Optimal competitiveness for the Rectilinear Steiner Arborescence problem

Erez Kantor *
erezk@csail.mit.edu

Shay Kutten †
kutten@ie.technion.ac.il

Abstract

We present optimal online algorithms for two related known problems involving Steiner Arborescence, improving both the lower and the upper bounds. One of them is the well studied continuous problem of the *Rectilinear Steiner Arborescence* (RSA). We improve the lower bound and the upper bound on the competitive ratio for RSA from $O(\log N)$ and $\Omega(\sqrt{\log N})$ to $\Theta(\frac{\log N}{\log \log N})$, where N is the number of Steiner points. This separates the competitive ratios of RSA and the Symetric-RSA (SRSA), two problems for which the bounds of Berman and Coulston is STOC 1997 were identical. The second problem is one of the Multimedia Content Distribution problems presented by Papadimitriou et al. in several papers and Charikar et al. SODA 1998. It can be viewed as the discrete counterparts (or a network counterpart) of RSA. For this second problem we present tight bounds also in terms of the network size, in addition to presenting tight bounds in terms of the number of Steiner points (the latter are similar to those we derived for RSA).

Keywords: Online Algorithm, Approximation Algorithm, Video-on-Demand

1 Introduction

Steiner trees, in general, have many applications, see e.g. [12] for a rather early survey that already included hundreds of items. In particular, Steiner Arborescences[1] are useful for describing the evolution of processes in time. Intuitively, directed edges represent the passing of time. Since there is no way to go back in time in such processes, all the directed edges are directed away from the initial state of the problem (the root), resulting in an arborescence. Various examples are given in the literature such as processes in constructing a Very Large Scale Integrated electronic circuits (VLSI), optimization problems computed in iterations (where it was not feasible to return to results of earlier iterations), dynamic programming, and problems involving DNA, see, e.g. [4, 6, 13, 3]. Papadimitriou at al. [18, 19] and Charikar et al. [5] presented the discrete version, in the context of Multimedia Content Delivery (MCD) to model locating and moving caches for titles on a path graph. The formal definition of (one of the known versions) of this problem, Directed-MCD, appears in Section 2.

*CSAIL, MIT, Cambridge, MA. Supported in a part by NSF Awards 0939370-CCF, CCF-1217506 and CCF-AF-0937274 and AFOSR FA9550-13-1-0042.

†Department of Industrial Engineering and Management, IE&M, Technion, Haifa, Israel. Supported in part by the ISF, Israeli ministry of science and by the Technion Gordon Center.

[1] A Steiner arborescence is a Steiner tree directed away from the root.

We present new tight lower and upper bounds for two known interrelated problems involving Steiner Arborescences: *Rectilinear Steiner Arborescence (*RSA*)* and Directed-MCD (DMCD). We also deal indirectly with a third known arborescence problem: the *Symmetric*-RSA (SRSA) problem by separating its competitive ratio from that of RSA. That is, when the competitive ratios of RSA and SRSA were discussed originally by Berman and Coulston [4], the same lower and upper bounds were presented for both problems.

The *RSA* problem: This is a rather heavily studied problem, described also e.g. in [16, 21, 4, 17, 9]. A rectilinear line segment in the plane is either horizontal or vertical. A rectilinear path contains only rectilinear line segments. This path is also *y-monotone* (respectively, *x-monotone*) if during the traversal, the y (resp., x) coordinates of the successive points are never decreasing. The input is a set of *requests* $\mathcal{R} = \{r_1 = (x_1, y_1), ..., r_N = (x_N, y_N)\}$ called Steiner terminals (or points) in the positive quadrant of the plane. A feasible solution to the problem is a set of rectilinear segments connecting all the N terminals to the origin $r_0 = (0, 0)$, where the path from the origin to each terminal is both x-monotone and y-monotone (rectilinear shortest path). The goal is to find a feasible solution in which the sum of lengths of all the segments is the minimum possible. The above mentioned third problem, SRSA was defined in the same way, except that the above paths were not required to be x-monotone (only y-monotone).

Directed-MCD defined in Section 2 is very related to RSA. Informally, one difference is that it is discrete (Steiner points arrive only at discrete points) whiling RSA is continuous. In addition, in DMCD each "X coordinates" represents a network nodes. Hence, the number of X coordinates is bounded from above by the network size. This resemblance turned out to be very useful for us, both for solving RSA and for solving DMCD.

The *online* version of *RSA* [4]: the given requests (terminals) are presented to the algorithm with nondecreasing y-coordinates. After receiving the i'th request $r_i = (x_i, y_i)$ (for $i = 1, ..., N$), the on-line RSA algorithm must extend the existing arborescence solution to incorporate r_i. There are two additional constraints: (1) a line, once drawn (added to the solution), cannot be deleted, and (2) a segment added when handling a request r_i, can only be drawn in the region between y_{i-1} (the y-coordinates of the previous request r_{i-1}) and upwards (grater y-coordinates). If an algorithm obeys constraint (1) but not constraint (2), then we term it a *pseudo online* algorithm. Note that quite a few algorithms known as "online", or as "greedy offline" fit this definition of "pseudo online".

Additional Related works. Online algorithms for RSA and SRSA were presented by Berman and Coulston [4]. The online algorithms in [4] were $O(\log N)$ competitive (where N was the number of the Steiner points) both for RSA and SRSA. Berman and Coulston also presented $\Omega(\sqrt{\log N})$ lower bounds for both continuous problems. Note that the upper bounds for both problems were equal, and were the squares of the lower bounds. A similar gap for MCD arose from results of Halperin, Latombe, and Motwani [11], who gave a similar competitive ratio of $O(\log N)$, while Charikar, Halperin, and Motwani [5] presented a lower bound of $\Omega(\sqrt{\log n})$ for various variants of MCD, where n was the size of the network. Their upper bound was again the square of the lower bound: $O(\min\{\log n, \log N\})$ (translating their parameter p to the parameter n we use).

Berman and Coulston also conjectured that to close these gaps, both the upper bound and the lower bound for both problems could be improved. This conjecture was disproved in the cases of SRSA and of MCD on undirected line networks [15]. The latter paper closed the gap by presenting

an optimal competitive ratio of $O(\sqrt{\log N})$ for SRSA and $O(\min\{\sqrt{n}, \sqrt{\log N}\})$ for MCD on the undirected line network with n nodes. They left the conjecture of Berman and Coulston open for RSA and for MCD on directed line networks. In the current paper, we prove this conjecture (for RSA and for Directed-MCD), thus separating RSA and SRSA in terms of their competitive ratios.

Charikar, Halperin, and Motwani [5] also studied the the offline case for MCD, for which they gave a constant approximation. The offline version of RSA is heavily studied. It was attributed to [17] who gave an exponential integer programming solution and to [9] who gave an exponential time dynamic programming algorithm. An exact and polynomial algorithm was proposed in [23], which seemed surprising, since many Steiner problems are NP Hard. Indeed, difficulties in that solution were noted by Rao, Sadayappan, Hwang, and Shor [21], who also presented an approximation algorithm. Efficient algorithms are claimed in [7] for VLSI applications. However, the problem was proven NP-Hard in [22]. (The rectilinear Steiner tree problem was proven NPH in [10]). Heuristics that are fast "in practice" were presented in [8]. A PTAS was presented by [16]. An optimal logarithmic competitive ratio for MCD on *general undirected* networks was presented in [2]. They also present a constant off-line approximation for MCD on grid networks.

On the relation between this paper and [15]. An additional contribution of the current paper is the further development of the approach of developing (fully) online algorithms in two stages: (a) develop a pseudo online algorithm; and (b) convert the pseudo online into an online algorithm. As opposed to the problem studied in [15] where a pseudo online algorithm was known, here the main technical difficulty was to develop such an algorithm. From [15] we also borrowed an interesting twist on the rather common idea to translate between instances of a discrete and a continuous problems: we translate in *both* directions, the discrete solutions helps in optimizing the continuous one *and vice versa*.

Our Contributions. We improve both the upper and the lower bounds of RSA to show that the competitive ratio is $\Theta(\frac{\log N}{\log \log N})$. This proves the conjecture for RSA of Berman and Coulston [4] and also separates the competitive ratios of RSA and SRSA. We also provide tight upper and lower bound for Directed-MCD, the network version of RSA (both in terms of n and of N). The main technical innovation is the specific pseudo online algorithm we developed here, in order to convert it later to an online algorithm. The previously known offline algorithms for RSA and for DMCD where *not* pseudo online, so we could not use them. In addition to the usefulness of the new algorithm in generating the online algorithm, this pseudo online algorithm may be interesting in itself: It is $O(1)$-competitive for DMCD and for RSA (via the transformation) for a *different* (but rather common) online model (where each request must be served before the next one arrives, but no time passes between requests).

Paper Structure. Definitions are given in Section 2. The pseudo online algorithm SQUARE for DMCD is presented and analyzed in Section 3. In Section 4, we transform SQUARE to a (fully) online algorithm D-LINE$^{\text{on}}$ for DMCD. Then, Section 5 describes the transformation of the online DMCD algorithm D-LINE$^{\text{on}}$ to become an optimal online algorithm for RSA, as well as a transformation back from RSA to DMCD to make the DMCD online algorithm also optimal in terms of n (not just N). These last two transformations are taken from [15]. Finally, a lower bound is given in Section 6. The best way to understand the algorithms in this paper may be from

a geometric point of view. Hence, we added multiple drawings to illustrate both the algorithms and the proofs.

2 Preliminaries

The network×time grid (Papadimitriou et. al, [19]). A *directed line network* $L(n) = (V_n, E_n)$ is a network whose node set is $V_n = \{1, ..., n\}$ and its edge set is $E_n = \{(i, i+1) \mid i = 1, ..., n-1\}$. Given a directed line network $L(n) = (V_n, E_n)$, construct "time-line" graph $\mathcal{L}(n) = (\mathcal{V}_n, \mathcal{E}_n)$, intuitively, by "layering" multiple replicas of $L(n)$, one per time unit, where in addition, each node in each replica is connected to the same node in the next replica (see Fig. 1). Formally, the node set $\mathcal{V}_n$ contains a *node replica* (sometimes called just a *replica*) (v, t) of every $v \in V_n$, coresponding to each time step $t \in \mathbb{N}$. That is, $\mathcal{V}_n = \{(v, t) \mid v \in V_n, t \in \mathbb{N}\}$. The set of directed edges $\mathcal{E}_n = \mathcal{H}_n \cup \mathcal{A}_n$ contains *horizontal directed edges* $\mathcal{H}_n = \{((u, t), (v, t)) \mid (u, v) \in E_n, t \in \mathbb{N}\}$, connecting network nodes in every time step (round), and directed *vertical edges*, called *arcs*, $\mathcal{A}_n = \{((v, t), (v, t+1)) \mid v \in V_n, t \in \mathbb{N}\}$, connecting different copies of V_n. When n is clear from the context, we may write just X rather than X_n, for every $X \in \{V, E, \mathcal{V}, \mathcal{H}, \mathcal{A}\}$. Notice that $\mathcal{L}(n)$ can be viewed geometrically as a grid of n by ∞ whose grid points are the replicas. Following Fig. 1, we consider the time as if it proceeds upward. We use such geometric presentations also in the text, to help clarifying the description.

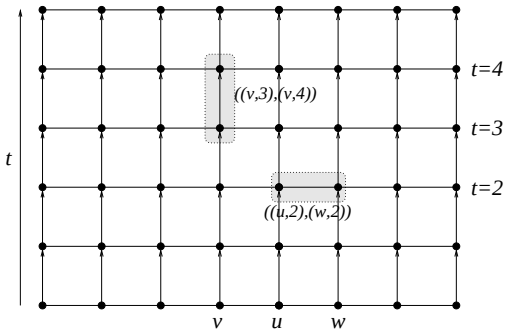

Figure 1: An example of a time-line graph $\mathcal{L}(n) = (\mathcal{V}, \mathcal{E} = \mathcal{H} \cup \mathcal{A})$. Each node in $\mathcal{V}$ is represented by a circle; each horizontal edge in $\mathcal{H}$ is represented by a horizontal segment (see, as an example, $((u, 2), (w, 2)) \in \mathcal{H}$ for an horizontal directed edge in the marked rectangle on the right); each arc in $\mathcal{A}$ is represented by a horizontal arrow (see, as an example, $((v, 3), (v, 4)) \in \mathcal{A}$ for an arc in the marked rectangle on the left).

The DMCD **problem.** We are given a directed line network $L(n)$, an *origin* node $v_0 \in V$, and a set of *requests* $\mathcal{R} \subseteq \mathcal{V}$. A feasible solution is a subset of directed edges $\mathcal{F} \subseteq \mathcal{E}$ such that for every request $r \in \mathcal{R}$, there exists a path in $\mathcal{F}$ from the origin $(v_0, 0)$ to r. Intuitively a directed horizontal edge $((u, t), (v, t))$ is for delivering a copy of a multimedia title from node u to node v at time t.

A directed vertical edge (arc) $((v, t), (v, t+1))$ is for storing a copy of the title at node v from time t to time $t+1$. For convenience, the endpoints $\mathcal{V}_\mathcal{F}$ of edges in $\mathcal{F}$ are also considered parts of the solution. For a given algorithm A, let $\mathcal{F}_A$ be the solution of A, and let $cost(A, \mathcal{R})$, (the cost of algorithm A), be $|\mathcal{F}_A|$. (We assume that each storage cost and each delivery cost is 1.) The goal is to find a minimum cost feasible solution. Let OPT be the set of edges in some optimal solution whose cost is $|\text{OPT}|$.

Online *DMCD*. In the online versions of the problem, the algorithm receives as input a sequence of events. One type of events is a request in the (ordered) set $\mathcal{R}$ of requests $\mathcal{R} = \{r_1, r_2, ..., r_N\}$, where the requests times are in a non-decreasing order, i.e., $t_1 \leq t_2 \leq ... \leq t_N$ (as in RSA). A second type of events is a time event (this event does not exists in RSA), where we assume a clock that tells the algorithm that no additional requests for time t are about to arrive (or that there are no requests for some time t at all). The algorithm then still has the opportunity to complete its calculation for time t (e.g., add arcs from some replica (v,t) to $(v,t+1)$). Then time $t+1$ arrives.

When handling an event ev, the algorithm only knows the following: (a) all the previous requests $r_1, ..., r_i$; (b) time t; and (c) the solution arborescence $\mathcal{F}_{ev}$ it constructed so far (originally containing only the origin). In each event, the algorithm may need to make decisions of two types, before seeing future events:

(1.DMCD) If the event is the arrival of a request $r_i = (v_i, t_i)$, then from which *current* (time t_i) cache (a point already in the solution arborescence $\mathcal{F}_{ev}$ when r_i arrives) to serve r_i by adding *horizontal* directed edges to $\mathcal{F}_{ev}$.

(2.DMCD) If this is the time event for time t, then at which nodes to store a copy for time $t+1$, for future use: select some replica (or replicas) (v,t) already in the solution $\mathcal{F}_{ev}$ and add to $\mathcal{F}_{ev}$ an edge directed from (v,t) to $(v,t+1)$.

Note that at time t, the online algorithm cannot add nor delete any edge with an endpoint that corresponds to previous times. Similarly to e.g. [2, 18, 20, 19, 5], at least one copy must remain in the network at all times.

General definitions and notations. Consider an interval $J = \{v, v+1, ..., v+\rho\} \subseteq V$ and two integers $s, t \in \mathbb{N}$, s.t. $s \leq t$. Let $J[s,t]$ (see Fig. 2) be the *"rectangle subgraph"* of $\mathcal{L}(n)$ corresponding to vertex set J and time interval $[s,t]$. This rectangle consists of the replicas and edges of the nodes of J corresponding to every time in the interval $[s,t]$. For a given subsets $\mathcal{V}' \subseteq \mathcal{V}$, $\mathcal{H}' \subseteq \mathcal{H}$ and $\mathcal{A}' \subseteq \mathcal{A}$, denote by (1) $\mathcal{V}'[s,t]$ replicas of $\mathcal{V}'$ corresponding to times $s, ..., t$. Define similarly (2) $\mathcal{H}'[s,t]$ for horizontal edges of $\mathcal{H}'$; and (3) $\mathcal{A}'[s,t]$ arcs of $\mathcal{A}'$. (When $s = t$, we may write $\mathcal{X}[t] = \mathcal{X}[s,t]$, for $\mathcal{X} \in \{J, \mathcal{V}', \mathcal{H}'\}$.) Consider also two nodes $v, u \in V$ s.t. $u \leq v$. Let $\mathcal{P}_{\mathcal{H}}[(u,t),(v,t)]$

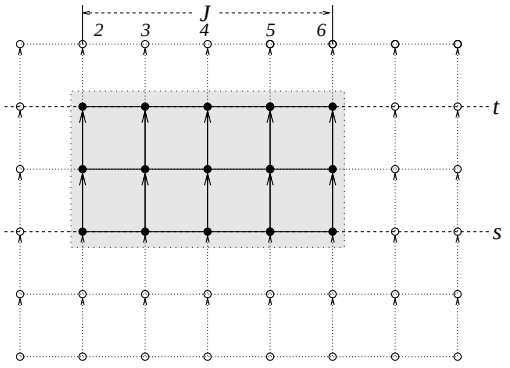

Figure 2: A subgraph rectangle $J[s,t]$, where $J = \{2,3,4,5,6\}$.

be the set of horizontal directed edges of the path from (u,t) to (v,t). Let $\mathcal{P}_{\mathcal{A}}[(v,s),(v,t)]$ be the set of arcs of the path from (v,s) to (v,t). Let $dist^{\rightarrow}_{\infty}((u,s),(v,t))$ be the "directed" distance from

(u, s) to (v, t) in L_∞ norm. Formally, $dist^{\rightarrow}_{\infty}((u, s), (v, t)) = \max\{t - s, v - u\}$, if $s \leq t$ and $u \leq v$ and $dist^{\rightarrow}_{\infty}((u, s), (v, t)) = \infty$, otherwise.

3 Algorithm SQUARE, a pseudo online algorithm

This section describes a pseudo online algorithm named SQUARE for the DMCD problem. Developing SQUARE was the main technical difficulty of this paper. Consider a requests set $\mathcal{R} = \{r_0 = (0, 0), r_1 = (v_1, t_1), ..., r_N = (v_N, t_N)\}$ such that $0 \leq t_1 \leq t_2 \leq ... \leq t_N$. When Algorithm SQUARE starts, the solution includes just $r_0 = (0, 0)$. Then, SQUARE handles, first, request r_1, then, request r_2, etc... In handling a request r_i, the algorithm may add some edges to the solution. (It never deletes any edge from the solution.) After handling r_i, the solution is an arborescence rooted at r_0 that spans the request replicas $r_1, ..., r_i$. Denote by SQUARE(i) the solution of SQUARE after handling the i'th request. For a given replica $r = (v, t) \in \mathcal{V}$ and a positive integer ρ, let

$$\mathcal{S}[r, \rho] = [v - \rho, v] \times [t - \rho, t]$$

denotes the rectangle subgraph (of the layered graph) whose top right corner is r induced by the set of replicas that contains every replica q such that (1) there is a directed path in the layer graph from q to r; and (2) the distance from q to r in L_∞ is at most ρ. For each request $r_i \in \mathcal{R}$, for $i = 1, ..., N$, SQUARE performs the following (The pseudo code of SQUARE is given in Fig. 4 and an example of an execution in Fig. 3).

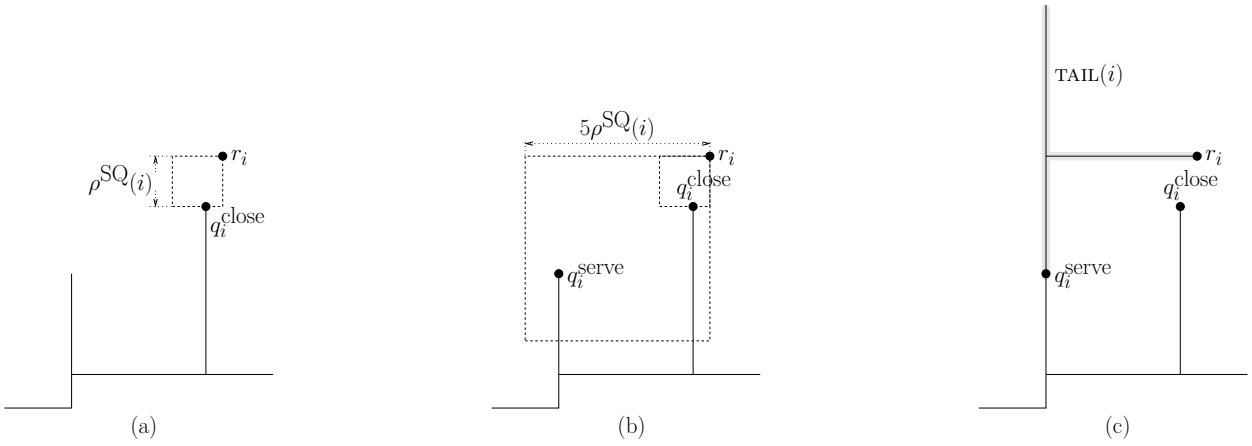

Figure 3: SQUARE execution example. (a) in case (SQ3), SQUARE$(i-1)$ (SQUARE's solution after handling point r_{i-1}); q_i^{close} defines the radius $\rho^{\mathsf{SQ}}(i)$. (b) the serving replica q_i^{serve} is the leftmost in $\mathcal{S}[r_i, 5\rho^{\mathsf{SQ}}(i)] \cap$ SQUARE$(i-1)$. (c) SQUARE(i).

