every point $x \in X$ let i_x be the largest index for which $x \in X_{i_x-1}$. Thus, in particular, $x \in Y_{i_x}$. Maintain for every $x \in X$ a vector $\mathbf{vec}_x$ of length i_x having constant time direct access, such that for $i \in \{0, \ldots, i_x - 1\}$, $\mathbf{vec}_x[i]$ is a pointer to the leaf representing x in T_i. Given a query $x, y \in X$ assume without loss of generality that $i_x \leq i_y$. It follows that $x, y \in X_{i_x-1}$. Locate the leaves $\hat{x} = \mathbf{vec}_x[i_x]$, and $\hat{y} = \mathbf{vec}_y[i_x]$ in T_{i_x}, and then compute $\Delta_{i_x}(\mathbf{lca}(\hat{x}, \hat{y}))$ to obtain an $O(\alpha)$ approximation to $d_X(x, y)$.

3.2 Correctness

Next, we prove that the construction outlined above achieves the efficiency parameters declared in Theorem 1.1. The proof follows [42, Lemma 2.1, Lemma 3.1, Theorem 3.2, Lemma 4.2].

Approximation factor of 128α: Fix a pair of points $x, y \in X$. Without loss of generality, let x be the first point in the pair to become padded. Let i be the unique index having $x \in Y_i$. As described above, the approximate distance value is set to be $\rho_i(x,y)$. Let $\{\mathscr{E}_k\}_{k=0}^{\infty}$ be the partition tree sampled in the i-th iteration. By definition $\rho_i(x,y) = 8^{-k}\phi$ for the largest integer k having $\mathscr{E}_k(x) = \mathscr{E}_k(y)$. Then $d_X(x,y) \le \operatorname{diam}(\mathscr{E}_k(x)) \le 8^{-k}\phi = \rho(x,y)$. On the other hand, by our assumption x is padded, *i.e.*, $\forall\, k \in \mathbb{N}, B_X\left(x, \frac{8^{-k}\phi}{16\alpha}\right) \subseteq \mathscr{E}_k(x)$. Since $\mathscr{E}_{k+1}(x) \ne \mathscr{E}_{k+1}(y)$, $y \notin B_X\left(x, \frac{8^{-k-1}\phi}{16\alpha}\right)$. Thus $d_X(x,y) > \frac{8^{-k-1}\phi}{16\alpha} = \frac{1}{128\alpha}\rho(x,y)$. As needed.

Before we continue with the rest of the proof we present a few useful lemmas.

Given a probability distribution over partitions $\mathscr{P}$ of a metric space (X, d_X). Define the *padding probability* of $x \in X$ with radius t to be $\Pr[B_X(x,t) \subseteq \mathscr{P}(x)]$. The following Lemma give a lower bound for the padding probability in a CKR random partition.

Lemma 3.2. *Given an n-point metric space (X, d_X). Let $\mathscr{P}$ be a random CKR partition of (X, d_X) created with scale $\Delta > 0$. Then for every $0 < t \le \Delta/8$ and every $x \in X$,*

$$\Pr[B_X(x,t) \subseteq \mathscr{P}(x)] \ge \left(\frac{|B_X(x,\Delta/8)|}{|B_X(x,\Delta)|}\right)^{\frac{16t}{\Delta}}.$$

Proof. Let π, R be the random parameters chosen in the creation of $\mathscr{P}$ using Algorithm 1 (**CKR-Partition**). For every $r \in [\Delta/4,\ \Delta/2]$,

$$\Pr[B_X(x,t) \subseteq \mathscr{P}(x) \mid R = r] \ge \frac{|B_X(x, r-t)|}{|B_X(x, r+t)|}. \tag{1}$$

Indeed, if $R = r$, then the triangle inequality implies that if in the random order induced by the partition π on the points of the ball $B_X(x, r+t)$ the minimal element is from the ball $B_X(x, r-t)$, then $B_X(x,t) \subseteq \mathscr{P}(x)$. The aforementioned event happens with probability $\frac{|B_X(x,r-t)|}{|B_X(x,r+t)|}$.

Write $\frac{\Delta}{8t} = k + \beta$, where $\beta \in [0,1)$ and k is a positive integer. Then

$$\begin{aligned}
\Pr\left[B_X(x,t) \subseteq \mathscr{P}(x)\right] &\geq \frac{4}{\Delta}\int_{\Delta/4}^{\Delta/2} \frac{|B_X(x,r-t)|}{|B_X(x,r+t)|}dr \qquad (2)\\
&= \frac{4}{\Delta}\sum_{j=0}^{k-1}\int_{\frac{\Delta}{4}+2jt}^{\frac{\Delta}{4}+2(j+1)t} \frac{|B_X(x,r-t)|}{|B_X(x,r+t)|}dr + \frac{4}{\Delta}\int_{\frac{\Delta}{4}+2kt}^{\frac{ta}{2}} \frac{|B_X(x,r-t)|}{|B_X(x,r+t)|}dr\\
&\geq \frac{4}{\Delta}\int_0^{2t}\sum_{j=0}^{k-1}\frac{\left|B_X\left(x,\frac{\Delta}{4}+2jt+s-t\right)\right|}{\left|B_X\left(x,\frac{\Delta}{4}+2jt+s+t\right)\right|}ds + \frac{4}{\Delta}\left(\frac{\Delta}{4}-2kt\right)\frac{\left|B_X\left(x,\frac{\Delta}{4}+2kt-t\right)\right|}{\left|B_X\left(x,\frac{\Delta}{2}+t\right)\right|}\\
&\geq \frac{4k}{\Delta}\int_0^{2t}\left[\prod_{j=0}^{k-1}\frac{\left|B_X\left(x,\frac{\Delta}{4}+2jt+s-t\right)\right|}{\left|B_X\left(x,\frac{\Delta}{4}+2jt+s+t\right)\right|}\right]^{\frac{1}{k}}ds + \left(1-\frac{8kt}{\Delta}\right)\frac{\left|B_X\left(x,\frac{\Delta}{4}+2kt-t\right)\right|}{\left|B_X\left(x,\frac{\Delta}{2}+t\right)\right|} \qquad (3)\\
&= \frac{4k}{\Delta}\cdot\int_0^{2t}\left[\frac{\left|B_X\left(x,\frac{\Delta}{4}+s-t\right)\right|}{\left|B_X\left(x,\frac{\Delta}{4}+2t(k-1)+s+t\right)\right|}\right]^{\frac{1}{k}}ds + \left(1-\frac{8kt}{\Delta}\right)\frac{\left|B_X\left(x,\frac{\Delta}{4}+2kt-t\right)\right|}{\left|B_X\left(x,\frac{\Delta}{2}+t\right)\right|}\\
&\geq \frac{8kt}{\Delta}\left[\frac{\left|B_X\left(x,\frac{\Delta}{4}-t\right)\right|}{\left|B_X\left(x,\frac{\Delta}{4}+2kt+t\right)\right|}\right]^{\frac{1}{k}} + \left(1-\frac{8kt}{\Delta}\right)\frac{\left|B_X\left(x,\frac{\Delta}{4}+2kt-t\right)\right|}{\left|B_X\left(x,\frac{\Delta}{2}+t\right)\right|}\\
&\geq \left[\frac{\left|B_X\left(x,\frac{\Delta}{4}-t\right)\right|}{\left|B_X\left(x,\frac{\Delta}{4}+2kt+t\right)\right|}\right]^{\frac{8t}{\Delta}}\cdot\left[\frac{\left|B_X\left(x,\frac{\Delta}{4}+2kt-t\right)\right|}{\left|B_X\left(x,\frac{\Delta}{2}+t\right)\right|}\right]^{1-\frac{8kt}{\Delta}} \qquad (4)\\
&= \left[\frac{\left|B_X\left(x,\frac{\Delta}{4}-t\right)\right|}{\left|B_X\left(x,\frac{\Delta}{4}+2kt+t\right)\right|}\cdot\frac{\left|B_X\left(x,\frac{\Delta}{4}+2kt-t\right)\right|}{\left|B_X\left(x,\frac{\Delta}{2}+t\right)\right|}\right]^{\frac{8t}{\Delta}}\cdot\left[\frac{\left|B_X\left(x,\frac{\Delta}{4}+2kt-t\right)\right|}{\left|B_X\left(x,\frac{\Delta}{2}+t\right)\right|}\right]^{\frac{8t}{\Delta}\left(\frac{\Delta}{8t}-k-1\right)}\\
&\geq \left[\frac{\left|B_X\left(x,\frac{\Delta}{4}-t\right)\right|}{\left|B_X\left(x,\frac{\Delta}{2}+t\right)\right|}\right]^{\frac{16t}{\Delta}} \geq \left[\frac{|B_X(x,\Delta/8)|}{|B_X(x,\Delta)|}\right]^{\frac{16t}{\Delta}}.
\end{aligned}$$

where in (2) we used (1), in (3) we used the arithmetic mean/geometric mean inequality, in (4) we used the elementary inequality $\theta a + (1-\theta)b \geq a^{\theta}b^{1-\theta}$, which holds for all $\theta \in [0,1]$ and $a, b \geq 0$, and in (5) we used the fact that $\frac{\Delta}{8t} - k - 1$ is negative. □

Lemma 3.3. *Given $\alpha \geq 1$ and an n-point metric space (X, d_X). Let $\{\mathscr{E}_k\}_{k=0}^{\infty}$ be a partition tree of X, created as described in Algorithm 2 (***RamseyPartitions-ADO***), line 4-line 11. Then, for every $x \in X$,*

$$\Pr\left[\forall\, k \in \mathbb{N},\ B_X\left(x, \frac{8^{-k}\phi}{16\alpha}\right) \subseteq \mathscr{E}_k(x)\right] \geq |X|^{-\frac{1}{\alpha}}.$$

Proof. For every $x \in X$ and every $k > 0$ we have $\mathscr{E}_k(x) = \mathscr{E}_{k-1}(x) \cap \mathscr{P}_k(x)$. Thus one proves inductively that

$$\forall\, k \in \mathbb{N},\ B_X\left(x, \frac{8^{-k}\phi}{16\alpha}\right) \subseteq \mathscr{P}_k(x) \implies \forall\, k \in \mathbb{N},\ B_X\left(x, \frac{8^{-k}\phi}{16\alpha}\right) \subseteq \mathscr{E}_k(x).$$

From Lemma 3.2 and the independence of $\{\mathscr{P}_k\}_{k=0}^{\infty}$ it follows that,

$$\begin{aligned}
\Pr\left[\forall\, k \in \mathbb{N},\ B_X\left(x, \frac{8^{-k}\phi}{16\alpha}\right) \subseteq \mathscr{E}_k(x)\right] &\geq \Pr\left[\forall\, k \in \mathbb{N},\ B_X\left(x, \frac{8^{-k}\phi}{16\alpha}\right) \subseteq \mathscr{P}_k(x)\right] \\
&= \prod_{k=0}^{\infty} \Pr\left[B_X\left(x, \frac{8^{-k}\phi}{16\alpha}\right) \subseteq \mathscr{P}_k(x)\right] \\
&\geq \prod_{k=0}^{\infty} \left[\frac{\left|B_X\left(x, 8^{-k-1}\phi\right)\right|}{\left|B_X\left(x, 8^{-k}\phi\right)\right|}\right]^{\frac{1}{\alpha}} \\
&= |B_X(x,\phi)|^{-\frac{1}{\alpha}} = |X|^{-\frac{1}{\alpha}}. \qquad \square
\end{aligned}$$

Lemma 3.4. *Given an n-point metric space (X, d_X) and approximation factor $\alpha \geq 1$. Let $X_0 = X, \ldots, X_s = \emptyset$ and $Y_1, \ldots, Y_s \subseteq X$ be the subsets of X defined in Algorithm 2 (***RamseyPartitions-ADO***). Then,*

1. *$\mathbb{E}[s] \leq \alpha n^{1/\alpha}$.*
2. *For $p = 1, 2 \quad \mathbb{E}\left[\sum_{j=0}^{s} \left|X_j\right|^p\right] \leq n^{p+1/\alpha}$.*

Proof. By Lemma 3.3 we have the estimate $\mathbb{E}\left[\left|Y_j\right| \middle| Y_1, \ldots, Y_{j-1}\right] \geq \left|X_{j-1}\right|^{1-1/\alpha}$. Set

1. For $p = 0 \quad x_p = \alpha$.
2. For $p = 1, 2 \quad x_p = 1$.

Then the the claim above can be written as: $\mathbb{E}\left[\sum_{j=0}^{s} \left|X_j\right|^p\right] \leq x_p n^{p+1/\alpha}$.

The proof is by induction on n. For $n = 1$ the claim is obvious, and if $n > 1$ then by the inductive hypothesis:

$$\begin{aligned}
\mathbb{E}\left[\sum_{j=0}^{s} \left|X_j\right|^p \middle| Y_1\right] &\leq n^p + x_p |X_1|^{p+1/\alpha} \\
&= n^p + x_p n^{p+1/\alpha}\left(1 - \frac{|Y_1|}{n}\right)^{p+1/\alpha} \leq n^p + x_p n^{p+1/\alpha}\left(1 - \frac{|Y_1|}{x_p n}\right) \\
&= n^p + x_p n^{p+1/\alpha} - n^{p-1+1/\alpha}|Y_1|.
\end{aligned}$$

Taking expectation with respect to Y_1 gives the required result.

$$\begin{aligned}
\mathbb{E}\left[\sum_{j=0}^{s} \left|X_j\right|^p\right] &\leq n^p + x_p n^{p+1/\alpha} - n^{p-1+1/\alpha}\mathbb{E}[|Y_1|] \\
&\leq n^p + x_p n^{p+1/\alpha} - n^{p-1+1/\alpha} n^{1-1/\alpha} \\
&= x_p n^{p+1/\alpha}. \qquad \square
\end{aligned}$$

Now we are ready to prove the rest of the bounds on the efficiency parameters.

$O\left(n^{2+1/k} \lg n\right)$ **expected preprocessing time:** In the i-th iteration, line 3-line 16, the natural implementation of the CKR random partition used in the algorithm takes $O\left(|X_{i-1}|^2\right)$ time. The construction of a partition tree requires performing $O(\lg \Phi)$ such decompositions where $\phi = diam(X_{i-1})$. This results in $O\left(|X_{i-1}|^2 \lg \Phi\right)$ preprocessing time to sample one partition tree from the distribution. A standard technique can be used to dispense with the dependence on the aspect ratio and obtain that the expected preprocessing time of one partition tree is $O\left(|X_{i-1}|^2 \lg |X_{i-1}|\right)$. We omit describing it here as it is detailed in full in Section 4.3. The other steps in the iteration can be easily done in $O\left(|X_{i-1}|^2\right)$ time.

