Frameworks for Nonclairvoyant Network Design with Deadlines or Delay

Noam Touitou[1]
Amazon, Tel Aviv, Israel

Abstract

Clairvoyant network design with deadlines or delay has been studied extensively, culminating in an $O(\log n)$-competitive general framework, where n is the number of possible request types (Azar and Touitou, FOCS 2020). In the nonclairvoyant setting, the problem becomes much harder, as $\Omega(\sqrt{n})$ lower bounds are known for certain problems (Azar et al., STOC 2017). However, no frameworks are known for the nonclairvoyant setting, and previous work focuses only on specific problems, e.g., multilevel aggregation (Le et al., SODA 2023).

In this paper, we present the first nonclairvoyant frameworks for network design with deadlines or delay. These frameworks are nearly optimal: their competitive ratio is $\tilde{O}(\sqrt{n})$, which matches known lower bounds up to logarithmic factors.

2012 ACM Subject Classification Theory of computation; Theory of computation → Design and analysis of algorithms; Theory of computation → Online algorithms

Keywords and phrases Online, Deadlines, Delay, Network Design, Nonclairvoyant

Digital Object Identifier 10.4230/LIPIcs.ICALP.2023.105

Category Track A: Algorithms, Complexity and Games

1 Introduction

In network design problems with deadlines, online connectivity requests arrive over time, such that every request must be served by its associated deadline. A solution serves pending requests by transmitting sets of items at various times, thus incurring some cost; the online algorithm constructs the solution by deciding, irrevocably, whether to make a transmission at any given time (and which items to transmit). A concrete example is Steiner tree with deadlines, in which a weighted graph with a designated root node is given offline, and each connectivity request names a terminal node to be connected to the root. Each transmission consists of a set of edges, and serves each pending request if its terminal is connected to the root by the transmitted edges; the cost of the transmission is the total cost of edges. In network design with delay, in lieu of a deadline, every request accumulates delay cost while pending, thus motivating quicker service by the algorithm.

A parameter that influences the difficulty of such problems is called *clairvoyance*. In the clairvoyant model, upon the arrival of a request, the algorithm learns its deadline (in the deadline case) or future delay accumulation (in the delay case). However, in the nonclairvoyant deadline model, the algorithm only learns the deadline of a request upon its expiration (and must then immediately serve the request if pending). Similarly, in the nonclairvoyant delay model, the algorithm is only aware of delay accumulated until the current time.

Various network design problems with deadlines or delay were studied in the clairvoyant setting. This includes problems such as multilevel aggregation [6, 11, 3, 5, 27], facility location [3, 7], TCP Acknowledgement [17, 24, 12] and joint replenishment [13, 10, 8, 16].

[1] This work does not relate to the author's position at Amazon.

For general, clairvoyant network design with deadlines or delay, algorithmic frameworks were presented in [4]. These frameworks yield logarithmic competitiveness with respect to multiple parameters: the number of requests, which we denote m; the number of request *types* (e.g., number of possible terminals for Steiner tree), which we denote n; and the number of items in the item set[2].

The nonclairvoyant setting, however, is less thoroughly studied. Some specific problems of the network design with deadlines/delay class gracefully handle nonclairvoyance; for example, for nonclairvoyant set cover with delay, logarithmic competitiveness is known [1]. Other problems, such as joint replenishment, multilevel aggregation, and facility location with deadlines, have $\Omega(\sqrt{n})$ and $\Omega(\sqrt{m})$ lower bounds on competitiveness in the nonclairvoyant setting, even for randomized algorithms [26][3]. (Note that an earlier, deterministic lower bound along the same lines appears in [2] for the service with delay problem.) Recently, Le et al. [26] presented matching and nearly-matching upper bounds for nonclairvoyant joint replenishment and multilevel aggregation with delay, respectively. However, unlike in the clairvoyant case, a general algorithmic framework for nonclairvoyant network design with deadlines or delay is still not known.

1.1 Our Results

We present the first frameworks for general network design problems with deadlines or delay in the nonclairvoyant setting. Specifically, with n the number of possible request types and m the number of requests, we present:

1. A deterministic, $O\left(\min\left\{\sqrt{n\log n}, \sqrt{m\log m}\right\}\right)$-competitive framework for network design with deadlines.
2. A deterministic, $O\left(\min\left\{\sqrt{n\log n}, \sqrt{m\log m}\right\}\right)$-competitive framework for network design with delay.

The competitiveness of our frameworks is nearly optimal, as implied by the lower bounds of $\Omega(\sqrt{n})$ and $\Omega(\sqrt{m})$ on competitiveness for some network design problems, e.g., multilevel aggregation [26].

While our frameworks provide nearly-optimal upper bounds for nonclairvoyant network design, some components require specific properties to be implemented in polynomial time. In Section 5, we show how to implement the frameworks in polynomial time for a class of network design problems; specifically, those problems that admit *Lagrangian prize-collecting* algorithms. In particular, we obtain the following results:

1. A poly-time, deterministic, $O\left(\min\left\{\sqrt{n\log n}, \sqrt{m\log m}\right\}\right)$-competitive algorithm for Steiner tree with deadlines/delay. Note that for Steiner tree, n equals the number of nodes in the graph.
2. A poly-time, deterministic, $O\left(\sqrt{m\log m}\right)$-competitive algorithm for facility location with deadlines/delay[4].

[2] Note that the bound with respect to the number of request types is implicit in [4].
[3] The lower bound in [26] is stated in terms of joint replenishment; however, joint replenishment can be seen as a special case of multilevel aggregation and facility location, and thus the lower bound applies to those problems as well.
[4] A fine point regarding facility location is that the number of request types n does not yield a meaningful bound, and thus we only state the bound with respect to the number of requests m. This is discussed in more detail in Section 5.

3. A poly-time, deterministic, $O\left(\min\left\{\sqrt{n}\log n, \sqrt{m}\log m\right\}\right)$-competitive algorithm for multicut with deadlines/delay on a tree. Note that for multicut, n equals the number of node pairs in the graph (i.e., quadratic in the number of nodes).

For nonclairvoyant facility location with deadlines/delay, our algorithm is the first algorithm. This is also the case for nonclairvoyant multicut with deadlines or delay. For nonclairvoyant Steiner tree, there existed no *explicit* previous algorithm; however, randomly embedding into an HST [18] then using the multilevel aggregation algorithm of [26] would yield a randomized $O(\sqrt{n}\log n)$-competitive algorithm, with no guarantee in m. Our algorithm for Steiner tree improves the power of the logarithm, is deterministic, and has a guarantee in m.

1.2 Other Related Work

A related problem to network design with deadlines/delay is online service, in which one is given a server (or multiple servers) on a metric space, which must then be moved to locations specified in incoming online requests. In fact, Steiner tree (and multilevel aggregation) with delay can be seen as a special case of this problem, in which the input forces the server to rest at the root node of the given graph (or tree) through an infinite stream of urgent requests at the root. Service with delay was first introduced by Azar et al. [2], and has since seen additional work [9, 3, 20, 25, 21].

The problem of set cover with delay was introduced in [14]; in this problem, the algorithm must transmit sets containing requested elements. While set cover is not usually considered a network design problem, set cover with delay falls within the network design with delay category, and thus the framework of [4] yields a logarithmic-competitive clairvoyant algorithm. As mentioned earlier, this problem also admits a logarithmic-competitive *nonclairvoyant* algorithm, as presented in [1] (and later derandomized in [26]). The best lower bound for this problem, given in [28], is nearly tight given the upper bound for network design implied by the framework in [4].

2 Preliminaries

We now introduce definitions and notation related to network design with deadlines/delay; to make these concrete, we also apply these definitions to the special case of Steiner tree with deadlines/delay.

Offline network design. In offline network design, one is given a set of items $\mathcal{E}$ with associated costs c, and a set of requests H. (In Steiner tree, an input graph is given with a designated root node; the items are the edges of the input graph, and every request demands connecting some terminal to the root.) A request can be served by any chosen subset of items, and we assume that serving requests is *upwards-closed*: that is, if request $h \in H$ is served by a subset of items $E \subseteq \mathcal{E}$, then it is also served by any superset of E. (In Steiner tree, a set of edges satisfies a request if it contains a path connecting the request's terminal to the root; note that satisfaction is indeed upwards-closed.) A solution to the offline problem is a subset of items $E \subseteq \mathcal{E}$ which serves all requests in H; the cost of the solution is the total cost of items in E.