(SQ1) Add the vertical path from $(0, t_{i-1})$ to $(0, t_i)$.

(SQ2) Let replica $q_i^{\mathrm{close}} = (u_i^{\mathrm{close}}, s_i^{\mathrm{close}})$ be such that q_i^{close} is already in the solution SQUARE$(i-1)$ and (1) the distance in L_∞ norm from q_i^{close} to r_i is minimum (over the replicas already in the solution); and (2) over those replicas choose the latest, that is, $s_i^{\mathrm{close}} = \max\{t \leq t_i \mid (u_i^{\mathrm{close}}, t) \in \text{SQUARE}(i-1)\}$. Define the *radius* of r_i as $\rho^{\mathrm{SQ}}(i) = dist^{\rightarrow}_{\infty}(q_i^{\mathrm{close}}, r_i) = \max\{|v_i - u_i^{\mathrm{close}}|, |t_i - s_i^{\mathrm{close}}|\}$. Call q_i^{close} the *closest* replica of the i'th request.

(SQ3) Choose a replica $q_i^{\text{serve}} = (u_i^{\text{serve}}, s_i^{\text{serve}}) \in \mathcal{S}[r_i, 5 \cdot \rho^{\text{SQ}}(i)]$ such that q_i^{serve} is already in the solution SQUARE$(i-1)$ and u_i^{serve} is the leftmost node (over the nodes corresponding to replicas of $\mathcal{S}[r_i, 5 \cdot \rho^{\text{SQ}}(i)]$ that are already in the solution). Call q_i^{serve} the *serving replica* of the i'th request.

(SQ4) Deliver a copy from q_i^{serve} to r_i via $(u_i^{\text{serve}}, t_i)$. This is done by storing a copy in node u_i^{serve} from time s_i^{serve} to time t_i, and then delivering a copy from $(u_i^{\text{serve}}, t_i)$ to (v_i, t_i) [2].

(SQ5) Store a copy in u_i^{serve} from time t_i to time $t_i + 4 \cdot \rho^{\text{SQ}}(i)$ [3].

Intuitively, steps SQ1–SQ4 utilize previous replicas in the solution, while step SQ5 prepares the contribution of r_i to serve later requests. Note that SQUARE is not an online algorithm, since in step SQ4, it may add to the solution some arcs corresponding to previous times. Such an action cannot be preformed by an online algorithm. Denote by $\mathcal{F}^{\text{SQ}} = \mathcal{H}^{\text{SQ}} \cup \mathcal{A}^{\text{SQ}}$ the feasible solution SQUARE(N) of SQUARE. Let BASE$(i) = \{(u, t_i) \mid u_i^{\text{serve}} \leq u \leq v_i\}$ and let BASE $= \cup_{i=1}^{N}$BASE(i) (notice that BASE $\subseteq \mathcal{F}^{\text{SQ}}$ because of step SQ4). Similarly, let TAIL$(i) = \{(u_i^{\text{serve}}, t) \mid t_i \leq t \leq t_i + 4\rho^{\text{SQ}}(i)\}$ be the nodes of the path $\mathcal{P}_{\mathcal{A}}[(u_i^{\text{serve}}, t_i), (u_i^{\text{serve}}, t_i + 4 \cdot \rho^{\text{SQ}}(i))]$ (added to the solution in step SQ5) and let TAIL $= \cup_{i=1}^{N}$TAIL(i). Note that $\mathcal{F}^{\text{SQ}}$ is indeed an arborescence rooted at $(0, 0)$.

- $\mathcal{F}^{\text{SQ}} \leftarrow \{(0,0)\}$ is the SQUARE's solution after handling request r_{i-1}.
- When request r_i arrives do:

1. $\mathcal{F}^{\text{SQ}} \leftarrow \mathcal{F}^{\text{SQ}} \cup \mathcal{P}_{\mathcal{A}}[(0, t_{i-1}), (0, t_i)]$.

2'. $\rho^{\text{SQ}}(i) \leftarrow \min\{dist_{\infty}^{\rightarrow}(q, r_i) \mid q \in \mathcal{F}^{\text{SQ}}\}$.

2. Choose a replica $q_i^{\text{close}} = (u_i^{\text{close}}, s_i^{\text{close}})$ such that q_i^{close} is in SQUARE$(i-1)$ and
   (a) $dist_{\infty}^{\rightarrow}(q_i^{\text{close}}, r_i) = \rho^{\text{SQ}}(i)$; and
   (b) $s_i^{\text{close}} = \max\{t \leq t_i \mid (u_i^{\text{close}}, t) \in \text{SQUARE}(i-1)\}$.

3. Choose the serving replica $q_i^{\text{serve}} = (u_i^{\text{serve}}, s_i^{\text{serve}}) \in \mathcal{S}[r_i, 5 \cdot \rho^{\text{SQ}}(i)] \cap \mathcal{F}^{\text{SQ}}$ such that $u_i^{\text{serve}} = \min\left\{u \mid \exists s \text{ such that } (u, s) \in \mathcal{S}[r_i, 5 \cdot \rho^{\text{SQ}}(i)] \cap \mathcal{F}^{\text{SQ}}\right\}$.
   $\triangleright$ u_i^{serve} is the leftmost node corresponding to the replicas of $\mathcal{S}[r_i, 5 \cdot \rho^{\text{SQ}}(i)] \cap \mathcal{F}^{\text{SQ}}$

4. $\mathcal{F}^{\text{SQ}} \leftarrow \mathcal{F}^{\text{SQ}} \cup \mathcal{P}_{\mathcal{A}}[(u_i^{\text{serve}}, s_i^{\text{serve}}), (u_i^{\text{serve}}, t_i)] \cup \mathcal{P}_{\mathcal{H}}[(u_i^{\text{serve}}, t_i), (v_i, t_i)]$.
   $\triangleright$ deliver a copy from q_i^{serve} to r_i via $(u_i^{\text{serve}}, t_i)$.

5. $\mathcal{F}^{\text{SQ}} \leftarrow \mathcal{F}^{\text{SQ}} \cup \mathcal{P}_{\mathcal{A}}[(u_i^{\text{serve}}, t_i), (u_i^{\text{serve}}, t_i + 4\rho^{\text{SQ}}(i))]$.
   $\triangleright$ leave a copy at u_i^{serve} from current time t_i to time $t_i + 4\rho^{\text{SQ}}(i)$.

Figure 4: Algorithm SQUARE.

[2] More formally, add the arcs of $\mathcal{P}_{\mathcal{A}}[(u_i^{\text{serve}}, s_i^{\text{serve}}), (u_i^{\text{serve}}, t_i)]$ and the horizontal directed edges of $\mathcal{P}_{\mathcal{H}}[(u_i^{\text{serve}}, t_i), (v_i, t_i)]$ to the solution.

[3] More formally, add the arcs of $\mathcal{P}_{\mathcal{A}}[(u_i^{\text{serve}}, t_i), (u_i^{\text{serve}}, t_i + 4 \cdot \rho^{\text{SQ}}(i))]$ to the solution.

3.1 Analysis of SQUARE

First, bound the cost of SQUARE as a function of the radii (defined in SQ2).

Observation 3.1 *cost*(SQUARE, $\mathcal{R}$) $\leq 14 \sum_{i=1}^{N} \rho^{\mathrm{SQ}}(i)$.

Proof: For each request $r_i \in \mathcal{R}$, Algorithm SQUARE pay a cost of $10\rho^{\mathrm{SQ}}(i)$ to the path between r_i's serving replica q_i^{serve} a r_i itself (see step SQ4) and additional cost of $4\rho^{\mathrm{SQ}}(i)$ for serving a copy to all replicas of TAIL(i) (see step SQ5). ∎

It is left to bound from below the cost of the optimal solution as a function of the radii.

Quarter balls. Our analysis is based on the following notion. A *quarter-ball*, or a Q-BALL, of *radius* $\rho \in \mathbb{N}$ centered at a replica $q = (v, t) \in \mathcal{V}$ contains every replica from which there exists a path of length ρ to q [4] . For every request $r_i \in \mathcal{R}$, denote by $Q\text{-BALL}^{\mathrm{SQ}}(r_i, \rho^{\mathrm{SQ}}(i))$ [5] (also $Q\text{-BALL}^{\mathrm{SQ}}(i)$ for short) the quarter-ball centered at r_i with radius $\rho^{\mathrm{SQ}}(i)$.

Intuitively, for every request $r_i \in \mathcal{R}'$ (where $\mathcal{R}'$ obey the observation's condition below), OPT's solution starts outside of $Q\text{-BALL}^{\mathrm{SQ}}(i)$, and must reach r_i with a cost of $\rho^{\mathrm{SQ}}(i)$ at least.

Observation 3.2 *Consider some subset $\mathcal{R}' \subseteq \mathcal{R}$ of requests. If the Q-balls, $Q\text{-BALL}^{\mathrm{SQ}}(i)$ and $Q\text{-BALL}^{\mathrm{SQ}}(j)$, of every two requests $r_i, r_j \in \mathcal{R}'$ are edges disjoint, then $|\mathrm{OPT}| \geq \sum_{r_i \in \mathcal{R}'} \rho^{\mathrm{SQ}}(i)$.*

Intuitively, for every request $r_i \in \mathcal{R}'$ (where $\mathcal{R}'$ obey the observation's condition), OPT's solution starts outside of $Q\text{-BALL}^{\mathrm{SQ}}(i)$, and must reach r_i with a cost of $\rho^{\mathrm{SQ}}(i)$ at least.

Proof: Consider some request $r_i \in \mathcal{R}'$. Any directed path from $(v_0, 0)$ to r_i must enter the quarter ball $Q\text{-BALL}^{\mathrm{SQ}}(i)$ of radius $\rho^{\mathrm{SQ}}(i)$ to reach r_i. The length of this path inside the $Q\text{-BALL}^{\mathrm{SQ}}(i)$ is $\rho^{\mathrm{SQ}}(i)$. All the Q-BALLs of $\mathcal{R}'$ are disjoint, which implies the observation. ∎

3.1.1 Covered and uncovered requests

Consider some request $r_i = (v_i, t_i)$ and its serving replica $q_i^{\mathrm{serve}} = (u_i^{\mathrm{serve}}, s_i^{\mathrm{serve}})$ (see step SQ3). We say that r_i is *covered*, if $v_i - u_i^{\mathrm{serve}} \geq \rho^{\mathrm{SQ}}(i)$ (see SQ2 and SQ3). Intuitively, this means the solution $\mathcal{F}^{\mathrm{SQ}}$ is augmented by the whole top of the square SQUARE$[r_i, \rho^{\mathrm{SQ}}(i)]$; see Figure 3 (a) and (b). Otherwise, we say that r_i is *uncovered*. Let COVER $= \{i \mid r_i$ is a covered request$\}$ and let UNCOVER $= \{i \mid r_i$ is an **un**covered request$\}$. Given Observation 3.2, the following lemma implies that

$$|\mathrm{OPT}| \geq \sum_{i \in \mathrm{COVER}} \rho^{\mathrm{SQ}}(i). \tag{1}$$

Lemma 3.3 *Consider two* **covered** *requests r_i and r_j.*
Then, the quarter balls $Q\text{-BALL}^{\mathrm{SQ}}(i)$ and $Q\text{-BALL}^{\mathrm{SQ}}(j)$ are edge disjoint.

Proof: Assume without loss of generality that, $i > j$. Thus, $\mathcal{P}_{\mathcal{H}}[(u_j^{\mathrm{serve}}, t_j), (v_j, t_j)]$ (see SQ4) is already in the solution when handling request i. Also, $\rho^{\mathrm{SQ}}(j) \geq v_j - u_j^{\mathrm{serve}}$, since r_j is covered. Consider three cases.

[4] This is, actually, the definition of the geometric place "ball". We term them "quarter ball" to emphasize that we deal with directed edges. That is, it is not possible to reach (v, t) from above nor from the right.

[5] Note that $Q\text{-BALL}^{\mathrm{SQ}}(r_i, \rho^{\mathrm{SQ}}(i))$ is different from $\mathcal{S}[r_i, \rho^{\mathrm{SQ}}(i)]$, since the first ball considers distances in L_2 norm and the last considers distances in L_∞ norm.

Case 1. $v_j \leq v_i$, see Figure 5. Since, r_i is covered, $\rho^{\text{SQ}}(i) \leq dist^{\rightarrow}_{\infty}(r_j, r_i) = \max\{v_i - v_j \ , \ t_i - t_j\}$. If $v_j \leq v_i - \rho^{\text{SQ}}(i)$, then these two Q-BALLs are edges disjoint. Otherwise, $v_i - v_j < \rho^{\text{SQ}}(i)$. Then $\rho^{\text{SQ}}(i) \leq t_i - t_j$ which implies that these two Q-BALLs are edges disjoint.

Case 2. $v_j - \rho^{\text{SQ}}(j) \leq v_i \leq v_j$, see Figure 6. Then, also $v_i \geq u_j^{\text{serve}}$, since r_j is covered. Thus, in particular, $(v_i, t_j) \in \text{SQUARE}(i-1)$. Hence, $\rho^{\text{SQ}}(i) \leq dist^{\rightarrow}_{\infty}((v_i, t_j), r_i = (v_i, t_i)) = t_i - t_j$, which implies that these two Q-BALLs are edges disjoint.

Case 3. $v_i < v_j - \rho^{\text{SQ}}(j)$. The Q-ball of r_j is on the right of any possible (radius) Q-ball with r_i as a center. Thus, these Q-balls are edges disjoint.

∎

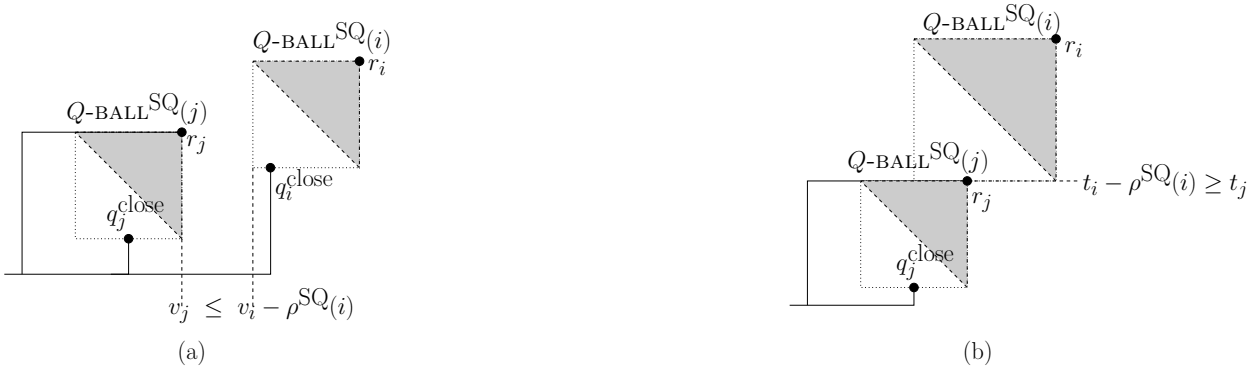

Figure 5: Two covered requests are edge disjoint, case 1; (a) Q-BALL$^{\text{SQ}}(i)$ is on the right of Q-BALL$^{\text{SQ}}(j)$, since $v_j \leq v_i - \rho^{\text{SQ}}(i)$; (b) $v_i - \rho^{\text{SQ}}(i) \leq v_j \leq v_i$ implying that the whole Q-BALL$^{\text{SQ}}(i)$ is above Q-BALL$^{\text{SQ}}(j)$.

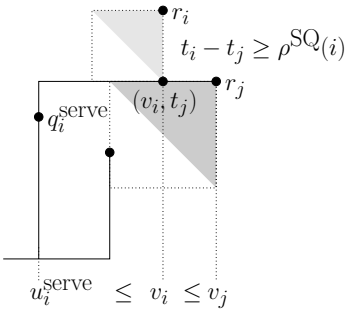

Figure 6: Two covered requests are edge disjoint, case 2.

By the above lemma and observations 3.1, 3.2, and Inequality (1), we have:

Observation 3.4 SQUARE*'s cost for covered requests is no more than* $14 \cdot$ OPT.

It is left to bound the cost of SQUARE for the uncovered requests.

3.1.2 Overview of the analysis of the cost of uncovered requests

Unfortunately, unlike the case of covered requests, balls of two *uncovered* requests may not be disjoint. Still, we managed to have a somewhat similar argument that we now sketch. (The formal analysis appears later in Subsection 3.2.) Below, we partition the balls of uncovered requests into disjoint subsets. Each has a representative request, a *root*. We show that the Q-BALL of roots *are* edge disjoint. This implies by Observation 3.1 and Observation 3.2 that the cost SQUARE pays for the roots is smaller than 14 times the total cost of an optimal solution. Finally, we show that the cost of SQUARE for all the requests in each subset is at most twice the cost of SQUARE for the root of the subset. Hence, the total cost of SQUARE for the uncovered requests is also just a constant times the total cost of the optimum.

To construct the above partition, we define the following relation: ball Q-BALL$^{\text{SQ}}(j)$ is the *child* of Q-BALL$^{\text{SQ}}(i)$ (for two *uncovered* requests r_i and r_j) intuitively, if the Q-BALL$^{\text{SQ}}(i)$ is the first ball (of a request later then r_j) such that Q-BALL$^{\text{SQ}}(i)$ and Q-BALL$^{\text{SQ}}(j)$ are *not* edge disjoint. Clearly, this parent-child relation induces a forest on the Q-BALLs of uncovered requests. The following observation follows immediately from the definition of a root.

Observation 3.5 *The quarter balls of every two root requests are edge disjoint.*

Proof: Consider two root requests r_i and r_j. Assume W.O.L.G that $j < i$. Also assume, toward contradiction that Q-BALL$^{\text{SQ}}(i)$ and Q-BALL$^{\text{SQ}}(j)$ are **not** edge disjoint. By the definition of the child parent relationship, either r_j is child of r_i, or r_j is a child of some other request r_ℓ for some $j < \ell < i$. In both cases, r_j has a parent, hence r_j is not a root request which contradict to choice of r_j as a root request. The observation follow. ∎

The above observation together with Observation 3.2, implies the following.

Observation 3.6 *The cost of* SQUARE *for the roots is* $14 \cdot |\text{OPT}|$ *at most.*

It is left to bound the cost that SQUARE pays for the balls in each tree (in the forest of Q-BALLs) as a constant function of the cost it pays for the tree root. Specifically, we show that the sum of the radii of the Q-BALLs in the tree (including that of the root) is at most twice the radius of the root. This implies the claim for the costs by Observation 3.1 and Observation 3.2. To show that, given any non leaf ball Q-BALL$^{\text{SQ}}(i)$ (not just a root), we first analyze only Q-BALL$^{\text{SQ}}(i)$'s "latest child" Q-BALL$^{\text{SQ}}(j)$. That is, $j = \max_k\{Q\text{-BALL}^{\text{SQ}}(k)$ is a child of $Q\text{-BALL}^{\text{SQ}}(i)\}$. We show that the radius of the latest child is, at most, a quarter of the radius of Q-BALL$^{\text{SQ}}(i)$. (See Property (P1) of Lemma 3.14 in Subsection 3.2.) Second, we show that the *sum* of the radii of the rest of the children (all but the latest child) is, at most, a quarter of the radius of Q-BALL$^{\text{SQ}}(i)$ too. Hence, the radius of a parent ball is at least twice as the sum of its children radii. This implies that the sum of the radii of all the Q-BALLs in a tree is at most twice the radius of the root.

The hardest technical part here is in the following lemma that, intuitively, states that "a lot of time" (proportional to the request's radius) passes between the time one child ball ends and the time the next child ball starts, see Fig. 7.

