Optimal competitiveness for the Rectilinear Steiner Arborescence problem

Erez Kantor[1]* and Shay Kutten[2]**

[1] MIT CSAIL, Cambridge, MA erezk@csail.mit.edu,
[2] Technion, Haifa 32000, Israel. 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).

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[3] 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. [19, 20] and Charikar et al. [5] presented the discrete version, in the context of Multimedia Content Delivery (MCD) to model locating and moving caches

* Supported in a part by NSF Awards 0939370-CCF, CCF-1217506 and CCF-AF-0937274 and AFOSR FA9550-13-1-0042.
** Supported in part by the ISF, Israeli ministry of science and by the Technion Gordon Center.

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

for titles on a path graph. The formal definition of (one of the known versions) of this problem, Directed-MCD, appears in Section 2.

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 [17, 22, 4, 18, 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 Halperinet al. [11], who gave a similar competitive ratio of $O(\log N)$, while Charikaret al. [5] presented a lower bound of $\Omega(\sqrt{\log n})$ for various variants of MCD, where n was the network size. Their upper bound was again the square of the lower bound. 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 et al. [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 [18] 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 [24], which seemed surprising, since many Steiner problems are NP-Hard. Indeed, difficulties in that solution were noted by Rao et al. [22], who also presented an approximation algorithm. Efficient algorithms are claimed in [7] for VLSI applications. However, the problem was proven NP-Hard in [23]. (The rectilinear Steiner tree problem was proven NP-Hard in [10]). Heuristics that are fast "in practice" were presented in [8]. A PTAS was presented by [17].

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.

Because of space considerations, some of the proofs are omitted. However, all the proofs are given in the full version [16]. Moreover, the best way to understand the algorithms in this paper may be from a geometric point of view. Hence, in [16], we added multiple drawings to illustrate both the algorithms and the proofs.

2 Preliminaries

The network$\times$time grid (Papadimitriou et. al, [20]). 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. 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. We consider the time as if it proceeds upward. We use such geometric presentations also in the text, to help clarifying the description.

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, 19, 21, 20, 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 $J[s, t]$ 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 $\mathcal{P}_{\mathcal{H}}[(u, t), (v, t)]$ 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 $dist_{\infty}^{\rightarrow}((u, s), (v, t))$ be the "directed" distance from (u, s) to (v, t) in L_{∞} norm. Formally, $dist_{\infty}^{\rightarrow}((u, s), (v, t)) = \max\{t - s, v - u\}$, if $s \leq t$ and $u \leq v$ and $dist_{\infty}^{\rightarrow}((u, s), (v, t)) = \infty$, 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.

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

(SQ2) Let replica $q_i^{\text{close}} = (u_i^{\text{close}}, s_i^{\text{close}})$ 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, $s_i^{\text{close}} = \max\{t \le t_i \mid (u_i^{\text{close}}, t) \in \text{SQUARE}(i-1)\}$. Define the *radius* of r_i as $\rho^{\text{SQ}}(i) = dist_\infty^{\rightarrow}(q_i^{\text{close}}, r_i) = \max\{|v_i - u_i^{\text{close}}|, |t_i - s_i^{\text{close}}|\}$. 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) .

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

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$(i) = \{(u, t_i) \mid u_i^{\text{serve}} \le u \le v_i\}$ and let BASE $= \cup_{i=1}^{N}$BASE(i) (notice that BASE $\subseteq \mathcal{F}^{\text{SQ}}$ because of step SQ4). Similarly, let TAIL$(i) = \{(u_i^{\text{serve}}, t) \mid t_i \le t \le t_i + 4\rho^{\text{SQ}}(i)\}$ 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 $(0, 0)$.

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

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

(For lack of space, some of the proofs are omitted. Still, Observation 1 is obvious from the description of SQUARE.) 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 $Q\text{-BALL}^{\text{SQ}}(r_i, \rho^{\text{SQ}}(i))$ [5] (also $Q\text{-BALL}^{\text{SQ}}(i)$ for short) the quarter-ball centered at r_i with radius $\rho^{\text{SQ}}(i)$.