Online network design with deadlines. In online network design with deadlines, one is also given a set of items $\mathcal{E}$ with costs. Now, however, the requests Q arrive over time; we denote by $m := |Q|$ the number of requests in the online input. Each request $q \in Q$ has a *type*, and we denote the set of all possible request types by $\mathcal{H}$, and define $n := |\mathcal{H}|$; in fact, every request type in the online problem corresponds to a possible request in the offline problem. We denote by r_q the release time of request q, and each request q also has a deadline time d_q by which it must be served. At any point in time, the algorithm may transmit any subset of items $E \subseteq \mathcal{E}$; such a transmission is also called a *service*, and incurs a cost of $c(E) := \sum_{e\in E} c(e)$ for the algorithm. Transmitting E serves all pending requests whose request type is served by E in the offline problem. The goal of the algorithm is to serve every request by its deadline, while minimizing the total cost of services, i.e., $\sum_{\text{service } S} \sum_{e\in E(S)} c(e)$ (where $E(S)$ is the set of items transmitted in S).

For concreteness, consider Steiner tree with deadlines, in which requests for connecting terminals to the root are served by transmitting subsets of edges. As transmissions are momentary, it is meaningful for multiple requests (with different release/deadline times) to request connecting the same terminal v; such requests would belong to the same *request type*, as they would be satisfied by exactly the same subsets of items (i.e., those that contain a path from v to the root). In particular, note that for Steiner tree the number of request types n is equal to the number of nodes in the graph.

Online network design with delay. In the (more general) model with delay, each request q has a nondecreasing, continuous delay function d_q, such that for every $t \geq r_q$ the amount $d_q(t)$ is the total delay cost incurred by q until time t (if it is still pending at that time). The goal of the algorithm in this case is to minimize the total cost of services plus the total delay cost. That is, denoting by C_q the time in which request q is served in the algorithm, the goal is to minimize $\sum_{\text{service } S} \sum_{e\in E(S)} c(e) + \sum_{q\in Q} d_q(C_q)$. For ease of notation, for every set of requests Q' and time t, we define $D(Q',t) := \sum_{q\in Q'} d_q(t)$.

Additional notation. For a request q in online network design with deadlines/delay, we define h_q to be the type of request q. For a set of requests Q', we define $H(Q') \subseteq \mathcal{H}$ to be $\left\{h_q | q \in Q'\right\}$. For a set of request types H, we use $\mathrm{ND}(H)$ to denote the minimum cost of serving these request types; slightly abusing notation, for a set of requests Q', we define $\mathrm{ND}(Q') := \mathrm{ND}(H(Q'))$.

For a set of request types H, we define $\ell(H) := \lceil \log \mathrm{ND}(\{H\}) \rceil$. When considering a single request type, we sometimes write $\ell(h)$ instead of $\ell(\{h\})$. Finally, for request q and set of requests Q', we define $\ell(q) := \ell(h_q)$ and $\ell(Q') := \ell(H(Q'))$.

3 Framework for Network Design with Deadlines

In this section, we present a framework for nonclairvoyant network design with deadlines. Analyzing this framework, we obtain the following theorem.

▶ **Theorem 1.** *There exists a deterministic,* $O(\min\left\{\sqrt{n\log n}, \sqrt{m\log m}\right\})$*-competitive algorithm for network design with deadlines.*

Algorithm's description. The algorithm assigns levels to each request according to the logarithm of the cost of serving that request. Upon the deadline of a pending request q, the algorithm starts a service λ that serves the request, thus incurring a cost of at most $2^{\ell(q)}$; the level of the service, denoted ℓ_λ, is defined to be the level of the triggering request q. If a service is *large*, it also serves additional requests; it does so by identifying a set of pending request types which can be served, subject to some specific budget for every request type.

Otherwise, a service is *small*, and does not serve additional requests. Whether a service is large or small is determined by the triggering request q, i.e., the request upon whose deadline the service is started; specifically, the service triggered by the deadline of q will be large if and only if the variable b_q is True at that time.

Specifically, suppose a large service λ takes place. The service will identify the *eligible* requests, i.e., the pending requests whose level is at most ℓ_λ. Denoting by H the set of request types of those eligible requests, the service will give a budget of $\tilde{\Theta}(2^{\ell_\lambda}/\sqrt{|H|})$ to each request type, and will attempt to find a subset of requests that can be served without exceeding the budget for those request types. Thus, the cost of such a large service is at most $\tilde{\Theta}(2^{\ell_\lambda} \cdot \sqrt{|H|})$.

The variable b_q, which controls whether q will trigger a large service, is initially True for every request q. However, if a large service of level at least $\ell(q)$ takes place while q is pending, the variable is set to False. Thus, a request that "experienced" a large service for which it was eligible will never trigger a large service upon its deadline. The formal description of the algorithm is given in Algorithm 1.

Algorithm 1 Nonclairvoyant framework for network design with deadlines.

```
Event Function UponRequest($q$)
    $b_q \leftarrow$ True.
Event Function UponDeadline($q$)
    start a new service $\lambda$, and set $\ell_\lambda = \ell(q)$.
    define $E_\lambda$ to be the set of pending requests of level at most $\ell_\lambda$, and define $H \leftarrow H(E_\lambda)$.
    transmit ND($\{h_q\}$), serving all requests of type $h_q$.
    if $b_q$ then
        let $x_\lambda \leftarrow 2^{\ell_\lambda} \cdot \sqrt{\frac{\log(1+|H|)}{|H|}}$.
        let $H' \subseteq H$ be a maximal subset such that ND$(H') \le x_\lambda \cdot |H'|$.
        transmit ND$(H')$, serving all requests of types in $H'$.
        // set $b_{q'}$ for eligible requests $q'$ which are still pending.
        let $Q_\lambda$ be the subset of requests served by $\lambda$.
        foreach $q' \in E_\lambda \setminus Q_\lambda$ do
            set $b_{q'} \leftarrow$ False.
```

3.1 Analysis

We now focus on proving Theorem 1.

▶ **Definition 2.** *We define the following terms:*

1. *We denote by Λ, Λ^* the set of services in the algorithm and in the optimal solution, respectively.*
2. *For a service $\lambda \in \Lambda$, we define Q_λ to be the set of requests served by (the transmissions of) λ. For an optimal service $\lambda^* \in \Lambda^*$, we define Q_{λ^*} in a similar way.*
3. *For a service $\lambda \in \Lambda$, we define $\ell_\lambda, E_\lambda, x_\lambda$ to be the values of the variables of those names in* UponDeadline.
4. *We define $H_\lambda = H(E_\lambda)$. As a shorthand, we define $y_\lambda = |H_\lambda|$.*
5. *We define the* triggering request *of λ, denoted $q^\star_\lambda$, to be the request upon whose deadline λ was initiated.*
6. *We define $c(\lambda)$ to be the total transmission cost incurred in service λ.*

7. *We say that an algorithm service $\lambda \in \Lambda$ is* charged *to an optimal service λ^* if $q^\star_\lambda \in Q_{\lambda^*}$. For every $\lambda^* \in \Lambda^*$, we define $\Lambda_{\lambda^*} \subseteq \Lambda$ to be the set of services charged to λ^*.*
8. *We call a service λ a* large *service if upon the start of λ we have $b_{q^\star_\lambda} = $ TRUE. Otherwise, λ is a* small *service.*
9. *For every service λ in the algorithm or in the optimal solution, denote by t_λ the time in which the service takes place.*

We define k to be the maximum size of $H(E_\lambda)$ over all services λ. In particular, note that $k \le \min\{n, m\}$, which allows us to prove our competitiveness bounds with respect to k. Fix henceforth any optimal service $\lambda^* \in \Lambda^*$.

▶ **Proposition 3.** *For every $\lambda \in \Lambda_{\lambda^*}$, it holds that $t_\lambda \ge t_{\lambda^*}$.*

Proof. Note that λ is triggered by the deadline of request $q^\star_\lambda$. From the definition of Λ_{λ^*}, we have that $q^\star_\lambda \in Q_{\lambda^*}$; thus, $t_{\lambda^*} \le d_{q^\star_\lambda} = t_\lambda$. ◀

We partition Λ_{λ^*}, into three parts, the costs of which we bound individually:

- $\Lambda^1_{\lambda^*}$: The large services in Λ_{λ^*}.
- $\Lambda^2_{\lambda^*}$: The small services λ such that $b_{q^\star_\lambda}$ was first set to FALSE at time smaller than t_{λ^*}.
- $\Lambda^3_{\lambda^*}$: The small services λ such that $b_{q^\star_\lambda}$ was first set to FALSE at time at least t_{λ^*}.