Using Lemma 3.4 with $p = 2$, the expected preprocessing time in this case is $O\left(\mathbb{E}\left[\sum_{j=0}^{s} \left|X_j\right|^2 \lg \left|X_j\right|\right]\right) = O\left(n^{2+1/\alpha} \lg n\right)$.

$O\left(n^{1+1/\alpha}\right)$ **Storage space:** The storage space needed to store the ultrametric $\left(X_{j-1}, \rho_j\right)$ is $O\left(\left|X_{j-1}\right|\right)$. Using Lemma 3.4 with $p = 1$, the expected storage space is $O\left(\mathbb{E}\left[\sum_{j=0}^{s} \left|X_j\right|\right]\right) = O\left(n^{1+1/\alpha}\right)$.

Constant query time: This is obvious from the query method as described in Section 3.1.

4 Preprocessing Ramsey partitions based ADOs for weighted graphs

In this section we present improved preprocessing time for ADOs on metrics defined by sparse weighted graphs.

Theorem 4.1. *Given $\alpha \geq 1$ and an undirected positively weighted graph with n vertices and m edges. It is possible to construct Ramsey partitions based $O(\alpha)$-ADO in $O\left(\alpha m n^{1/\alpha} \lg^3 n\right)$ expected time and $O\left(n^{1+1/\alpha}\right)$ storage space.*

Most of this section is devoted to the proof of Theorem 4.1. We show how to obtain $O\left(m \lg^3 n\right)$ expected time for each iteration of Algorithm 2 (**RamseyPartitions-ADO**), line 3-line 16. By Lemma 3.4 we know that the expected number of iterations is $\alpha n^{1/\alpha}$, and so the total expected prerocessing time is $O\left(\alpha m n^{1/\alpha} \lg^3 n\right)$.

In the i-th iteration, let $U \subseteq V$ be the set of vertices which are not part of the padded set in any of iterations $j < i$. The iteration consists of the following steps. In the rest of the section we explain how to execute them in the required time bound.

Remark 4.1. When the metric is represented as a subset of a weighted graph (U, d_G) where $G = (V, E)$ and $U \subseteq V$, the running time of each iteration is linear in the size of the graph G, which may be much larger than $|U|$. This is because all vertices and edges in the graph are part of the graph shortest path metric and have to be processed for distance calculations.

1. Computing the diameter, line 4: $\operatorname{diam}(U)$ can be $1/2$-approximated using USSSP algorithm, starting with arbitrary vertex. The oracle approximation is multiplied by a factor of 2, and becomes 256α. The rest of the oracle efficiency parameters are the same.

2. Computing a CKR partition, line 8: Algorithm 3 (**GraphPartition-CKR**) in Section 4.1.

3. Picking the padded vertices set, line 12: Set $\phi = \operatorname{diam}(U)$. We show how to compute

$$\left\{ v \in U \mid \forall\, k \in \mathbb{N},\ B_G\left(v, \frac{8^{-k}\phi}{16\alpha}\right) \subseteq \mathscr{E}_k(v) \right\}.$$

Lemma 4.2. *For each $v \in U$,*

$$\forall\, k \in \mathbb{N},\ B_G\left(v, \frac{8^{-k}\phi}{16\alpha}\right) \subseteq \mathscr{P}_k(v) \Longleftrightarrow \forall\, l \in \mathbb{N},\ B_G\left(v, \frac{8^{-l}\phi}{16\alpha}\right) \subseteq \mathscr{E}_l(v).$$

Proof. $\Longleftarrow$ This is implied from the fact that $\forall\, k \in \mathbb{N}$, $\mathscr{E}_k$ is a refinement of $\mathscr{P}_k$.

$\Longrightarrow$ We prove that $\forall\, l \in \mathbb{N}$, $B_G\left(v, \frac{8^{-l}\phi}{16\alpha}\right) \subseteq \mathscr{E}_l(v)$ inductively (as mentioned in [42]). For $l = 0$ this is trivial. Suppose it is true for l. Let $v \in U$ having $\forall\, k \in \mathbb{N}$, $B_G\left(v, \frac{8^{-k}\phi}{16\alpha}\right) \subseteq \mathscr{P}_k(v)$. By the induction hypothesis we know that $B_G\left(v, \frac{8^{-l}\phi}{16\alpha}\right) \subseteq \mathscr{E}_l(v)$, so $B_G\left(v, \frac{8^{-l-1}\phi}{16\alpha}\right) \subseteq B_G\left(v, \frac{8^{-l}\phi}{16\alpha}\right) \subseteq \mathscr{E}_l(v) \cap \mathscr{P}_{l+1}(v) = \mathscr{E}_{l+1}(v)$. As needed. □

So for computing the set of padded vertices for the partition tree it is enough to compute them for each $\mathscr{P}_k$ separately and then take the intersection. We do that using Algorithm 4 (**PaddedVertices**) in Section 4.2.

4. Dispensing with the $\lg\phi$ factor, line 3-line 16: Naturally, the partition tree includes $\lg\phi$ partitions. Constructing the partition tree and picking the padded vertices set can be done using $O(\lg\phi)$ applications of Algorithm 3 (**GraphPartition-CKR**) and Algorithm 4 (**PaddedVertices**) each done in $O\left(m\lg^2 n\right)$ time. The construction of the ultrametric take additional $O(n\lg\phi)$ time. Resulting in total running time of $O\left(m\lg^2 n\lg\phi\right)$. In Section 4.3 we show how to dispense with the $\lg\phi$ factor and replace it with a $\lg n$ factor. That is, the resulting total running time is $O\left(m\lg^3 n\right)$.

4.1 Computing a CKR random partition

Given an undirected positively weighted graph $G = (V, E, w)$ with n vertices and m edges, $U \subseteq V$ and $\Delta > 0$. We show how to implement Algorithm 1 (**CKR-Partition**) in $O\left(m\lg^2 n\right)$ expected time, resulting in partition $\mathscr{P}$ of U. Furthermore, we extend $\mathscr{P}$ definition to include all V vertices. That extension is used in the next section.

First, we compute the random permutation π. π can be generated in linear time using several methods *e.g.,* Knuth Shuffle (see [11]). We set R value in the range $\left[\frac{\Delta}{4}, \frac{\Delta}{2}\right]$.

We use Dijkstra's algorithm for computing balls. The algorithm performs $|U|$ iterations, in the i-th iteration, all yet-unpartitioned vertices in $B_G(x_{\pi(i)}, R)$ are put in C_i. In order to gain the improved running time, we change Dijkstra's algorithm as follows. Consider the i-th iteration and let $\delta(v)$ be the variable that holds the Dijkstra algorithms current estimate on the distance between $\pi(i)$ and v. Usually, in Dijsktras algorithm $\delta(v)$ is initialized to ∞ and then gradually decreased until u is extracted from the priority queue, at which point $\delta(v) = d_G(\pi(i), v)$. In the variant of the algorithm we use, the $\delta(\cdot)$ are not reinitialized when the value of i is changed. This means that now at the end of the $(i-1)$-th iteration, $\delta(v) = \min_{j<i} d_G(\pi(j), v)$, which in turn means that an edge (u, w) is relaxed in the i-th iteration only when $\pi(i)$ is the closest center to u, and w among $\pi(j)$, $j \geq i$. This dramatically reduces the number of relaxations being done, and does not hurt the correctness of the algorithm.

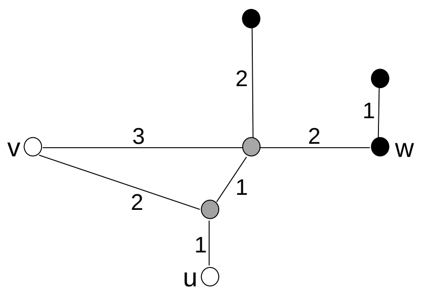

Figure 1: *A schematic description of the behavior of the modified Dijkstra's algorithm. It is assumed that* $U = \{v, w\}$, $\pi(1) = v$, $\pi(2) = w$ *and* $R = 4$. *The white vertices are those visited in the first iteration only. The gray vertices are those visited in both iterations. The black vertices are those visited in the second iteration only. Observe that u is not visited in the second iteration although its distance from w is* 4.

Algorithm 3 GraphPartition-CKR

```
Input: Graph $G = (V, E, w)$, $U \subseteq V$, scale $\Delta > 0$
Output: Partition $\mathscr{P}$ of $V$
 1: Generate random permutation $\pi$ of $U$
 2: Chose randomly $R$ in $\left[\frac{\Delta}{4}, \frac{\Delta}{2}\right]$
 3: for all $v \in V$ do
 4:     $\delta(v) := \infty$
 5:     $\mathscr{P}(v) := 0$
 6: end for
 7: for $i := 1$ to $|U|$ do  // Perform modified Dijkstra's algorithm starting from $\pi(i)$
 8:     $Q := \{\pi(i)\}$  // $Q$ is a priority queue with $\delta$ being the key
 9:     $\delta(\pi(i)) = 0$
10:     while $Q$ is not empty do
11:         Extract $w \in Q$ with minimal $\delta(w)$  // $w$ is visited now
12:         if $\mathscr{P}(w) = 0$ then
13:             $\mathscr{P}(w) := i$
14:         end if
15:         for all $(u, w) \in E$ do  // Relax $w$ edges
16:             if $\delta(u) > \delta(w) + w(u, w)$ and $\delta(w) + w(u, w) \leq R$ then
17:                 $\delta(u) = \delta(w) + w(u, w)$
18:                 $Q := Q \cup \{u\}$
19:             end if
20:         end for
21:     end while
22: end for
```

Lemma 4.3. *For all* $v \in V$, *let* $\delta^*(v)$ *be* $\delta(v)$ *value just before the i-th iteration, and* $\delta'(v)$, *the value immediately after that. In the i-th iteration, the modified Dijkstra's algorithm visits all the vertices* v *where* $d_G(\pi(i), v) \leq R$ *and there exists a shortest path* $\pi(i) = v_0, v_1, \ldots, v_\ell = v$ *between* $\pi(i)$ *and* v *where* *for every $t \in \{1, \ldots, \ell\}$, $d_G(\pi(i), v_t) < \delta^*(v_t)$. For each vertex, $\delta'(v) = d_G(\pi(i), v)$ is computed correctly.*

Proof. Similar modification is used in [52, Lemma 4.2] where a visited vertex relaxes its edges if its distance from the source is no more than a pre-defined constant. The correctness proof for both cases is similar and we omit the details. $\square$

Lemma 4.4. *After the i-th iteration of the loop on lines 7–22 of Algorithm 3,*

$$\delta(v) = \begin{cases} \min_{j\le i} d_G(\pi(j), v) & \text{if } \min_{j\le i} d_G(\pi(j), v) \le R, \\ \infty & \text{otherwise.} \end{cases} \tag{5}$$

Proof. Proof by induction on i. When $i = 1$, the proof is followed from Lemma 4.3.

Assume inductively that (5) is correct for $i-1$. If $\min_{j\le i-1} d_G(\pi(j), v) \le d_G(\pi(i), v)$, then clearly, by Lemma 4.3, the i-th iteration will not change $\delta(v)$, and by the inductive hypothesis we are done.

Assume now that $\min_{j\le i-1} d_G(\pi(j), v) > d_G(\pi(i), v)$, and $d_G(\pi(i), v) \le R$. Let $\pi(i) = v_0, v_1, \ldots, v_\ell = v$ be a shortest path between $\pi(i)$ and v. We claim that for every $t \in \{1, \ldots, \ell\}$, $\min_{j\le i-1} d_G(\pi(j), v_t) > d_G(\pi(i), v_t)$, since otherwise we had

$$\min_{j\le i-1} d_G(\pi(j), v) \le \min_{j\le i-1} d_G(\pi(j), v_t) + d_G(v_t, v) \le d_G(\pi(i), v_t) + d_G(v_t, v) = d_G(\pi(i), v).$$

Hence, by Lemma 4.3, v will be visited in the i-th iteration, and in the end of the iteration, $\delta(v) = d_G(\pi(i), v)$. $\square$

Lemma 4.5. *Given an undirected positively weighted graph $G = (V, E, w)$ with n vertices and m edges, $U \subseteq V$ and $\Delta > 0$. Algorithm 3 (***GraphPartition-CKR***) samples a CKR partition, $\mathcal{P}$, of U at scale Δ. Furthermore, $\mathcal{P}$ is extended to include all the vertices of V as follows. Let π, R be the random parameters defining $\mathcal{P}$ then for $v \in V$, $\mathcal{P}(v) = \min\{i \mid v \in B_G(\pi(i), R)\}$. The expected running time is $O\left(m \lg^2 n\right)$.*

Proof. We first prove the correctness of the sampling algorithm: $\mathcal{P}(v) = \min\{i \mid v \in B_G(\pi(i), R)\}$ for every $v \in V$. Let $i_0 = \min\{i \mid d_G(\pi(i), v) \le R\}$. This means that $\min_{j<i_0} d_G(\pi(j), v) > R \ge d_G(\pi(i_0), v)$. By Lemma 4.4 at the beginning of the i_0-th iteration, $\delta(v) = \infty$, and hence $\mathcal{P}(v) = 0$ and by the end of the (i_0)-th iteration, $\delta(v) = d_G(\pi(i_0), v)$, and necessarily $\mathcal{P}(v) = i$. Note that once $\mathcal{P}(v)$ is set to a value different than 0, it does not change until the end.

Bounding the running time, we will that each edge of G undergone $O(\lg n)$ relaxations in expectation. We fix $v \in V$, and bound the expected number of relaxations performed for each adjacent edge, via v. Consider the non-increasing sequence $a_i = \min_{j\le i} d_G(\pi(j), v)$. In the i-th iteration the edges adjacent to v are relaxed if and only if $a_{i-1} > a_i$. Note that $a_{i-1} > a_i$ means that $d_G(\pi(i), v)$ is the minimum among $\{d_G(\pi(j), v) \mid j \le i\}$, and the probability (over π) for this to happen is at most $1/i$. Therefore, the expected number of relaxations edges adjacent to v undergo via v is at most

$$\sum_{i=1}^{n} \frac{1}{i} \le 1 + \ln n.$$

By linearity of expectation, the expected total number of relaxations is

$$O\left(\sum_{v\in V} \ln n \cdot \deg(v)\right) = O(m \lg n).$$

The total running time of Dijkstra's algorithm in iteration i is $O\left(m'_i \lg n\right)$, where m'_i is the number of relaxations. As was shown above, $\sum_i m'_i = O(m \lg n)$, and therefore the total running time is $O\left(m \lg^2 n\right)$.