Lemma 3.7 *Consider some uncovered request r_i which has at least two children. Let Q-BALL$^{\text{SQ}}(j)$, Q-BALL$^{\text{SQ}}(k)$ some two children of Q-BALL$^{\text{SQ}}(i)$, such that $k < j$. Then, $t_j - \rho^{\text{SQ}}(j) \geq t_k + 4\rho^{\text{SQ}}(k)$.*

Intuitively, the radius of a parent Q-BALL is covered by the radii of its children Q-BALLs, plus the tails (see step SQ5) between them. Restating the lemma, the time of the earliest replica in Q-BALL$^{\text{SQ}}(j)$ is not before the time of the latest replica in TAIL(k). Intuitively, recall that the tail

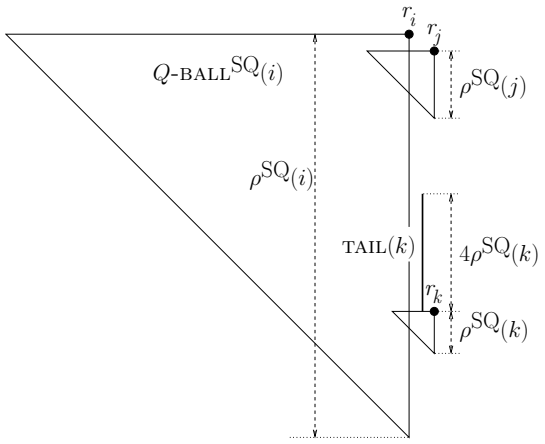

Figure 7: Geometric look on a parent Q-BALL$^{\mathrm{SQ}}(i)$ (note that a Q-BALL is a triangle) and its children Q-BALL$^{\mathrm{SQ}}(j)$ and Q-BALL$^{\mathrm{SQ}}(k)$.

length of a request is much grater than the radius of the request's Q-BALL. Hence, the fact that the radius of a latest child is at most a quarter of the radius of its parent, together with Lemma 3.7, imply that the sum of the childrens radii is less than half of the radius of the parent Q-BALL.

The full proof of Lemma 3.7 (appears in Subsection 3.2) uses geometric considerations. Outlining the proof, we first establish an additional lemma. Given any two requests r_j and r_ℓ such that $j > \ell$, the following lemma formalizes the following: Suppose that the node v_j of request r_j is "close in space (or in the network)" to the node v_ℓ of another request r_ℓ. Then, the whole Q-BALL of r_j is "far in time" (and later) from r_j.

Lemma 3.8 *Suppose that, $j > \ell$ and $v_j - \rho^{\mathrm{SQ}}(j) + 1 \leq u_\ell^{\mathrm{serve}} \leq v_j$. Then, the time of the earliest replica in Q-BALL$^{\mathrm{SQ}}(j)$ is not before the time of the latest replica in* TAIL(ℓ)*, i.e., $t_j - \rho^{\mathrm{SQ}}(j) \geq t_\ell + 4\rho^{\mathrm{SQ}}(\ell)$.*

Intuitively, Lemma 3.8 follows thanks to the tail left in step SQ5 of SQUARE, as well as to the action taken in SQ3 for moving u^{serve} further left of u^{close}. In the proof of Lemma 3.7, we show that in the case that two requests r_k and r_j are siblings, either **(1)** they satisfy the conditions of Lemma 3.8, or **(2)** there exists some request r_ℓ such that $k < \ell < j$ such that r_ℓ and r_j satisfy the conditions of Lemma 3.8. Moreover, the time of the last replica in TAIL(ℓ) is even later then the time of the last replica in TAIL(k). In both cases, we apply Lemma 3.8 to show that the time of the earliest replica in Q-BALL$^{\mathrm{SQ}}(j)$ is not before the time of the latest replica in TAIL(k) as needed for the lemma.

To summarize, we show (1) For *covered* requests the cost of SQUARE is $O(1)$ of $|\mathrm{OPT}|$; see Observation 3.4. (2) For *uncovered* requests, we prove two facts: (2.a) the Q-BALLs of the root requests are edges disjoint, and hence by Observation 3.6, the sum of their radii is $O(1)$ of $|\mathrm{OPT}|$ too. (2.b) On the other hand, the sum of root's radii is at least half of the sum of the radii of all the uncovered requests. This establishes Theorem 3.9 (which prove appears in Subsection 3.2).

Theorem 3.9 *Algorithm* SQUARE *is $O(1)$-competitive for* DMCD *under the pseudo online model.*

3.2 Formal analysis of the cost of uncovered requests

We start with a formal definitions of the forest of parent-child relationships.

Forest of balls. For any uncovered request r_i, define the following notations.

1. Let $parent(i) \triangleq j$ be the minimal index grater than i such that $Q\text{-BALL}^{\text{SQ}}(i)$ and $Q\text{-BALL}^{\text{SQ}}(j)$ are not edges disjoint, if such exists, otherwise $parent(i) \triangleq \perp$.
2. $\text{CHILDREN}(i) \triangleq \{j \mid parent(j) = i\}$.
3. $\text{TREE}(i) \triangleq \bigcup_{j \in \text{CHILDREN}(i)} \text{TREE}(j)$, if $\text{CHILDREN}(i) \neq \emptyset$, otherwise $\text{TREE}(i) \triangleq \{i\}$.
4. $\text{ROOTS} = \{i \mid parent(i) = \perp\}$.

(We also abuse the definition and say that request r_j is child of request r_i; and j is child of i, if $Q\text{-BALL}^{\text{SQ}}(j)$ is child of $Q\text{-BALL}^{\text{SQ}}(i)$.)

We now, state four observations about uncovered requests. The main lemmas use these observations heavily. Recall that, $q_i^{\text{close}} = (u_i^{\text{close}}, s_i^{\text{close}})$ is the closest replica of r_i (see SQ2).

Observation 3.10 *The radius of an uncovered request is determined by the* time difference *from its closest replica. That is, if a request r_i is uncovered, then $\rho^{\text{SQ}}(i) = t_i - s_i^{\text{close}}$.*

Proof: If r_i is uncovered, then $v_i - u_i^{\text{serve}} < \rho^{\text{SQ}}(i)$. In addition, $v_i - u_i^{\text{close}} \leq v_i - u_i^{\text{serve}}$, since $u_i^{\text{serve}} \leq u_i^{\text{close}}$ (see SQ2). Thus, $v_i - u_i^{\text{close}} < \rho^{\text{SQ}}(i)$ too. Therefore, $\rho^{\text{SQ}}(i) = t_i - s_i^{\text{close}}$, since $\rho^{\text{SQ}}(i) = dist_{\infty}^{\rightarrow}(q_i^{\text{close}}, r_i) = \max\{v_i - u_i^{\text{close}}, t_i - s_i^{\text{close}}\}$ (see SQ2). ∎

Observation 3.11 *The replicas of the "rectangle-graph" $[v_i - 5\rho^{\text{SQ}}(i), u_i^{\text{serve}} - 1] \times [t_i - 5\rho^{\text{SQ}}(i), t_i]$ are not in* $\text{SQUARE}(i)$.

Proof: Assume by the way of contradiction that some replica $q = (w, t) \in [v_i - 5\rho^{\text{SQ}}(i), u_i^{\text{serve}} - 1] \times [t_i - 5\rho^{\text{SQ}}(i), t_i]$ is in $\text{SQUARE}(i)$. This implies that $v_i - 5\rho^{\text{SQ}}(i) \leq w < u_i^{\text{serve}}$, contradicting the choice (in step SQ3) of node u_i^{serve} of the serving replica $q_i^{\text{serve}} = (u_i^{\text{serve}}, s_i^{\text{serve}})$ as the leftmost node over all replicas that are in the solution and in $\mathcal{S}[r_i, 5 \cdot \rho^{\text{SQ}}(i)]$. ∎

Observation 3.12 *Consider some request r_i. Assume that its closest replica q_i^{close}) is added to* SQUARE*'s solution when handling request r_j (for some $j < i$). Then, $s_i^{\text{close}} \geq t_j$ (the time of the i'th closest replica is not before the time t_j of r_j).*

Proof: The replica $q_i^{\text{close}} = (u_i^{\text{close}}, s_i^{\text{close}})$ is added to the solution in step SQ4 or in step SQ5 while SQUARE is handling request r_j. If q_i^{close} is added to the solution in step SQ4, then the replica $(u_i^{\text{close}}, t_j)$ is added to the solution in that step too; otherwise, q_i^{close} is added in step SQ5, and then a replica of u_i^{close} at time t (for some time $t > t_j$) is added to the solution. This implies that $s_i^{\text{close}} \geq t_j$, see step SQ2 for the selection of q_i^{close}. ∎

Observation 3.13 *If there exists a replica (w, t_i) in the solution of* $\text{SQUARE}(i-1)$ *such that $0 \leq v_i - w \leq 5\rho^{\text{SQ}}(i)$, then r_i is a covered request.*

Proof: By the definition, $\rho^{\text{SQ}}(i) \leq dist_{\infty}^{\rightarrow}((w, t_i), r_i)$, since the distance from w to v_i is a candidate for $\rho^{\text{SQ}}(i)$. The observation now follows from the definition of a covered request. ∎

3.2.1 Parent ball in tree larger then its child

As promised (in the overview), Property (P1) of Lemma 3.14 below implies that a parent ball in tree is at least four times larger than its "last child". In fact, the lemma is more general (Property (P2) is used in the proof of other lemmas)[6].

Lemma 3.14 *Consider two* **uncovered** *requests r_i and r_j such that $i > j$. If Q-BALL$^{\mathrm{SQ}}(i)$ and Q-BALL$^{\mathrm{SQ}}(j)$ are not edges disjoint, then the following properties hold.*
(P1) $\rho^{\mathrm{SQ}}(i) \geq 4 \cdot \rho^{\mathrm{SQ}}(j)$; and
(P2) $v_j - \rho^{\mathrm{SQ}}(j) \leq v_i < u_j^{\mathrm{serve}} \leq v_j$ (the leftmost replicas of Q-BALL$^{\mathrm{SQ}}(j)$ are on left of r_i; r_i is on the left of r_j and even on the left of the j'th serving replica).

Proof: We first prove Property (P2). Since Q-BALL$^{\mathrm{SQ}}(i)$ and Q-BALL$^{\mathrm{SQ}}(j)$ are **not** edges disjoint,

$$v_j - \rho^{\mathrm{SQ}}(j) < v_i \quad \text{and} \quad v_i - \rho^{\mathrm{SQ}}(i) < v_j \ . \tag{2}$$

Since also $i > j$ (see Figure 8),

$$\rho^{\mathrm{SQ}}(i) = t_i - s_i^{\mathrm{close}} > t_i - t_j, \tag{3}$$

where the equality below follows from Observation 3.10, since r_i is uncovered; and the inequality holds since, on the one hand, Q-BALL$^{\mathrm{SQ}}(i)$ and Q-BALL$^{\mathrm{SQ}}(j)$ are not edge disjoint, hence has a common edge; on the other hand, (1) for every edge in Q-BALL$^{\mathrm{SQ}}(i)$, at least one of its corresponding replicas corresponds to time strictly grater than $t_i - \rho^{\mathrm{SQ}}(i)$; however, non of the edges of Q-BALL$^{\mathrm{SQ}}(j)$ corresponding to replicas of time strictly grater than t_j .

The left inequality of Property (P2) holds by the left inequality of (2). The right inequality of Property (P2) holds trivially, see step SQ3. Assume by the way of contradiction that the remaining inequality does not holds, i.e., $v_i \geq u_j^{\mathrm{serve}}$. Consider two cases.

Case 1. $u_j^{\mathrm{serve}} \leq v_i \leq v_j$, see Figure 9. Then, (v_i, t_j) is in SQUARE(j) (SQUARE's solution after handling request r_j). Thus, $\rho^{\mathrm{SQ}}(i) \leq dist_{\infty}^{\rightarrow}((v_i, t_j), r_i = (v_i, t_i)) = t_i - t_j$, contradicting Inequality (3).

Case 2. $v_j \leq v_i$, see Figure 10. In this case, $\rho^{\mathrm{SQ}}(i) \leq dist_{\infty}^{\rightarrow}(r_j, r_i) = \max\{v_i - v_j \ , \ t_i - t_j\}$. By the second inequality of (2), $v_i - v_j < \rho^{\mathrm{SQ}}(i)$. Hence, $\rho^{\mathrm{SQ}}(i) \leq t_i - t_j$. Again, this contradicts Inequality (3).

These two cases shows that Property (P2) holds. We next show that Property (P1) holds too. For that, consider two cases.

Case A: $u_i^{\mathrm{close}} < v_j - 5\rho^{\mathrm{SQ}}(j)$. In other words, the closest replica $q_i^{\mathrm{close}} = (u_i^{\mathrm{close}}, s_i^{\mathrm{close}})$ to r_i is on the left of $\mathcal{S}[r_j, 5\rho^{\mathrm{SQ}}(j)]$ (see Figure 11). Recall that the closest replica q_i^{close} defines the radius $\rho^{\mathrm{SQ}}(i)$ (see SQ2), i.e., $\rho^{\mathrm{SQ}}(i) = \max\{v_i - u_i^{\mathrm{close}}, t_i - s_i^{\mathrm{close}}\}$. We have (the second inequality bellow follows by substituting v_i using the first inequality of (2)),

$$\rho^{\mathrm{SQ}}(i) \geq v_i - u_i^{\mathrm{close}} \geq (v_j - \rho^{\mathrm{SQ}}(j)) - (v_j - 5\rho^{\mathrm{SQ}}(j)) = 4\rho^{\mathrm{SQ}}(j)$$

[6] Actually, this lemma shows that property for any other child too, but for the other children this is not helpful, since there may be too many of them.

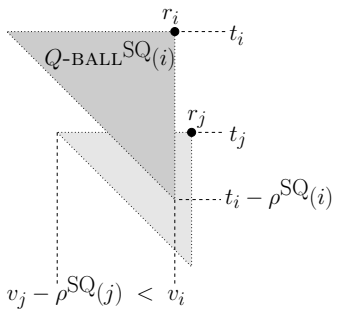

Figure 8: Q-BALL$^{\mathrm{SQ}}(i)$ and Q-BALL$^{\mathrm{SQ}}(j)$ are *not* edges disjoint implying inequalities (2) and (3).

as needed for Property (P1). (Intuitively, since Q-BALL$^{\mathrm{SQ}}(i)$ intersect Q-BALL$^{\mathrm{SQ}}(j)$ on the left, v_i's is at most $\rho^{\mathrm{SQ}}(j)$ left of v_j; however, we assumed that u_i^{close} is at least $5\rho^{\mathrm{SQ}}(j)$ left of v_j; hence, $\rho^{\mathrm{SQ}}(i) \geq 4\rho^{\mathrm{SQ}}(j)$.)

Case B: $v_j - 5\rho^{\mathrm{SQ}}(j) \leq u_i^{\mathrm{close}}$. We have that (see Figure 12),

$$v_j - 5\rho^{\mathrm{SQ}}(j) \leq u_i^{\mathrm{close}} < u_j^{\mathrm{serve}} \ , \tag{4}$$

where the inequality on the right holds by Property (P2) of this lemma. Assume that q_i^{close} is added to the solution when SQUARE handles some request r_k (for some $k < i$). By Observation 3.12, $s_i^{\mathrm{close}} \geq t_k$. If $k \geq j$, then $t_k \geq t_j$, which means that $s_i^{\mathrm{close}} \geq t_j$ contradicting Inequality (3). Thus $k < j$. Therefore, by Observation 3.11, $q_i^{\mathrm{close}} \notin [v_j - 5\rho^{\mathrm{SQ}}(j), u_j^{\mathrm{serve}} - 1] \times [t_j - 5\rho^{\mathrm{SQ}}(j), t_j]$. However, by Inequality (4), $u_i^{\mathrm{serve}} \in [v_j - 5\rho^{\mathrm{SQ}}(j), u_j^{\mathrm{serve}} - 1]$. Also by Inequality (3), $s_i^{\mathrm{close}} < t_j$. Hence, $s_i^{\mathrm{close}} < t_j - 5\rho^{\mathrm{SQ}}(j)$, implying $\rho^{\mathrm{SQ}}(i) > 5\rho^{\mathrm{SQ}}(j)$. Property (P1) follows.

As showed above Property (P1) and Property (P2) hold, the lemma follows too. ∎

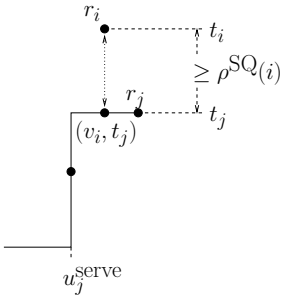

Figure 9: r_i is on the right of q_j^{serve} and on the left of r_j (case 1, $u_j^{\mathsf{serve}} \leq v_i \leq v_j$).

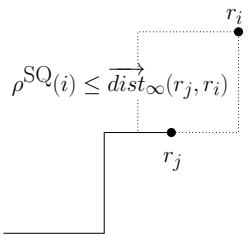

Figure 10: r_i is on the right of r_j (case 2, $v_j \leq v_i$).

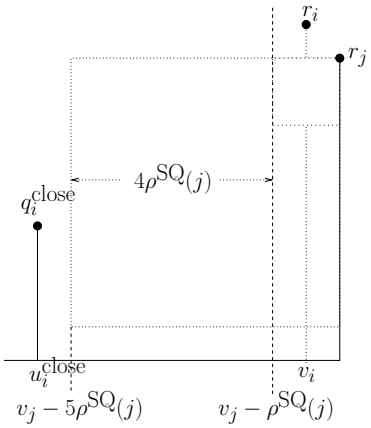

Figure 11: q_i^{close} is on the left of $\mathcal{S}[r_j, 5\rho^{\mathsf{SQ}}(j)]$.

3.2.2 Uncovered request has at least two children

The previous lemma suffices for the case that an uncovered request has only one child. We now consider the case where an uncovered request has at least two children. We first establish Lemma 3.8 (which state in the proof overview) that deals with the case that the quarter ball of request r_j is later than the tail of some previous request r_ℓ (for some $\ell < j$). Before representing the proof of this lemma, let us make two "geometric" definitions. Consider two given requests r_j and r_i such that $i > j$. Intuitively, Q-BALL$^{\mathrm{SQ}}(i)$ is *later* than TAIL(j), if the time of earliest replica of Q-BALL$^{\mathrm{SQ}}(i)$ is not before the time of the last replica of TAIL(j). Formally, Q-BALL$^{\mathrm{SQ}}(i)$ is later than TAIL(j), if $t_i - \rho^{\mathrm{SQ}}(i) \geq t_j + 4\rho^{\mathrm{SQ}}(j)$. In addition, we say that TAIL(j) (which contains only replicas of u_j^{serve}) is in the *range* of Q-BALL$^{\mathrm{SQ}}(i)$ (which contains replicas of the nodes of $\{v_i - \rho^{\mathrm{SQ}}(i), ..., v_i\}$), if $v_i - \rho^{\mathrm{SQ}}(i) < u_j^{\mathrm{serve}} \leq v_i$ (in other words, $u_j^{\mathrm{serve}} \neq v_i - \rho^{\mathrm{SQ}}(i)$ and there exists a replica of u_j^{serve} in Q-BALL$^{\mathrm{SQ}}(i)$).

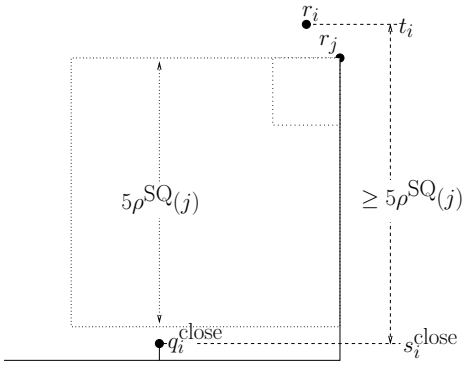

Figure 12: q_i^{close} is below $\mathcal{S}[r_j, 5\rho^{\mathsf{SQ}}(j)]$.

Before presenting the proof of Lemma 3.8, let us remaind and a bit restate this lemma (using formal notations).

Lemma 3.8. *Consider two requests r_ℓ and r_j such that $j > \ell$. Suppose that,* TAIL(ℓ) *is in the range of* Q-BALL$^{\mathrm{SQ}}(j)$*, i.e., $v_j - \rho^{\mathrm{SQ}}(j) < u_\ell^{\mathrm{serve}} \leq v_j$. Then,* Q-BALL$^{\mathrm{SQ}}(j)$ *is later than* TAIL(ℓ)*. That is, $t_j - \rho^{\mathrm{SQ}}(j) \geq t_\ell + 4\rho^{\mathrm{SQ}}(\ell)$.*

Proof of Lemma 3.8: Consider two requests r_j and r_ℓ that satisfy the conditions of the lemma. We begin by showing a slightly weaker assertion, that r_j itself is later than TAIL(ℓ). That is, $t_j > t_\ell + 4\rho^{\mathrm{SQ}}(\ell)$. Assume the contrary, that $t_j \leq t_\ell + 4\rho^{\mathrm{SQ}}(\ell)$. Note that the replicas of TAIL(ℓ) of time no later than t_j (if such do exists) are "candidates" for the closest and the serving replicas of the j'th request (since they belong to the solution SQUARE$(j-1)$). Thus,

$$\rho^{\mathrm{SQ}}(j) \leq dist_\infty^{\rightarrow}((u_\ell^{\mathrm{serve}}, t_j), r_j) = v_j - u_\ell^{\mathrm{serve}} \ .$$

That is, the inequality holds since $(u_\ell^{\mathrm{serve}}, t_j) \in$ SQUARE$(j-1)$ (see step SQ2); the equality holds since $r_j = (v_j, t_j)$ and $u_\ell^{\mathrm{serve}} \leq v_j$. This means that the complete j'th quarter-ball is on the right of the ℓ'th serving replica q_ℓ^{serve}, i.e.,

$$u_\ell^{\mathrm{serve}} \leq v_j - \rho^{\mathrm{SQ}}(j).$$

This contradicts the condition of the lemma, hence $t_j > t_\ell + 4\rho^{\mathrm{SQ}}(\ell)$ as promised.