▶ **Proposition 4.** *For every level ℓ, there exists at most one level-ℓ service in $\Lambda^1_{\lambda^*}$.*

Proof. Assume otherwise that there exist $\lambda_1, \lambda_2 \in \Lambda_{\lambda^*}$ which are both large, and assume without loss of generality that $t_{\lambda_2} > t_{\lambda_1}$. From the previous claim, we have that $t_{\lambda_1} \ge t_{\lambda^*}$, and thus $q^\star_{\lambda_2}$ is pending at t_{λ_1}. Moreover, as $\ell\left(q^\star_{\lambda_2}\right) = \ell_{\lambda_1} = \ell$, we have that $q^\star_{\lambda_2} \in E_{\lambda_1}$. But Line 13 of λ_1 sets $b_{q^\star_{\lambda_2}}$ to be FALSE, and this value is maintained until λ_2. This is in contradiction to λ_2 being a large service. ◀

▶ **Proposition 5.** *For every ℓ, we have $\sum_{\lambda \in \Lambda^1_{\lambda^*} | \ell_\lambda = \ell} c(\lambda) = O(\sqrt{k \log k}) \cdot 2^\ell$.*

Proof. From Proposition 4, it holds that $\Lambda^1_{\lambda^*}$ contains at most one (large) service of level ℓ, and thus the left-hand side of the equation contains at most one summand. The cost of a large service λ of level ℓ consists of two costs: the first cost is the cost of transmitting $\mathrm{ND}(h_{q^\star_\lambda})$, which is at most 2^ℓ, using the fact that $\ell(q^\star_\lambda) = \ell$; the second cost is the cost of the transmission in Line 10, which is at most

$$y_\lambda \cdot x_\lambda \le 2^\ell \cdot \sqrt{y_\lambda \log(1 + y_\lambda)} = O(\sqrt{k \log k}) \cdot 2^\ell$$ ◀

▶ **Proposition 6.** *For every ℓ, we have*

$$\sum_{\lambda \in \Lambda^2_{\lambda^*} | \ell_\lambda = \ell} c(\lambda) = O(\sqrt{k / \log k}) \cdot c(\lambda^*).$$

Proof. Fix any ℓ. We define $\Lambda' := \left\{\lambda \in \Lambda^2_{\lambda^*} | \ell_\lambda = \ell\right\}$ and, as a shorthand, define $z := |\Lambda'|$. We define R to be the set of triggering requests of services in Λ'. First, we claim that the request types of triggering requests of services in Λ' are distinct, i.e., that $|H(R)| = |\Lambda'| = z$. To prove this claim, assume for contradiction that there exist two services in $\lambda_1, \lambda_2 \in \Lambda'$ with triggering requests of the same type, and further assume without loss of generality that $t_{\lambda_1} < t_{\lambda_2}$. Since $q^\star_{\lambda_2} \in Q_{\lambda^*}$, it must be pending at t_{λ^*}; moreover, Proposition 3 implies that $t_{\lambda^*} \le t_{\lambda_1}$, and thus $q^\star_{\lambda_2}$ is pending at t_{λ_1}. But then λ_1 would serve $q^\star_{\lambda_2}$ in Line 6, in contradiction to $q^\star_{\lambda_2}$ triggering λ_2. Thus, the proof of the claim is complete.

Now, consider the first large service $\lambda \in \Lambda$ after which for every triggering request q of a service in Λ' we have $b_q = \textsc{False}$; it must be that $\ell_\lambda \geq \ell$. From the definition of Λ', it holds that $t_\lambda < t_{\lambda^*}$; combining this with Proposition 3, we have that $t_\lambda < t_{\lambda'}$ for every $\lambda' \in \Lambda'$. In particular, all triggering requests of services from Λ' must be pending at t_λ, and since they are of level ℓ, these requests are also in E_λ. However, they all remain pending after λ, which means that $\mathrm{ND}(H(R)) \geq x_\lambda \cdot |H(R)|$. (Otherwise, we would get a contradiction to the maximality of the set H' defined in Line 9, as $\mathrm{ND}(H(R))$ could be added to the solution without exceeding budget.) We thus have

$$c(\lambda^*) \geq \mathrm{ND}(H(R)) \geq x_\lambda \cdot |H(R)| \geq 2^\ell \cdot \sqrt{\frac{\log(1+k)}{k}} \cdot z \tag{1}$$

where the third inequality uses the fact that $y_\lambda \leq k$. Meanwhile, the total cost of services in Λ' is at most $z \cdot 2^\ell$. Combining with Equation (1) completes the proof. ◀

▶ **Proposition 7.** *For every ℓ, we have $\sum_{\lambda\in\Lambda^3_{\lambda^*}|\ell_\lambda=\ell} c(\lambda) = O(\sqrt{k/\log k}) \cdot c(\lambda^*)$.*

Proof. For ease of notation, define $\Lambda' := \left\{\lambda \in \Lambda^3_{\lambda^*} | \ell_\lambda = \ell\right\}$. We also define R to be the set of triggering requests for requests in Λ'. We define $z := |\Lambda'|$; using an identical argument to that in the proof of Proposition 6, it holds that $|H(R)| = z$. Let λ be the first large service of level at least ℓ such that $t_\lambda \geq t_{\lambda^*}$. Note that the following hold:

1. $t_\lambda < t_{\lambda'}$ for every $\lambda' \in \Lambda'$ (stems from the definition of Λ').
2. R are all pending at t_λ and eligible for λ (as $t_\lambda \geq t_{\lambda^*}$).
3. λ changes b_q from $\textsc{True}$ to $\textsc{False}$ for every $q \in R$.

Since R were all eligible for λ but were not served, it must be the case that $\mathrm{ND}(H(R)) \geq x_\lambda \cdot z \geq 2^\ell \cdot \sqrt{z \log(1+z)}$; since λ^* serves R, it thus holds that $c(\lambda^*) \geq 2^\ell \cdot \sqrt{z\log(1+z)}$. We therefore have $\sum_{\lambda\in\Lambda^3_{\lambda^*}|\ell_\lambda=\ell} c(\lambda) \leq 2^\ell \cdot z \leq \sqrt{z/\log(1+z)} \cdot c(\lambda^*)$. We now note that all requests in R were pending during λ; thus, $z \leq k$, which completes the proof. ◀

Proof of Theorem 1. We first observe that $\mathrm{ALG} = \sum_{\lambda\in\Lambda} c(\lambda) = \sum_{\lambda^*\in\Lambda^*}\sum_{\lambda\in\Lambda_{\lambda^*}} c(\lambda)$. We claim that for every $\lambda^* \in \Lambda^*$, we have $\sum_{\lambda\in\Lambda_{\lambda^*}} c(\lambda) \leq O(\sqrt{k\log k}) \cdot c(\lambda^*)$; since $\mathrm{OPT} = \sum_{\lambda^*\in\Lambda^*} c(\lambda^*)$, proving this claim completes the proof of the theorem.

To prove the claim, fix any optimal service $\lambda^* \in \Lambda^*$, and define $\ell := \lceil \log c(\lambda^*) \rceil$. Note that for every service $\lambda \in \Lambda_{\lambda^*}$ we have $\ell_\lambda \leq \ell$; this is since $q^\star_\lambda$ is served by λ^*, which implies $\mathrm{ND}(q^\star_\lambda) \leq c(\lambda^*)$. We partition Λ_{λ^*} into $\Lambda^1_{\lambda^*}, \Lambda^2_{\lambda^*}, \Lambda^3_{\lambda^*}$ as before, and bound each set separately.

First, we bound the cost of $\Lambda^1_{\lambda^*}$ as follows.

$$\begin{aligned}\sum_{\lambda\in\Lambda^1_{\lambda^*}} c(\lambda) &= \sum_{\ell'\leq\ell}\sum_{\lambda\in\Lambda^1_{\lambda^*}|\ell_\lambda=\ell'} c(\lambda) \leq \sum_{\ell'\leq\ell} O(\sqrt{k\log k})\cdot 2^{\ell'} \\ &\leq O(\sqrt{k\log k}) \cdot 2^\ell \leq O(\sqrt{k\log k}) \cdot c(\lambda^*)\end{aligned} \tag{2}$$

where the first inequality uses Proposition 5, and the second inequality bounds a geometric series.