□

4.2 Picking $O(\alpha)$-padded vertices

Assuming we are given a CKR partition $\mathscr{P}$ of U (at scale Δ, generated using a permutation π and $R \in \left[\frac{\Delta}{4}, \frac{\Delta}{2}\right]$) and a padding radius t. Our goal in this section is to pick the t-padded points in $\mathscr{P}$.

We use the extension of $\mathscr{P}$ to V where for $v \in V \setminus U$, $\mathscr{P}(v) = \min\{i \mid v \in B_G(\pi(i), R)\}$ as produced by Algorithm 3 (**GraphPartition-CKR**).

Algorithm 4 PaddedVertices

```
Input: Graph $G = (V, E, w)$, $U \subseteq V$, partition $\mathscr{P}$ of $V$, padding radius $t > 0$
Output: $S \subseteq U$
1: Create a new dummy vertex $\gamma \in V$
2: for all $(u, v) \in E$ where $\mathscr{P}(v) \neq \mathscr{P}(u)$ do
3:     Connect $v$ to $\gamma$ with edge length $w(u, v)$
4: end for
5: For all $v \in U$ compute the distance $d_G(\gamma, v)$ using Dijkstra's algorithm
6: $S := \{v \in U \mid d_G(\gamma, v) > t\}$
```

Lemma 4.6. *Given an undirected positively weighted graph $G = (V, E, w)$ with n vertices and m edges, a subset of the vertices, $U \subseteq V$, an extended CKR partition $\mathscr{P}$ of V generated using a permutation π and $R \in \left[\frac{\Delta}{4}, \frac{\Delta}{2}\right]$ and a padding radius t. Algorithm 4 (***PaddedVertices***) computes a padded set, where a padded vertex $v \in U$ satisfies $U \cap B_G(v, t) \subseteq \mathscr{P}(v)$. For random π, R, the probability of a vertex v to be picked is at least $\left(\frac{|U\cap B_G(v,\Delta/8)|}{|U\cap B_G(v,\Delta)|}\right)^{\frac{16t}{\Delta}}$ (as in Lemma 3.2). The running time of the algorithm is $O(m \lg n)$.*

Proof. The output is exactly the set of vertices $u \in U$ for which $B_G(u, t) \subseteq \mathscr{P}(u)$. Obviously, u is padded. In order to estimate the size of the output, observe that the proof of Lemma 3.2 actually bounds the number of points $u \in U$ for which $\pi(\min\{j \mid \pi(j) \in B_G(u, R + t)\}) \in B_G(u, R - t)$. Algorithm 4 produces those points, and therefore the quantitative analysis of in the proof Lemma 3.2 also holds for the padded set produced by Algorithm 4.

Regarding the running time, observe that the initialization steps take $O(1)$ time. Defining the new vertex γ takes a constant time for each edge, and the running time of Dijkstra's algorithm is $O(m \lg n)$.

□

4.3 Dispensing with the $\lg \phi$ factor

In this section we explain how we can dispense with the $O(\lg \phi)$ factor in the running time of the algorithm and replace it with $O(\lg n)$. The method being used is standard. Similar ideas appeared previously, *e.g.*, in [3, 29, 42]. However, the context here is slightly different. Also, the designated time bound is $\tilde{O}(m)$, which is faster then the time bounds in the papers we are aware of. We give here a detailed description of the implementation.

In the naïve approach, the number of scales processed for constructing $\{\mathscr{E}_k\}_{k=0}^{\infty}$ is $\Theta(\lg \phi)$. For each integer $k \geq 0$, we compute a CKR partition $\mathscr{P}_k$ of U at scale $\Delta_k = 8^{-k} \cdot \operatorname{diam}(U)$ and pick the $\frac{\Delta_k}{16\alpha}$-padded vertices in $O\left(m \lg^2 n\right)$ time. S is the set of vertices padded in all scales. For creating the ultrametric (U, ρ) HST representation we process the partition tree $\{\mathscr{E}_k\}_{k=0}^{\infty}$ in time linear in its size, $O(n \lg \phi)$.

Here we bound the total processing time of those tasks by $O\left(m \lg^3 n\right)$. We define for each scale an appropriate quotient of the input graph. We show how the processing can be performed on those substitutive graph metrics while retaining the properties of the original algorithm. Also, using those quotients, not all scales need to be processed. The total size of the quotient graphs in all processed scales is $O(m \lg n)$.

We begin with defining a scale Δ quotient graph for $\Delta \geq 0$, and prove its useful property: retaining the padding probability. For $Y, Y' \subseteq X$, let $d_X(Y, Y') = \min\{d_X(x, y) \mid x \in Y, y \in Y'\}$.

Definition 4.7. The space (Y, d_Y) is called *a scale* Δ *quotient of* (X, d_X) if Y is a Δ-bounded partition of X, and

$$d_Y(y, y') = \min\left\{ \sum_{j=1}^{l} d_X\left(y_{j-1}, y_j\right) \,\middle|\, y_0, \ldots, y_l \in Y, y_0 = y, y_l = y' \right\}$$

Lemma 4.8. *Fix $\Delta > 0$, and let (Y, d_Y) be a $\frac{\Delta}{2n}$ scale quotient of (X, d_X). Set $\sigma : X \to Y$ to be the natural projection, assigning each vertex $x \in X$ to its cluster $Y(x)$. Let $\mathscr{L}$ be a CKR partition of Y at scale $\Delta/2$. $\mathscr{L}$ is a probability distribution* Pr *such that for every $0 < t \leq \Delta/16$ and every $x \in X$,*

$$\Pr[B_Y(\sigma(x), t) \subseteq \mathscr{L}(\sigma(x))] \geq \left(\frac{|B_X(x, \Delta/16)|}{|B_X(x, \Delta)|}\right)^{\frac{16t}{\Delta}}. \tag{6}$$

Additionally, let $\mathscr{P}$ be the pullback of $\mathscr{L}$ under σ, i.e., *$\mathscr{P} = \left\{\sigma^{-1}(A) \,\middle|\, A \in \mathscr{L}\right\}$. $\mathscr{P}$ is a Δ-bounded partition of X such that for every $0 < t \leq \Delta/16$ and every $x \in X$,*

$$B_Y(\sigma(x), t) \subseteq \mathscr{L}(\sigma(x)) \Longrightarrow B_X(x, t) \subseteq \mathscr{P}(x). \tag{7}$$

Proof. From Lemma 3.2 we know that for every $0 < t \leq \Delta/16$ and every $x \in Y$ we have

$$\Pr[B_Y(x, t) \subseteq \mathscr{L}(x)] \geq \left(\frac{|B_Y(x, \Delta/16)|}{|B_Y(x, \Delta/2)|}\right)^{\frac{16t}{\Delta}}. \tag{8}$$

For each $x, y \in X$ [41, Lemma 5]:

$$d_X(x,y) - \frac{\Delta}{2} \le d_Y(\sigma(x), \sigma(y)) \le d_X(x,y). \tag{9}$$

The upper bound in (9) is immediate from the definition of a quotient metric. The lower bound in (9) is proved as follows. There are points $x = x_0, \dots, x_k = y$ in X such that $d_Y(\sigma(x), \sigma(y)) = \sum_{j=1}^{k} d_Y\left(\sigma\left(x_{j-1}\right), \sigma\left(x_j\right)\right)$. For $j \in \{1, \dots, k\}$ let $a_j \in \sigma\left(x_{j-1}\right)$ and $b_j \in \sigma\left(x_j\right)$ be such that $d_X\left(a_j, b_j\right) = d_X\left(\sigma\left(x_{j-1}\right), \sigma\left(x_j\right)\right)$. Then, since $k \le n-1$ and for all $z \in X$ we have $\operatorname{diam}(\sigma(z)) = \max_{a,b\in\sigma(z)} d_X(a,b) \le \frac{\Delta}{2n}$, we get that

$$\begin{aligned} d_X(x,y) \le d_X(x,a_1) + \sum_{j=1}^{k} d_X\left(a_j, b_j\right) + \sum_{j=1}^{k-1} d_X\left(b_j, a_{j+1}\right) + d_X(b_k, y) \\ \le \frac{\Delta}{2n} + d_Y(\sigma(x), \sigma(y)) + (n-2)\frac{\Delta}{2n} + \frac{\Delta}{2n}, \end{aligned}$$

implying the lower bound in (9).

Inequality (9) implies that for every $x \in X$ we have,

$$\sigma^{-1}\left(B_Y(\sigma(x), \Delta/2)\right) \subseteq B_X(x, \Delta) \tag{10}$$

and for every $t > 0$,

$$\sigma^{-1}\left(B_Y(\sigma(x), t)\right) \supseteq B_X(x, t) \tag{11}$$

Observe that (8), (10) and (11) implies (6). Also, (11) implies (7). Finally, (9) means that $\mathscr{P}$ is Δ bounded. □

The next Lemma, 4.10, together with Lemma 4.8 is the basis for dispensing with the dependence on the aspect ratio. It states that given a graph $G = (V, E, w)$, $U \subseteq V$ and $\Delta > 0$, some of the edges and vertices in G are irrelevant for computing a random CKR partition of U at scale Δ.

We start with defining $\mathit{Limited}_\Delta(G)$ as the graph G reduced to edges with maximum weight Δ and non-isolated vertices.

Definition 4.9. Given a weighted graph $G = (V, E, w)$ and $\Delta > 0$. Define the graph $\mathit{Limited}_\Delta(G) = (V^* \subseteq V, E^* \subseteq E, w^*)$ as follows. $E^* = \{(u,v) \in E \mid w(u,v) \le \Delta\}$, $V^* = \{u \in V \mid \exists v \in V,\ (u,v) \in E^*\}$ and $w^*(u,v) = w(u,v)$. Additionally, given $U \subseteq V$, define $\mathit{Limited}_\Delta(U) = U \cap V^*$.

Lemma 4.10. *Given a weighted graph $G = (V, E, w)$, $U \subseteq V$ and $\Delta > 0$. Let $G^* = (V^*, E^*, w^*)$ where $G^* = \mathit{Limited}_\Delta(G)$ and $U^* = \mathit{Limited}_\Delta(U)$. Let $\mathscr{L}$ be a random CKR partition of U^*, using the metric induced by G^*, at scale Δ. $\mathscr{P} = \mathscr{L} \cup \{\{v\} \mid v \in U \setminus U^*\}$ is a random CKR partition of U, using the metric induced by G, at scale Δ.*

Proof. Observe that for computing a CKR partition of (U, d_G) at scale Δ we only need to use paths at G with weight up to Δ. That is, no edge weighting more than Δ is needed. Also, for each $v \in U \setminus U^*$, $B_G(v, \Delta) \setminus \{v\} = \emptyset$, *i.e.*, in any CKR partition of U, v is partitioned alone. □

We next sketch the scheme we use. Construct an ultrametric ρ on V, represented by an HST $H = (T, \Gamma)$ such that for every $u, v \in V$, $d_G(u,v) \le \rho(u,v) \le n d_G(u,v)$. Given $\Delta \ge 0$, set $\sigma_\Delta : T \to T$ such that for $v \in T$, $\sigma_\Delta(v)$ is the highest ancestor u of v for which $\Gamma(u) \le \frac{\Delta}{2n}$. In Algorithm 6 (**Init**) we explain how to construct H and its supportive data structures in $O(m \lg n)$ time.

Use σ_Δ for constructing a weighted graph $G(\Delta)$ as follows. $G(\Delta) = (V(\Delta), E(\Delta), w(\Delta))$ where $V(\Delta) = \{\sigma_\Delta(v) \mid v \in V\}$ and $E(\Delta) = \{(\sigma_\Delta(u), \sigma_\Delta(v)) \mid (u,v) \in E, \sigma_\Delta(u) \neq \sigma_\Delta(v)\}$. For $(u,v) \in E(\Delta)$, $w(\Delta)(u,v) = \min\{w(w,z) \mid \sigma_\Delta(w) = u, \sigma_\Delta(z) = v\}$. Also, $U(\Delta) = \{\sigma_\Delta(v) \mid v \in U\}$.

Then, $(U(\Delta), d_{G(\Delta)})$ is a scale $\frac{\Delta}{2n}$ quotient of (U, d_G).

Given an integer $j \ge 0$. Set $\Delta_j = 8^{-j} \cdot \operatorname{diam}(U)$, $G_j = \left(V_j, E_j, w_j\right)$ where $G_j = \mathit{Limited}_{\Delta_j/2}\left(G\left(\Delta_j\right)\right)$ and $U_j = \mathit{Limited}_{\Delta_j/2}\left(U\left(\Delta_j\right)\right)$.

Let *Processed* be the set of integers $j \ge 0$ where $V_j \neq \emptyset$. The following Lemma gives upper bound on the total size of the graphs G_j for all $j \in \mathit{Processed}$.

Lemma 4.11.

$$\sum_{j \in \mathit{Processed}} \left(\left|V_j\right| + \left|E_j\right|\right) = O(m \lg n).$$

Proof. Given $j \in \mathit{Processed}$. For each $(u,v) \in E$ observe that $\left(\sigma_{\Delta_j}(u), \sigma_{\Delta_j}(v)\right) \in E\left(\Delta_j\right)$ if and only if $\sigma_{\Delta_j}(u) \neq \sigma_{\Delta_j}(v)$. Then, by $E\left(\Delta_j\right)$ definition, $w(u,v) \ge d_G(u,v) \ge \frac{\Delta_j}{2n^2}$. Also, given $\left(\sigma_{\Delta_j}(u), \sigma_{\Delta_j}(v)\right) \in E\left(\Delta_j\right)$, $\left(\sigma_{\Delta_j}(u), \sigma_{\Delta_j}(v)\right) \in E_j$ if and only if $w\left(\sigma_{\Delta_j}(u), \sigma_{\Delta_j}(v)\right) \le \frac{\Delta_j}{2}$. That is, each edge in G is represented in G_j only when $w(u,v) \in \left[\frac{\Delta_j}{2n^2}, \frac{\Delta_j}{2}\right]$. A total of $O(\lg n)$ scales. By definition, G_j contains only non-isolated vertices, so $\forall j$, $\left|V_j\right| \le \left|E_j\right|$. □

For all $j \in \mathit{Processed}$ we use Algorithm 7 (**Quotient**) to compute G_j and U_j. The algorithm initialization time is $O(m \lg n)$ and the total running time of all its subsequent calls is $O\left(m \lg^2 n\right)$.