We now prove the lemma's assertion that $t_j - \rho^{\mathrm{SQ}}(j) \geq t_\ell + 4\rho^{\mathrm{SQ}}(\ell)$. Denote by q_ℓ^{last} the latest replica in TAIL(ℓ), i.e., $q_\ell^{\mathrm{last}} = (u_\ell^{\mathrm{serve}}, t_\ell + 4\rho^{\mathrm{SQ}}(\ell))$. Note that q^{last} is a candidate for the closest replica of the j'th request, since $q_\ell^{\mathrm{last}} \in$ SQUARE$(j-1)$ and the time of q^{last} is earlier than the time of r_j (i.e., $t_j > t_\ell + 4\rho^{\mathrm{SQ}}(\ell)$). Thus, the radius $\rho^{\mathrm{SQ}}(j)$ of the j'th request is at most as the distance between q_ℓ^{last} to r_j, see step SQ2. That is,

$$\rho^{\mathrm{SQ}}(j) \leq dist_\infty^{\rightarrow}(q_\ell^{\mathrm{last}} \ , \ r_j). \tag{5}$$

In addition, by the condition of the lemma,

$$v_j - u_\ell^{\mathrm{serve}} < \rho^{\mathrm{SQ}}(j). \tag{6}$$

Thus, by Inequalities (5) and (6)

$$v_j - u_\ell^{\mathrm{serve}} < dist_\infty^{\rightarrow}(q_\ell^{\mathrm{last}} \ , \ r_j). \tag{7}$$

Recall that, $dist^{\rightarrow}_{\infty}(q^{\text{last}}_{\ell}\ ,\ r_j) = \max\{v_j - u^{\text{serve}}_{\ell}\ ,\ t_j - (t_\ell + 4\rho^{\text{SQ}}(\ell))\}$. Hence, by Ineq. (7),

$$dist^{\rightarrow}_{\infty}(q^{\text{last}}_{\ell}\ ,\ r_j) = t_j - (t_\ell + 4\rho^{\text{SQ}}(\ell)).$$

Combining this with Ineq. (5), we get also that $\rho^{\text{SQ}}(j) \leq t_j - (t_\ell + 4\rho^{\text{SQ}}(\ell))$. The Lemma follows. ∎

Now, we are ready to show the main lemma (Lemma 3.7), which intuitively, shows that "a lot of time" (proportional to the request's radius) passes between the time the one child ball ends and the time the next child ball starts.

We begins by remaining this lemma and restate it a bit (using formal notations).

Lemma 3.7. *Consider some uncovered request r_i such that $|\text{CHILDREN}(i)| \geq 2$. Let $j, k \in \text{CHILDREN}(i)$ such that $k < j$. Then, Q-$\text{BALL}^{\text{SQ}}(j)$ is later than $\text{TAIL}(k)$. That is, $t_j - \rho^{\text{SQ}}(j) \geq t_k + 4\rho^{\text{SQ}}(k)$.*

Proof of Lemma 3.7: We consider two cases regarding the relation between the serving replica u^{serve}_k of the k'th request and the node v_j of the j'th replica.

Case 1: $u^{\text{serve}}_k \leq v_j$. This is the simpler case. Apply Lemma 3.8 with the requests j and $\ell = k$. First, note that $j > k$ as required to apply Lemma 3.8. To use this lemma, it is also required to show that

$$v_j - \rho^{\text{SQ}}(j) < u^{\text{serve}}_k \leq v_j. \tag{8}$$

The right inequality holds by the assumption of this case. The left inequality holds since

$$v_j - \rho^{\text{SQ}}(j) < v_i \leq u^{\text{serve}}_k,$$

where the first inequality holds by Lemma 3.14 Property (P2) with i and j; the second inequality holds by Lemma 3.14 Property (P2) with i and k. Thus, in this case, the lemma follows by Lemma 3.8.

Case 2: $v_j < u^{\text{serve}}_k$ (that is, v_j is on the left of the k'th serving replica u^{serve}_k). Note that, unlike the previous case, $\text{TAIL}(k)$ is *not* in the range of Q-$\text{BALL}^{\text{SQ}}(j)$. Thus, the condition of Lemma 3.8 does not holds, and we cannot apply Lemma 3.8 with j and $\ell = k$. Fortunately, we show that in this case, we can use another request for which Lemma 3.8 can be applied. That is, we claim that in this case, there exists a request r_ℓ that has the following three properties.

(P1) $k < \ell < j$;

(P2) $\text{TAIL}(\ell)$ is in the range of Q-$\text{BALL}^{\text{SQ}}(j)$ (it satisfies the condition of Lemma 3.8); and

(P3) $\rho^{\text{SQ}}(\ell) \geq \rho^{\text{SQ}}(k)$.

Note that if indeed such a request r_ℓ (that has the above three properties) does exists, then applying Lemma 3.8, we will get that

$$t_j - \rho^{\text{SQ}}(j) \geq t_\ell + 4\rho^{\text{SQ}}(\ell) \geq t_k + 4\rho^{\text{SQ}}(k).$$

The last inequality follows from Property (P3) and since $t_j \geq t_\ell$ (since $j > \ell$). This will imply the lemma. It is left to show that such a request r_ℓ **must** exist. Let

$$\text{REC} = [v_i, v_j] \times [t_k - 4\rho^{\text{SQ}}(k), t_i],$$

and let

$$\ell^* = \min_l \textsc{Square}(l) \cap \textsc{rec} \neq \emptyset,$$

the index of the first request in which the solution $\textsc{Square}(\ell^*)$ contains some replicas in REC. Note that REC is well defined since $v_i > v_j$ by Lemma 3.14, Property (P2). We completes the proof by showing that ℓ^* exists and has properties (P1)-(P3). Hence, we can choose $r_\ell = r_{\ell^*}$ and the lemma will follow.

1. ℓ^* **has Property (P1), i.e.,** $k < \ell^* < j$**.**
   We first show that $\textsc{Square}(k)$ does not contain any replica from the rectangle graph REC. That is,

$$\textsc{Square}(k) \cap \textsc{rec} = \emptyset. \tag{9}$$

   Then, we show that $\textsc{Square}(j-1)$ does contain some replicas from the rectangle graph REC. That is,

$$\textsc{Square}(j-1) \cap \textsc{rec} \neq \emptyset. \tag{10}$$

   Once we prove the above two inequalities, they will imply that ℓ^* does exist, and in particular, $k < \ell^* < j$ as needed.

   Proving Ineq (9): Note that when Algorithm SQUARE handles r_k, it does not add any replica in the above rectangle, since it only adds replicas on the right hand side of u_k^{serve}. (Recall that, $v_i < v_j$ and we are now analysing case (2) where $v_j < u_k^{\text{serve}}$, i.e., $[v_i, v_j] \subseteq [v_i, u_k^{\text{serve}} - 1]$.)

   It is left to prove that $\textsc{Square}(k-1)$ does not include a replica in REC. By Observation 3.11, it follows that $\textsc{Square}(k)$ and $\textsc{Square}(k-1)$ do not contain any replica from the "bottom part" of REC, since

$$\{(v,t) \in \textsc{rec} \mid t \leq t_k\} \subseteq [v_k - 4\rho^{\text{SQ}}(k), u_k^{\text{serve}} - 1] \times [t_k - 5\rho^{\text{SQ}}(k), t_k],$$

   where the inequality holds since $v_k - 4\rho^{\text{SQ}}(k) \leq v_i$ (by Lemma 3.14, Property (P2)); and $v_j < u_k^{\text{serve}}$ (the assumption of case (2)).
   It is left to prove that $\textsc{Square}(k-1)$ does not contain any replica from the "top part" of REC.

   Assume by the way of contradiction that there exists a replica $q = (u,s) \in \textsc{rec} \cap \textsc{Square}(k-1)$ such that $s > t_k$. Let r_l be the request in which SQUARE added q to the solution (that is, when SQUARE was handling r_l, it added q to the solution). The assumption that $q \in \textsc{Square}(k-1)$ implies that such a request r_l does exist, and in particular, $l \leq k-1$. Thus, $t_k \geq t_l$, and hence, $s > t_l$. This implies that q is added to the solution in step SQ5 and $q \in \textsc{tail}(l) = \mathcal{P}_{\mathcal{A}}[(u_l^{\text{serve}}, t_l), (u_l^{\text{serve}}, t_l + 4 \cdot \rho^{\text{SQ}}(l))]$. Therefore, also, $(u_l^{\text{serve}}, t_k) \in \textsc{tail}(l)$ (since $t_l \leq t_k$ and $t_k \leq s \leq t_l + 4 \cdot \rho^{\text{SQ}}(l)$), and in particular,

$$(u_l^{\text{serve}}, t_k) \in \textsc{Square}(k-1).$$

In addition, $u_l^{\text{serve}} \in [v_i, v_j]$, since $q \in \text{REC}$, and also

$$v_k - \rho^{\text{SQ}}(k) \leq v_i \leq u_l^{\text{serve}} \leq v_j < u_k^{\text{serve}} \leq v_k \ ,$$

where the first and the last inequalities hold by Lemma 3.14 Property (P2) with i and k; the second and the third inequalities hold since $u_l^{\text{serve}} \in [v_i, v_j]$; and the fourth inequality holds by the assumption of case (2).

Therefore, in particular, $0 \leq v_k - u_l^{\text{serve}} \leq \rho^{\text{SQ}}(k)$. Thus, by Observation 3.13, r_k is a covered request, contradicting the assumption that k is child of i (covered requests have no parents). Therefore, $\text{SQUARE}(k-1) \cap \text{REC} = \emptyset$ and (as mentioned) also $\text{SQUARE}(k) \cap \text{REC} = \emptyset$. Hence, Ineq. (9) holds.

Proving Ineq. (10): Recall that the j'th closest replica $q_j^{\text{close}} = (u_j^{\text{close}}, s_j^{\text{close}}) \in \text{SQUARE}(j-1)$, see step SQ2. Thus, to show that Ineq. (10) holds, it is sufficient to show that $q_j^{\text{close}} \in \text{REC}$.

The assumption that $Q\text{-BALL}^{\text{SQ}}(j)$ and $Q\text{-BALL}^{\text{SQ}}(i)$ are not edge disjoint implies that the j'th serving and closest replicas are on the right of r_i. That is,

$$v_j - \rho^{\text{SQ}}(j) < v_i < u_j^{\text{serve}} \leq u_j^{\text{close}} \leq v_j < u_k^{\text{serve}},$$

where the first and the second inequalities hold by Lemma 3.14, Property (P2); the third and the forth inequalities hold by steps SQ2 and SQ3; and the fifth inequality is the assumption in the current case (2).

This implies, in particular, that

$$v_j - \rho^{\text{SQ}}(j) < v_i < u_j^{\text{close}} \leq v_j \ . \tag{11}$$

In addition, by Observation 3.10, the radius of an uncovered request is the *time* difference between the request and its closest replica, that is, $\rho^{\text{SQ}}(j) = t_j - s_j^{\text{close}}$, and equivalently

$$s_j^{\text{close}} = t_j - \rho^{\text{SQ}}(j). \tag{12}$$

Recall that k and j are children of i, thus $Q\text{-BALL}^{\text{SQ}}(j)$ and $Q\text{-BALL}^{\text{SQ}}(k)$ are edges disjoint. This, together with inequalities (11) and (12) imply that

$$s_j^{\text{close}} \geq t_k \ . \tag{13}$$

Hence, $q_j^{\text{close}} \in \text{REC}$ by inequalities (11) and (13) and since $s_j^{\text{close}} \leq t_j \leq t_i$. Thus, Ineq. (10) holds as promised.

We have shown that inequalities (9) and (10) hold as we argued above this implies that r_{ℓ^*} has Property (P1).

2. ℓ^* **has Property (P2), i.e.,** $\text{TAIL}(\ell^*)$ **is in the range of** $Q\text{-BALL}^{\text{SQ}}(j)$**.** Recall that $i > j$; and $Q\text{-BALL}^{\text{SQ}}(i)$ and $Q\text{-BALL}^{\text{SQ}}(j)$ are not edge disjoint, thus by Lemma 3.14, Part 2,

$$v_j - \rho^{\text{SQ}}(j) < v_i \ < v_j \ . \tag{14}$$

We show that

$$v_i \ < u_{\ell^*}^{\text{serve}} \leq v_j \ , \tag{15}$$

which implies together with Ineq. (14) that $v_j - \rho^{\text{SQ}}(j) < u_{\ell^*}^{\text{serve}} \ < v_j$ as needed (for showing that $\text{TAIL}(\ell^*)$ is in the range of $Q\text{-BALL}^{\text{SQ}}(j)$).

It remains to show that Ineq. (15) holds. Note that, on the one hand, the choice of r_{ℓ^*} (as the first request which the solution $\text{SQUARE}(\ell^*)$ contains a replica in REC) implies that some replica $q' = (u', t') \in \text{REC}$ is added to the solution when SQUARE handles r_{ℓ^*}. On the other hands, when Algorithm SQUARE handles r_{ℓ^*}, it only adds replicas (in steps SQ4 and SQ5) to the right of $u_{\ell^*}^{\text{serve}}$ and to the left of v_{ℓ^*}. Thus, on the one hand, $v_i \leq u' \leq v_j$, and on the other hand, $u_{\ell^*}^{\text{serve}} \ \leq u' \leq v_{\ell^*}$. Hence, also

$$v_i \ \leq v_{\ell^*} \quad \text{and} \quad u_{\ell^*}^{\text{serve}} \leq v_j \ . \tag{16}$$

This already establish the right inequality of (15). To show that its left inequality holds too, assume toward contradiction that $u_{\ell^*}^{\text{serve}} \leq v_i$. Combining this with the left inequality of (16), we have

$$u_{\ell^*}^{\text{serve}} \leq v_i \leq v_{\ell^*} \ .$$

This implies that, when Algorithm SQUARE handles r_{ℓ^*} , it added the replica (v_i, t_{ℓ^*}), in step SQ4 to the solution. Hence, $(v_i, t_{\ell^*}) \in \text{SQUARE}(\ell^*)$, and is a candidates for the i'th close replica (see step SQ2). Thus,

$$\rho^{\text{SQ}}(i) \leq dist_{\infty}^{\rightarrow}((v_i, t_{\ell^*}, r_i)) = t_i - t_{\ell^*} \ .$$

Hence, the time of each of $Q\text{-BALL}^{\text{SQ}}(i)$'s replicas is at least t_{ℓ^*}. Recall that, $t_{\ell^*} \geq t_k$ (since $\ell^* > k$); and that in each edge e of $Q\text{-BALL}^{\text{SQ}}(i)$ at least one of e's endpoints is corresponds to time later than $v_i - \rho^{\text{SQ}}(i)$. Therefore, $Q\text{-BALL}^{\text{SQ}}(i)$ and $Q\text{-BALL}^{\text{SQ}}(k)$ are edge disjoint, which contradicts the choice of k as a child of i. Hence, $v_i < u_{\ell^*}^{\text{serve}}$, Ineq. (15) holds and ℓ^* maintains Property (P2) as promised.

3. **ℓ^* has Property (P3), i.e., $\rho^{\mathbf{SQ}}(\ell^*) \geq \rho^{\mathbf{SQ}}(k)$.**

We first show that the time $s_{\ell^*}^{\text{serve}}$ of the serving replica $q_{\ell^*}^{\text{serve}}$ of the ℓ^*'th request is before $t_k - 5\rho^{\text{SQ}}(k)$. That is,

$$s_{\ell^*}^{\text{serve}} \leq t_k - 5\rho^{\text{SQ}}(k). \tag{17}$$

The choice of ℓ^* implies that $\text{SQUARE}(\ell^* - 1) \cap \text{REC} = \emptyset$. On the other hand, the serving replica $q_{\ell^*}^{\text{serve}}$ does belong to $\text{SQUARE}(\ell^* - 1)$ (see step SQ3). This implies, in particular, that

$$q_{\ell^*}^{\text{serve}} = (u_{\ell^*}^{\text{serve}}, s_{\ell^*}^{\text{serve}}) \notin \text{REC} \ = \ [v_i, v_j] \times [t_k - 5\rho^{\text{SQ}}(k), t_i] \ .$$

Recall that $u_{\ell^*}^{\text{serve}} \in [v_i, v_j]$ by Ineq. (15), hence $s_{\ell^*}^{\text{serve}} \notin [t_k - 5\rho^{\text{SQ}}(k), t_i]$. Inequality (17) holds, since $s_{\ell^*}^{\text{serve}} \leq t_{\ell^*} \leq t_i$.

Summarizing what we know so far, $t_{\ell^*} \geq t_k$ and $s_{\ell^*}^{\text{serve}} \leq t_k - 5\rho^{\text{SQ}}(k)$. Thus, on one hand,

$$dist_{\infty}^{\rightarrow}(q_{\ell^*}^{\text{serve}}, r_{\ell^*}) \geq t_{\ell^*} - s_{\ell^*}^{\text{serve}} \geq 5\rho^{\text{SQ}}(k). \tag{18}$$

On the other hand, $q_{\ell^*}^{\text{serve}} \in \mathcal{S}[r_{\ell^*}, 5 \cdot \rho^{\text{SQ}}(\ell^*)]$ (see step SQ3), which implies that

$$5\rho^{\text{SQ}}(\ell^*) \geq dist_{\infty}^{\rightarrow}(q_{\ell^*}^{\text{serve}}, r_{\ell^*}) \ . \tag{19}$$

Inequalities (18-19), imply that $\rho^{\text{SQ}}(\ell^*) \geq \rho^{\text{SQ}}(k)$ as needed. Hence, ℓ^* maintains Property (P3).

We have shown that r_{ℓ^*} maintains the three properties, implying the lemma for case (2) too. [Lemma 3.7] ∎

The previous lemma shows that a lot of time pass between the time of the last replica in the quarter ball of a child and the time of first replica in the quarter ball of the next child. The next, lemma use this property to show that the radius of a root is at least half of the sum of the radii of its children in its tree.

Lemma 3.15 *Consider some root request $r_{i^*} \in$ ROOTS. Then,*

$$2\rho^{\text{SQ}}(i^*) \geq \sum_{i \in \text{TREE}(i^*)} \rho^{\text{SQ}}(i).$$

Proof: We begin by showing that the radius of each ball Q-BALL$^{\text{SQ}}(i)$ in the tree is at least, twice the sum of the radii of its children. Consider some non leaf request $r_i \in$ TREE(i^*) (that is, CHILDREN$(i) \neq \emptyset$). Let us first, show that

$$\rho^{\text{SQ}}(i) \geq 2 \sum_{j \in \text{CHILDREN}(i)} \rho^{\text{SQ}}(j). \tag{20}$$

If CHILDREN$(i) = \{j\}$ (i has exactly one child), then (20) follows from property (P1) of Lemma 3.14. Otherwise, CHILDREN$(i) = \{j_1, j_2 ..., j_\nu\}$, where $\nu \geq 2$ and $j_1 \leq j_2 \leq ... \leq j_\nu$. (For simplicity, to avoid double subscripts, we may write $t(l)$ instead of t_l.) By Lemma 3.7 with $k = j_l$ and $j = j_{l+1}$, it follows that

$$t(j_l) + 4\rho^{\text{SQ}}(j_l) + \rho^{\text{SQ}}(j_{l+1}) \leq t(j_{l+1}), \tag{21}$$

for every $l = 1, ..., \nu - 1$. Now, (see Figure 13)

$$\rho^{\text{SQ}}(i) \geq t_i - t(j_1) \geq 4 \sum_{l=1}^{\nu-1} \rho^{\text{SQ}}(j_l), \tag{22}$$

where the first inequality holds since the Q-BALL$^{\text{SQ}}(i)$ and Q-BALL$^{\text{SQ}}(j_1)$ are *not* edges disjoint; the second inequality holds by Inequality (21), since $t_i \geq t(j_\nu)$. In addition, by Property (P1) of Lemma 3.14,

$$\rho^{\text{SQ}}(i) \geq 4\rho^{\text{SQ}}(j_\nu), \tag{23}$$

which implies Inequality (20) that implies the lemma. ∎

So far, we have shown that (1) the quarter-ball of the covered requests are edges disjoint; (2) the quarter-ball of the root requests are edges disjoint, and hence by Observation 3.1 and Observation 3.2, the sum of their radii of the covered request and the root requests is no more than 28 times

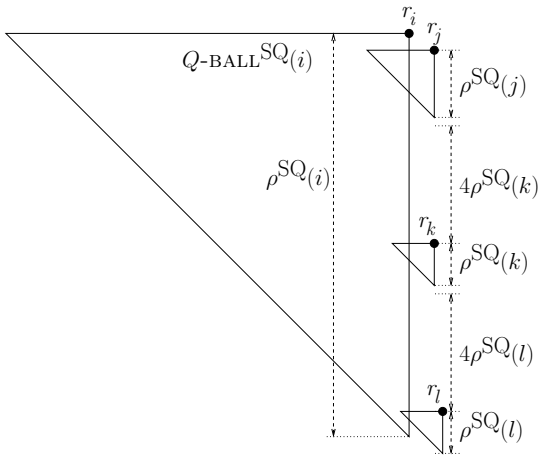

Figure 13: Geometric vision on a parent and its children relationships.

the cost of OPT. On the other hand, the sum of root's radii is at least half of the sum of the radii of the uncovered requests. This, in fact, establishes Theorem 3.9.