Defining $\gamma := \lceil \log k \rceil$, we bound the cost of $\Lambda^2_{\lambda^*}$.

$$\begin{aligned}\sum_{\lambda\in\Lambda^2_{\lambda^*}} c(\lambda) &= \sum_{\lambda\in\Lambda^2_{\lambda^*}|\ell_\lambda\leq\ell-\gamma} c(\lambda) + \sum_{\lambda\in\Lambda^2_{\lambda^*}|\ell-\gamma<\ell_\lambda\leq\ell} c(\lambda) \leq \sum_{\ell'\leq\ell-\gamma} k\cdot 2^{\ell'} + \sum_{\lambda\in\Lambda^2_{\lambda^*}|\ell-\gamma<\ell_\lambda\leq\ell} c(\lambda) \\ &\leq \sum_{\ell'\leq\ell-\gamma} k\cdot 2^{\ell'} + \gamma\cdot O(\sqrt{k/\log k})\cdot c(\lambda^*) \leq 2\cdot 2^\ell + \gamma \cdot O(\sqrt{k/\log k}) \cdot c(\lambda^*) \\ &\leq O(\sqrt{k\log k}) \cdot c(\lambda^*)\end{aligned} \tag{3}$$

Here, the first inequality is due to the fact that the total cost of level-ℓ' small services in Λ_{λ^*} cannot exceed $k \cdot 2^{\ell'}$. (As seen in the proof of Proposition 6, the requests of level ℓ' triggering small services are of distinct request types, and are all eligible in a single service λ; thus, their number is at most k.) The second inequality is through using Proposition 6, the third inequality is through the definition of γ and through summing a geometric sequence, and the fourth inequality notes that through the definition of ℓ we have $c(\lambda^*) \geq 2^{\ell-1}$.

Replacing Proposition 6 with Proposition 7, an identical argument to the one used for bounding $\Lambda^2_{\lambda^*}$ can be used for bounding $\Lambda^3_{\lambda^*}$, yielding the following:

$$\sum_{\lambda \in \Lambda^3_{\lambda^*}} c(\lambda) = O(\sqrt{k \log k}) \cdot c(\lambda^*) \tag{4}$$

Combining Equations (2) to (4) yields $\sum_{\lambda \in \Lambda_{\lambda^*}} c(\lambda) \leq O(\sqrt{k \log k}) \cdot c(\lambda^*)$, completing the proof. ◀

4 Framework for Network Design with Delay

In this section, we present a framework for nonclairvoyant network design with delay. Using this framework, we prove the following theorem.

▶ **Theorem 8.** *There exists a deterministic, $O\left(\min\left\{\sqrt{n \log n}, \sqrt{m \log m}\right\}\right)$-competitive algorithm for network design with delay.*

4.1 The Algorithm

We now describe the framework for nonclairvoyant network design with delay. For every time t and set of requests Q' which are pending at t, we say that Q' are *critical* at t if $D(Q', t) \geq \text{ND}(Q')$.

Framework's description. The framework for delay contains many analogues to the deadline framework of Section 3. In the deadline case, a service was triggered upon the deadline of a pending request; in the delay case, the framework initiates a service whenever a set of pending requests becomes "critical", which is when its accumulated delay justifies its service. In the deadline case, whether a service triggered by request q was large is determined by the associated variable b_q. In the delay case, we also maintain the variable b_q for every pending request q.

However, the delay case introduces an additional complication: the triggering set contains multiple requests, and thus multiple values for the variables b_q. Where services for deadlines were either "large" or "small", in the delay case this distinction is no longer binary: services identify a budget for expansive service which depends on the large requests in the triggering set.

The service considers the requests q in its triggering set with $b_q = \textsc{True}$, and finds the largest subset of those requests whose delay is at least a constant fraction of its service cost. This subset is considered "mature" enough to justify expansive service, and its service cost is used as a budget for serving pending requests. Thus, where in deadlines the level of the service was simply the level of the triggering requests, for delay the level depends on the cost of the "mature" subset of large requests.

Algorithm 2 Nonclairvoyant framework for network design with delay.

```
1  Event Function UponRequest($q$)
2      $b_q \leftarrow$ True.
3  Event Function UponCritical($R$) // called when the delay of some set $R$ exceeds ND($R$)
4      Start a new service $\lambda$; denote the current time by $t$.
5      Define $R^\top \leftarrow \{q \in R | b_q = \text{True}\}$.
6      Let $R^\star \subseteq R^\top$ be a maximal subset such that $D(R^\star, t) \geq \frac{\text{ND}(H(R^\star))}{2}$.
7      Define $\ell_\lambda \leftarrow \ell(R^\star)$.
8      Define $E$ to be the set of pending requests of level at most $\ell_\lambda$, and define $H \leftarrow H(E)$.
9      Transmit ND($H(R)$), serving all requests of types $H(R)$.
10     Let $x_\lambda \leftarrow 2^{\ell_\lambda} \cdot \sqrt{\frac{\log(1+|H|)}{|H|}}$.
11     Let $H' \subseteq H$ be a maximal subset such that $\text{ND}(H') \leq |H'| \cdot x_\lambda$.
12     Transmit ND($H'$), serving all requests of types in $H'$.
       // set $b_{q'}$ for eligible requests $q'$ which are still pending.
13     let $Q_\lambda$ be the subset of pending requests served by $\lambda$.
14     foreach $q' \in E \setminus Q_\lambda$ do
15         set $b_{q'} \leftarrow$ False.
```

4.2 Analysis

Fix any optimal service $\lambda^* \in \Lambda^*$.

▶ **Definition 9.** *For a service $\lambda \in \Lambda$, define $R_\lambda, R_\lambda^\top, R_\lambda^\star, E_\lambda$ to be the values of the variables $R, R^\top, R^\star, E$ in the call to* UponCritical *which started λ. Moreover, we define $R_\lambda^\perp := R_\lambda \backslash R_\lambda^\top$; these are the requests $q \in R_\lambda$ such that $b_q =$ False at t_λ. In addition, define ℓ_λ, Q_λ as they are defined in the call to* UponCritical.

For a service $\lambda \in \Lambda$ and an optimal service $\lambda^* \in \Lambda^*$, for ease of notation, when referring to a set of jobs related to λ we add λ^* to the subscript to intersect this set with Q_{λ^*}. For example, we define $R_{\lambda,\lambda^*} := R_\lambda \cap Q_{\lambda^*}$, $R_{\lambda,\lambda^*}^\top := R_\lambda^\top \cap Q_{\lambda^*}$, $R_{\lambda,\lambda^*}^\perp := R_\lambda^\perp \cap Q_{\lambda^*}$ and $R_{\lambda,\lambda^*}^\star := R_\lambda^\star \cap Q_{\lambda^*}$. We also define $\ell_{\lambda,\lambda^*} := \ell\left(R_{\lambda,\lambda^*}^\star\right)$.

We define the cost of a service λ, denoted by $c(\lambda)$, to be the total transmission cost in λ, plus the total delay cost of requests served in λ. We define $c(\lambda^*)$ identically for an optimal service $\lambda^* \in \Lambda^*$.

▶ **Proposition 10.** *For a service λ, it holds that*

$$c(\lambda) \leq O(1) \cdot \sum_{\lambda^* \in \Lambda^*} D\left(R_{\lambda,\lambda^*}^\perp, t_\lambda\right) + O(\sqrt{k \log k}) \cdot \sum_{\lambda^* \in \Lambda^*} D\left(R_{\lambda,\lambda^*}^\star, t_\lambda\right)$$

Proof. Since a service is triggered whenever a set of requests becomes critical, the algorithm maintains that for every set Q' of pending requests at any time t we have $D(Q', t) \leq \text{ND}(Q')$. In particular, this holds for the set of requests served by λ with respect to the service time t_λ. Thus, the delay cost of the service can be charged to the transmission costs of the service; we hence focus on bounding the transmission costs of λ.