Using Algorithm 3 (**GraphPartition-CKR**) and Algorithm 4 (**PaddedVertices-CKR**) we calculate for all $j \in \mathit{Processed}$, $\left(\mathscr{L}_j, R_j\right)$ where $\mathscr{L}_j$ is a CKR partition of U_j, using the metric induced by G_j, at scale $\frac{\Delta_j}{2}$ and R_j is the set of $\frac{\Delta_j}{16\alpha}$-padded vertices in total $O\left(m \lg^3 n\right)$ running time .

For $j \notin \mathit{Processed}$, let $\mathscr{L}_j = \emptyset$. For $j \ge 0$, let $\mathscr{T}_j = \mathscr{L}_j \cup \left\{\{u\} \mid u \in U\left(\Delta_j\right) \setminus U_j\right\}$. Lemma 4.10 implies that for all $j \ge 0$, $\mathscr{T}_j$ is a CKR partition of $U\left(\Delta_j\right)$ at scale $\Delta_j/2$. Let $\mathscr{P}_j = \left\{\sigma_{\Delta_j}^{-1}(A) \;\middle|\; A \in \mathscr{T}_j\right\}$. Lemma 4.8 implies that for all $j \ge 0$, $\mathscr{P}_j$ padding probability is at least as for a CKR partition of U at scale Δ_j. Let $\mathscr{E}_0 = \{U\}$ and $\mathscr{E}_j$ be the refinement of $\mathscr{P}_j$ and $\mathscr{E}_{j-1}$. Lemma 3.3 implies that $\left\{\mathscr{E}_j\right\}_{j=0}^{\infty}$ is Ramsey partitions on U which are $O(\alpha)$-padded with exponent $O(\alpha)$. Lemmas 4.8 and 3.3 imply that the set

$$S = \left\{v \in V \mid \forall j \ge 0, B_{G(\Delta_j)}\left(\sigma_{\Delta_j}(v), \frac{\Delta_j}{16\alpha}\right) \subseteq \mathscr{T}_j\left(\sigma_{\Delta_j}(v)\right)\right\}$$

is $O(\alpha)$-padded set with expected size $\Omega\left(n^{1-1/\alpha}\right)$. As computing $\left\{\mathscr{E}_j\right\}_{j=0}^{\infty}$ and S directly is too time-consuming, we do it indirectly as follows.

Given $i, j \in \mathit{Processed}$, $i < j$, define the refinement of $\mathscr{L}_j$ with $\mathscr{L}_i$ as

$$\left\{C \subseteq C' \in \mathscr{L}_j \;\middle|\; \forall u, v \in C, \sigma_{\Delta_i}(u) \neq \sigma_{\Delta_i}(v) \Rightarrow \mathscr{L}_i(\sigma_{\Delta_i}(u)) = \mathscr{L}_i(\sigma_{\Delta_i}(v))\right\}.$$

Set $\mathscr{W}_0 = \mathscr{L}_0$. Algorithm 9 (**Refinement**) calculate $\left\{\mathscr{W}_j\right\}_{j\in Processed}$ where $\mathscr{W}_j$ is the result of iteratively refining $\mathscr{L}_j$ with $\{\mathscr{L}_k\}_{k<j,\ k\in Processed}$ in $O\left(m\lg^2 n\right)$ time.

For $j \notin Processed$, let $U_j = \mathscr{W}_j = \emptyset$. It is easy to verify that for $j \geq 0$, $\mathscr{E}_j$ is equivalent to

$$\left\{\sigma_{\Delta_j}^{-1}(A) \;\middle|\; A \in \left(\mathscr{W}_j \cup \left\{\{u\} \mid u \in U\left(\Delta_j\right) \setminus U_j\right\}\right)\right\}.$$

For all $j \geq 0$, the isolated vertices in $U\left(\Delta_j\right)$, which are exactly $U\left(\Delta_j\right) \setminus U_j = \left\{v \in V \mid \sigma_{\Delta_j}(v) \notin U_j\right\}$, are $\frac{\Delta_j}{16\alpha}$-padded in $\mathscr{T}_j$. We can deduce that S is equivalent to

$$\left\{v \in V \mid \forall j \in Processed, \sigma_{\Delta_j}(v) \in U_j \Rightarrow \sigma_{\Delta_j}(v) \in R_j\right\}.$$

Algorithm 8 (**Padded**) process $\left\{R_j\right\}_{j\in Processed}$ in $O(m\lg n)$ time and produce S.

Algorithm 10 (**UM**) process $\left\{\mathscr{W}_j\right\}_{j\in Processed}$ in $O\left(m\lg^2 n\right)$ time and create the HST representation of the ultrametric (U,ρ) defined by the partition tree $\left\{\mathscr{E}_j\right\}_{j=0}^{\infty}$ as $\rho(u,v) = \Delta_k$ where $k := \max\left\{j \mid \mathscr{E}_j(u) = \mathscr{E}_j(v)\right\}$.

Lemma 4.12. *Given an undirected positively weighted graph $G = (V,E,w)$ with n vertices and m edges, $U \subseteq V$ and $\alpha \geq 1$. Algorithm 5 (***Graph-RamseyPartitions***) construct a subset S of U with expected size $\Omega\left(n^{1-1/\alpha}\right)$ and an ultrametric (U,ρ) in HST representation where the metric d_G restricted to the set of pairs $u \in U$ and $v \in S$ is $O(\alpha)$ equivalent to the ultrametric (U,ρ). The running time of the algorithm is $O\left(m\lg^3 n\right)$.*

4.4 Preprocessing time / storage space trade-off

We conclude the section with preprocessing time / storage space trade-offs when the metrics are given in matrix representation.

Theorem 4.13. *Fix integers $k \geq 1$, $\beta \geq 1$ and an undirected positively weighted graph on n vertices and m edges. Let $\alpha = (2k-1)\cdot\beta$. It is possible to construct Ramsey partitions based $O(\alpha)$-ADO in $O\left(km + \alpha n^{1+1/k+1/\beta}\lg^3 n\right)$ time and $O\left(n^{1+1/\beta}\right)$ storage space.*

For the proof we use the following theorem,

Theorem 4.14. *[6] Given a weighted graph $G = (V,E,w)$ with n vertices and m edges, and an integer $k \geq 1$, a spanner of $(2k-1)$-stretch and $O\left(kn^{1+1/k}\right)$ size can be computed in $O(km)$ expected time.*

Proof of Theorem 4.13. First, using Theorem 4.14, we construct in $O(km)$ time an $2k-1$−spanner of G with $O\left(kn^{1+1/k}\right)$ edges. Then we compute $O(\beta)$-ADO as explained in this section on the spanner. The running time is $O\left(\beta\left(kn^{1+1/k}\right)n^{1/\beta}\lg^3 n\right) = O\left(\alpha n^{1+1/k+1/\beta}\lg^3 n\right)$. The approximation factor is $O(k\beta) = O(\alpha)$. □

Using threom 4.13 we can derive the following result for general metric spaces.

Algorithm 5 Graph-RamseyPartitions

```
Input: Graph $G = (V, E, w)$, $U \subseteq V$, approximation factor $\alpha \geq 1$
Output: An HST $H = (T, \Gamma)$, $S \subseteq U$
 1: $H :=$ Init$(G)$
 2: $\phi := \mathrm{diam}(U)$
 3: $Processed := \left\{\text{integer } j > 0 \;\middle|\; (u,v) \in E, 8^{-j}\phi \in \left[2 \cdot w(u,v), 2|V|^2 \cdot w(u,v)\right]\right\}$
 4: for all $j \in Processed$ in increasing order do  // $V_j \neq \emptyset$
 5:     $\Delta_j := 8^{-j}\phi$
 6:     $(G_j, U_j) :=$ Quotient $(H, G, U, \Delta_j)$
 7:     $\mathscr{L}_j^* :=$ GraphPartition-CKR $\left(G_j, U_j, \frac{\Delta_j}{2}\right)$
 8:     $R_j :=$ PaddedVertices $\left(G_j, U_j, \mathscr{L}_j^*, \frac{\Delta_j}{16\alpha}\right)$
 9:     $\mathscr{L}_j := \left\{C \cap U_j \;\middle|\; C \in \mathscr{L}_j^*\right\}$
10: end for
11: $S :=$ Padded $\left(H, U, \left\{U_j, R_j\right\}_{j \in Processed}\right)$
12: $\left\{\mathscr{W}_j\right\}_{j \in Processed} :=$ Refinement $\left(H, \left\{U_j, \mathscr{L}_j\right\}_{j \in Processed}\right)$
13: $(T, \Gamma) :=$ UM $\left(H, U, \left\{U_j, \mathscr{W}_j, \Delta_j\right\}_{j \in Processed}\right)$
```

Algorithm 6 Init

```
Input: Graph $G = (V, E, w)$
Output: An HST $H = (T, \Gamma)$ which is $(|V| - 1)$ approximation of $G$ supporting the following
    queries:
      i. Given $u \in T$, compute $l(u) = \min\{i \mid v_i$ is a descendant of $u\}$ in $O(1)$ time.
     ii. Given $u \in T$, compute $s(u) = |\{w \mid w$ is a descendant of $u\}|$ in $O(1)$ time.
    iii. Given $u \in T$, $\Delta \geq 0$, compute $\sigma_\Delta(u)$ = the highest ancestor of $u$ for which $\Gamma(u) \leq \frac{\Delta}{2|V|}$.
 1: Construct an ultrametric $\rho$ on $V$ in HST representation $H = (T, \Gamma)$ such that for every
    $u, v \in V$, $d_G(u,v) \leq \rho(u,v) \leq |V| d_G(u,v)$
    // The fact that such HST exists is shown in [4, Lemma 3.6], and it can be constructed in time
    $O(|E| + |V| \lg |V|)$. This implementation is done in [29, Section 3.2].
 2: Let $T = \{u_1, \dots, u_{2|V|-1}\}$
 3: for all $u \in T$ do  // Use depth-first-search
 4:     $l(u) = \min\{i \mid u_i$ is a descendant of $u\}$
 5:     $s(u) = |\{w \mid w$ is a descendant of $u\}|$
 6: end for
 7: Process $H$ such that, given $u \in T$ and $\Delta \geq 0$ the following query is answered in $O(\lg |V|)$
    time: $\sigma_\Delta(u)$, the highest ancestor $v$ of $u$, where $\Gamma(v) \leq \frac{\Delta}{2|V|}$ (or undefined if $\Gamma(u) > \frac{\Delta}{2|V|}$)
    // The implementation idea originates at [29, Section 3.5]. The level-ancestor problem is defined
    as follows. Suppose a rooted l-vertex tree is given for preprocessing. Answer quickly queries of the
    following form. Given a vertex v and an integer i, find an ancestor of v whose level is i, where the
    level of the root is 0. Bender and in Farach-Colton [8] show how we can answer level ancestor
    queries in constant time using $O(l)$ preprocessing time. For all $u \in T$, we use binary search applying
    $O(\lg |V|)$ level ancestors queries to locate $\sigma_\Delta(u)$ in $O(\lg |V|)$ time.
```

Algorithm 7 Quotient

```
Input: An HST $H = (T, \Gamma)$ as produced by Algorithm 6, graph $G = (V, E, w)$, $U \subseteq V$, $\Delta \geq 0$
Output: Graph $G^* = (V^*, E^*, w^*)$, $U^* \subseteq V^*$ // $G^* = Limited_{\Delta/2}(G(\Delta))$, $U^* = Limited_{\Delta/2}(U(\Delta))$
 1: Sort $E$ by weight in time $O(|E| \lg |V|)$ only once for all subsequent calls to Alg. 7
 2: $Edges := \left\{(u,v) \in E \mid \frac{\Delta}{2n|V|^2} \leq w(u,v) < \frac{\Delta}{2}\right\}$ // Use binary search to locate Edges in $O(|Edges| + \lg |V|)$ time. Observe that $(u,v) \in Edges$ in $O(\lg |V|)$ scales $\Delta \in \left\{\Delta_j\right\}_{j \in Processed}$.
 3: $Edges := \{(u,v) \in Edges \mid \sigma_\Delta(u) \neq \sigma_\Delta(v)\}$ // Remove edges with points in the same class in $O(|Edges| \lg n)$ time.
 4: $V^* := \{\sigma_\Delta(u) \mid (u,v) \in Edges\}$
 5: $E^* := \{(\sigma_\Delta(u), \sigma_\Delta(v)) \mid (u,v) \in Edges\}$
 6: for all $(u,v) \in Edges$ do
 7:     $w^*(\sigma_\Delta(u), \sigma_\Delta(v)) := \min\{w^*(\sigma_\Delta(u), \sigma_\Delta(v)), w(u,v)\}$
 8: end for
 9: $G^* := (V^*, E^*, w^*)$
10: $U^* := \{\sigma_\Delta(u) \mid u \in U, (u,v) \in Edges\}$
```

Algorithm 8 Padded

```
Input: An HST $H = (T, \Gamma)$ as produced by Algorithm 6, $U \subseteq T$ leaves, $\left\{U_j, R_j\right\}_{j \in Processed}$ where $\forall j$, $U_j \subseteq T$ was produced by Algorithm 7 and $R_j \subseteq U_j$ was produced by Algorithm 4
Output: $S \subseteq U$ // $S = \left\{v \mid \forall j \in Processed, \sigma_{\Delta_j}(v) \in U_j \Rightarrow \sigma_{\Delta_j}(v) \in R_j\right\}$
 1: $T_{root} := T$ root
 2: $Y := \cup_{j \in Processed} U_j \setminus R_j$ // $Y$ is the set of unpadded vertices in $T$, $T_{root} \notin Y$
 3: $T^* := T \setminus Y$
 4: $S \subseteq U$ is the set of leaves in $T^*$ which are connected to $T_{root}$ // A vertex is padded if all its ancestors are padded
```