Proof of Theorem 3.9: The ratio for covered request follows Inequality (1). For uncovered requests it follows from Observation 3.5 and Observation 3.2 that $|\mathrm{OPT}| \geq \sum_{i \in \mathrm{ROOTS}} \rho^{\mathrm{SQ}}(i)$. Combining this with Lemma 3.15, we have, $2|\mathrm{OPT}| \geq \sum_{i \in \mathrm{UNCOVER}} \rho^{\mathrm{SQ}}(i)$. Thus, also, $3|\mathrm{OPT}| \geq \sum_{r_i \in \mathcal{R}} \rho^{\mathrm{SQ}}(i)$. The Theorem follows from Observation 3.1. ∎

4 Algorithm D-LINE$^{\mathrm{on}}$ - the "real" online algorithm

In this section, we transform the pseudo online algorithm SQUARE of Section 3 into a (fully) online algorithm D-LINE$^{\mathrm{on}}$ for DMCD[7]. Let us first give some intuition here.

The reason Algorithm SQUARE is *not* online, is one of the the actions it takes at step SQ4. There, it stores a copy at the serving replica u_i^{serve} for request r_i from time s_i^{serve} to time t_i. This requires "going back in time" in the case that the time $s_i^{\mathrm{serve}} < t_i$. A (full) online algorithm cannot perform such an action. Intuitively, Algorithm D-LINE$^{\mathrm{on}}$ "simulates" the impossible action by (1) storing additional copies (beyond those stored by SQUARE); and (2) shifting the delivery to request r_i (step SQ4 of SQUARE) from an early time to time t_i of r_i. It may happen that the serving node u_i^{serve} of r_i does not have a copy (in SQUARE) at t_i. In that case, Algorithm D-LINE$^{\mathrm{on}}$ also (3) delivers first a copy to $(u_i^{\mathrm{serve}}, t_i)$ from some node w on the left of u_i^{serve}. Simulation step (1) above (that we term the storage phase) is the one responsible for ensuring that such a node w exists, and is "not too far" from u_i^{serve}.

For the storage phase, Algorithm D-LINE$^{\mathrm{on}}$ covers the network by "intervals" of various lengthes (pathes that are subgraphs of the network graph). There are overlaps in this cover, so that each node is covered by intervals of various lengthes. Let the length of some interval I be $length(I)$. Intuitively, given an interval I and a time t, if SQUARE kept a copy in a node of interval I "recently"

[7] We comment that it bears similarities to the transformation of the pseudo online algorithm Triangle to a (full) online algorithm for *undirected* MCD in [15]. The transformation here is harder, since there the algorithm sometimes delivered a copy to a node v from some node on v's right, which we had to avoid here (since the network is directed to the right).

("recent" is proportional to $length(I)$), then D-LINE$^{\text{on}}$ makes sure that a copy is kept at the left most node of this interval, or "nearby" (in some node in the interval just left to I).

Now, we (formally) illustrated Algorithm D-LINE$^{\text{on}}$. We begins by giving some definitions.

Partitions of $[1,n]$ into intervals Consider some positive integer δ to be chosen later. For convenience, we assume that n is a power of δ. (It is trivial to generalize it to other values of n.) Define $\log_\delta n + 1$ *levels* of partitions of the interval $[1,n]$. In level l, partition $[1,n]$ into n/δ^l intervals, $I\langle\delta\rangle^l_1$, $I\langle\delta\rangle^l_2$,...,$I\langle\delta\rangle^l_{n/\delta^l}$, each of size δ^l. That is, $I\langle\delta\rangle^l_j = \{(j-1)\cdot\delta^l + k \mid k = 1,...,\delta^l\}$, for every $1 \le j \le n/\delta^l$ and every $0 \le l \le \log_\delta n$. Let $\mathcal{I}\langle\delta\rangle$ be the set of all such intervals. When it is clear from the context, we may omit $\langle\delta\rangle$ from $\mathcal{I}\langle\delta\rangle$ and $I\langle\delta\rangle^l_j$ and write $\mathcal{I}$ and I^l_j, respectively. Let $\ell(I)$ be the *level* of an interval $I \in \mathcal{I}$, i.e., $\ell(I^l_j) = l$. For a given interval $I^l_j \in \mathcal{I}$, denote by $\overrightarrow{N}^L(I^l_j)$, for $1 < j \le n/\delta^l$ the *neighbor* interval of level l that is on the left of I^l_j. That is, $\overrightarrow{N}^L(I^l_j) = I^l_{j-1}$. Define that $\overrightarrow{N}^L(I^i_1) = \{0\}$. Let $\overrightarrow{N}(I) = \overrightarrow{N}^L(I) \cup I$. We say that $\overrightarrow{N}(I)$ is the *neighborhood* of I.

Denote by $I^l(v)$ (for every node $v \in V$ and every level $l = 0,...,\log_\delta n$) the interval in level l that contains v. That is, $I^l(v) = I^l_k$, where $k = \lfloor \frac{v}{\delta^l} \rfloor + 1$. The *neighborhood* $\overrightarrow{N}^l(v)$ of a node v contains all those nodes in the neighborhood $\overrightarrow{N}(I^l(v))$ (of the interval of level l of v) that are left of v. That is, $\overrightarrow{N}^l(v) = \{u \in \overrightarrow{N}(I^l(v)) \mid u \le v\}$.

Active node Consider some node $v \in V$, some level $0 \le l \le \log_\delta n$. Node v is called $\langle l,\delta\rangle$-*active* at time t, if $(\text{BASE} \cup \text{TAIL}) \cap v[t-\delta^l, t] \neq \emptyset$. Intuitively, Algorithm SQUARE kept a movie copy in v, at least once, and "not to long" before time t. We say that v is $\langle l,\delta\rangle$-*stays-active*, intuitively, if v is **not** "just about to stop being $\langle l,\delta\rangle$-active", that is, if $(\text{BASE} \cup \text{TAIL}) \cap v[t-\delta^l+1, t] \neq \emptyset$.

Let us now construct $\mathcal{C}_{t+1}$, the set of replicas corresponding to the nodes that store copies from time t to time $t+1$ in a D-LINE$^{\text{on}}$ execution. Let $\mathcal{C}_0 = \{(v_0 = 0, 0)\}$. (The algorithm will also leave a copy in $v_0 = 0$ always.) To help us later in the analysis, we also added an auxiliary set COMMIT $\subseteq \{\langle I,t\rangle \mid I \in \mathcal{I}\langle\delta\rangle \text{ and } t \in \mathbb{N}\}$. Initially, COMMIT $\leftarrow \emptyset$. For each time $t = 0,1,2,...$, consider first the case that there exists at least one request corresponding to time t, i.e., $\mathcal{R}[t] = \{r_j,...,r_k\} \neq \emptyset$. Then, for each request $r_i \in \mathcal{R}[t]$, D-LINE$^{\text{on}}$ simulates SQUARE to find the radius $\rho^{\text{SQ}}(i)$ and the serving node u^{serve}_i of the serving replica $q^{\text{serve}}_i = (u^{\text{serve}}_i, s^{\text{serve}}_i)$ of r_i. Unfortunately, we may not be able to deliver, at time t, a copy from q^{serve}_i may be $t > s^{\text{serve}}_i$. Hence, D-LINE$^{\text{on}}$ delivers a copy to r_i via (u^{serve}_i, t) (this is called the *"delivery phase"*). That is, for each $i = j,...,k$ do:

(D1) choose a closest (to (u^{serve}_i, t)) replica $q^{\text{on}}_i = (u^{\text{on}}_i, t)$ on the left of u^{serve}_i of time $t = t_i$ already in the solution;

(D2) add the path $\mathcal{H}^{\text{on}}(i) = \mathcal{P}_{\mathcal{H}}[q^{\text{on}}_i, r_i]$ to the solution.

Let $\mathcal{V}^{\text{on}}(i) = \{r \mid (r,q) \in \mathcal{H}^{\text{on}}(i)\}$. (Note that r_j is served from $\mathcal{C}_t$, after that, the path $\mathcal{H}^{\text{on}}(j)$ is added; and r_{j+1} is served from $\mathcal{C}_t \cup \mathcal{V}^{\text{on}}(j)$, etc.)

Recall that before the delivery phase, the replicas of $\mathcal{C}_t$ have copies. It is clear, that the delivery phase of time t ensures that the replicas of $\text{BASE}[t] \cup \text{TAIL}[t]$ have copies too. That is, at the end of the delivery phase of time t, at least the replicas of $\mathcal{C}_t \cup \text{BASE}[t] \cup \text{TAIL}[t]$ have copies. It is left to decide which of the above copies to leave for time $t+1$. That is (the *"storage phase"*), D-LINE$^{\text{on}}$ chooses the set $\mathcal{C}_{t+1} \subseteq \mathcal{C}_t \cup \text{BASE}[t] \cup \text{TAIL}[t]$. Initially, $C_{t+1} \leftarrow \{(v_0, t+1)\} \cup \{(u, t+1) \mid (u,t) \in \text{TAIL}\}$ (as we choose to leave copy at the replicas of the tails and to leave a copy at v_0 always). Then, for

each level $l = 0, ..., \log_\delta n$, in an *increasing* order, the algorithm goes over and each node $v = 1, ..., n$, in an *increasing* order, selects as follows.

(S1) Choose a node v such that (1) v is level $\langle l, \delta \rangle$-stays-active at t; but (2) no replica has been selected in level l v's neighborhood ($\mathcal{C}_{t+1} \cap \overrightarrow{N}^l(v)[t+1] = \emptyset$). If such a node v does exist, then perform steps (S1.1–S1.3) below.

(S1.1) Add the tuple $\langle I^l(v), t \rangle$ to the auxiliary set COMMIT; we say that the interval $I^l(v)$ *commits* at level l at time t.

(S1.2) Select a node $u \in \overrightarrow{N}^l(v)$ such that a replica of u at time t is in $\text{BASE}[t] \cup \mathcal{C}_t$ (by Observation 4.1 below, such a replica does exist, recall that all these replicas have copies at this time).

(S1.3) Add $(u, t+1)$ to $\mathcal{C}_{t+1}$ and add the arc $((u,t),(u,t+1))$ to the solution.

The solution constructed by D-LINE$^{\text{on}}$ is denoted $\overrightarrow{\mathcal{F}}^{\text{on}} = \overrightarrow{\mathcal{H}}^{\text{on}} \cup \overrightarrow{\mathcal{A}}^{\text{on}}$, where $\overrightarrow{\mathcal{H}}^{\text{on}} = \cup_{i=1}^{N} \mathcal{H}^{\text{on}}(i)$ represents the horizontal edges added in the delivery phases and $\overrightarrow{\mathcal{A}}^{\text{on}} = \{((v,t),(v,t+1)) \mid (v,t+1) \in \mathcal{C}_{t+1} \text{ and } t = 0, ..., t_N\}$ represents the arcs added in the storage phase.

Observation 4.1 ("WELL DEFINED"). *If a node $v \in V$ is level $\langle l, \delta \rangle$-stays-active at time t, then there exists a replica $(u,t) \in \mathcal{C}_t \cup \text{BASE}[t] \cup \text{TAIL}[t]$ such that $(v,t) \in \overrightarrow{N}^l(v)$.*

Proof: Consider some node $v \in V$ and a time t. If $\langle l, \delta \rangle$-stays-active at time t, then either $(v,t) \in \text{BASE} \cup \text{TAIL}$ or $(v,t) \notin \text{BASE}[t] \cup \text{TAIL}[t]$ and v is also $\langle l, \delta \rangle$-stays-active at time $t-1$ (and $\mathcal{C}_t \cap \overrightarrow{N}^l(v)[t] \neq \emptyset$); hence, $(\text{BASE}[t] \cup \text{TAIL}[t] \cup \mathcal{C}_t) \cap \overrightarrow{N}^l(v)[t] \neq \emptyset$. The observation follows. ∎

Moreover, a stays-active node v has a copy in its neighborhood longer (for an additional round).

Observation 4.2 ("A $\langle l, \delta \rangle$-ACTIVE NODE HAS A NEAR BY COPY"). *If a node v is $\langle l, \delta \rangle$-active at time t, then, either (1) $(\text{BASE} \cup \text{TAIL}) \cap \overrightarrow{N}^l(v)[t] \neq \emptyset$, or (2) $\overrightarrow{N}^l(v)[t] \cap \mathcal{C}_t \neq \emptyset$.*

Proof: Consider a node $v \in V$ that is $\langle l, \delta \rangle$-active at time t. If $(\text{BASE} \cup \text{TAIL}) \cap \overrightarrow{N}^l(v)[t] \neq \emptyset$, then the observation follows. Assume that $(\text{BASE} \cup \text{TAIL}) \cap \overrightarrow{N}^l(v)[t] = \emptyset$. Then, the fact that v is $\langle l, \delta \rangle$-active at t, but $(v,t) \notin \text{BASE} \cup \text{TAIL}$, implies also, that v is $\langle l, \delta \rangle$-stays-active at time $t-1$. Thus, either (1) $I^l(v)$ commit at $t-1$ (at step (S1.1)) which "cause" adding an additional replica to $\mathcal{C}_t$ from $\overrightarrow{N}^l(v)$ (at step (S1.2)); or (2) $I^l(v)$ does not commit at $t-1$, since $\mathcal{C}_t$ has, already, a replica from $\overrightarrow{N}^l(v)$. ∎

Observation 4.3 ("BOUND FROM ABOVE ON $|\overrightarrow{\mathcal{A}}^{\text{on}}|$"). $|\overrightarrow{\mathcal{A}}^{\text{on}} \setminus \mathcal{P}_{\mathcal{A}}[(v_0,0),(v_0,t_N)]| \leq |\text{COMMIT}|$.

Proof: Let $\overrightarrow{\mathcal{A}}^{\text{on}}_{-v_0} = \overrightarrow{\mathcal{A}}^{\text{on}} \setminus \mathcal{P}_{\mathcal{A}}[(v_0,0),(v_0,t_N)]$. Now we prove that $|\overrightarrow{\mathcal{A}}^{\text{on}}_{-v_0}| = |\text{COMMIT}|$. Every arc in $\overrightarrow{\mathcal{A}}^{\text{on}}_{-v_0}$ (that add at step (S1.3)) corresponds to exactly one tuple $\langle I, t \rangle$ of an interval I that commits at time t (in step (S1.1)); and every interval commits at most once in each time t that corresponds to exactly one additional arc in $\mathcal{A}_{-v_0}$. Thus, $|\overrightarrow{\mathcal{A}}^{\text{on}}_{-v_0}| = |\text{COMMIT}|$. The observation follows. ∎

Analysis of D-Line$^{\text{on}}$ We, actually, compare the cost of Algorithm D-LINE$^{\text{on}}$ to that of the pseudo online Algorithm SQUARE. The desired competitive ratio for D-LINE$^{\text{on}}$ will follow, since we have shown that SQUARE approximates the optimum (Theorem 3.9). A similar usage of a (very different) pseudo online algorithm utilized in [15]. $\frac{cost(\text{D-LINE}^{\text{on}},\mathcal{R})}{cost(\text{SQUARE},\mathcal{R})} = O(\frac{\log n}{\log \log n})$. This implies

the desired competitive ratio of $O(\frac{\log n}{\log\log n})$ by Theorem 3.9. We first show, that the number of horizontal edges in $\overrightarrow{\mathcal{H}}^{\text{on}}$ (*"delivery cost"*) is $O\left(\delta \cdot cost(\text{SQUARE}, \mathcal{R})\right)$. Then, we show, that the the number of arcs in $\overrightarrow{\mathcal{A}}^{\text{on}}$ (*"storage cost"*) is $O\left(\log_\delta n \cdot cost(\text{SQUARE}, \mathcal{R})\right)$. Optimizing δ, we get a competitiveness of $O(\frac{\log n}{\log\log n})$.

Delivery cost analysis. For each request $r_i \in \mathcal{R}$, the delivery phase (step (D2)) adds $\mathcal{H}^{\text{on}}(i) = \mathcal{P}_{\mathcal{H}}[q_i^{\text{on}}, r_i]$ to the solution. Define the *online* radius of r_i as $\rho_i^{\text{on}} = d(q_i^{\text{on}}, r_i)$. We have,

$$|\overrightarrow{\mathcal{H}}^{\text{on}}| \leq \sum_{i=1}^{N} \rho_i^{\text{on}}. \tag{24}$$

It remains to bound ρ_i^{on} as a function of $\rho^{\text{SQ}}(i)$ from above. Restating Observation 4.2 somewhat differently we can use the distance $v_i - u_i^{\text{serve}} \leq 5\rho^{\text{SQ}}(i)$ (see (SQ3)) and the time difference $t_i - s_i^{\text{serve}} \leq 5\rho^{\text{SQ}}(i)$ for bounding ρ_i^{on}. That is, we show that D-LINE$^{\text{on}}$ has a copy at time t_i (of r_i) at a distance at most $10\delta\rho^{\text{SQ}}(i)$ from u_i^{serve} (of q_i^{serve} of SQUARE). Since, $v_i - u_i^{\text{serve}} \leq 5\rho^{\text{SQ}}(i)$, D-LINE$^{\text{on}}$ has a copy at distance at most $(10\delta+5)\rho^{\text{SQ}}(i)$ from v_i (of r_i).

Lemma 4.4 $\rho_i^{\text{on}} \leq (10\delta+5) \cdot \rho^{\text{SQ}}(i)$.

Proof: The following claim restating Observation 4.2 somewhat differently and help us to prove that the serving replica has a "near by" copy.

Claim 4.5 *Consider some base replica $(v,t) \in \text{BASE} \cup \text{TAIL}$ and some $\rho > 0$, such that, $t+\rho \leq t_N$. Then, there exists a replica $(w, t+\rho) \in \mathcal{C}_{t+\rho}$ such that $v - w \leq 2\delta\rho$.*

Proof: Assume that $(v,t) \in \text{BASETAIL}$. Consider an integer $\rho > 0$. Let $l = \lceil \log_\delta \rho \rceil$. Node v is $\langle l, \delta\rangle$-active at time $t+\rho$. Thus, by Observation 4.2, there exists some node $w \in \overrightarrow{N}^l(v)$ that keep a copy for time $t+\rho$. That is, a replica $(w, t+\rho) \in \overrightarrow{N}^l(v)[t+\rho] \cap \mathcal{C}_{t+\rho}$ does exists. The fact that $w \in \overrightarrow{N}^l(v)$ implies that $v - w \leq 2\delta^l$. The claim follows, since $\rho > \delta^{l-1}$. ∎

Recall that SQUARE serves request $r_i = (v_i, t_i)$ from some base replica $q_i^{\text{serve}} = (u_i^{\text{serve}}, s_i^{\text{serve}})$ already include in the solution. That q_i^{serve} may correspond to some earlier time. That is, $s_i^{\text{serve}} \leq t_i$. In the case that $s_i^{\text{serve}} = t_i$, D-LINE$^{\text{on}}$ can serve r_i from q_i^{serve}. Hence, $\rho_i^{\text{on}} \leq 5\rho^{\text{SQ}}(i)$. In the more interesting case, $s_i^{\text{serve}} < t_i$. By Claim 4.5 (substituting $v = u_i^{\text{serve}}$, $t = s_i^{\text{serve}}$, and $\rho = t_i - s_i^{\text{serve}} \leq 5\rho^{\text{SQ}}(i)$), there exists a replica $(w, t_i) \in \mathcal{C}_{t_i}$ such that $u_i^{\text{serve}} - w \leq 10\delta\rho^{\text{SQ}}(i)$. Recall that $v_i - u^{\text{serve}} \leq 5\rho^{\text{SQ}}(i)$ (see (SQ3)). Thus, $v_i - w \leq (10\delta+5)\rho^{\text{SQ}}(i)$. Hence, $\rho_i^{\text{on}} \leq (10\delta+5)\rho^{\text{SQ}}(i)$ as well. ∎

The following corollary holds, by combining together the above lemma with Inequality (24).