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

Observation 2 *Consider some subset $\mathcal{R}' \subseteq \mathcal{R}$ of requests. If the Q-balls, $Q\text{-BALL}^{\text{SQ}}(i)$ and $Q\text{-BALL}^{\text{SQ}}(j)$, of every two requests $r_i, r_j \in \mathcal{R}'$ are edges disjoint, then $|\text{OPT}| \geq \sum_{r_i \in \mathcal{R}'} \rho^{\text{SQ}}(i)$.*

Covered and uncovered requests. Consider some request $r_i = (v_i, t_i)$ and its serving replica $q_i^{\text{serve}} = (u_i^{\text{serve}}, s_i^{\text{serve}})$ (see step SQ3). We say that r_i is *covered*, if $v_i - u_i^{\text{serve}} \geq \rho^{\text{SQ}}(i)$ (see SQ2 and SQ3). Intuitively, this means the solution $\mathcal{F}^{\text{SQ}}$ is augmented by the whole top of the square SQUARE$[r_i, \rho^{\text{SQ}}(i)]$.Otherwise, we say that r_i is *uncovered*. Let COVER $= \{i \mid r_i \text{ is a covered request}\}$ and let UNCOVER $= \{i \mid r_i$ is an **un**covered request$\}$. Given Observation 2, the following lemma implies that

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

Lemma 1. *Consider two* **covered** *requests r_i and r_j. The quarter balls $Q\text{-BALL}^{\text{SQ}}(i)$ and $Q\text{-BALL}^{\text{SQ}}(j)$ are edge disjoint.*

The lemma follows easily from geometric considerations, see figures 5–6 and the proof in [16]. By observations 1, 2, and Inequality (1), we have:

Observation 3 SQUARE*'s cost for covered requests is no more than* $14 \cdot \text{OPT}$.

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

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 in [16]. Below, we partition the balls

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

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 1 and Observation 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 $Q\text{-BALL}^{\text{SQ}}(j)$ is the *child* of $Q\text{-BALL}^{\text{SQ}}(i)$ (for two *uncovered* requests r_i and r_j) intuitively, if the $Q\text{-BALL}^{\text{SQ}}(i)$ is the first ball (of a request later then r_j) such that $Q\text{-BALL}^{\text{SQ}}(i)$ and $Q\text{-BALL}^{\text{SQ}}(j)$ 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 4 *The quarter balls of every two root requests are edge disjoint.*

The above observation together with Observation 2, implies the following.

Observation 5 *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 1 and Observation 2. To show that, given any non leaf ball $Q\text{-BALL}^{\text{SQ}}(i)$ (not just a root), we first analyze only $Q\text{-BALL}^{\text{SQ}}(i)$'s "latest child" $Q\text{-BALL}^{\text{SQ}}(j)$. That is, $j = \max_k\{Q\text{-BALL}^{\text{SQ}}(k) \text{ is a child of } Q\text{-BALL}^{\text{SQ}}(i)\}$. We show that the radius of the latest child is, at most, a quarter of the radius of $Q\text{-BALL}^{\text{SQ}}(i)$. 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 $Q\text{-BALL}^{\text{SQ}}(i)$ 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. 1.

Lemma 2. *Consider some uncovered request r_i which has at least two children. Let $Q\text{-BALL}^{\text{SQ}}(j)$, $Q\text{-BALL}^{\text{SQ}}(k)$ some two children of $Q\text{-BALL}^{\text{SQ}}(i)$, 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 $Q\text{-BALL}^{\text{SQ}}(j)$ is not before the time of the latest replica in $\text{TAIL}(k)$. Intuitively, recall that the tail 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

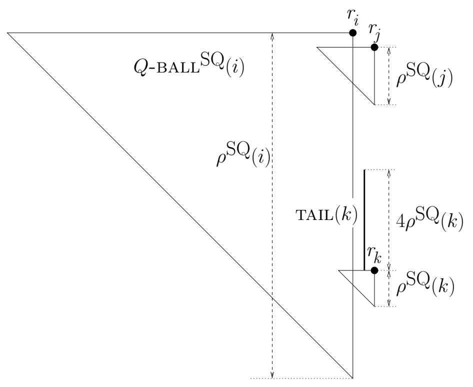

Fig. 1. Geometric look on a parent Q-BALL$^{\text{SQ}}(i)$ (note that a Q-BALL is a triangle) and its children Q-BALL$^{\text{SQ}}(j)$ and Q-BALL$^{\text{SQ}}(k)$.