Algorithm 9 Refinement

```
Input: An HST $H = (T, \Gamma)$ as produced by Algorithm 6, $\left\{U_j, \mathscr{L}_j\right\}_{j \in Processed}$ where $\forall j$, $U_j \subseteq T$ was produced by Algorithm 7 and $\mathscr{L}_j$ is a partition of $U_j$ produced by Algorithm 3
Output: $\left\{\mathscr{W}_j\right\}_{j \in Processed}$ where $\forall j$, $\mathscr{W}_j$ is a partition of $U_j$ // $\mathscr{W}_j$ is the result of iteratively refining $\mathscr{L}_j$ with $\{\mathscr{L}_k\}_{k<j,\ k \in Processed}$
 1: $T_{root} := T$ root
 2: $\mathscr{P} := \mathscr{W}_0 := \mathscr{L}_0 := \{\{T_{root}\}\}$
 3: $Pre_j := 0$
 4: for all $j \in Processed$ in increasing order do
 5:     $\mathscr{W}_j := \left\{C \subseteq C' \in \mathscr{L}_j \,\middle|\, \forall u,v \in C, \sigma_{\Delta_{Pre_j}}(u) \neq \sigma_{\Delta_{Pre_j}}(v) \Rightarrow \mathscr{P}\left(\sigma_{\Delta_{Pre_j}}(u)\right) = \mathscr{P}\left(\sigma_{\Delta_{Pre_j}}(v)\right)\right\}$
        // Refine $\mathscr{L}_j$ with $\mathscr{P}$
 6:     $\mathscr{P} := \left\{C \setminus \sigma_{\Delta_{Pre_j}}\left(U_j\right) \,\middle|\, C \in \mathscr{P}\right\} \setminus \{\emptyset\} \cup \mathscr{W}_j$ // Refine $\mathscr{P}$ with $\mathscr{W}_j$
 7:     $Pre_j := j$
 8: end for
```

Algorithm 10 UM

```
Input: An HST $H = (T, \Gamma)$ as produced by Algorithm 6, $U \subseteq T$ leaves, $\left\{U_j, \mathscr{W}_j, \Delta_j\right\}_{j \in Processed}$
    where $\forall j,\ U_j \subseteq T$ was produced by Algorithm 7, $\mathscr{W}_j$ is a partition of $U_j$ produced by
    Algorithm 9 and $\Delta_j > 0$
Output: An HST $H^* = (T^*, \Gamma^*)$ // For $j \notin Processed$, let $U_j = \mathscr{W}_j = \emptyset$. For $j \geq 0$, let
    $\mathscr{E}_j = \left\{\sigma_{\Delta_j}^{-1}(A) \,\middle|\, A \in \left(\mathscr{W}_j \cup \left\{\{u\} \mid u \in U\left(\Delta_j\right) \setminus U_j\right\}\right)\right\}$ then $\Gamma^*(\mathbf{lca}((u,1,0),(v,1,0))) = \Delta_k$ where
    $k := \max\left\{j \mid \mathscr{E}_j(u) = \mathscr{E}_j(v)\right\}$
 1: for all $j \in Processed$, $k \in \{-1, +1\}$, $j + k \notin Processed$ do
 2:     $U_{j+k} := \mathscr{W}_{j+k} := \emptyset$
 3:     $\Delta_{j+k} := 8^{-k}\Delta_j$
 4: end for
 5: for all $j \in Processed$ do
 6:     $\mathscr{W}^*_{j-1} := \left\{\{u\} \mid u \in \sigma_{\Delta_{j-1}}\left(U_j\right) \setminus U_{j-1}\right\}$
 7:     $\mathscr{W}^{**}_{j+1} := \left\{\{u\} \mid u \in U_j \setminus \sigma_{\Delta_j}\left(U_{j+1}\right)\right\}$
 8: end for
 9: $Edges := \emptyset$
10: for all $j, j-1 \in Processed$, $C \in \mathscr{W}_j \cup \mathscr{W}^{**}_j$ do // C identify with a node in the resulted tree.
    Each node is represented uniquely as $(l, s, j)$ where l is the least ordered leaf under C and s is the
    number of leafs under C.
11:     $Parent(C) := C^* \in \left(\mathscr{W}_{j-1} \cup \mathscr{W}^*_{j-1}\right)$, $\sigma_{\Delta_{j-1}}(C) \subseteq C^*$
12:     $node_1 := \{\min_{v \in C} l(v), \sum_{v \in C} s(v), j\}$
13:     $node_2 := \left\{\min_{v \in Parent(C)} l(v), \sum_{v \in Parent(C)} s(v), j-1\right\}$
14:     $Edges := Edges \cup \{(node_1, node_2)\}$
15: end for
16: $E := \{(l_1, s_1, j_1), (l_2, s_2, j_2) \in Edges \mid\ s_1 \neq s_2\}$
17: $V := \{node_1 \mid (node_1, node_2) \in Edges\}$
18: for all $(l, s, j_1), (l, s, j_2) \in V$, $j_1 < j_2$ do // Sort the nodes by l, s, j and eliminate duplicates
19:     $j_1 := j_2$
20: end for
21: for all $(l, s, j) \in V$ do // Set labels $\Gamma^*$ for $T^*$ vertices
22:     if $s > 1$ then
23:         $\Gamma^*(l, s, j) := \Delta_j$
24:     else
25:         $\Gamma^*(l, s, j) := 0$
26:     end if
27: end for
28: $T^* = (V, E)$
```

Theorem 4.15. *Given $\alpha \geq 1$ and an n-point metric space (X, d_X) represented by distance matrix. It is possible to construct Ramsey partitions based $O(\alpha)$-ADO in $O\left(n^2 \lg^3 n\right)$ time and $O\left(n^{1+1/\alpha}\right)$ storage space.*

Proof. Represent the general metric space as a complete weighted graph, and use Theorem 4.13 with $k = 2$ and $\beta = \alpha$. $\square$

Observation 4.16. Similar trade-offs can be applied to every ADO construction where the number of edges in the input graph dominate the preprocessing time.

5 Source restricted ADOs

Sometimes we are only interested in approximate distances from a subset S of a given metric space. This is called *S-source restricted ADOs*, and both the preprocessing time and the storage space can be reduced in this case. This topic was first studied by Roditty, Thorup and Zwick [47], where it was shown that a variant of the Thorup-Zwick ADOs [52] achieves, for a set of sources S of size s, expected preprocessing time of $O\left(ms^{1/k}\lg n\right)$ and storage space $O\left(kns^{1/k}\right)$. In this section we give asymptotically tight bounds on S-source restricted ADOs constructions.

We start with a simple extension of any α-ADO construction that gives S-source restricted 3α-ADO.

Theorem 5.1. *Given $\alpha \geq 1$ and an undirected positively weighted graph $G = (V, E, w)$ with n vertices and m edges. Let $S \subseteq V$ be a set of sources with s vertices. Given O_S, α-ADO on the metric space (S, d_G). In $O(m)$ time and additional $O(n - s)$ storage space we can modify O_S into S-source restricted 3α-ADO on the metric space (V, d_G).*

Proof. For each $v \in V \setminus S$ store $d_G(v, S)$ and $p(v) \in S$ where $d_G(p(v), v) = d_G(v, S)$. Given $v \in V$ and $s \in S$, using the triangle inequality,

$$\begin{aligned}
d_G(v, s) \leq d_G(v, p(v)) + O_S(p(v), s) &\leq d_G(v, p(v)) + \alpha d_G(p(v), s) \\
&\leq \alpha\left(d_G(v, p(v)) + d_G(p(v), s)\right) \\
&\leq \alpha\left(d_G(v, p(v)) + d_G(p(v), v) + d_G(v, s)\right) \\
&\leq 3\alpha d_G(v, s)
\end{aligned}$$

That is, the distance $d_G(v, s)$ is 3α-approximated by $d_G(v, p(v)) + O_S(p(v), s)$. □

Observation 5.2. Ramsey partitions based ADO is $O(\alpha)$-ADO construction with $O\left(n^{1+1/\alpha}\right)$ storage space. Using Theorem 5.1 we derive upper bound of $O\left(s^{1+1/\alpha} + n\right)$ on the storage space required for S-source restricted $O(3\alpha)$-ADO. From the discussion in Section 1.2 we know that every α-ADO construction for metric space with s points requires at least $\Omega\left(s^{1+1/\alpha}\right)$ storage space. That is, a trivial lower bound for S-source restricted ADOs storage space is $\Omega\left(s^{1+1/\alpha} + n\right)$. We see that the upper and lower bounds asymptotically match.

Next, we focus on Ramsey partitions based ADOs. We show how slight changes to the preprocessing algorithm give us S-source restricted ADOs while the approximation factor remain the same. Comparing to using Theorem 5.1 with Ramsey partitions based ADOs, we save factor of 3 in the approximation.

We first show S-source restricted Ramsey partitions based ADOs with parameters asymptotically to Thorup-Zwick S-source restricted ADOs [47].

Theorem 5.3. *Given $\alpha \geq 1$ and an undirected positively weighted graph $G = (V, E, w)$ with n vertices and m edges. Let $S \subseteq V$ be a set of sources with s vertices. It is possible to construct S-source restricted Ramsey partitions based $O(\alpha)$-ADO in $O\left(\alpha ms^{1/\alpha}\lg^3 n\right)$ time and $O\left(\alpha ns^{1/\alpha}\right)$ storage space.*

Proof. Let (X, d_X) be the metric induced by G. We use Algorithm 2 (**RamseyPartitions-ADO**) implemented as proved in Section 4, with the following changes:

- Line 1: $X_0 := S$
- Line 13: Construct the ultrametric ρ_i on the set X.

For every $x \in S$ there is a unique index j where $x \in Y_j$. For every $y \in X$, the ultrametric ρ_j $O(\alpha)$-approximates the distance $d_X(x, y)$. From Lemma 3.4 we conclude that the expected number of iterations until the algorithm terminates is $\alpha s^{1/\alpha}$. □

Next, we show improved storage space. However the preprocessing time increases.

Theorem 5.4. *Given $\alpha \geq 1$ and an undirected positively weighted graph $G = (V, E, w)$ with n vertices and m edges. Let $S \subseteq V$ be a set of sources with s vertices. It is possible to construct S-source restricted Ramsey partitions based $O(\alpha)$-ADO in $O\left(\alpha m n^{1/\alpha} \lg^3 n\right)$ time and $O\left(\alpha n^{1/\alpha} s + n\right)$ storage space.*

Proof. Let (X, d_X) be the metric induced by G. We use Algorithm 2 (**RamseyPartitions-ADO**) implemented as proved in Section 4, with the following changes:

- Line 13: Construct the ultrametric ρ_i on the set $Y_i \cup S$.

For every $x \in X$ there is a unique index j where $x \in Y_j$. For every $y \in S$, the ultrametric ρ_j $O(\alpha)$-approximates the distance $d_X(x, y)$. From Lemma 3.4 we conclude that the expected number of iterations until the algorithm terminates is $\alpha n^{1/\alpha}$. Also, observe that each $x \in X \setminus S$ is included in a unique ultrametric, while each $x \in S$ is included in all ultrametrics. □

We finish this section with a simple lower bound proposition strengthening the trivial lower bound presented in Observation 5.2. Using this lower bound we conclude that the storage space obtained in 5.4 is asymptotically tight.

The proof is similar to the proof appearing in [52, Proposition 5.1]. The counting argument being used is common and appear in various papers, *e.g.*, [40].

Let $m_g(n)$ be the maximal number of edges in an n-vertices graph with girth g. In Section 1.2 we mentioned the conjectured bound $m_{2k+2}(n) = \Omega\left(n^{1+1/k}\right)$.

Theorem 5.5. *Given an integer $\alpha \geq 1$, let $t < 2\alpha + 1$. For the family of graphs with at most n vertices and at most s sources, any stretch t compact distance data structure must use $\Omega\left(\frac{s}{n} m_{2\alpha+2}(n) + n\right)$ bits of storage space for at least one graph in the family.*

Proof. Let $G = (V, E)$ be an unweighted graph with n vertices, $m_{2\alpha+2}(n)$ edges, and girth $2\alpha + 2$. Let $S \subseteq V$ be the set of s vertices with the highest degree, and let $u_0 \in S$ be an arbitrary vertex in S. Define

$$E' = \{(u, v) \in E \mid u \in S\} \cup \{(u_0, v) \mid v \in V,\ \forall (u, v) \in E,\ u \notin S\}.$$

All the edges of (V, E') have at least one endpoint in S, $|E'| = \Omega\left(\frac{s}{n} m_{2\alpha+2}(n) + n\right)$ edges, and the girth of (V, E') is at least $2\alpha + 2$.

For subset of the edges $H \subseteq E'$, define d_H to be the shortest path metric on (V, H) (with the convention that pairs in different connected components have infinite distance), and $\rho_H = \min\{d_H(u,v), 2\alpha + 1\}$. Denote by $O_H : V \times V \to [0, \infty)$ the distances on V as defined by the compact data structure holding stretch t approximation of ρ_H. For any two subsets $H_1, H_2 \subseteq E'$, $H_1 \neq H_2$, there exists (u,v) in the symmetric difference of H_1 and H_2. Assume without loss of generality that $(u,v) \in H_1 \setminus H_2$. Then $O_{H_1}(u,v) \leq t \cdot \rho_{H_1}(u,v) = t < 2\alpha + 1$, but $O_{H_2}(u,v) \geq \rho_{H_2}(u,v) \geq 2\alpha + 1$. Consequently, all the $2^{|E'|}$ subsets H of E' with the metric ρ_H have different stretch t compact distance data structure, and therefore at least one of those requires $\Omega\left(\frac{s}{n} m_{2\alpha+2}(n) + n\right)$ bits to represent. $\square$

Table 3 summaries the best known upper bounds on storage space for Thorup-Zwick and Ramsey partitions based S-source restricted ADOs.