Corollary 4.6 $|\overrightarrow{\mathcal{H}}^{\text{on}}| \leq (10\delta+5) \cdot cost(\text{SQUARE}, \mathcal{R})$.

Analysis of the storage cost By Observation 4.3, it remains to bound the size of $|\text{COMMIT}|$ from above. Let $commit(I,t) = 1$ if $\langle I, t\rangle \in \text{COMMIT}$ (otherwise 0). Hence, $|\text{COMMIT}| = \sum_{I\in\mathcal{I}} \sum_{t=0}^{t_N} commit(I,t)$. We begin by bounding the number of commitments in D-LINE$^{\text{on}}$ made by nodes for level $l = 0$. Observation 4.7 below follows directly from the definitions of *commit* and stays-active.

Observation 4.7 $\sum_{I\in\mathcal{I}:\ell(I)=0} \sum_{t=0}^{t_N} commit(I,t) \leq \left|\mathcal{F}^{\text{SQ}}\right|$.

Proof: Consider some commitment $\langle I, t\rangle \in \text{COMMIT}$, where interval I is of level $\ell(I) = 0$. Interval I commit at time t only if there exists a node $v \in I$ such that v is $\langle l = 0, \delta\rangle$-stays-active at t

(see step (S1) in D-LINE$^{\text{on}}$). This stays-active status at time t occur only if $(v,t) \in \text{BASE} \cup \text{TAIL}$. Hence, each base replica causes at most one commitment at t of one interval of level $l = 0$. ∎

The following lemma is not really new. The main innovation of the paper is the special pseudo online algorithm we developed here. The technique for simulating the pseudo online algorithm by a "true" online one, as well as the following analysis of the simulation, are not really new. For completeness we still present a (rather detailed) proof sketch for Lemma 4.8. Its more formal analysis is deferred to the full paper (and a formal proof of a very similar lemma for very similar mapping of undirected MCD) can be found in Lemma 3.8 of [14].

Lemma 4.8 $|\text{COMMIT}| \leq (1 + 4\log_\delta n)\left|\mathcal{F}^{\text{SQ}}\right|$.

Proof sketch: The 1 term in the statement of the lemma follows from Observation 4.7 for commitments of nodes for level $l = 0$. The rest of the proof deals with commitments of nodes for level $l > 0$.

Let us group the commitments of each such interval (of level $l > 0$) into *"bins"*. Later, we shall "charge" the commitments in each bin on certain costs of the pseudo online algorithm SQUARE. Consider some level $l > 0$ interval $I \in \mathcal{I}\langle\delta\rangle$ an input $\mathcal{R}$. We say that I is a *committed-interval* if I commits at least once in the execution of D-LINE$^{\text{on}}$ on $\mathcal{R}$. For each committed-interval I (of level $\ell(I) > 0$), we define (almost) non-overlapping *"sessions"* (one session may end at the same time the next session starts; hence, two consecutive sessions may overlap on their boundaries). The first session of I does *not* contain any commitments (and is termed an *uncommitted-session*); it begins at time 0 and ends at the first time that I contains some base replica. Every other session (of I) contains at least one commitment (and is termed a *committed-session*).

Each commitment (in D-LINE$^{\text{on}}$) of I belongs to some committed session. Denote by $pivot(I)$ the leftmost node in I, i.e., $pivot(I) = \min\{v \mid v \in I\}$. Given a commitment $\langle I,t\rangle \in \text{COMMIT}$ that I makes at time t, let us identify $\langle I,t\rangle$'s session. Let $t^- < t$ be the last time (before t) there was a base replica in $pivot(I)$. Similarly, let $t^+ > t$ be the next time (after t) there will be a base replica in $pivot(I)$ (if such a time does exist; otherwise, $t^+ = \infty$). The session of commitment $\langle I,t\rangle$ starts at t^- and ends at t^+. Similarly, when talking about the i's session of interval I, we say that the session starts at $t_i^-(I)$ and ends at $t_i^+(I)$. When I is clear from the context, we may omit $(I)$and write t_i^-, t_i^+. A bin is a couple (I,i) of a committed-interval and the ith commitment-session of I. Clearly, we assigned all the commitments (of level $l > 0$ intervals) into bins.

Before proceeding, we claim that the bins indeed do not overlap (except, perhaps, on their boundaries). This is because the boundaries of the sessions are times when $pivot(I)$ has a BASE replicas. At such a times t^*, I does not commit. This is because the pivot of I is $\langle l = 0,\delta\rangle$-stays-active at t^* and hence keeps a copy. On the other hand, I is of higher level (we are dealing with the case of $l > 0$); hence, it is treated later by the algorithm (see step (S1)). Hence, I indeed does not commit at t^*. Therefore, there is no overlap between the sessions, except the ending and the starting times. That is, $t_0^- \leq t_0^+ \leq t_1^- < t_1^+ \leq, ..., \leq t_{i'}^- < t_{i'}^+$, where i' is the number of bins that I has.

Let us now point at costs of algorithm SQUARE on which we "charge" the set of commitments $\text{COMMIT}(I,i)$ in bin (I,i) for the ith session of I. We now consider only a bin (I,i) whose committed session is not the last. Note that the bin corresponds to a rectangle of $|I|$ by $t_i^+ - t_i^-$ replicas. Expand the bin by $|I|$ replicas left, if such exist. This yields the *payer* of bin (I,i); that is the payer is a rectangle subgraph of $|\overrightarrow{N}^L(I) \cup I|$ by $t_i^+ - t_i^-$ replicas. We point at specific costs SQUARE had in this payer.

Recall that every non last session of I ends with a *base* replica in $pivot(I)$, i.e., $(pivot(I), t_i^+) \in$ BASE $\cup$ TAIL. The solution of SQUARE contains a route (SQUARE route) that starts at the root and reaches $(pivot(I), t_i^+)$ by the definition of a base replica. For the charging, we use *some* (detailed below) of the edges in the intersection of that SQUARE route and the payer rectangle.

The easiest case is that the above SQUARE route enters the payer at the payer's bottom (t_i^-) and stays in the payer until t_i^+. In this case (**EB**, for Entrance from Below), each time $(t_i^- < t < t_i^+)$ there is a commitment in the bin, there is also an arc a_t in the SQUARE route (from time t to time $t+1$). We charge that commitment on that arc a_t. The remaining case (**SE**, for Side Entrance) is that the SQUARE route enters the payer from the left side of the payer. (That is, SQUARE delivers a copy to $pivot(I)$ from some other node u outside I's neighborhood, rather than stores copies at $pivot(I)$'s neighborhood from some earlier time). Therefore, the route must "cross" the left neighbor interval of I in that payer. Thus, there exists at least $|I| = \delta^{\ell(I)}$ horizontal edges in the intersection between the payer ($payer(I,i)$), of (I,i) and the SQUARE route.

Unfortunately, the number of commitments in bin (I,i) can be much grater than $\delta^{\ell(I)}$. However, consider some replica $(v,t^*) \in$ (BASE $\cup$ TAIL) $\cap\, I[t^*]$, where t^* is the last time there was a base replica in I at its i'th session. The number of commitments in bin (I,i) corresponding to the times *after* t^* is $\delta^{\ell(I)}$ at most. (To commit, an interval must have an active node; to be active, that node needs a base replica in the last $\delta^{\ell(I)}$ times.) The commitments of times t^* to t_i^+ are charged on the horizontal edges in the intersection between $payer(I,i)$ and SQUARE's route that reach $(pivot(I), t_i^+)$. Recall that, on the one hand, there are $\delta^{\ell(I)}$ commitments at most in bin (I,i) corresponding to times $t^* \le t \le t^+$. On the other hand, there exists at least $\delta^{\ell(I)}$ horizontal edges in the intersection between SQUARE route and $payer(I)$.

We charge the commitments of times t_i^- to t^*-1 on the arcs in the intersection between the payer ($payer(I,i)$), of (I,i) and the SQUARE's route that reaches (v,t^*). (The route of SQUARE that reach (v,t^*) must contain an arc $a_t = ((u,t),(u,t+1))$ in $payer(I,i)$ for every time $t \in [t_i^-, t^*-1]$; this implies that in each time $(t_i^- < t < t^*)$ there is a commitment in the bin, there is also an arc a_t in SQUARE solution (from time t to time $t+1$); we charge that commitment on that arc a_t.)

For each interval I, it is left to account for commitments in I's last session. That is, we now handle the bin (I,i') where I has i' commitment-sessions. This session may not end with a base replica in the pivot of I, so we cannot apply the argument above (that SQUARE must have a route reaching the pivot of I at $t_{i'}^+$). On the other hand, the first session of I (the uncommitted-session) does end with a base replica in $pivot(I)$, but has no commitments. Intuitively, we use the payer of the first session of I to pay for the commitments of the last session of I. Specifically, in the first session, the SQUARE route must enter the neighborhood of I from the left side; Hence, we apply the argument of case SE above.

To summarize, **(1)** each edge that belongs to SQUARE's solution may be charged at most once to each payer that it belongs too. **(2)** each edge belongs to $4\log_\delta n$ payers at most (there are $\log_\delta n$ levels; the payer rectangle of each level is two times wider than the bins; two consecutive sessions may intersect only at their boundaries)[8]. This leads to the term $4\log_\delta n$ before the $|\mathcal{F}^{\mathrm{SQ}}|$ in the statement of the lemma. ∎

We now optimize a tradeoff between the storage coast and the delivery cost of D-LINE$^{\mathrm{on}}$. On the one hand, Lemma 4.8 shows that a large δ reduces the number of commitments. By Observation 4.3, this means a large δ reduces the storage cost of D-LINE$^{\mathrm{on}}$. On the other hand, corollary 4.6

[8] Note that, unlike the analysis of LINE$^{\mathrm{on}}$ for undirected line network [15, 14], we don't claim that each arc is charged just for constant number of times.

shows that a *small* δ reduces the delivery cost. To optimize the tradeoff (in an order of magnetite), fix $\delta = \lceil \frac{\log n}{\log \log n} \rceil$. Thus, $\log_\delta n = \Theta(\frac{\log n}{\log \log n})$. Corollary 4.6, Lemma 4.8 and Observation 4.3 imply that $cost(\text{D-LINE}^{\text{on}}, \mathcal{R}) = O(\frac{cost(\text{SQUARE}, \mathcal{R}) \log n}{\log \log n})$. Thus, by Theorem 3.9, we have the proof of the following theorem.

Theorem 4.9 *Algorithm* D-LINE$^{\text{on}}$ *is* $O(\frac{\log n}{\log \log n})$*-competitive for* DMCD *problem.*

5 Optimal algorithm for RSA and for DMCD

Algorithm D-LINE$^{\text{on}}$ in Section 4 solves DMCD. To solve also RSA, we transform Algorithm D-LINE$^{\text{on}}$ to an algorithm RSA$^{\text{on}}$ that solves RSA. First, let us view the reasons why the solution for DMCD (Section 4) does not yet solve RSA. In DMCD, the X coordinate of every request (in the set $\mathcal{R}$) is taken from a known set of size n (the network nodes $\{1, 2, ..., n\}$). On the other hand, in RSA, the X coordinate of a *point* is arbitrary. (A lesser obstacle is that the Y coordinate is a real number, rather than an integer.) The main idea is to make successive guesses of the number of Steinr points and of the largest X coordinate and solve under is proven wrong (e.g. a point with a larger X coordinate arrives) then readjust the guess for future request. Fortunately, the transformation is exactly the same as the one used in [14, 15] to transform the algorithm for undirected MCD to solve *SRSA*. For completeness, we nevertheless present the transformation here.

5.1 Proof Outline

The following outline is taken (almost) word for word from [15]. (We made minor changes, e.g. replacing the word *SRSA* by the word RSA).

First, let us view the reasons why the solution for DMCD (Section 4) does not yet solve RSA. In DMCD, the X coordinate of every request (in the set $\mathcal{R}$) is taken from a known set of size n (the network nodes $\{1, 2, ..., n\}$). On the other hand, in RSA, the X coordinate of a *point* is arbitrary. (A lesser obstacle is that the Y coordinate is a real number, rather than an integer.) The main idea is to make successive guesses of the number of Steinr points and of the largest X coordinate and solve under is proven wrong (e.g. a point with a larger X coordinate arrives) then readjust the guess for future request. Let us now transform, in three conceptual stages, D-LINE$^{\text{on}}$ into an optimal algorithm for the online problem of RSA:

1. Given an instance of RSA, assume temporarily (and remove the assumption later) that the number N of points is known, as well as M, the maximum X coordinate any request may have. Then, simulate a network where $n \geq N$ and $\sqrt{\log n} = O(\sqrt{\log N})$, and the n nodes are spaced evenly on the interval between 0 and M. Transform each RSA request to the nearest grid point. Solve the resulting DMCD problem.

2. Translate these results to results of the original RSA instance.

3. Get rid of the assumptions.

The first stage is, of course, easy. It turns out that "getting rid of the assumptions" is also relatively easy. To simulate the assumption that M is known, guess that M is some M_j. Whenever a guess fails, (a request $r_i = (x_i, t_i)$ arrives, where $x_i > M_j$), continue with an increased guess M_{j+1}. A similar trick is used for guessing N. In implementing this idea, our algorithm turned out paying a cost of ΣM_j. (This is M_j per failed guess, since each application of SQUARE to a new instance, for a new guess, starts with delivering a copy to every node in the simulated network; see the description

of Algorithm SQUARE.) On the other hand, an (optimal) algorithm that knew M could have paid M only once. IF M_{j+1} is "sufficiently" larger than M_j, then $\Sigma M_j = O(M)$.

The second stage above (translate the results) proved to be somewhat more difficult, even in the case that N and M are known (and even if they are equal). Intuitively, following the first stage, each request $r_i = (x_i, t_i)$ is inside a grid square. The solution of DMCD passes via a corner of the grid square. To augment this into a solution of RSA, we need to connect the corner of the grid square to r_i. This is easy in an offline algorithm. However, an online algorithm is not allowed to connect a point at the top of the grid square (representing some time t) to a point somewhere inside the grid square (representing some earlier time $t - \epsilon$).

Somewhat more specifically, following the first stage, each request $r_i = (x_i, t_i)$ is in some grid square, where the corners of the square are points of the simulated DMCD problem. If we normalize M to be N, then the left bottom left corner of that square is $(\lfloor x_i \rfloor, \lfloor t_i \rfloor))$. Had we wanted an *offline* algorithm, we could have solved an instance of DMCD, where the points are $(\lfloor x_1 \rfloor, \lfloor t_1 \rfloor), (\lfloor x_2 \rfloor, \lfloor t_2 \rfloor), (\lfloor x_3 \rfloor, \lfloor t_3 \rfloor), ...$ Then, translating the results of DMCD would have meant just augmenting with segments connecting each $(\lfloor x_i \rfloor, \lfloor t_i \rfloor)$ to (x_i, t_i). Unfortunately, this is not possible in an *online* algorithm, since (x_i, t_i) is not yet known at $(\lfloor t_i \rfloor)$. Similarly, we cannot use the upper left corner of the square (for example) that way, since at time $\lceil t_i \rceil$, the algorithm may no longer be allowed to add segments reaching the earlier time t_i.

5.2 Informal description of the transformed *RSA* algorithm assuming $n/2 \leq \max_x \mathcal{Q} \leq n$ and $\sqrt[4]{n} \leq N \leq n$ and n is known

The algorithm under the assumptions above appears in Figure 15. Below, let us explain the algorithm and its motivation informally.

When describing the solution of DMCD, it was convenient for us to assume that the network node were $\{1, ..., n\}$. In this section (when dealing with RSA), it is more convenient for us to assume that D-LINE$^{\text{on}}$ solves DMCD with the set of network nodes being $\{0, ..., n-1\}$. Clearly, it is trivial (though cumbersome) to change D-LINE$^{\text{on}}$ to satisfy this assumption.

Assume we are given a set of points $\mathcal{Q} = \{p_1 = (x_1, y_1, ..., (x_N, y_N))\}$ for RSA. We now translate RSA points to DMCD requests (Fig. 14). That is, each point $p_i = (x_i, y_i)$ that is not already on a grid node, is located inside some square whose corners are the grid vertices. We move point $p_i = (x_i, y_i)$ to the grid vertex (replica) $r_i = (v_i, t_i)$ on the left top corner of this square. That is, we move p_i (if needed) somewhat later in time, and somewhat left on the X axis. We apply D-LINE$^{\text{on}}$ to solve the resulting DMCD. This serves $r_i = (v_i, t_i)$ from some other replica (u, t_i), where t_i may be slightly later than the time y_i we must serve p_i. After SQUARE solves the DMCD instance, we modify the DMCD solution to move the whole horizontal route $\mathcal{H}^{\text{on}}(i)$ of request r_i (route from $q_i^{\text{on}} = (u_i^{\text{on}}, t_i)$ to $r_i = (v_i, t_i)$ somewhat earlier in time (from time t_i to time y_i). This now serves a point (v_i, y_i), where v_i may be slightly left of x_i. Hence, we extend the above horizontal route by the segment from (v_i, y_i) to $p_i = (x_i, y_i)$. In addition, the transformed algorithm leaves extra copies in every network node along the route $\mathcal{H}^{\text{on}}(i)$, until time t_i (see Fig. 16(d)); a little more formally, the algorithm adds to the solution of RSA the vertical line segment $L_{ver}\langle(k, y_i), (k, t_i)\rangle$ (a vertical segment between the points (k, y_i) and (k, t_i)), for every k such that $(k, t_i) \in \mathcal{V}^{\text{on}}(i)$.

There is a technical point here: D-LINE$^{\text{on}}$ had a copy in (u, t_i) and we need a "copy" in (u, y_i) where $t_i - 1 < y \leq t_i$. That is, we need that the solution of RSA problem will already includes (u, y_i).

Observation 5.1 *The solution of* RSA *problem already includes* (u, y_i).

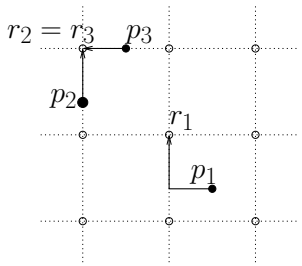

Figure 14: Point p_1 is transformed upward and leftward to γ_1; p_2 is transformed upward and p_3 is is transformed leftward; the points transform to the same vertex point.

Proof: To make sure such a copy in (u, y_i) does exist, let us consider the way the copy reached (u, t_i) in D-LINE$^{\text{on}}$. If D-LINE$^{\text{on}}$ stored a "copy" in u from time $t-1$ to t (see Fig. 16(c)), then also (u, y_i) belong to the solution. Otherwise, D-LINE$^{\text{on}}$ moved the copy to (u, t_i) over a route $\mathcal{P}_{\mathcal{H}}$ from some other grid vertex (w, t_i).

Note that (w, t_i) appeared in the transformed algorithm because that algorithm served a point $p_i = (x_i, y_i)$, of a time $t_i - 1 < y_i < t_i$ (see Fig. 16(d)). The transformed algorithm moved this route $\mathcal{P}_{\mathcal{H}}[(w,t),(u,t)]$ earlier in time to $\tilde{\mathcal{P}}_{\mathcal{H}}[(w,y_i),(u,y_i)]$ and left copies in those network node until time t_i (see Fig. 16(d)). In particular, it leave a copy also in u from time y_i to time t_i, hence (u, y_i) is already in the solution of the transform algorithm. ∎

So far, we described how to transform the delivery phase of D-LINE$^{\text{on}}$. The storage phase of D-LINE$^{\text{on}}$ does not need to be transformed. (Actually, DMCD even has some minor extra difficulty that does not exist in RSA; consider some request $r_{i-1} = (v_{i-1}, t_{i-1})$ in DMCD, and suppose that the next request $r_i = (v_i, t_i)$ is at time $t_i = t_{i-1} + 10$; then time $t+1$ arrives, and D-LINE$^{\text{on}}$ must make some decisions, without knowing that the next request will be at time $t_{i-1}+10$; then time $t+2$ arrives, etc; no such notion of time passing (without new points arriving) exists in the definition of RSA; that is, the Y coordinate y_i of the next request $p_i = (x_i, y_i)$ is known right after the algorithm finished handling $p_i = (x_i, y_i)$; the storage phase of the transformed algorithm does not make any use of this extra freedom in RSA and simulates the "times", or the Y coordinates, one by one; note that for that purpose, the transformation of the delivery phase ensured the following property: that if a copy in DMCD exists in a replica (v, t) in D-LINE$^{\text{on}}$, this replica also contains a copy in the transformed algorithm.) Denote the solution of RSA$_n^{\text{on}}$ on $\mathcal{Q}$ by $\mathcal{F}_n^{\text{RSA}}(\mathcal{Q})$. For the pseudo code, see Fig. 15.