The service λ performs two transmissions, one at Line 9 and one at Line 12. The first transmission costs ND(R_λ); since R_λ is critical at t_λ, it holds that

$$\text{ND}(R_\lambda) = D(R_\lambda, t_\lambda) = D\left(R_\lambda^\perp, t_\lambda\right) + D\left(R_\lambda^\top \backslash R_\lambda^\star, t_\lambda\right) + D\left(R_\lambda^\star, t_\lambda\right).$$

First, let us bound the delay of requests in $R^{\top}_{\lambda}\backslash R^{\star}_{\lambda}$. From the choice of $R^{\star}_{\lambda}$, it must be the case that $D\big(R^{\top}_{\lambda}\backslash R^{\star}_{\lambda}, t_{\lambda}\big) < \frac{\mathrm{ND}(R^{\top}_{\lambda}\backslash R^{\star}_{\lambda})}{2}$; assuming otherwise, we would have the following:

$$D\big(R^{\top}_{\lambda}, t_{\lambda}\big) = D\big(R^{\star}_{\lambda}, t_{\lambda}\big) + D\big(R^{\top}_{\lambda}\backslash R^{\star}_{\lambda}, t_{\lambda}\big) \geq \frac{\mathrm{ND}(R^{\star}_{\lambda})}{2} + \frac{\mathrm{ND}(R^{\top}_{\lambda}\backslash R^{\star}_{\lambda})}{2} \geq \frac{\mathrm{ND}(R^{\top}_{\lambda})}{2},$$

in contradiction to the maximality in the definition of $R^{\star}_{\lambda}$; thus, $D\big(R^{\top}_{\lambda}\backslash R^{\star}_{\lambda}, t_{\lambda}\big) < \frac{\mathrm{ND}(R^{\top}_{\lambda}\backslash R^{\star}_{\lambda})}{2} \leq \frac{\mathrm{ND}(R_{\lambda})}{2}$. However, we know that $D(R_{\lambda}, t_{\lambda}) = \mathrm{ND}(R_{\lambda})$, and therefore conclude that

$$\mathrm{ND}(R_{\lambda}) \leq 2\big(D\big(R^{\perp}_{\lambda}, t_{\lambda}\big) + D\big(R^{\star}_{\lambda}, t_{\lambda}\big)\big)$$

We have thus bounded the cost of the first transmission.

To bound the cost of the second transmission in λ, let H be the set of pending request types in E_{λ}. We know that the cost of the second transmission is at most $x_{\lambda} \cdot |H| = 2^{\ell_{\lambda}} \cdot \sqrt{|H| \log(1+|H|)}$. Additionally, note that $2^{\ell_{\lambda}} \leq 2 \cdot \mathrm{ND}(R^{\star}_{\lambda}) \leq 4 \cdot D\big(R^{\star}_{\lambda}, t_{\lambda}\big)$, where the second inequality is due to the definition of $R^{\star}_{\lambda}$. Overall, the cost of the second transmission is at most $4 \cdot D\big(R^{\star}_{\lambda}, t_{\lambda}\big) \cdot \sqrt{k \log(1+k)}$.

To summarize, we proved that

$$\begin{aligned} c(\lambda) &\leq O(1) \cdot D\big(R^{\perp}_{\lambda}, t_{\lambda}\big) + O(\sqrt{k \log k}) \cdot D\big(R^{\star}_{\lambda}, t_{\lambda}\big) \\ &= O(1) \cdot \sum_{\lambda^* \in \Lambda^*} D\Big(R^{\perp}_{\lambda,\lambda^*}, t_{\lambda}\Big) + O(\sqrt{k \log k}) \cdot \sum_{\lambda^* \in \Lambda^*} D\Big(R^{\star}_{\lambda,\lambda^*}, t_{\lambda}\Big). \end{aligned}$$ ◀

For every $\lambda \in \Lambda, \lambda^* \in \Lambda^*$ we define the *joint cost* of λ and λ^* as the following.

$$c(\lambda, \lambda^*) := D\Big(R^{\perp}_{\lambda,\lambda^*}, t_{\lambda}\Big) + \sqrt{k \log(1+k)} \cdot D\Big(R^{\star}_{\lambda,\lambda^*}, t_{\lambda}\Big).$$

Through Proposition 10, we have

$$\mathrm{ALG} \leq O(1) \cdot \sum_{\lambda \in \Lambda} \sum_{\lambda^* \in \Lambda^*} c(\lambda, \lambda^*). \tag{5}$$

We henceforth focus on bounding joint costs.

▶ **Proposition 11.** *For every optimal service $\lambda^* \in \Lambda^*$, it holds that*

$$\sum_{\lambda \in \Lambda | t_{\lambda} \leq t_{\lambda^*}} c(\lambda, \lambda^*) \leq O(\sqrt{k \log k}) \cdot c(\lambda^*).$$

Proof. For every service $\lambda \in \Lambda$ such that $t_{\lambda} \leq t_{\lambda^*}$, it holds that

$$c(\lambda, \lambda^*) \leq \sqrt{k \log(1+k)} \cdot D\big(R_{\lambda,\lambda^*}, t_{\lambda}\big) \leq \sqrt{k \log(1+k)} \cdot D\big(R_{\lambda,\lambda^*}, t_{\lambda^*}\big)$$

Note that every request in Q_{λ^*} is critical in at most one service in the algorithm, as critical requests in a service are always served by that service. Thus, summing over different services, we get the following:

$$\begin{aligned} \sum_{\lambda \in \Lambda | t_{\lambda} \leq t_{\lambda^*}} c(\lambda, \lambda^*) &\leq \sqrt{k \log(1+k)} \cdot \sum_{\lambda \in \Lambda | t_{\lambda} \leq t_{\lambda^*}} D\big(R_{\lambda,\lambda^*}, t_{\lambda^*}\big) \\ &\leq \sqrt{k \log(1+k)} \cdot D(Q_{\lambda^*}, t_{\lambda^*}) \leq \sqrt{k \log(1+k)} \cdot c(\lambda^*) \end{aligned}$$ ◀

We thus bounded the joint cost of services prior to λ^*. It remains to bound the joint cost of services at time at least t_{λ^*}.

▶ **Proposition 12.** *For every optimal service λ^*, it holds that*

$$\sum_{\lambda\in\Lambda|t_\lambda>t_{\lambda^*}} D\left(R^\star_{\lambda,\lambda^*}, t_\lambda\right) \le O(1)\cdot c(\lambda^*).$$

Proof. Let $\Lambda' \subseteq \Lambda$ be the subset of services that occurred after t_{λ^*}. First, we claim that for every integer ℓ, there exists at most one service in Λ' with $\ell_{\lambda,\lambda^*} = \ell$. To prove the claim, assume for contradiction that there are two services $\lambda_1, \lambda_2 \in \Lambda'$ such that $\ell_{\lambda_1,\lambda^*} = \ell_{\lambda_2,\lambda^*} = \ell$, and assume without loss of generality that $t_{\lambda_2} > t_{\lambda_1} > t_{\lambda^*}$. As $t_{\lambda_1} > t_{\lambda^*}$, it must be that all requests in $R^\star_{\lambda_2,\lambda^*}$ were pending before and after t_{λ_1}. Moreover, every request $q \in R^\star_{\lambda_2,\lambda^*}$ must have $\ell(q) \le \ell$, as $\ell = \ell\left(R^\star_{\lambda_2,\lambda^*}\right) \ge \ell(q)$. However, $\ell_{\lambda_1} \ge \ell_{\lambda_1,\lambda^*} = \ell$, and thus $R^\star_{\lambda_2,\lambda^*} \subseteq E_{\lambda_1}$. But, Line 15 in λ_1 sets $b_q = \textsc{False}$ for every $q \in R^\star_{\lambda_2,\lambda^*}$, in contradiction to having $b_q = \textsc{True}$ at t_{λ_2}. We conclude that $R^\star_{\lambda_2,\lambda^*} = \emptyset$; however, this implies that $\ell_{\lambda_2,\lambda^*} = -\infty$, in contradiction to $\ell_{\lambda_2,\lambda^*} = \ell$.

Using this claim, for every level ℓ, there exists at most one service $\lambda \in \Lambda'$ such that $\ell_{\lambda,\lambda^*} = \ell$. For such λ, it holds that $D\left(R^\star_{\lambda,\lambda^*}, t_\lambda\right) \le \mathrm{ND}(R^\star_{\lambda,\lambda^*}) \le 2^\ell$ (note that delay cannot exceed service cost since critical sets trigger a service). Also note that $\max_{\lambda\in\Lambda'} \ell_{\lambda,\lambda^*} \le \ell(Q_{\lambda^*})$. Summing over the possible levels, we get that

$$\sum_{\lambda\in\Lambda|t_\lambda>t_{\lambda^*}} D\left(R^\star_{\lambda,\lambda^*}, t_\lambda\right) \le 2\cdot 2^{\max_{\lambda\in\Lambda'}\ell_{\lambda,\lambda^*}} \le 2\cdot 2^{\ell(Q_{\lambda^*})} \le 4\cdot c(\lambda^*).$$ ◀

At this point, the only missing ingredient for the main theorem is the following lemma.

▶ **Lemma 13.** *For every optimal service $\lambda^* \in \Lambda$, it holds that*

$$\sum_{\lambda\in\Lambda|t_\lambda>t_{\lambda^*}} D\left(R^\perp_{\lambda,\lambda^*}, t_\lambda\right) \le O(\sqrt{k}\log k)\cdot c(\lambda^*)$$

The proof of Lemma 13 appears in Appendix A; assuming Lemma 13 holds, we proceed to prove Theorem 8.