	Integer s	Storage Space	Reference
Thorup-Zwick. Approximation: $3 \cdot (2k-1)$	$s = O\left(n^{\frac{k}{k+1}}\right)$	$O(kn)$	Theorem 5.1
	$s = O\left(n^{\frac{1}{1+1/k-1/(3k-1)}}\right)$	$O\left(ks^{1+\frac{1}{k}} + n\right)$	Theorem 5.1
	$s \leq n$	$O\left(kns^{\frac{1}{3k-1}}\right)$	[47]
Ramsey partitions based. Approximation: $3 \cdot 256k$	$s = O\left(n^{\frac{k}{k+1}}\right)$	$O(kn)$	Theorem 5.4, Theorem 5.1
	$s = O\left(n^{1-\frac{1}{3k}}\right)$	$O(kn)$	Theorem 5.4
	$s \leq n$	$O\left(kn^{\frac{1}{3k}} s + n\right)$	Theorem 5.4

Table 3: *Best known upper bounds on storage space for S-source restricted Thorup-Zwick ADOs and Ramsey partitions based ADOs.*

6 Parallel ADOs preprocessing

When working in parallel environment we would like to achieve maximum utilization of the concurrent processing force without increasing much the total work done. In this section we present a priority CRCW PRAM implementation of Thorup-Zwick ADOs (see Section 1.2) and Ramsey partitions based ADOs in parallel polylogaritmic time with only small increase in the total work.

We assume an unlimited number of available processors and avoid describing the method of allocating tasks to specific processors and of communication between processors. For a discussion of those issues see [30].

Theorem 6.1. *Given integer $\alpha \geq 1$ and an undirected positively weighted graph with n vertices and m edges. Fix $\delta < 1, \delta = \Omega(1/\lg\lg n)$ and $1 > \epsilon > 0$. Let $d = \left(\frac{\lg n}{\epsilon}\right)^{1/\delta^2}$. It is possible to construct Thorup-Zwick $(1+\epsilon)(2\alpha-1)$-ADO using the priority CRCW PRAM model in $O\left(d\lg^2 n\right)$ expected time using $O\left(mn^\delta + \alpha\left(m + n^{1+\delta}\right)n^{1/\alpha} \cdot d\lg n\right)$ work and $O\left(\alpha n^{1+1/\alpha}\right)$ storage space.*

Theorem 6.2. *Given $\alpha \geq 1$ and an undirected positively weighted graph with n vertices and m edges. Fix $\delta < 1, \delta = \Omega(1/\lg\lg n)$ and $1 > \epsilon > 0$. Let $d = \left(\frac{\lg n}{\epsilon}\right)^{1/\delta^2}$. It is possible to construct Ramsey partitions based $O(\alpha)$-ADO using the priority CRCW PRAM model in $O\left(d\lg^2 n\right)$ expected time using $O\left(mn^\delta + \alpha\left(m + n^{1+\delta}\right)n^{1/\alpha} \cdot d\lg^4 n\right)$ work and $O\left(n^{1+1/\alpha}\lg n\right)$ storage space.*

It is also possible to parallelize the results of Section 4.4 using parallel construction of spanners [6].

We start with describing a parallel algorithm for the undirected single source shortest paths [USSSP] problem, **Parallel-USSSP**(G, S), which is heavily used in the ADOs. The input for the algorithm is a weighted graph $G = (V, E, w)$ and a set of sources $S \subseteq V$. The output is $\delta : V \to [0, \infty)$ where $\delta(v) = d_G(S, v)$.

Currently, no algorithm for *exact* USSSP achieving *polylog* (n) parallel running time and using near linear work is known. Algorithms achieving tight bounds in one of those criteria have $O(n\lg n)$ parallel time using $O(m + n\lg n)$ work [28], or $O\left(\lg^2 n\right)$ parallel time using $O\left(n^3 (\lg\lg n/\lg n)^{1/3}\right)$ work [20]. However, if we are content with an approximate solution, there exists better trade-offs, due to Cohen [16], which we describe next.

A (d, ϵ)-hop set of a weighted graph $G = (V, E, w)$ is a collection of weighted edges E^* such that for every pair of vertices, the minimum-weight path in $(V, E \cup E^*)$ between them with no more than d edges has weight within $(1+\epsilon)$ of the corresponding shortest path in (V, E). The objective, typically, is to obtain sparse hop sets with small factor d and good approximation quality.

The following result [2] from [16, Theorem 1.3] gives an upper bound on the parallel construction of hop sets.

Theorem 6.3. *Given a graph with n vertices and m edges. Fix $1 > \epsilon > 0$ and $\delta < 1, \delta = \Omega(1/\lg\lg n)$. Let $d = \left(\frac{\lg n}{\epsilon}\right)^{1/\delta^2}$. An $(O(d), \epsilon)$-hop set E^* of size $O\left(n^{1+\delta}\right)$ can be computed in $O(d)$ time by randomized EREW PRAM algorithm using $O\left(mn^\delta\right)$ work. The probability for success is $1 - O(1/n)$.*

[2]The original statement does not contain explicit parameters, but we have computed them from the proof. See Appendix A.

Hop-sets can be employed as follows to solve approximately the parallel USSSP problem. A *d-edge* shortest path is a minimum weight path among the paths that contain at most d edges. When d is small, d-edge shortest paths can be computed efficiently in parallel by d-iteration Bellman-Ford algorithm in $O(d \lg n)$ time using $O(dm)$ work (see, *e.g.*, [15, Algorithm 3.1]). Suppose we are given (d, ϵ)-hop set, E^*, for a weighted graph $G = (V, E, w)$. In order to approximate distances in G, it suffices to compute respective d-edge shortest paths in $(V, E \cup E^*)$. Hence, $(1 + \epsilon)$-approximate USSSP problem can be solved efficiently in $O(d \lg n)$ time using $O(dm)$ work.

The *M-induced subgraph*, $G(M)$, of a graph $G = (V, E)$ where $M \subseteq V$ is defined as $G(M) = (M, E(M))$ where $E(M) = E \cap M \times M$. Obviously, $G(M)$ can be constructed from G in $O(1)$ parallel time using $O(|\partial(M)|)$ work where $\partial(M)$ is the set of edges having at least one point incident at M. Compute an $(O(d), \epsilon)$-hop set E^* of G. In the rest of the section we implement calls to $1 + \epsilon$ approximate **Parallel-USSSP** $(G(M), S)$ where $S \subseteq M$ using $O(d)$-iteration parallelized Bellman-Ford algorithm on $G'(M)$ where $G' = (V, E \cup E^*)$.

6.1 Thorup-Zwick ADOs

The sequential algorithm of Thorup and Zwick [52, Section 4.1] for construction α-ADO is presented here as algorithm 11. Its running time is $\alpha m n^{1/\alpha}$ on graph with n vertices and m edges. Thorup and Zwick prove that the resulting ADO achieves $2\alpha - 1$ approximation, $O\left(\alpha n^{1+1/\alpha}\right)$ storage space and $O(\alpha)$ query time.

Algorithm 11 Thorup-Zwick ADO

```
Input: Graph $G = (V, E, w)$, integer $\alpha > 1$
Output:   i. $\forall v$ and $0 \le i \le \alpha - 1$: $p_i(v)$ and $d_G(A_i, v)$.
         ii. A 2-level hash table holding $\forall v \in V$ and $w \in B(V)$: $d_G(v, w)$.
 1: $A_0 := V$
 2: $A_\alpha := \emptyset$
 3: for $i = 1$ to $\alpha - 1$ do
 4:     Let $A_i$ contain each element of $A_{i-1}$, independently, with probability $n^{-1/\alpha}$
 5: end for
 6: for all $v \in V$ do
 7:     $\delta(A_\alpha, v) := \infty$
 8: end for
 9: for $i = \alpha - 1$ downto 0 do
10:     for $v \in V$ do
11:         Compute $\delta(A_i, v) = d_G(A_i, v)$, find $p_i(v) \in A_i$ such that $d_G(p_i(v), v) = d_G(A_i, v)$
12:         if $\delta(A_i, v) = \delta(A_{i+1}, v)$ then
13:             $p_i(v) := p_{i+1}(v)$
14:         end if
15:     end for
16:     for $w \in A_i \setminus A_{i+1}$ do
17:         $C(w) := \{v \in V \mid d_G(w, v) < \delta(A_{i+1}, v)\}$
18:     end for
19: end for
20: for $v \in V$ do
21:     $B(v) := \{w \in V \mid v \in C(w)\}$
22: end for
23: Build a 2-level hash table holding $\forall v \in V$ and $w \in B(V)$: $d_G(v, w)$
```

We next describe the non-trivial steps needed for implementing Algorithm 11 (**Thorup-Zwick ADO**) in $O\left(d \lg^2 n\right)$ expected parallel time using $O\left(\left(mn^{\delta}\right) + \alpha\left(m + n^{1+\delta}\right) n^{1/\alpha} \cdot d \lg n\right)$ work.

Computing $p_i(v)$ for all $v \in V$ (Line 11) in parallel is done by an algorithm that recursively splits V in $O(\lg n)$ phases as follows. In phase k, the set A_i is divided into 2^k sets, $\left\{B_{k,j} \mid j \in \left\{1, \ldots, 2^k\right\}\right\}$ of roughly the same size. We compute $P_{k,j} = \left\{v \in V \mid p_i(v) \in B_{k,j}\right\}$. In phase 0, trivially, $B_{0,1} = A_i$ and $P_{0,1} = V$. In phase k, for $1 \le j \le 2^{k-1}$, we divide $B_{k-1,j}$ into two roughly equal size sets, $B_{k,2j-1}$ and $B_{k,2j}$. We call to **Parallel-USSSP**$\left(G\left(P_{k-1,j}\right), B_{k,2j-1}\right)$ and **Parallel-USSSP**$\left(G\left(P_{k-1,j}\right), B_{k,2j}\right)$. We divide $P_{k-1,j}$ into $P_{k,2j-1}$ and $P_{k,2j}$ according to the results. The correctness is easily proved by induction on the phase, k.

Computing $C(w)$ for all $w \in A_i \setminus A_{i+1}$ (Line 17) in parallel is done similarly in $O(\lg |A_i \setminus A_{i+1}|) = O(\lg n)$ phases. In the k-th phase k, the set $A_i \setminus A_{i+1}$ is divided into 2^k sets of roughly the same size, $B_{k,j}$. For each $B_{k,j}$ we compute $P_{k,j}$ as the set of vertices $v \in V$ in maximal distance $\delta(A_{i+1}, v)$ from $B_{k,j}$ using a call to **Parallel-USSSP**$\left(G\left(P_{k-1,\lceil j/2 \rceil}\right), B_{k,j}\right)$.

6.2 Ramsey partitions based ADOs

We next describe the non-trivial steps needed for implementing Algorithm 2 (**RamseyPartition-ADO**) in $O\left(d\lg^2 n\right)$ expected parallel time using $O\left(\left(mn^{\delta}\right) + \alpha\left(m + n^{1+\delta}\right)n^{1/\alpha} \cdot d\lg^4 n\right)$ work.

Parallel construction of the ultrametrics

Input: Graph $G = (V, E, w)$ with n vertices and m edges, approximation factor $\alpha \geq 1$
Output: Set of HSTs $\bigcup (T_i, \Delta_i)$

Algorithm 2 (**RamseyPartition-ADO**) works iteratively. In each iteration (Lines 3-16) the set of padded points is deleted, until no point remains. Alternative way, as outline in [42, Observation 4.3] is to repeatedly and independently sample Ramsey partitions from the given metric space and construct the resulted padded set and ultrametric. The expected number of ultrametrics before all points are part of the padded set in at least one ultrametric is $O\left(\alpha n^{1/\alpha}\lg n\right)$.

Our algorithm process $O\left(\alpha n^{1/\alpha}\lg n\right)$ parallel tasks (where the constant is determined by the probability of success required). In each, we sample Ramsey partitions on the given metric space and construct the padded set and the ultrametric using a parallel implementation of Algorithm 5 (**Graph-RamseyPartitions**).

The oracle storage space and total work are multiplied by a factor of $O(\lg n)$.

Algorithm 5 (Graph-RamseyPartitions)

Input: Graph $G = (V, E, w)$ with n vertices and m edges, $U \subseteq V$, approximation factor $\alpha \geq 1$
Output: An HST $H = (T, \Gamma)$, $S \subseteq U$

Approximating the diameter $\phi = \operatorname{diam}(U)$ can be done in $O(d\lg n)$ parallel time and $O(dm)$ work using one call to **Parallel-USSSP**.

Constructing the set $Processed := \left\{\text{integer } j > 0 \,\middle|\, (u, v) \in E, 8^{-j}\phi \in \left[2 \cdot w(u, v), 2n^2 \cdot w(u, v)\right]\right\}$ can be done in $O(\lg n)$ time using m parallel tasks. Each task is assigned an edge $(u, v) \in E$ and enumerate the scales in its given range.

Algorithm 6 (Init)

Input: Graph $G = (V, E, w)$ with n vertices and m edges
Output: An HST $H = (T, \Gamma)$ which is $(n - 1)$ approximation of G supporting the following queries:

i. Given $u \in T$, compute $l(u) = \min\{i \mid v_i \text{ is a descendant of } u\}$ in $O(1)$ time.
ii. Given $u \in T$, compute $s(u) = |\{w \mid w \text{ is a descendant of } u\}|$ in $O(1)$ time.
iii. Given $u \in T$, $\Delta \geq 0$, compute $\sigma_{\Delta}(u) =$ the highest ancestor of u for which $\Gamma(u) \leq \frac{\Delta}{2n}$.

The algorithm is a parallel version of the algorithm presented by Har-Peled and Mendel in [29, Section 3.2]. In the implementation we use the following known algorithms.

- Finding a minimum spanning tree of a weighted graph with n vertices and m edges in $O(\lg n)$ expected parallel time and $O(m)$ work [18].
- A *separator* [37] is a partition of the vertices of an n-vertex tree into two sets A, B having one common vertex and $|A|, |B| \leq \frac{2n}{3}$, such that no edge of the tree goes between a vertex in A and a vertex in B. Using Brent's method [12] finding a tree separator is done in $O(\lg n)$ parallel time and $O(n \lg n)$ work.
- Berkman and Vishkin [10, 9] show how to preprocess an n-vertex tree in $O(\lg n)$ parallel time using $O(n \lg n)$ work and create a data structure such that level-ancestor queries are answered in constant time.
- Sorting n elements in $O(\lg n)$ time using $O(n \lg n)$ work [17].

Compute the minimum spanning tree B of G. Divide B using the tree separator. Recursively and in parallel build an HST H from each subtree and merge them. A one vertex tree is an HST and a one edge tree, (u, v), defines an HST with a root labeled $(n-1) \cdot w(u, v)$ and two leafs, u and v. The merging of two HSTs H_1, and H_2 with a common leaf u, is done as follows. For $i \in \{1, 2\}$ we use level-ancestor queries to locate all the ancestors of u in H_i and disconnecting H_i by removing all edges on the path from u to the root. Sort the subtrees of H_1, H_2 obtained in this process in non-increasing order by the label of their roots. Connect each subtree as a child of the root of the subtree whose label is immediately larger than his.