Analysis sketch of the transformed algorithm with known parameters It is not hard to see that an optimal solution for that instance of DMCD is "not that far" from an optimal solution of the original instance of RSA. To see that, given an optimal solution of RSA, one can derive a feasible solution of the resulting DMCD by adding 2 segments of length at most 1 for each point p. (One vertical such segment plus a horizontal one are enough to connect a point p to the replica (v, t) where we moved p). The total of those distances is $2n$ at most. On the other hand, an optimal solution of RSA would need to pay at least $\max_x \mathcal{Q} \geq n/2$. Hence, an optimal solution for DMCD would have implied a constant approximation of RSA. Intuitively, an approximation (and a competitive ratio) for DMCD implies an approximation (and a competitive ratio) of RSA in a similar way. For a given Algorithm A for RSA and a set $\mathcal{Q}$ of input points, let $cost(A, \mathcal{Q})$ be the cost of A on $\mathcal{Q}$. Let OPT be an optimal algorithm for RSA.

1. For p_1 do:
   (a) compute the translated request $r_1 = (v_1, t_1)$ of p_1; $\mathcal{R} \leftarrow \{r_1\}$.
   (b) $\mathcal{F}_n^{\mathrm{RSA}} \leftarrow \{L_{ver}\langle(0,0),(0,y_1)\rangle, L_{hor}\langle(0,y_1),(x_1,y_1)\rangle\}$;
2. For each point $p_i \in \mathcal{Q} \setminus \{p_1\} = \{p_2 = (x_2, y_2), ..., p_N = (y_N, y_N)\}$ do:
   (a) compute the translate request $r_i = (v_i, t_i)$ of p_i;
   (b) $\mathcal{R} \leftarrow \mathcal{R} \cup \{r_i\}$.
   (c) "Vertical phase"
      i. If $t_i > t_{i-1}$, then for each time $t = t_{i-1}, ..., t_i - 1$ do:
         A. "Simulate" D-LINE$^{\mathrm{on}}$ on $\mathcal{R}$ to find $\mathcal{C}_{t+1}$.
         B. $\mathcal{F}_n^{\mathrm{RSA}} \leftarrow \mathcal{F}_n^{\mathrm{RSA}} \cup \{L_{ver}\langle(v,t),(v,t+1)\rangle \mid (v,t+1) \in \mathcal{C}_{t+1}\}$.
   (d) "Horizontal phase"
      i. "Simulate" D-LINE$^{\mathrm{on}}$ on $\mathcal{R}$ to find $\mathcal{V}^{\mathrm{on}}(i)$.
      ii. $\mathcal{F}_n^{\mathrm{RSA}} \leftarrow \mathcal{F}_n^{\mathrm{RSA}} \cup \{L_{hor}\langle(u_i^{\mathrm{on}}, y_i),(v_i,y_i)\rangle\}$.
      iii. $\mathcal{F}_n^{\mathrm{RSA}} \leftarrow \mathcal{F}_n^{\mathrm{RSA}} \cup \{L_{hor}\langle(v_i,y_i),(x_i,y_i)\rangle\}$
      iv. $\mathcal{F}_n^{\mathrm{RSA}} \leftarrow \mathcal{F}_n^{\mathrm{RSA}} \cup \{L_{ver}\langle(u,y_i),(u,t_i)\rangle \mid (u,t_i) \in \mathcal{P}_{\mathcal{V}}^{\mathrm{on}}(i)\}$.
3. Return $\mathcal{F}_n^{\mathrm{RSA}}(\mathcal{Q})$

Figure 15: Subroutine $\mathrm{RSA}_n^{\mathrm{on}}$ assumes the knowledge of n and that $n/2 \leq \max_x \mathcal{Q} \leq n$ and $\sqrt[4]{n} \leq N \leq n$.

Lemma 5.2 *Assume that $max_x \mathcal{Q} \leq n$ and $N \leq n$. Then, $cost(\mathrm{RSA}_n^{\mathrm{on}}, \mathcal{Q}) = O(\frac{\log n}{\log\log n}(cost(\mathrm{OPT}, \mathcal{Q}) + n))$. If also $\sqrt[4]{n} \leq N$ and $n/2 \leq max_x \mathcal{Q}$, then $\mathrm{RSA}_n^{\mathrm{on}}$ is $O(\frac{\log N}{\log\log N})$-competitive for* RSA.

Proof: It is easy to verify that $\mathrm{RSA}_n^{\mathrm{on}}$ computes a feasible solution (see the "technical point" comments in parentheses in section 5.2). Consider some input point set $\mathcal{Q} = \{p_1 = (x_1, y_1), ..., p_N = (x_N, y_N)\}$ such that $\max_x \mathcal{Q} \leq n$ and $N \leq n$. Let $\mathcal{R} = \{r_1 = (v_1, t_1), ..., r_N = (v_N, t_N)\}$ be the translated instance of the MCD problem.

Recall how does $\mathrm{RSA}_n^{\mathrm{on}}(\mathcal{Q})$ translate the solution of D-LINE$^{\mathrm{on}}(\mathcal{R})$. An horizontal edge $((u,t_i),(u+1,t_i)) \in \mathcal{P}_{\mathcal{H}}^{\mathrm{on}}(i)$ (that D-LINE$^{\mathrm{on}}$ add to its solution when handling request r_i, see step (D2) in D-LINE$^{\mathrm{on}}$) is translated into a horizontal line segment $L_{hor}\langle(u,y_i),(u+1,y_i)\rangle$. An arc $((u,t),(u,t+1)) \in \mathcal{A}^{\mathrm{on}}$ (of D-LINE$^{\mathrm{on}}$'s solution on $\mathcal{R}$) is translated into a vertical line segment $L_{ver}\langle(u,t),(u,t+1)\rangle$. Hence, the total cost of those parts of the solution of $\mathrm{RSA}_n^{\mathrm{on}}(\mathcal{Q})$ is exactly the same as the cost of the solution of D-LINE$^{\mathrm{on}}(\mathcal{R})$.

Thus, the cost of $\mathrm{RSA}_n^{\mathrm{on}}$ on $\mathcal{Q}$ differ from the cost of D-LINE$^{\mathrm{on}}$ on $\mathcal{R}$ only by two kinds of "short" segments (Segment of length at most 1). For the first kind, recall (technical point in Section 5.2) that for every moved horizontal path $\tilde{\mathcal{P}}_{\mathcal{H}}[(w,\tilde{y}),(u,\tilde{y})]$, $\mathrm{RSA}_n^{\mathrm{on}}$ added a short vertical segment for every network node w' of that path from $(w',\tilde{y})$ to $(w',\lceil y \rceil)$. The second kind of addition is an horizontal short segment connecting the input point $p = (x,y)$ to (u,y), where $u = \lfloor x \rfloor$.

The total cost of the second kind is bounded by n, since $|v_i - x_i| \leq 1$. We claim that the total cost of the short segment of the first kind is $cost(\text{D-LINE}^{\text{on}}, \mathcal{R})$ at most. To see that, notice that we have at most 1 such "short" segment (shorter than 1) per replica that appears in the solution of D-LINE$^{\text{on}}$ on $\mathcal{R}$. That solution of D-LINE$^{\text{on}}$ contains at least as many edges as it contains replicas. Formally, the cost of RSA_n^{on} is at most,

$$\begin{aligned}
&cost(\text{RSA}_n^{\text{on}}, \mathcal{Q}) = \\
&cost(\text{D-LINE}^{\text{on}}, \mathcal{R}) + \sum_{i=1}^{n} \left(|\mathcal{P}_{\mathcal{V}}^{\text{on}}(i)| \cdot (t_i - y_i) + |v_i - x_i| \right) \\
&\leq 2cost(\text{D-LINE}^{\text{on}}, \mathcal{R}) + n.
\end{aligned}$$

Thus, by Theorem 4.9,

$$cost(\text{RSA}_n^{\text{on}}, \mathcal{Q}) \leq c_1 \frac{\log n}{\log \log n} \cdot cost(\text{OPT}, \mathcal{R}) + n, \tag{25}$$

where c_1 is some constant.

Let us look the other direction, from an *optimal* solution of RSA for $\mathcal{Q}$ to optimal solution of DMCD for $\mathcal{R}$. Recall that r_i can be served from p_i at a cost of 2 (at most). Hence,

$$cost(\text{OPT}, \mathcal{R}) \leq cost(\text{OPT}, \mathcal{Q}) + 2n. \tag{26}$$

Thus, by Inequalities (25) and (26),

$$\begin{aligned}
&cost(\text{RSA}_n^{\text{on}}, \mathcal{Q}) \qquad (27) \\
&\leq c_1 \frac{\log n}{\log \log n} \cdot (cost(\text{OPT}, \mathcal{Q}) + 2n) + n \\
&= O(\frac{\log n}{\log \log n} \cdot (cost(\text{OPT}, \mathcal{Q}) + n).
\end{aligned}$$

The first statement of the lemma holds. Now, let us prove the second statement of the lemma. Assume that $\max_x \mathcal{Q}/2 \leq N \leq n$ and $\sqrt[4]{n} \leq N \leq n$. Thus also,

$$cost(\text{OPT}, \mathcal{Q}) \geq n/2. \tag{28}$$

Therefore, by Inequalities (27) and (28),

$$\begin{aligned}
&\frac{cost(\text{RSA}_n^{\text{on}}, \mathcal{Q})}{cost(\text{OPT}, \mathcal{Q})} \\
&\leq \frac{c_1 \frac{\log n}{\log \log n} \cdot cost(\text{OPT}, \mathcal{Q})}{cost(\text{OPT}, \mathcal{Q})} + \frac{c_1 \frac{\log n}{\log \log n} \cdot 2n + n}{n/2} \\
&\leq (5c_1 + 1) \frac{\log n}{\log \log n}.
\end{aligned}$$

The lemma follows, since $\sqrt[4]{\log n} \leq N$. ∎

Below, RSA_n^{on} is used as a module in another algorithm, responsible for implementing the assumptions. In each execution of the other algorithm, RSA_n^{on} is invoked multiple times, for multiple subsets of the input. Unfortunately, not every time, the other algorithm uses RSA_n^{on}, all the assumptions are ensured. This is the reason of the "extra" factor $n\sqrt{\log n}$ in the first part of the above lemma above. Fortunately, these extra factors of all the invocations are bounded separately later.

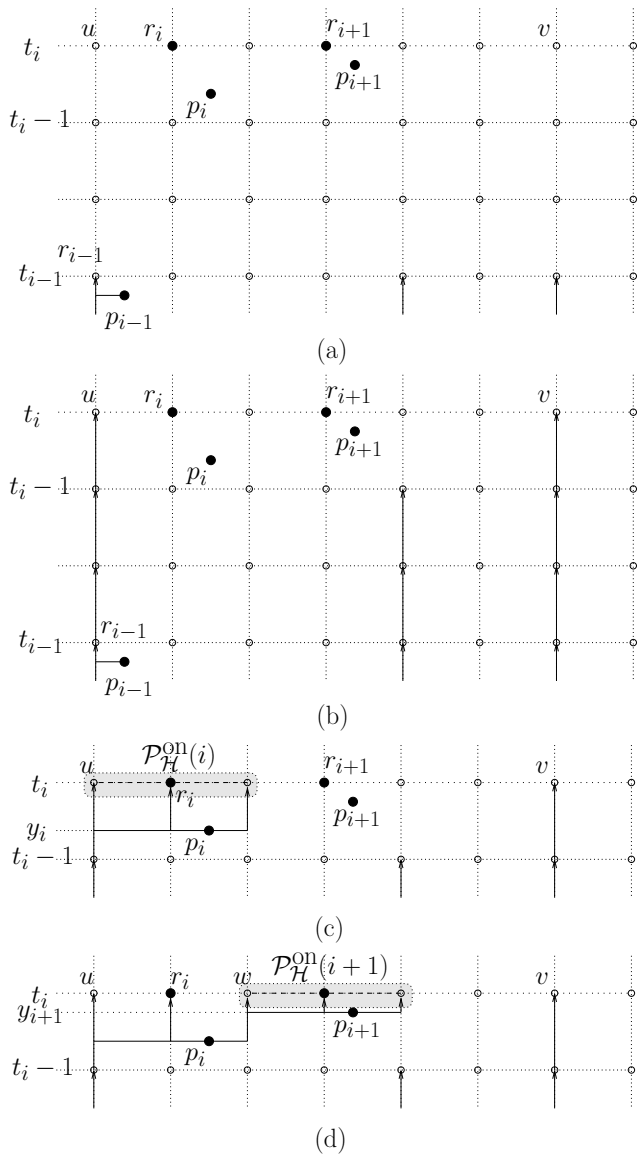

Figure 16: Example of execution of $\text{RSA}_n^{\mathsf{on}}$. (a) $\text{RSA}_n^{\mathsf{on}}$'s solution after handling point p_{i-1}; (b) $\text{RSA}_n^{\mathsf{on}}$ simulates the storage phase of D-LINE$^{\mathsf{on}}$ on $\mathcal{R}$ for times $t = t_{i-1}, ..., t_i - 1$; (c) $\text{RSA}_n^{\mathsf{on}}$ handles point p_i, moves $\mathcal{P}^{\mathsf{on}}_{\mathcal{H}}(i)$ from "time" t_i to "time" y_i (it serves this path from (u, y_i), who "receives a copy" when D-LINE$^{\mathsf{on}}$ handles time $t_i - 1$ in the storage phase), and "leaves copies" at the nodes $\mathcal{P}^{\mathsf{on}}_{\mathcal{V}}(i)$ from "time" y_i to "time" t_i; (d) $\text{RSA}_n^{\mathsf{on}}$ handles point p_{i+1}, moves $\mathcal{P}^{\mathsf{on}}_{\mathcal{H}}(i+1)$ from "time" t_i to "time" y_{i+1} (it serves this path from (w, y_{i+1}), who "receives a copy" when handling point p_i), and "leaves copies" at the nodes of $\mathcal{P}^{\mathsf{on}}_{\mathcal{V}}(i+1)$ from "time" y_{i+1} to "time" t_i.

5.3 Getting rid of the assumption that $M=N$

We now describe an online algorithm $\text{RSA}^{\text{on}}_{M,n,p}$ that is somewhat more general than $\text{RSA}^{\text{on}}_{n}$. Algorithm $\text{RSA}^{\text{on}}_{M,n,p}$ is *not* based on the assumption that the upper bound M on $\max_x \mathcal{Q}$ is also the number of points. That is, we now do *not* assume that $M = N$. Getting rid of this assumption is straightforward. The new online algorithm $\text{RSA}^{\text{on}}_{M,n,p}$ transforms the X coordinate of each input point to the interval $[0, n]$. Algorithm $\text{RSA}^{\text{on}}_{M,n,p}$ passes the transformed point to the online algorithm $\text{RSA}^{\text{on}}_{n}$ of Section 5.2 that is assumed to be executing in parallel. The transformation of a point is, though, a little more involved, as detailed below.

Later on (in Section 5.4), $\text{RSA}^{\text{on}}_{M,n,p}$ will be used by an even more general algorithm in a similar way. For that, it is more convenient for us to define algorithm $\text{RSA}^{\text{on}}_{M,n,p}$ a somewhat more general algorithm then is needed by the description so far. We now assume that the origin is not necessarily $(0, 0)$, but is rather some $p_0 = (0, y_0)$. (Meanwhile, we still assume that $x_0 = 0$). Hence, algorithm $\text{RSA}^{\text{on}}_{M,n,p}$ translates the X coordinate of each input point $p = (x, y)$ to $f(x) = x \cdot \frac{N}{M}$. To keep the proportion between the axes, the y coordinate y is translated to $f(y) = (y - y_0) \cdot \frac{N}{M}$. (Recall that $y \geq t_0$.) Finally, the solution of $\text{RSA}^{\text{on}}_{n}$ is translated back to the coordinates of $\text{RSA}^{\text{on}}_{M,n,p}$ applying the transformation f^- to every point of the solution. (Clearly, this is a polynomial task, since the solution is described using a polynomial number of points). The pseudo code appears in Fig. 17. By Lemma 5.2 and the description of $\text{RSA}^{\text{on}}_{M,n,p}$, it is easy to see the following.

Observation 5.3 *Assume that* $max_x\mathcal{Q} \leq M$ *and* $N \leq n$. *Then,* $cost(\text{RSA}^{\text{on}}_{M,n,p}, \mathcal{Q}) = O(\frac{\log n}{\log\log n}(cost(\text{OPT}, \mathcal{Q}) + M))$. *If also* $\sqrt[4]{n} \leq N$ *and* $M/2 \leq max_x\mathcal{Q}$, *then* $\text{RSA}^{\text{on}}_{M,n,p}$ *is* $O(\frac{\log N}{\log\log N})$*-competitive for* RSA.

- origin is $p_0 = (x_0, y_0)$.
  1. $\mathcal{Q}' \leftarrow \emptyset$.
  2. For each point $p_i \in \mathcal{Q}$ do:
     (a) $p'_i = (x'_i, y'_i) \leftarrow f(p_i, M, n, y_0)$;
     (b) $\mathcal{Q}' \leftarrow \mathcal{Q}' \cup \{p'_i\}$.
     (c) Call $\text{RSA}^{\text{on}}_{n}$ as a subroutine on $\mathcal{Q}'$ to find $\mathcal{F}^{\text{RSA}}_{n}(\mathcal{Q}')$;
     (d) $\mathcal{F}^{\text{RSA}}_{M,n,p} \leftarrow f^{-1}(\mathcal{F}^{\text{RSA}}_{n}(\mathcal{Q}'), M, n, y_0)$;

Figure 17: Algorithm $\text{RSA}^{\text{on}}_{M,n,p}$.

5.4 Getting rid of the knowledge assumptions

To give up the assumption that $\max_x \mathcal{Q}$ is known, we use a standard trick. We first guess that $\max_x \mathcal{Q}$ is "about" twice the X coordinate of the first point. Whenever the guess for $\max_x \mathcal{Q}$ is proven wrong (some $p_i = (x_i, y_i)$ arrives with x_i larger then our guess for $\max_x \mathcal{Q}$), we double the guess. We do not change the solution for the points we already served. Simply, the points that arrive from now on, are treated as a new instance of RSA, to be solved (by $\text{RSA}^{\text{on}}_{M,n,p}$) by a translation to a new instance of DMCD. Intuitively, every instance of DMCD may need to pay an

additional cost that is proportional to our current guess of $\max_x \mathcal{Q}$. This is justified by the fact that (1) the new guess is at least double our previous guess of $\max_x \mathcal{Q}$; and (2) any optimal algorithm would need now to pay $\max_x \mathcal{Q}$ guessed before. (A minor technical point is that the origin of the new instance of RSA may not be $p_0 = (0, 0)$; instead, the new origin is $(0, y_{i-1})$, where y_{i-1} is the y-coordinates of the last point served.)

For justifying the other assumption, that the number of points is known in advance, we use a similar trick; however, its justification is more complex. That is, if the number of points grows larger beyond our current guess, *n-guess*, we increase our guess of the number of points. We then start a new instance of $\text{RSA}^{\text{on}}_{M,n,p}$ with the new guess. (In turn, this leads to a new activation of D-LINE$^{\text{on}}$ with $n\text{-}guess^{new}$ as the new network size.) Hence, we start a new DMCD instance with an increased "network size". The "new" guess $n\text{-}guess^{new}$ of the number of RSA points is (not doubled but) the power of 4 of our "current" *n-guess* (yielding a double exponential sequence). Each new DMCD instance is associated with a cost of $O(\sqrt{\log n\text{-}guess^{new}} \max_x \mathcal{Q})$ at most. Thanks to using a double exponential groth rather than an exponential growth, this would increase the competitive ratio just by a factor of $O(\log \log N)$. Clearly, one should not increase the guess (of the number of points) more than polynomially each time (since otherwise, for the last guess $\widetilde{n\text{-}guess}$, the value would have been too high compared to the desired $\frac{\log N}{\log \log N}$ competitive ratio.) Summarizing the above informal description, given an instance of RSA, we use "guesses" of $\max_x \mathcal{Q}$ and N to partition the points $\mathcal{Q}$ into subsets. Each such subset defines a problem we translate separately to DMCD via $\text{RSA}^{\text{on}}_{M,n,p}$.