Proof of Theorem 8. Equation (5) implies that it is enough to bound the sum of joint costs. Fix any optimal service λ^*; we have

$$\begin{aligned}\sum_{\lambda\in\Lambda} c(\lambda,\lambda^*) &= \sum_{\lambda\in\Lambda|t_\lambda<t_{\lambda^*}} c(\lambda,\lambda^*) + \sum_{\lambda\in\Lambda|t_\lambda\ge t_{\lambda^*}} c(\lambda,\lambda^*)\\ &\le O(\sqrt{k}\log k)\cdot c(\lambda^*) + \sum_{\lambda\in\Lambda|t_\lambda\ge t_{\lambda^*}} \left(D\left(R^\perp_{\lambda,\lambda^*}, t_\lambda\right) + \sqrt{k}\log(1+k)\cdot D\left(R^\star_{\lambda,\lambda^*}, t_\lambda\right)\right)\\ &\le O(\sqrt{k}\log k)\cdot c(\lambda^*)\end{aligned}$$

where the first inequality uses Proposition 11 and the second inequality uses Proposition 12 and Lemma 13. ◀

5 Polynomial Time through Lagrangian Prize Collecting

While the algorithms in this paper yield proper competitiveness bounds, it is not clear how to implement some of their components in polynomial time. In this section, we focus on a subset of network design problems that admit a *Lagrangian prize-collecting* approximation algorithms, and describe a polynomial-time implementation of the framework. For conciseness, we focus on the delay framework of Section 4; the result for deadlines is a special case of the result for delay.

Considering the framework in Algorithm 2 of Section 4, we note the following components which might take super-polynomial time:

1. The framework for the delay model waits until the delay cost of a subset of pending requests exceeds the cost of serving their request types.
2. The framework solves the offline network design problem optimally, i.e., makes calls to ND (e.g., in Lines 7 and 12).
3. The framework finds a subset of requests whose delay exceeds a constant fraction of their service cost (Line 6).
4. The framework includes a component which, given a penalty x and a set of request types H', finds a maximal subset $H' \subseteq H$ such that $\mathrm{ND}(H') \geq |H'| \cdot x$ (Line 11).

The prize-collecting problem. In the offline network design problems we considered thus far, a valid solution must serve all given requests. However, we now consider a more general model of these problems, which is the *prize-collecting* model. In the (offline) prize-collecting model, in addition to the given connectivity requests H, we are also given a penalty function $\pi : H \to \mathbb{R}^+$; a valid solution in this model can now serve only a subset of the input requests, and pay the penalty for the remaining requests. The total cost of the solution is thus the total service cost plus the total penalty cost; for prize-collecting input (H, π), we use $\mathrm{PCND}(H, \pi)$ to refer to the minimum total cost of a feasible solution to the input. Approximation algorithms are known for many such prize-collecting network design problems; given an approximation algorithm $\widetilde{\mathrm{PCND}}$, we again use $\widetilde{\mathrm{PCND}}(H, \pi)$ to refer to the total cost of $\widetilde{\mathrm{PCND}}$ on the input (H, π). We also use the subscripts b and p to refer to the service and penalty costs of an algorithm, respectively (e.g., $\widetilde{\mathrm{PCND}}_{\mathrm{b}}(H, \pi)$).

To give a polynomial-time implementation to the framework, we require an approximation algorithm for the prize-collecting version of the offline network design problem. In fact, we need a slightly stronger notion of approximation, in which the algorithm's penalty cost is more closely bound to the optimal solution than the service cost; we now define this notion, called *Lagrangian* approximation.

▶ **Definition 14.** *We say that an algorithm for the prize-collecting problem is a Lagrangian γ-approximation if for every prize-collecting input (H, π) it holds that*

$$\widetilde{PCND}_b(H, \pi) + \gamma \cdot \widetilde{PCND}_p(H, \pi) \leq \gamma \cdot PCND(H, \pi).$$

In this section, we present an algorithm which proves the following theorem.

▶ **Theorem 15.** *For an online network design problem with deadlines/delay, whose offline network design prize-collecting problem admits a Lagrangian γ approximation, there exists a poly-time algorithm which achieves the competitiveness of Theorem 8 up to a factor polynomial in γ. Specifically, it achieves a competitive ratio of $O(\gamma^3 \cdot \min\left\{\sqrt{n \log n}, \sqrt{m \log m}\right\})$.*

5.1 Applications

To demonstrate the use of Theorem 15, we apply it to some network design problems for which Lagrangian prize-collecting approximation algorithms are known.

Steiner tree. Goemans and Williamson [19] gave a Lagrangian 2-approximation for the prize-collecting Steiner tree problem. Thus, we obtain the following corollary of Theorem 15.

▶ **Corollary 16.** *There exists an $O(\min\left\{\sqrt{|V| \log |V|}, \sqrt{m \log m}\right\})$-competitive nonclairvoyant algorithm for Steiner tree with deadlines/delay on a graph with vertex set V which runs in polynomial time.*

Facility location. First, we explain the way facility location conforms to the network design setting. The set of items consists of two types: an *opening* item for each location, of the cost of opening a facility at that location; and a *connection* item for each (location, request) pair, of the cost of connecting the request to a facility at the given location. To satisfy a request, there must exist a location for which both the opening item, and the connection item to the request, have been bought; note that the upwards-closed property of network design problems holds. Also note that each request requires a separate connection item; thus, no two requests belong to the same type, and thus $n \geq m$. Hence, we do not state competitiveness in terms of n for this problem.

Charikar et al. [15] gave a Lagrangian 3-approximation for the prize-collecting facility location problem. This implies the following corollary of Theorem 15 for facility location.

▶ **Corollary 17.** *There exists an $O(\min\left\{\sqrt{m \log m}\right\})$-competitive algorithm for facility location with deadlines/delay which runs in polynomial time.*

Multicut on a tree. Hou et al. [23] gave a Lagrangian 2-approximation for prize-collecting multicut where the underlying graph is a tree. Note that for multicut on a tree, it holds that n is the number of vertex pairs in the tree, i.e., quadratic in the number of vertices. Thus, we obtain the following corollary of Theorem 15 for multicut on a tree.

▶ **Corollary 18.** *There exists an $O(\min\left\{|V|\sqrt{\log |V|}, \sqrt{m \log m}\right\})$-competitive algorithm for multicut with deadlines/delay on a tree with vertices V which runs in polynomial time.*

5.2 The Algorithm

Consider a problem which admits a Lagrangian γ-approximation, which we denote by $\widetilde{\text{PCND}}$. As a shorthand, we use $\widetilde{\text{ND}}$ to refer to the offline γ-approximation obtained from using $\widetilde{\text{PCND}}$ with the penalties set to ∞.

The procedure PCSOLVE. The algorithm uses $\widetilde{\text{PCND}}$ in the procedure PCSOLVE (Algorithm 3), which receives a set of request types $\hat{H}$ and penalties π to those requests, and outputs a subset of request types $H' \subseteq \hat{H}$ and a solution S to H'. We prove the following properties of PCSOLVE:

1. It holds that the cost of solution S to H' is at most $\gamma\pi(H')$.
2. For every subset $H'' \subseteq \hat{H} \backslash H'$, it holds that $\text{ND}(H'') \geq \pi(H'')$.

These two properties make PCSOLVE a useful primitive, which is used several times in the algorithm.

Algorithm's Description The algorithm is given in Algorithm 4. The algorithm periodically runs the procedure PCSOLVE on the set of pending requests, where the penalty of every request type is the total current delay of pending requests of that type. Whenever PCSOLVE returns a non-empty set of request types H' to serve, the procedure starts the service, and the set of pending requests of types H' is called *critical*. This triggers a call to UPONCRITICAL.

Inside UPONCRITICAL, as in Algorithm 2, the algorithm chooses the subset of critical requests whose variable b_q is TRUE, and looks for a maximal subset of them whose service cost is at most some factor from their delay cost. However, this factor is now linear in the approximation factor γ rather than a constant. To find this request set $R^{\star}$, the algorithm makes a call to PCSOLVE.

Now, the service makes its transmissions. First, it transmits the approximate solution previously calculated for the critical requests, thus serving them. Then, it uses PCSolve to find some subset of the request types of eligible requests to serve in the second transmission; it does so by providing a uniform penalty function to PCSolve. The proof of Theorem 15 using Algorithm 4 appears in Appendix B.