Observe that in each stage of the construction, each HST corresponds to a connected component of B. It can be easily proved, using induction, that the resulting tree is a valid HST and for $w, v \in V$, the distance in H between w and v is $(n-1) \cdot w(e)$ where e be the heaviest edge on the path between w and v in B. As T is a minimum spanning tree, and the maximum number of edges on a path along it is $n-1$, the approximation is at most $n-1$. The recursion depth is $O(\lg n)$, resulting in the time and work bounds given.

Let $H = (T, \Gamma)$. Assume a total order on V. Observe that V identifies with the leaves of T. For all $u \in T$ we compute the values $l(u)$, the least ordered leaf under u at H and $s(u)$, the number of leaves under u at H in $O(\lg n)$ time and $O(m \lg n)$ work using the tree contraction method presented in [43].

We process T in $O(\lg n)$ parallel time using $O(n \lg n)$ work such that it supports level-ancestor queries in constant time [10, 9]. For $v \in T$ and $\Delta \geq 0$ where $\Gamma(v) \leq \frac{\Delta}{2n}$, define $\sigma_\Delta(v)$, to be the highest ancestor u of v, such that $\Gamma(u) \leq \frac{\Delta}{2n}$. We use $O(\lg n)$ level ancestors queries to locate $\sigma_\Delta(v)$.

Algorithm 7 (Quotient)

Input: An HST $H = (T, \Gamma)$ as produced by Algorithm 6, graph $G = (V, E, w)$ with n vertices and m edges, $U \subseteq V$, $\Delta \geq 0$
Output: Graph $G^* = (V^*, E^*, w^*)$, $U^* \subseteq V^*$

We sort the edges by their weight in $O(\lg n)$ time using $O(n \lg n)$ work [17] once, for all invocations of the algorithm. Then, to construct the set $Edges := \left\{(u, v) \mid \frac{\Delta}{2n^2} \le w(u, v) < \frac{\Delta}{2}\right\}$, we use binary search in $O(\lg n)$ time and $O(|Edges| + \lg n)$ work.

The rest is easily implemented in $O(\lg n)$ time and total $O(|Edges| \lg n)$ work.

Algorithm 3 (GraphPartition-CKR)

Input: Graph $G = (V, E, w)$ with n vertices and m edges, $U \subseteq V$, scale $\Delta > 0$
Output: Partition $\mathscr{P}$ of V

We implement Algorithm 3 (**GraphPartition-CKR**) in $O\left(d \lg^2 n\right)$ parallel time and $O\left(dm \lg^2 n\right)$ expected work.

Given a permutation π of U, for $1 \le i \le |U|$ define

$$M_i = \left\{ v \in V \mid d_G(v, \pi(i)) < \min_{1 \le j < i} d_G(v, \pi(j)) \right\},$$

that is, M_i is the set of vertices in V which are closer to $\pi(i)$ than to all the vertices before it in the permutation π. Using the argument from section 4.1, we know that $\mathbb{E}\left[\sum_i |M_i|\right] = O(n \lg n)$. Set D as a constant function where for all $u \in V$, $D(u) = \mathrm{diam}(U) + 1$. A call to Algorithm 12, **CloserSet**(G, π, D), computes M_i for all i in $O\left(d \lg^2 n\right)$ time and expected total work $O\left(dm \lg^2 n\right)$.

To compute the extended CKR Partition $\mathscr{L}$ (as in Section 4.1) we first generate in parallel a random permutation π of U in $O(\lg n)$ time and $O(n)$ work. There exist many algorithms achieving that, *e.g.*, [27].

Next, we run Algorithm 12, **CloserSet**(G, π, D).

Finally, set $\forall v \in V$, $\mathscr{L}(v) = |U|$. We run in parallel $|U|$ processes where the priority of process i is $|U| - i$, In process i we set $\delta =$**Parallel-USSSP**$(G(M_i), \{\pi(i)\})$. For each vertex $v \in M_i$ where $\delta(\pi(i), v) \le R$ and $\mathscr{L}(v) > i$, set $\mathscr{L}(v) = i$.

Algorithm 12 CloserSet

```
Input: Graph $G = (V, E)$, $\pi(i), \ldots, \pi(j)$, $d : V \to [0, \infty)$
Output: $\{M_k \subseteq V\}_{i \le k \le j}$
 1: $mid := \lfloor (i + j)/2 \rfloor$
 2: $d_1 :=$ Parallel-USSSP($G, \{\pi(k) \mid i \le k \le mid\}$)
 3: $S_1 := \{v \in V \mid d_1(v) < d(v)\}$
 4: if $i = j$ then
 5:     $M_i := S_1$
 6: else
 7:     $d_2 :=$ Parallel-USSSP($G, \{\pi(k) \mid mid + 1 \le k \le j\}$)
 8:     $S_2 := \{v \in V \mid d_2(v) < \min\{d(v), d_1(v)\}\}$
 9:     Perform in Parallel
10:         CloserSet($G(S_1), \pi(i), \ldots, \pi(mid), d$)
11:         CloserSet($G(S_2), \pi(mid + 1), \ldots, \pi(j), \min\{d(v), d_1(v)\}$)
12: end if
```

Algorithm 4 (PaddedVertices)

Input: Graph $G = (V, E, w)$ with n vertices and m edges, $U \subseteq V$, partition $\mathscr{P}$ of V, padding radius $t \ge 0$
Output: $S \subseteq U$

We trivially implement Algorithm 4 (**PaddedVertices**) in $O(d \lg n)$ parallel time and $O(dm)$ work using one call to **Parallel-USSSP**.

Algorithm 8 (Padded)

Input: An $O(n)$-vertex HST $H = (T, \Gamma)$ as produced by Algorithm 6, $U \subseteq T$ leaves, $\left\{U_j, R_j\right\}_{j \in Processed}$ where $\forall j$, $U_j \subseteq T$ was produced by Algorithm 7 and $R_j \subseteq U_j$ was produced by Algorithm 4, $\sum_{j \in Processed} \left|U_j\right| = O(m \lg n)$
Output: $S \subseteq U$

We parallelize Algorithm 8 (**Padded**) using a parallel algorithm for identifying connected components in $O\left(\lg^{1+1/2} n\right)$ time using $(m \lg n)$ work [32].

Algorithm 9 (Refinement)

Input: An $O(n)$-vertex HST $H = (T, \Gamma)$ as produced by Algorithm 6, $\left\{U_j, \mathscr{L}_j\right\}_{j \in Processed}$ where $\forall j$, $U_j \subseteq T$ was produced by Algorithm 7 and $\mathscr{L}_j$ is a partition of U_j produced by Algorithm 3, $\sum_{j \in Processed} \left|U_j\right| = O(m \lg n)$
Output: $\left\{\mathscr{W}_j\right\}_{j \in Processed}$ where $\forall j$, $\mathscr{W}_j$ is a partition of U_j

Let $\mathscr{W}_j$ be the refinement of $\{\mathscr{L}_k\}_{k<j,\, k \in Processed}$. Given $C \in \mathscr{L}_j$ and $i < j$. Let $u, v \in C$ be the pair of vertices with maximal **lca** (u, v) value. We know that $\frac{\Gamma(\mathbf{lca}(u,v))}{n} < \frac{\Delta_j}{2} < \frac{\Delta_i}{2}$. Also, if $\Gamma(\mathbf{lca}(u, v)) < \frac{\Delta_i}{2n}$, then $\forall w_1, w_2 \in C, \mathscr{L}_i(\sigma_{\Delta_i}(w_1)) = \mathscr{L}_i(\sigma_{\Delta_i}(w_2))$.

We conclude that in order to construct $\mathscr{W}_j$, it is enough to refine $\mathscr{L}_j$ with the set of partitions $\mathscr{L}_i$ where $\Delta_i \in \left(\Delta_j, n^2\Delta_j\right]$. That is, a total of $O(\lg n)$ partitions.

Also, the j-th iteration of Algorithm 9 (**Refinement**), lines 4- 8 can be easily performed in $O(\lg n)$ parallel time using $\sum_{C\in\mathscr{L}_j} |C| \lg n$ work.

Using those observations, for $i \in Processed_i$ we evoke in parallel Algorithm 9 (**Refinement**) with input $H, \left\{U_j, \mathscr{L}_j\right\}_{j\in Processed,\ \Delta_j\in(\Delta_i, n^2\Delta_i]}$. $\mathscr{W}_i$ is set by process i.

The algorithm runs in $O\left(\lg^2 n\right)$ parallel time using $O\left(m \lg^3 n\right)$ work.

Algorithm 10 (UM)

Input: An $O(n)$-vertex HST $H = (T, \Gamma)$ as produced by Algorithm 6, $U \subseteq T$ leaves, $\left\{U_j, \mathscr{W}_j, \Delta_j\right\}_{j\in Processed}$ where $\forall j$, $U_j \subseteq T$ was produced by Algorithm 7, $\mathscr{W}_j$ is a partition of U_j produced by Algorithm 9 and $\Delta_j > 0$, $\sum_{j\in Processed} \left|U_j\right| = O(m \lg n)$
Output: An HST $H^* = (T^*, \Gamma^*)$

We implement Algorithm 10 (**UM**) in $O(\lg n)$ parallel time using $O\left(m \lg^2 n\right)$ work by performing all its loops in parallel.

7 Open problems

In this thesis we gave various results for ADOs. Still, there are many open problems in the area. We focus here mainly on those concerning Ramsey partitions based ADOs.

- The approximation of the construction is $128k$. It is possible to improve the constant. What is the smallest constant that can be achieved?

- In an earlier version of this work we used the conditional expectation method to devise, for distances matrix representation, a deterministic preprocessing algorithm with $O\left(n^{2+1/k}\right)$ running time. A similar result was achieved by Tao and Naor [unpublished]. Bartal [unpublished] showed a deterministic preprocessing algorithm for weighted graph representation with $O\left(mn^{1+1/k}\right)$ running time. Is it possible to achieve better results?

- The expected running time of the preprocessing algorithm for weighted graphs proved in this thesis is $O\left(mn^{1/k}\lg^3 n\right)$. Can some of the $\lg n$ factors be removed? One obvious place for improvement is the running time of Algorithm 3 (**GraphPartition-CKR**). A little more careful analysis will bound its running time with $O((m + n\lg n)\lg n)$. Also, it is probably can be reduced to $O(m\lg n)$ by replacing Dijsktra's based USSSP procedure with Thorup's USSSP $O(m)$ time algorithm [50, 51] modified similarly to Dijkstra's algorithm.

- Construct an asymptotically tight storage space $(2k-1)$-ADO with constant query time.

- Prove a lower bound on the preprocessing time for weighted graphs of $O\left(n^{1+1/k}\right)$ storage space $(2k-1)$-ADOs with a constant query time. Currently, the best known lower bound is the trivial $\Omega(m)$ while the best known upper bound is $O\left(kmn^{1/k}\right)$.

- CKR partitions have found many algorithmic (as well as mathematical) applications. They were introduced as part of an approximation algorithm to the 0-extension problem [14, 22]. Fakcharoenphol, Rao and Talwar [23] used them to obtain (asymptotically) optimal probabilistic embeddings into trees, which we call FRT-embedding. Probabilistic embeddings are used in many of the best known approximation and online algorithms as a reduction step from general metrics into tree metrics, see for example [3]. Mendel and Naor [41] have shown that FRT-embeddings posses a stronger embedding property, which they coined "maximum gradient embedding". Recently, Räcke [46] has used FRT-embeddings to obtain hierarchical decompositions for congestion minimization in networks, and used them to give $O(\lg n)$ competitive online algorithm for the oblivious routing problem. Krauthgamer et. al [33] used CKR-partitions to reprove Bourgains embedding theorem. Can the improved running time of the sampling of CKR partitions improve the running time for those applications?

A Appendix: Parallel hop set construction parameters

Cohen [16] gives a parameterized parallel algorithm for constructing a (d, ϵ)-hop set E^* of a weighted graph G. The algorithm allows us to choose the desired trade-off between hop-diameter, approximation, running time and work. Here we show the computation steps leading from the result as presented in [16, Section 8], to Theorem 6.3.

Parameters for the algorithm are:

- $\delta < 1, \delta = \Omega(1/\lg\lg n)$
- $\mu \le 1, \mu = \Omega(1/\lg\lg n)$
- $\rho \gg 4$
- An integer $v > 1$
- $\gamma \ge 0$

The construction properties are:

- Hop set size: $|E^*| = O\left(\lceil \lg_v(n/d) \rceil \left(O\left(\lg^3 n\right)^{\lceil \lg_{1-\delta}\mu \rceil} n^{1+\mu} + n^{2\delta}\lg^5 n\right)\right)$.
- Approximation: $1+\epsilon = \left((1+\gamma)(1+4/\rho)^{\lceil \lg_{1-\delta}\mu \rceil}\right)^{\lceil \lg_v(n/d) \rceil}$
- Hop-diameter: $d = O(\rho \lg n)^{\lceil \lg_{1-\delta}\mu \rceil}$
- Running time for the construction: $O\left(\lceil \lg_v(n/d) \rceil \left(vd\gamma^{-1}\delta^{-1}\lg^3 n\right)\right)$
- Total work for the construction: $O\left(\lceil \lg_v(n/d) \rceil m' \left(n^\delta \lg^4 n + O\left(\lg^3 n\right)^{\lceil \lg_{1-\delta}\mu \rceil} n^\mu \lg n\right)\right)$
  where $m' \le m + O(|E'|)$.

We set the parameters as advised in [16, Section 8] in order to achieve *polylog* (n) running time: $\delta = \mu$, $\rho = O(\lg^r n)$ for integer $r > 1$, $v = 2$ and $\gamma = 0$.

For $\frac{1}{2} < \delta < 1$, $\lceil \lg_{1-\delta}\delta \rceil = 1$.

For $0 < \delta \le \frac{1}{2}$ we know that $e^{-2\delta} \le 1-\delta \le e^{-\delta}$. So, $-2\delta \le \lg_e(1-\delta) \le -\delta$. That gives us $\lg_{1-\delta}\delta = \frac{\lg(\delta)}{\lg(1-\delta)} = \Theta\left(\frac{\lg(1/\delta)}{\delta}\right)$.