Given an instance of RSA, we now define its partition of multiple instances. For that, we define the partition of $\mathcal{Q}$ into subsets $\mathcal{Q}\langle 1\rangle$, $\mathcal{Q}\langle 2\rangle$, ... The first $|\mathcal{Q}\langle 1\rangle|$ points will belong to $\mathcal{Q}\langle 1\rangle$, the next $|\mathcal{Q}\langle 2\rangle|$ will belong to $\mathcal{Q}\langle 2\rangle$, etc. We shall also show how to detect online the first point in $\mathcal{Q}\langle 2\rangle$, the first in $\mathcal{Q}\langle 3\rangle$, etc. Before that, we must tackle some technicality. The original RSA problem with defined for an origin of $X = 0$ and $Y = 0$. However, after solving for the RSA instance $\mathcal{Q}\langle 1\rangle$, the next point is at Y coordinate that is larger than zero. Moreover, when solving DMCD, we allowed the origin to be at any node (that is, in any X coordinate). Hence, it is convenient to generalize the definition of the RSA to the setting were the input includes an origin point $p_0 = (x_0, y_0)$, in the positive quadrant. The input point set $\mathcal{Q}\langle k\rangle$ includes only points (in the positive quadrant), whose y-coordinates are grater than or equal to y_0.

Consider a point set $\mathcal{Q} = \{p_1, ..., p_N\}$. Algorithm RSA^{on} partitions $\mathcal{Q}$ into subsets as follows. For every $i = 1, ..., N$ define that

$$M\text{-}guess(i) = 2^{l'}, \tag{29}$$

where $l' = \lceil \log \max\{x_j \mid j = 1, ..., i\} \rceil$, and

$$n\text{-}guess(i) = 2^{2^{2l^*}}, \tag{30}$$

where l^* is integer such that $l^* = \min_l(2^{2^{2l}} \geq i)$. Note that, $2^{2^{2l+1}} = 2^{(4 \cdot 2^l)} = (2^{2^{2l}})^4$. Hence, the growth of the sequence $2^{2^{2 \cdot 0}}, 2^{2^{2 \cdot 1}}, 2^{2^{2 \cdot 2}}, ...$ is for the power of 4.

Let us use the above guesses to generate the subset. Specifically, we generate a sequence $g_1 < g_2 < ... < g_\tau$ (for some g_τ) of separators between consecutive subsets. That is, $\mathcal{Q}\langle 1\rangle = \{p_{g_1}, ..., p_{g_2-1}\}$, then $\mathcal{Q}\langle 2\rangle = \{p_{g_2}, ..., p_{g_3-1}\}$, etc. A separator is the index of a point where one of the guess fails. Specifically, let $g_1 = 1$ and if $M\text{-}guess(g_k) < M\text{-}guess(N)$ or $n\text{-}guess(g_k) < n\text{-}guess(N)$, then let

$$g_{k+1} \triangleq \min_i \quad \big(M\text{-}guess(g_k) < M\text{-}guess(i) \text{ or } n\text{-}guess(g_k) < n\text{-}guess(i)\big). \tag{31}$$

Define that the guess *n-guess* of $\mathcal{Q}\langle k\rangle$ is $n_k = 2^{2^{2(n\text{-}guess(g_k))}}$ and the guess *M-guess* of $\mathcal{Q}\langle k\rangle$ is $M_k = 2^{M\text{-}guess(g_k)}$, for every $k = 1, ..., \tau$. The origin points of these subsets are defined as follows: Let y^k_{last} = be the y-axis of the *last* point $p_{g_{k+1}-1}$ in $\mathcal{Q}\langle k\rangle$ and let $y^1_0 = 0$ and $y^k_0 = y^{k-1}_{last}$ (for $k = 2, ..., \tau$). The origin point of $\mathcal{Q}\langle k\rangle$ is $p^k_0 = (0, y^k_0)$, for every $k = 1, ..., \tau$ (see Fig. 18).

All the above functions can be computed online. As sketched, Algorithm RSA$^{\text{on}}$ handles a point after point, and a subset after subset. For every point $p_i \in \mathcal{Q}$, RSA$^{\text{on}}$ finds the subset $\mathcal{Q}\langle k\rangle$ that p_i belongs to (i.e., $p_i \in \mathcal{Q}\langle k\rangle$), then RSA$^{\text{on}}$ passes the point to an instance of RSA$^{\text{on}}_{M,n,p}$ executing (in parallel to RSA$^{\text{on}}$) on $\mathcal{Q}\langle k\rangle$, with the origin point $p^k_0 = (0, y^k_0)$, and with the *M-guess* parameter $M = M_k$ and the *n-guess* parameter $n = n_k$. Denote the solution of RSA$^{\text{on}}_{M,n,p}$ on $\mathcal{Q}\langle k\rangle$ by $\mathcal{F}^{\text{RSA}}_{M,n,p}(\mathcal{Q}\langle k\rangle)$. The solution of RSA$^{\text{on}}$ is the union of the solutions of RSA$^{\text{on}}_{M,n,p}$ on all the subsets. That is, RSA$^{\text{on}}$'s solution is $\mathcal{F}^{\text{RSA}}(\mathcal{Q}) \equiv \cup_{k=1}^{\tau} \mathcal{F}^{\text{St}}_{M,n}(\mathcal{Q}\langle k\rangle)$. The pseudo code of RSA$^{\text{on}}$ is given in Fig. 19.

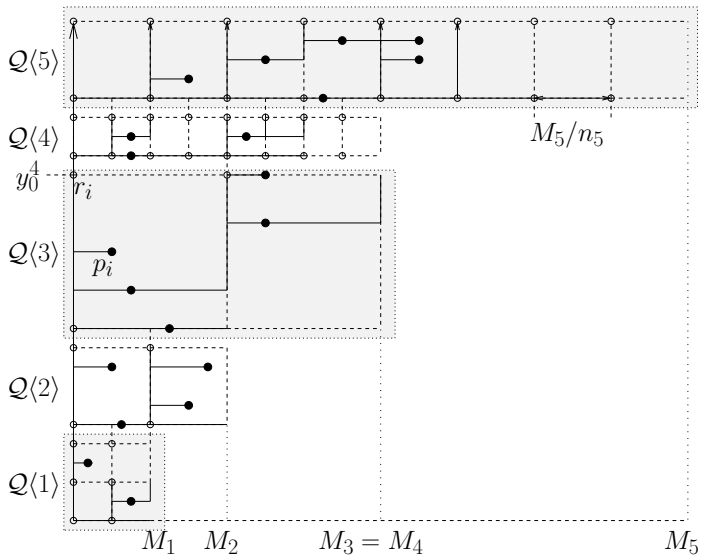

Figure 18: Partitioning $\mathcal{Q}$ into subsets $\mathcal{Q}\langle 1\rangle, \mathcal{Q}\langle 2\rangle, ..., \mathcal{Q}\langle 5\rangle$; each instance corresponds to a subset $\mathcal{Q}\langle k\rangle$, origin $(0, y^k_0)$, *M-guess* M_k and *n-guess* n_k.

Theorem 5.4 *Algorithm* RSA$^{\text{on}}$ *is optimal and is* $O(\frac{\log N}{\log\log N})$*-competitive.*

5.5 Optimizing DMCD for a small number of requests

Algorithm D-LINE$^{\text{on}}$ was optimal only as the function of the network size. Recall that our solution for RSA was optimal as a function of the number of requests. We obtain this property for the solution of DMCD too, by transforming our RSA algorithm back to solve DMCD, and obtain the promised competitiveness, $O(\min\{\frac{\log N}{\log\log N}, \frac{\log n}{\log\log n}\})$.

Algorithm D-LINE$^{\text{on}}$ was optimal as the function of the network size (Theorem 4.9). This means that it may not be optimal in the case that the number of requests is much smaller than the network size. In this section, we use Theorem 5.4 and algorithm RSA$^{\text{on}}$ to derive an improve algorithm for MCD. This algorithm, D-LINE$^{\text{on}}_{+}$, is competitive optimal (for DMCD) for any number of requests. Intuitively, we benefit from the fact that RSA$^{\text{on}}$ is optimal for any number of

1. when the first point p_1 arrives
   (a) $k \leftarrow 1$; $\mathcal{Q}\langle 1\rangle \leftarrow \{p_1\}$; $M_1 \leftarrow 2^{M\text{-}guess(1)}$; $n_1 \leftarrow 4$; $g_1 \equiv 1$; and origin $p_0^1 = (0,0)$.
   (b) start an instance of $\text{RSA}^{\text{on}}_{M,n,p}$ on $\mathcal{Q}\langle 1\rangle$;
2. when an input point arrives $p_i = (x_i, y_i)$ (for $i > 1$), /* the points are $i = 2, ..., N$ */
   (a) if $x_i \leq M_k$ and $i \leq n_k$, then $\mathcal{Q}\langle k\rangle \leftarrow \mathcal{Q}\langle k\rangle \cup \{p_i\}$.
   (b) Otherwise, (p_i "open a new instance"), then
      i. $k \leftarrow k + 1$;
      ii. $\mathcal{Q}\langle k\rangle \leftarrow \{i\}$;
      iii. $M_k \leftarrow 2^{M\text{-}guess(i)}$;
      iv. $n_k \leftarrow 2^{2^{2(n\text{-}guess(i))}}$;
      v. $p_0^k \equiv (0, y_{i-1})$;
      vi. $g_k \leftarrow i$;
      vii. start an instance of $\text{RSA}^{\text{on}}_{M,n,p}$ on $\mathcal{Q}\langle k\rangle$;
3. pass p_i to the instance of $\text{RSA}^{\text{on}}_{M,n,p}$ executing on $\mathcal{Q}\langle k\rangle$ with origin p_0^k; $M = M_k$; and $n = n_k$ and compute $\mathcal{F}^{\text{RSA}}_{M,n,p}(\{p_{g_k}, ..., p_{i+1}\})$.
4. $\mathcal{F}^{\text{RSA}} \leftarrow \mathcal{F}^{\text{RSA}} \cup \mathcal{F}^{\text{RSA}}_{M,n,p}(\{p_{g_k}, ..., p_{i+1}\})$.

Figure 19: Algorithm RSA^{on}.

points (no notion of network size exists in RSA).

This requires the solution of some delicate point. Given an instance DMCD^a of DMCD, we would have liked to just translate the set $\mathcal{R}^a$ of DMCD requests into a set $\mathcal{Q}$ of RSA points and apply RSA^{on} on them. This may be a bit confusing, since RSA^{on} performs by converting back to DMCD. Specifically, recall that RSA^{on} breaks $\mathcal{Q}$ into several subsets, and translates back first the first subset $\mathcal{Q}_1$ into an the requests set $\mathcal{R}_1^b$ of a new instance DMCD_1^b of DMCD. Then, RSA^{on} invokes $\text{D-LINE}^{\text{on}}$ on this new instance DMCD_1^b. The delicate point is that DMCD_1^b is different than DMCD_a.

In particular, the fact that $\mathcal{Q}_1$ contains only *some* of the points of $\mathcal{R}^a$, may cause RSA^{on} to "stretch" their X coordinates to fit them into the network of DMCD_a. Going carefully over the manipulations performed by RSA^{on} reveals that the solution of RSA^{on} may not be a feasible solution of DMCD (even though it applied $\text{D-LINE}^{\text{on}}$ plus some manipulations). Intuitively, the solution of RSA^{on} may "store copies" in places that are not grid vertices in the grid of DMCD_a. Thus the translation to a solution of DMCD_1 is not immediate.

Intuitively, to solve this problem, we translate a solution of RSA^{on} to a solution of DMCD_a in a way that is similar to the way we translated a solution of $\text{D-LINE}^{\text{on}}$ to a solution of RSA. That is, each request of DMCD_a we move to a "nearby" point of RSA^{on}. This is rather straightforward, given the description of our previous transformation (of Section 5.2). The details are left for the full paper.

Theorem 5.5 *Algorithm* D-LINE$_+^{\text{on}}$ *is optimal and it*
$O(\min\{\frac{\log N}{\log\log N}, \frac{\log n}{\log\log n}\})$*-competitive.*

6 Lower Bound for RSA

In this section, we prove the following theorem, establishing a tight lower bound for RSA and for DMCD on directed line networks. Interestingly, this lower bound is not far from the one proven by Alon and Azar for *undirected* Euclidian Steiner trees [1]. Unfortunately, the lower bound of [1] does not apply to our case since their construct uses edges directed in what would be the wrong direction in our case (from a high Y value to a low one).

Theorem 6.1 *The competitive ratio of any deterministic online algorithm for* DMCD *in directed line networks is* $\Omega(\frac{\log n}{\log\log n})$*, implying also an* $\Omega(\frac{\log N}{\log\log N})$ *lower bound for* RSA.

Proof: We first outline the proof. Informally, given a deterministic online algorithm ONALG$_{\text{MCD}}$, we construct an adversarial input sequence. Initially, the request set includes the set DIAG $= \{(k,k) \mid 0 \leq k \leq n\}$. That is, at each time step t, the request (t,t) is made. In addition, if the algorithm leaves "many copies" then the lower bound is easy. Otherwise, the algorithm leaves "too few copies" from some time $t-1$ until time t. For each such time, the adversary makes another request at $(t-k,t)$ for some k defined later. The idea is that the adversary can serve this additional request from the diagonal copy at $(t-k,t-k)$ paying the cost of k. On the other hand, the algorithm is not allowed at time t to decide to serve from $(t-k,t-k)$. It must serve from a copy it did leave. Since the algorithm left only "few" copies to serve time t the replica, $(t,t-k)$ can be chosen at least at distance $k(\log n)$ from any copy the algorithm did leave. Hence, the algorithm's cost for such a time t is $\Omega(\log n)$ times greater than that of the adversary.

More formally, let $\delta = \lceil \log n \rceil$. Partition the line at time $t \in \{n/2, \ldots, n\}$ into $\lfloor \log_\delta n - 1 \rfloor$ intervals: $I_i(t) = (t-\delta^{i+1}, t-\delta^i]$, where $i \in \{1,2,\ldots,\lfloor \log_\delta n - 1 \rfloor\}$. (Note that the intervals are well defined, since $\lfloor \log_\delta n - 1 \rfloor \leq \lfloor \log_\delta t \rfloor$, for every $n/2 \leq t \leq n$, which implies that $\delta^i \leq t$ for every $i = 1, ..., \lfloor \log_\delta n - 1 \rfloor$.) Given an online algorithm ONALG$_{\text{MCD}}$, the adversary constructs the set of requests $\mathcal{R}$ as follows. Initially, $\mathcal{R} \leftarrow$ DIAG. For each time $t \geq n/2$, denote by $V_{\text{ALG}}(t)$ the set of nodes that hold the movie for time t (just before ONALG$_{\text{MCD}}$ receives the requests for time t). The adversary may add a request at t according to $V_{\text{ALG}}(t)$. In particular, if ONALG$_{\text{MCD}}$ leaves a copy in at least one of the nodes of every such intervals $I_i(t)$, for $i = 1, ..., \lfloor \log_\delta n - 1 \rfloor$, then the only adversary request for time t is (t,t) (while ONALG$_{\text{MCD}}$ left copies in at least $\lfloor \log_\delta n - 1 \rfloor$ nodes). Otherwise, the adversary adds the request $(t-\delta^{i^*}, t)$ to $\mathcal{R}$, where i^* is an arbitrary index such that $I_{i^*}(t) \cap V_{\text{ALG}}(t) = \emptyset$. That is, the adversary request set of time t is $\{(t,t)\}$ in the first case and $\{(t-\delta^{i^*}, t), (t,t)\}$ in the second case.

For each time $t = \lfloor n/2 \rfloor, ..., n$, one of the following two cases hold: **(1)** ONALG$_{\text{MCD}}$ pays at least $\lfloor \log_\delta n - 1 \rfloor = \Omega(\frac{\log n}{\log\log n})$ for storing at least $\lfloor \log_\delta n - 1 \rfloor$ copies from time $t-1$ to time t, while the adversary pays just $2 = O(1)$ (to serves request (t,t)); or **(2)** ONALG$_{\text{MCD}}$ pays, at least, $\delta^{i^*+1} - \delta^{i^*} = \Omega(\delta^{i^*+1})$ for delivering a copy to $(t-\delta_t^{i^*}, t)$ from some node outside the interval $I_{i^*}(t)$, while the adversary pays $O(\delta^{i^*})$ for storing the movie in node $t-\delta^{i^*}$ from time $t-\delta^{i^*}$ to time t (that is, serving from replica $(t-\delta^*, t-\delta^*)$ on the diagonal) and additional two edges (to serve request (t,t)). Thus, in that case, ONALG$_{\text{MCD}}$ pays at least $O(\log n)$ times more than the adversary. This establishes Theorem 6.1. ∎

References

[1] N. Alon and Y. Azar. On-line Steine trees in the euclidean plane. *Discrete & Computational Geometry*, 10:113–121, 1993.

[2] R. Bar-Yehuda, E. Kantor, S. Kutten, and D. Rawitz. Growing half-balls: Minimizing storage and communication costs in CDNs. In *ICALP*, pages 416–427, 2012.

[3] W. Bein, M. Golin, L. Larmore, and Y. Zhang. The Knuth-Yao quadrangle-inequality speedup is a consequence of total monotonicity. *ACM Transactions on Algorithms*, 6(1), 2009.

[4] P. Berman and C. Coulston. On-line algorrithms for Steiner tree problems. In *STOC*, pages 344–353, 1997.

[5] M. Charikar, D. Halperin, and R. Motwani. The dynamic servers problem. In *9th Annual Symposium on Discrete Algorithms (SODA)*, pages 410–419, 1998.

[6] X. Cheng, B. Dasgupta, and B. Lu. Polynomial time approximation scheme for symmetric rectilinear Steiner arborescence problem. *J. Global Optim.*, 21(4):385–396, 2001.

[7] J. D. Cho. A min-cost flow based min-cost rectilinear Steiner distance-preserving tree construction. In *ISPD*, pages 82–87, 1997.

[8] J. Cong, A. B. Kahng, and K. S. Leung. Efficient algorithms for the minimum shortest path Steiner arborescence problem with applications to vlsi physical design. *IEEE Trans. on CAD of Integrated Circuits and Systems*, 17(1):24–39, 1998.

[9] R. R. Ladeira de Matos. A rectilinear arborescence problem. *Dissertation, University of Alabama*, 1979.

[10] M. R. Garey and D. S. Johnson. The rectilinear Steiner tree problem is NP-complete. *SIAM J. Appl. Math.*, 32(4):826–834, 1977.

[11] D. Halperin, J. C. Latombe, and R. Motwani. Dynamic maintenance of kinematic structures. In *J.P. Laumond and M. Overmars, editors, Algorithmic Foundations of Robotics. A.K. Peters Publishing*, pages 155–170, 1997.

[12] F. K. Hwang and D. S. Richards. Steiner tree problems. *Networks*, 22(1):55–897, 1992.

[13] A. Kahng and G. Robins. On optimal interconnects for VLSI. *Kluwer Academic Publishers*, 1995.

[14] E. Kantor and S. Kutten. Optimal competitiveness for symmetric rectilinear Steiner arborescence and related problems. *CoRR*, abs/1307.3080, 2013.

[15] E. Kantor and S. Kutten. Optimal competitiveness for symmetric rectilinear Steiner arborescence and related problems. In *ICALP(2)*, pages 520–531, 2014.

[16] B. Lu and L. Ruan. Polynomial time approximation scheme for rectilinear Steiner arborescence problem. *Combinatorial Optimization*, 4(3):357–363, 2000.

[17] L. Nastansky, S. M. Selkow, and N. F. Stewart. Cost minimum trees in directed acyclic graphs. *Z. Oper. Res.*, 18:59–67, 1974.

[18] C.H. Papadimitriou, S. Ramanathan, and P.V. Rangan. Information caching for delivery of personalized video programs for home entertainment channels. In *IEEE International Conf. on Multimedia Computing and Systems*, pages 214–223, 1994.

[19] C.H. Papadimitriou, S. Ramanathan, and P.V. Rangan. Optimal information delivery. In *6th ISAAC*, pages 181–187, 1995.

[20] C.H. Papadimitriou, S. Ramanathan, P.V. Rangan, and S. Sampathkumar. Multimedia information caching for personalized video-on demand. *Computer Communications*, 18(3):204–216, 1995.

[21] S. Rao, P. Sadayappan, F. Hwang, and P. Shor. The Rectilinear Steiner Arborescence problem. *Algorithmica*, pages 277–288, 1992.

[22] W. Shi and C. Su. The rectilinear Steiner arborescence problem is NP-complete. In *SODA*, pages 780–787, 2000.

[23] V.A. Trubin. Subclass of the Steiner problems on a plane with rectilinear metric. *Cybernetics and Systems Analysis*, 21(3):320–324, 1985.