Algorithm 3 Prize-collecting procedure.

```
Function PCSolve $(H, \pi)$
    Set $\hat{H} \leftarrow H$.
    Set $S \leftarrow \emptyset$.
    while True do
        Run PCND$(\hat{H}, \pi)$ to obtain a solution $S'$ which serves some subset $H' \subseteq \hat{H}$ of request types.
        if $H' = \emptyset$ then break
        Set $S \leftarrow S \cup S'$, $\hat{H} \leftarrow \hat{H} \setminus H'$
    return $(S, H \setminus \hat{H})$
```

Algorithm 4 Polynomial Time Framework for Nonclairvoyant Network Design with Delay.

```
Event Function UponRequest$(q)$
    $b_q \leftarrow$ True.
Function TestCritical() // called continuously
    Let $t$ be the current time, and let $Q'$ be the set of currently-pending requests.
    Define $\pi$ which maps from request type $h \in H(Q')$ to $\sum_{q \in Q' | h_q = h} d_q(t)$.
    Call PCSolve$(Q', \pi)$, and obtain the output $(H', S)$.
    if $H' \neq \emptyset$ then define $R \leftarrow \{q \in Q' | h_q \in H'\}$.
    call UponCritical$(R, S)$.
Event Function UponCritical$(R, S)$
    Start a new service $\lambda$; denote the current time by $t$.
    Define $R^\top \leftarrow \{q \in R | b_q = \text{True}\}$.
    Let $\pi_1$ map from request type $h$ to $2\gamma \cdot \sum_{q \in R^\star | h_q = h} d_q(t)$.
    Call PCSolve$(H(R^\top), \pi_1)$, let $H^\star \subseteq H(R^\top)$ be the request types served by the output, and let $S^1$ be the returned solution for $H'$.
    Define $R^\star \leftarrow \{q \in R^\top | h_q \in H^\star\}$
    Define $\hat{\ell}_\lambda \leftarrow \lceil \log(c(S^1)) \rceil$.
    Define $E$ to be the set of pending requests $q$ such that $\check{\ell}(q) \leq \hat{\ell}_\lambda$, and define $H \leftarrow H(E)$.
    Transmit $S$, serving all requests of types $H(R)$.
    Let $x_\lambda \leftarrow 2^{\hat{\ell}_\lambda} \cdot \sqrt{\frac{\log(1+|H|)}{|H|}}$.
    Let $\pi_2$ map from $h \in H$ to $x_\lambda$.
    Call PCSolve$(H, \pi_2)$; let $H' \subseteq H$ and solution $S^2$ be the output.
    Transmit $S^2$, serving all requests of types in $H'$.
    // Set $b_{q'}$ for eligible requests $q'$ which are still pending.
    Let $Q_\lambda$ be the subset of pending requests served by $\lambda$.
    foreach $q' \in E \setminus Q_\lambda$ do
        set $b_{q'} \leftarrow$ False.
```

6 Conclusions and Future Directions

In this paper, we presented frameworks for obtaining $\tilde{O}(\min\{\sqrt{n}, \sqrt{m}\})$-competitive algorithms for network design with deadlines or delay. For some problems, in particular facility location and multilevel aggregation, lower bounds of $\Omega(\sqrt{k})$ and $\Omega(\sqrt{m})$ exist, making these frameworks optimal up to a logarithmic factor. We then discussed running time, and presented a class of problems (namely those that admit Lagrangian prize-collecting approximations) for which these frameworks can be implemented in polynomial time.

An interesting direction for future work would be to implement this framework in polynomial time for additional problems. This could require a different direction from the one in this paper, as not all network design problems seem amenable to Lagrangian prize-collecting approximations. In particular, for Steiner forest, a Lagrangian prize-collecting approximation implies an approximation of similar ratio for k-Steiner forest; however, no subpolynomial approximation for k-Steiner forest is known (see e.g. [22]).

Additionally, we made little attempt to optimize the dependence of the poly-time framework's competitive ratio on the approximation ratio γ of the Lagrangian approximation algorithm. This is since γ is constant for the problems we consider in this paper. However, improving this dependence could be useful for problems which are harder to approximate; we conjecture that a linear dependence is possible.

References

1 Yossi Azar, Ashish Chiplunkar, Shay Kutten, and Noam Touitou. Set cover with delay – clairvoyance is not required. In Fabrizio Grandoni, Grzegorz Herman, and Peter Sanders, editors, *28th Annual European Symposium on Algorithms, ESA 2020, September 7-9, 2020, Pisa, Italy (Virtual Conference)*, volume 173 of *LIPIcs*, pages 8:1–8:21. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2020. doi:10.4230/LIPIcs.ESA.2020.8.

2 Yossi Azar, Arun Ganesh, Rong Ge, and Debmalya Panigrahi. Online service with delay. In *Proceedings of the 49th Annual ACM SIGACT Symposium on Theory of Computing, STOC 2017, Montreal, QC, Canada, June 19-23, 2017*, pages 551–563, 2017. doi:10.1145/3055399.3055475.

3 Yossi Azar and Noam Touitou. General framework for metric optimization problems with delay or with deadlines. In *60th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2019, Baltimore, Maryland, USA, November 9-12, 2019*, pages 60–71, 2019. doi:10.1109/FOCS.2019.00013.

4 Yossi Azar and Noam Touitou. Beyond tree embeddings – A deterministic framework for network design with deadlines or delay. In *61st IEEE Annual Symposium on Foundations of Computer Science, FOCS 2020, Durham, NC, USA, November 16-19, 2020*, pages 1368–1379. IEEE, 2020. doi:10.1109/FOCS46700.2020.00129.

5 Marcin Bienkowski, Martin Böhm, Jaroslaw Byrka, Marek Chrobak, Christoph Dürr, Lukáš Folwarczný, Lukasz Jez, Jirí Sgall, Kim Thang Nguyen, and Pavel Veselý. New results on multi-level aggregation. *Theor. Comput. Sci.*, 861:133–143, 2021. doi:10.1016/j.tcs.2021.02.016.

6 Marcin Bienkowski, Martin Böhm, Jaroslaw Byrka, Marek Chrobak, Christoph Dürr, Lukáš Folwarczný, Lukasz Jez, Jiri Sgall, Nguyen Kim Thang, and Pavel Veselý. Online algorithms for multi-level aggregation. In *24th Annual European Symposium on Algorithms, ESA 2016, August 22-24, 2016, Aarhus, Denmark*, pages 12:1–12:17, 2016. doi:10.4230/LIPIcs.ESA.2016.12.

7 Marcin Bienkowski, Martin Böhm, Jaroslaw Byrka, and Jan Marcinkowski. Online facility location with linear delay. In Amit Chakrabarti and Chaitanya Swamy, editors, *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques, APPROX/RANDOM 2022, September 19-21, 2022, University of Illinois, Urbana-Champaign, USA (Virtual Conference)*, volume 245 of *LIPIcs*, pages 45:1–45:17. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2022. doi:10.4230/LIPIcs.APPROX/RANDOM.2022.45.

8 Marcin Bienkowski, Jaroslaw Byrka, Marek Chrobak, Lukasz Jez, Dorian Nogneng, and Jirí Sgall. Better approximation bounds for the joint replenishment problem. In *Proceedings of the Twenty-Fifth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2014, Portland, Oregon, USA, January 5-7, 2014*, pages 42–54, 2014. doi:10.1137/1.9781611973402.4.

9 Marcin Bienkowski, Artur Kraska, and Pawel Schmidt. Online service with delay on a line. In *Structural Information and Communication Complexity – 25th International Colloquium, SIROCCO 2018, Ma'ale HaHamisha, Israel, June 18-21, 2018, Revised Selected Papers*, pages 237–248, 2018. doi:10.1007/978-3-030-01325-7_22.

10 Carlos Fisch Brito, Elias Koutsoupias, and Shailesh Vaya. Competitive analysis of organization networks or multicast acknowledgment: How much to wait? *Algorithmica*, 64(4):584–605, 2012. doi:10.1007/s00453-011-9567-5.

11 Niv Buchbinder, Moran Feldman, Joseph (Seffi) Naor, and Ohad Talmon. O(depth)-competitive algorithm for online multi-level aggregation. In *Proceedings of the Twenty-Eighth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2017, Barcelona, Spain, Hotel Porta Fira, January 16-19*, pages 1235–1244, 2017. doi:10.1137/1.9781611974782.80.