When $\delta = \Omega(1/\lg\lg n)$, $\lg_{1-\delta}\delta = O(\lg n)$. Also, $1/\delta^2 = \Omega\left(\frac{\lg(1/\delta)}{\delta}\right)$, so $O(\lg n)^{O\left(\frac{\lg(1/\delta)}{\delta}\right)} = O\left(\lg^{1/\delta^2} n\right)$. Finally, $O\left(\lg^{1/\delta^2} n\right) = O\left(n^\delta\right)$.

We immediately get:

- Hop set size:

$$\left|E^*\right| = O\left(n^{1+\delta} O(\lg n)^{O\left(\frac{\lg(1/\delta)}{\delta}\right)} + n^{2\delta}\lg^{O(1)} n\right) = O\left(n^{1+\delta}\lg^{1/\delta^2} n\right) = O\left(n^{1+\delta}\lg^{1/\delta^2} n\right) = O\left(n^{1+2\delta}\right).$$

- Approximation: using the well known equality, $\lim_{n\to\infty}\left(1+\frac{1}{n}\right)^n = e$,

$$1+\epsilon = \left(1+O\left(\frac{1}{\lg^r n}\right)\right)^{O\left(\lg^2 n\right)} = e^{-\lg^{r-2} n} = \left(1+O\left(\frac{1}{\lg^{r-2} n}\right)\right).$$

- Hop-diameter:

$$d = O\left(\lg^{r+1} n\right)^{O\left(\frac{\lg(1/\delta)}{\delta}\right)} = O\left(\frac{\lg^3 n}{\epsilon}\right)^{O\left(\frac{\lg(1/\delta)}{\delta}\right)} = O\left(\left(\frac{\lg n}{\epsilon}\right)^{1/\delta^2}\right).$$

- Running time for construction:

$$O\left(\lg^{r+1} n\right)^{O\left(\frac{\lg(1/\delta)}{\delta}\right)} \lg^4 n\delta^{-1} = O\left(\left(\frac{\lg n}{\epsilon}\right)^{1/\delta^2}\right).$$

- Total work for construction:

$$O\left(\left(m+O\left(\left|E^*\right|\right)\right)\left(n^\delta \lg^{O(1)} n + O\left(\lg^{O(1)} n\right)^{O\left(\frac{\lg(1/\delta)}{\delta}\right)} n^\delta\right)\right) = O\left(\left(m+\left|E^*\right|\right)n^{2\delta}\right).$$

Set $\delta^* = 4\delta$, $\delta^* < 1$, $\delta^* = \Omega\left(\lg\lg n\right)$. Then:

- Hop set size: $|E^*| = O\left(n^{1+\delta^*}\right)$.
- Hop-diameter: $d = O\left(\left(\frac{\lg n}{\epsilon}\right)^{1/\delta^{*2}}\right)$.
- Running time for construction: $O\left(\left(\frac{\lg n}{\epsilon}\right)^{1/\delta^{*2}}\right)$.
- Total work for construction: $\left(mn^{\delta^*}\right)$.

References

[1] I. Althöfer, G. Das, D. P. Dobkin, D. Joseph, and J. Soares. On sparse spanners of weighted graphs. *Discrete & Computational Geometry*, 9:81–100, 1993.

[2] M. J. Atallah and S. Fox. *Algorithms and Theory of Computation Handbook*. CRC Press, Inc., Boca Raton, FL, USA, 1998.

[3] Y. Bartal. Probabilistic approximations of metric space and its algorithmic application. In *Proc. 37th Ann. IEEE Symp. Foundations of Computer Science (FOCS'96)*, pages 183–193. Elsevier Science Publishers B. V., Amsterdam, The Netherlands, The Netherlands, 1996.

[4] Y. Bartal, N. Linial, M. Mendel, and A. Naor. On metric Ramsey type phenomena. *Ann. of Math.*, 162(2):643–709, 2005.

[5] S. Baswana and T. Kavitha. Faster algorithms for approximate distance oracles and all-pairs small stretch paths. In *Proc. 47th Ann. IEEE Symp. Foundations of Computer Science (FOCS'06)*, pages 591–602. IEEE Computer Society, Washington, DC, USA, 2006.

[6] S. Baswana and S. Sen. A simple and linear time randomized algorithm for computing sparse spanners in weighted graphs. *Random Struct. Algorithms*, 30(4):532–563, 2007.

[7] M. Ben-Or. Lower bounds for algebraic computation trees. In *Proc. 15th Ann. ACM Symp. Theory of computing (STOC'83)*, pages 80–86. ACM, New York, NY, USA, 1983.

[8] M. A. Bender and M. Farach-Colton. The level ancestor problem simplified. *Theor. Comput. Sci.*, 321(1):5–12, 2004.

[9] O. Berkman and U. Vishkin. Recursive *-tree parallel data structure. *SIAM J. Comput.*, 22(2):221–242, 1993.

[10] O. Berkman and U. Vishkin. Finding level-ancestors in trees. *J. Comput. Syst. Sci.*, 48(2):214–230, 1994.

[11] D. Berman and M. S. Klamkin. A reverse card shuffle. *SIAM Review*, 18(3):491–492, 1976.

[12] R. P. Brent. The parallel evaluation of general arithmetic expressions. *J. ACM*, 21(2):201–206, 1974.

[13] S. Cabello. Many distances in planar graphs. In *Proc. 17th Ann. ACM-SIAM Symp. Discrete Algorithms (SODA'06)*, pages 1213–1220. ACM, New York, NY, USA, 2006.

[14] G. Calinescu, H. Karloff, and Y. Rabani. Approximation algorithms for the 0-extension problem. In *Proc. 12th Ann. ACM-SIAM Symp. Discrete Algorithms (SODA'01)*, pages 8–16. SIAM, Philadelphia, PA, USA, 2001.

[15] E. Cohen. Using selective path-doubling for parallel shortest-path computations. *J. Alg.*, 22:30–56, 1997.

[16] E. Cohen. Polylog-time and near-linear work approximation scheme for undirected shortest paths. *J. ACM*, 47(1):132–166, 2000.

[17] R. Cole. Parallel merge sort. *SIAM J. Comput.*, 17(4):770–785, 1988.

[18] R. Cole, P. N. Klein, and R. E. Tarjan. Finding minimum spanning forests in logarithmic time and linear work using random sampling. In *Proc. 8th Ann. ACM Symp. on Parallel Algorithms and Architectures (SPAA'96)*, pages 243–250. ACM, New York, NY, USA, 1996.

[19] E. W. Dijkstra. A note on two problems in connexion with graphs. *Numer. Math*, 1:269–271, 1959.

[20] J. R. Driscoll, H. N. Gabow, R. Shrairman, and R. E. Tarjan. Relaxed heaps: an alternative to fibonacci heaps with applications to parallel computation. *Commun. ACM*, 31(11):1343–1354, 1988.

[21] P. Erdős. Extremal problems in graph theory. In *Proc. Symp. Theory of Graphs and its Applications (smolenice, 1963)*, pages 29–36. Publ. House Czechoslovak Acad. Sci., Prague, Czechoslovak, 1964.

[22] J. Fakcharoenphol, C. Harrelson, S. Rao, and K. Talwar. An improved approximation algorithm for the 0-extension problem. In *Proc. 14th Ann. ACM-SIAM Symp. Discrete Algorithms (SODA'03)*, pages 257–265. ACM, New York, NY, USA, 2003.

[23] J. Fakcharoenphol, S. Rao, and K. Talwar. A tight bound on approximating arbitrary metrics by tree metrics. *J. Comput. System Sci.*, 69(3):485–497, 2004.

[24] S. Fortune. Robustness issues in geometric algorithms. In *Proc. 1st Workshop Applied Computational Geometry: Towards Geometric Engineering (WACG'96)*. Springer, Berlin, Germany, 1996.

[25] M. L. Fredman and R. E. Tarjan. Fibonacci heaps and their uses in improved network optimization algorithms. *J. ACM*, 34(3):596–615, 1987.

[26] A. Gupta, M. T. Hajiaghayi, and H. Räcke. Oblivious network design. In *Proc. 17th Ann. ACM-SIAM Symp. Discrete Algorithms (SODA'06)*, pages 970–979. ACM, New York, NY, USA, 2006.

[27] T. Hagerup. Fast parallel generation of random permutations. In *Proc. 18th Int. Coll. Automata, Languages, and Programming (ICALP'91)*, pages 405–416. Springer, Berlin, Germany, 1991.

[28] J. Y. Han, V. Pan, and J. Reif. Efficient parallel algorithms for computing all pair shortest paths in directed graphs. In *Proc. 4th Ann. ACM Symp. on Parallel Algorithms and Architectures (SPAA'92)*, pages 353–362. ACM, New York, NY, USA, 1992.

[29] S. Har-Peled and M. Mendel. Fast construction of nets in low dimensional metrics, and their applications. *SIAM J. Comput.*, 35:1148–1184, 2006.

[30] J. JáJá. *An Introduction to Parallel Algorithms*. Addison Wesley Longman Publishing Co., Inc., Redwood City, CA, USA, 1992.

[31] D. R. Karger, D. Koller, and S. J. Phillips. Finding the hidden path: time bounds for all-pairs shortest paths. *SIAM J. Comput.*, 22(6):1199–1217, 1993.

[32] D. R. Karger, N. Nisan, and M. Parnas. Fast connected components algorithms for the erew pram. *SIAM J. Comput.*, 28(3):1021–1034, 1998.

[33] R. Krauthgamer, J. R. Lee, M. Mendel, and A. Naor. Measured descent: A new embedding method for finite metrics. In *Proc. 45th Ann. IEEE Symp. Foundations of Computer Science (FOCS'04)*, pages 434–443. IEEE Computer Society, Washington, DC, USA, 2004.

[34] F. Lazebnik, V. A. Ustimenko, and A. J. Woldar. A new series of dense graphs of high girth. *Bull Amer. Math. Soc.*, 32(1):73–79, 1995.

[35] F. Lazebnik, V. A. Ustimenko, and A. J. Woldar. A characterization of the components of the graphs $d(k,q)$. In *Proc. 6th Conf. Formal Power Series and Algebraic Combinatorics (FPSAC'96)*, pages 271–283. Elsevier Science Publishers B. V., Amsterdam, The Netherlands, The Netherlands, 1996.

[36] C. Li, S. Pion, and C. K. Yap. Recent progress in exact geometric computation. *J. of Logic and Algebraic Programming*, 64:85–111, 2005.

[37] R. J. Lipton and R. E. Tarjan. A planar separator theorem. *SIAM J. Comput.*, 36(2):177–189, 1979.

[38] A. Lubotzky, R. Phillips, and P. Sarnak. Ramanujan graphs. *Combinatorica*, 8(3):261–277, 1988.

[39] M. O. Mark de Berg, Mark van Kreveld and O. Schwarzkopf. *Computational Geometry - Algorithms and Applications*. Springer, Berlin, Germany, 1997.

[40] J. Matoušek. On the distortion required for embedding finite metric space into normed spaces. *Israel J. Math.*, 93:333–344, 1996.

[41] M. Mendel and A. Naor. Maximum gradient embeddings and monotone clustering. In *Proc. Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques, 10th Int. Workshop (APPROX'07) and 11th Int. Workshop (RANDOM'07)*, pages 242–256. Springer, Berlin, Germany, 2007.

[42] M. Mendel and A. Naor. Ramsey partitions and proximity data structures. *J. European Math. Soc.*, 9(2):253–275, 2007.

[43] G. L. Miller and J. H. Reif. Parallel tree contraction and its application. In *Proc. 26th Ann. IEEE Symp. Foundations of Computer Science (FOCS'85)*, pages 478–489. IEEE Computer Society, Washington, DC, USA, 1985.

[44] D. Peleg and J. D. Ullman. An optimal synchronizer for the hypercube. *SIAM J. Comput.*, 18(4):740–747, 1989.

[45] M. C. Peter Buergisser and A. Shokrollahi. *Algebraic Complexity Theory*. Springer, Berlin, Germany, 1996.

[46] H. Räcke. Optimal hierarchical decompositions for congestion minimization in networks. In *Proc. 40th Ann. ACM Symp. Theory of computing (STOC'08)*, pages 255–264. ACM, New York, NY, USA, 2008.

[47] L. Roditty, M. Thorup, and U. Zwick. Deterministic constructions of approximate distance oracles and spanners. In *Proc. 32th Int. Coll. Automata, Languages, and Programming (ICALP'05)*, pages 261–272. Springer, Berlin, Germany, 2005.

[48] A. Schönhage. On the power of random access machines. In *Proc. 6th Int. Coll. Automata, Languages, and Programming (ICALP'79)*, pages 520–529. Springer, Berlin, Germany, 1979.

[49] D. Stevenson. *ANSI/IEEE 754-1985, Standard for Binary Floating-Point Arithmetic*. IEEE Computer Society, Washington, DC, USA, 1985.

[50] M. Thorup. Undirected single-source shortest paths with positive integer weights in linear time. *J. ACM*, 46(3):362–394, 1999.

[51] M. Thorup. Floats, integers, and single source shortest paths. *J. Algorithms*, 35(2):189–201, 2000.

[52] M. Thorup and U. Zwick. Approximate distance oracles. *J. ACM*, 52(1):1–24, 2005.

[53] J. F. Traub. A continuous model of computation. *Phys. Today*, May:39–43, 1999.

[54] P. van Emde Boas. Machine models and simulation. In *Handbook of Theoretical Computer Science*, volume A, pages 1–66. Elsevier Science Publishers B. V., Amsterdam, The Netherlands, The Netherlands, 1990.

[55] V. Vassilevska, R. Williams, and R. Yuster. All-pairs bottleneck paths for general graphs in truly sub-cubic time. In *Proc. 39th Ann. ACM Symp. Theory of computing (STOC'07)*, pages 585–589. ACM, New York, NY, USA, 2007.

[56] U. Zwick. Exact and approximate distances in graphs—a survey. In *Proc. 9th Ann. European Symp. Algorithms (ESA'01)*, pages 33–48. Springer, Berlin, Germany, 2001.

[57] U. Zwick. A slightly improved sub-cubic algorithm for the all pairs shortest paths problem with real edge lengths. *Algorithmica*, 46(2):181–192, 2006.