Lemma 2, 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 2 (appears in [16]) 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. *Suppose that, $j > \ell$ and $v_j - \rho^{\text{SQ}}(j) + 1 \leq u_\ell^{\text{serve}} \leq v_j$. Then, the time of the earliest replica in Q-BALL$^{\text{SQ}}(j)$ is not before the time of the latest replica in TAIL(ℓ), i.e., $t_j - \rho^{\text{SQ}}(j) \geq t_\ell + 4\rho^{\text{SQ}}(\ell)$.*

Intuitively, Lemma 3 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 2, 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, 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. 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 to show that the time of the earliest replica in Q-BALL$^{\text{SQ}}(j)$ 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 $|\text{OPT}|$; see Observation 3. (2) For *uncovered* requests, we prove in [16] (as overviewed above) two facts: (2.a) the Q-BALLs of the root requests are edges disjoint, and hence by Observation 5, the sum of their radii is $O(1)$ of $|\text{OPT}|$ 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 6.

Theorem 6. *Algorithm* SQUARE *is* $O(1)$*-competitive for* DMCD *under the pseudo online model.*

4 Algorithm D-Line$^{\text{on}}$ - the "real" online algorithm

In this section, we transform the pseudo online algorithm Square of Section 3 into a (fully) online algorithm D-Line$^{\text{on}}$ for DMCD. The full details as well as the formal proof of this transformation appears in [16][6]. Let us nevertheless 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 $s_i^{\text{serve}} < t_i$. A (full) online algorithm cannot perform such an action. Intuitively, Algorithm D-Line$^{\text{on}}$ "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$^{\text{on}}$ also (3) delivers first a copy to $(u_i^{\text{serve}}, t_i)$ 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$^{\text{on}}$ 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" ("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).

Theorem 7. 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

[6] 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).

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 $SRSA$.

Theorem 8. *Algorithm* RSA$^{\text{on}}$ *is optimal and is* $O(\frac{\log N}{\log\log N})$*-competitive.*

5.1 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, $O(\min\{\frac{\log N}{\log\log N}, \frac{\log n}{\log\log n}\})$, see [16].

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 9. *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. The full proof appears in [16]. ∎

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. *ICALP*, pp 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) , 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. E. Kantor and S. Kutten. Optimal competitiveness for the rectilinear steiner arborescence problem. *CoRR*, arxiv.org/abs/1504.08265, 2015.
17. B. Lu and L. Ruan. Polynomial time approximation scheme for rectilinear Steiner arborescence problem. *Combinatorial Optimization*, 4(3):357–363, 2000.
18. L. Nastansky, S. M. Selkow, and N. F. Stewart. Cost minimum trees in directed acyclic graphs. *Z. Oper. Res.*, 18:59–67, 1974.
19. 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.
20. C.H. Papadimitriou, S. Ramanathan, and P.V. Rangan. Optimal information delivery. In *6th ISAAC*, pages 181–187, 1995.
21. 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.
22. S. Rao, P. Sadayappan, F. Hwang, and P. Shor. The Rectilinear Steiner Arborescence problem. *Algorithmica*, pages 277–288, 1992.
23. W. Shi and C. Su. The rectilinear Steiner arborescence problem is NP-complete. In *SODA*, pages 780–787, 2000.
24. V.A. Trubin. Subclass of the Steiner problems on a plane with rectilinear metric. *Cybernetics and Systems Analysis*, 21(3):320–324, 1985.