12 Niv Buchbinder, Kamal Jain, and Joseph Naor. Online primal-dual algorithms for maximizing ad-auctions revenue. In *Algorithms – ESA 2007, 15th Annual European Symposium, Eilat, Israel, October 8-10, 2007, Proceedings*, pages 253–264, 2007. doi:10.1007/978-3-540-75520-3_24.

13 Niv Buchbinder, Tracy Kimbrel, Retsef Levi, Konstantin Makarychev, and Maxim Sviridenko. Online make-to-order joint replenishment model: primal dual competitive algorithms. In *Proceedings of the Nineteenth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2008, San Francisco, California, USA, January 20-22, 2008*, pages 952–961, 2008. URL: http://dl.acm.org/citation.cfm?id=1347082.1347186.

14 Rodrigo A. Carrasco, Kirk Pruhs, Cliff Stein, and José Verschae. The online set aggregation problem. In *LATIN 2018: Theoretical Informatics – 13th Latin American Symposium, Buenos Aires, Argentina, April 16-19, 2018, Proceedings*, pages 245–259, 2018. doi:10.1007/978-3-319-77404-6_19.

15 Moses Charikar, Samir Khuller, David M. Mount, and Giri Narasimhan. Algorithms for facility location problems with outliers. In S. Rao Kosaraju, editor, *Proceedings of the Twelfth Annual Symposium on Discrete Algorithms, January 7-9, 2001, Washington, DC, USA*, pages 642–651. ACM/SIAM, 2001. URL: http://dl.acm.org/citation.cfm?id=365411.365555.

16 Ryder Chen, Jahanvi Khatkar, and Seeun William Umboh. Online weighted cardinality joint replenishment problem with delay. In Mikolaj Bojanczyk, Emanuela Merelli, and David P. Woodruff, editors, *49th International Colloquium on Automata, Languages, and Programming, ICALP 2022, July 4-8, 2022, Paris, France*, volume 229 of *LIPIcs*, pages 40:1–40:18. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2022. doi:10.4230/LIPIcs.ICALP.2022.40.

17 Daniel R. Dooly, Sally A. Goldman, and Stephen D. Scott. TCP dynamic acknowledgment delay: Theory and practice (extended abstract). In *Proceedings of the Thirtieth Annual ACM Symposium on the Theory of Computing, Dallas, Texas, USA, May 23-26, 1998*, pages 389–398, 1998. doi:10.1145/276698.276792.

18 Jittat Fakcharoenphol, Satish Rao, and Kunal Talwar. A tight bound on approximating arbitrary metrics by tree metrics. *J. Comput. Syst. Sci.*, 69(3):485–497, 2004. doi:10.1016/j.jcss.2004.04.011.

19 Michel X. Goemans and David P. Williamson. A general approximation technique for constrained forest problems. In *Proceedings of the Third Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '92, pages 307–316, Philadelphia, PA, USA, 1992. Society for Industrial and Applied Mathematics. URL: http://dl.acm.org/citation.cfm?id=139404.139468.

20 Anupam Gupta, Amit Kumar, and Debmalya Panigrahi. Caching with time windows. In Konstantin Makarychev, Yury Makarychev, Madhur Tulsiani, Gautam Kamath, and Julia Chuzhoy, editors, *Proccedings of the 52nd Annual ACM SIGACT Symposium on Theory of Computing, STOC 2020, Chicago, IL, USA, June 22-26, 2020*, pages 1125–1138. ACM, 2020. doi:10.1145/3357713.3384277.

21 Anupam Gupta, Amit Kumar, and Debmalya Panigrahi. A hitting set relaxation for k-server and an extension to time-windows. In *62nd IEEE Annual Symposium on Foundations of Computer Science, FOCS 2021, Denver, CO, USA, February 7-10, 2022*, pages 504–515. IEEE, 2021. doi:10.1109/FOCS52979.2021.00057.

22 Mohammad Taghi Hajiaghayi and Kamal Jain. The prize-collecting generalized steiner tree problem via a new approach of primal-dual schema. In *Proceedings of the Seventeenth Annual ACM-SIAM Symposium on Discrete Algorithm*, SODA '06, pages 631–640, Philadelphia, PA, USA, 2006. Society for Industrial and Applied Mathematics. URL: http://dl.acm.org/citation.cfm?id=1109557.1109626.

23 Xin Hou, Wen Liu, and Bo Hou. An approximation algorithm for the k-prize-collecting multicut on a tree problem. *Theor. Comput. Sci.*, 844:26–33, 2020. doi:10.1016/j.tcs.2020.07.014.

24 Anna R. Karlin, Claire Kenyon, and Dana Randall. Dynamic TCP acknowledgment and other stories about e/(e-1). *Algorithmica*, 36(3):209–224, 2003.

25 Predrag Krnetic, Darya Melnyk, Yuyi Wang, and Roger Wattenhofer. The k-server problem with delays on the uniform metric space. In Yixin Cao, Siu-Wing Cheng, and Minming Li, editors, *31st International Symposium on Algorithms and Computation, ISAAC 2020, December 14-18, 2020, Hong Kong, China (Virtual Conference)*, volume 181 of *LIPIcs*, pages 61:1–61:13. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2020. doi:10.4230/LIPIcs.ISAAC.2020.61.

26 Ngoc Mai Le, William Umboh, and Ningyuan Xie. The power of clairvoyance for multi-level aggregation and set cover with delay. In *To appear in Symposium on Discrete Algorithms (SODA) 2023*, 2023.

27 Jeremy McMahan. A d-competitive algorithm for the multilevel aggregation problem with deadlines. *CoRR*, abs/2108.04422, 2021. arXiv:2108.04422.

28 Noam Touitou. Nearly-tight lower bounds for set cover and network design with deadlines/delay. In Hee-Kap Ahn and Kunihiko Sadakane, editors, *32nd International Symposium on Algorithms and Computation, ISAAC 2021, December 6-8, 2021, Fukuoka, Japan*, volume 212 of *LIPIcs*, pages 53:1–53:16. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2021. doi:10.4230/LIPIcs.ISAAC.2021.53.

A Proof of Lemma 13

Henceforth, fix any optimal service $\lambda^* \in \Lambda^*$. For ease of notation, define $R^\perp := \bigcup_{\lambda\in\Lambda|t_\lambda>t_{\lambda^*}} R^\perp_{\lambda,\lambda^*}$. Also define $\ell^* := c(\lambda^*)$. Note that

$$\sum_{\lambda\in\Lambda|t_\lambda>t_{\lambda^*}} D\left(R^\perp_{\lambda,\lambda^*}, t_\lambda\right) \le \sum_{\lambda\in\Lambda|t_\lambda>t_{\lambda^*}} \sum_{h\in H(R^\perp_{\lambda,\lambda^*})} \mathrm{ND}(h)$$

We claim that no request type appears twice in the summation on the right-hand side of the above equation. That is, we claim that $\sum_{h\in H(R^\perp)} \mathrm{ND}(h) = \sum_{\lambda\in\Lambda|t_\lambda>t_{\lambda^*}} \sum_{h\in H(R^\perp_{\lambda,\lambda^*})} \mathrm{ND}(h)$. Indeed, note that there cannot be two services $\lambda_1, \lambda_2 \in \Lambda$ such that $t_{\lambda^*} \le t_{\lambda_1} < t_{\lambda_2}$ and requests $q_1 \in R^\perp_{\lambda_1,\lambda^*}, q_2 \in R^\perp_{\lambda_2,\lambda^*}$ such that $h_{q_1} = h_{q_2}$: otherwise, λ_1 would serve q_2. We conclude that

$$\sum_{\lambda\in\Lambda|t_\lambda>t_{\lambda^*}} D\left(R^\perp_{\lambda,\lambda^*}, t_\lambda\right) \le \sum_{h\in H(R^\perp)} \mathrm{ND}(h) \tag{6}$$

and it is thus enough to bound $\sum_{h\in H(R^\perp)} \mathrm{ND}(h)$. Note that every request $q \in R^\perp$ has had b_q set to FALSE at some point in the algorithm. Define $R^\perp_1 \subseteq R^\perp$ to be the subset of requests q such that b_q was first set to FALSE prior to t_{λ^*}, and define $R^\perp_2 := R^\perp \backslash R^\perp_1$. For every ℓ, further define $R^\perp_{1,\ell} := \left\{q \in R^\perp_1 \middle| \ell(q) = \ell\right\}$; define $R^\perp_{2,\ell}$ analogously.

▶ **Proposition 19.** *For every optimal service λ^*, and for every ℓ, it holds that*

$$\sum_{h\in H(R^{\perp}_{1,\ell})} ND(h) \le O\left(\sqrt{k/\log k}\right)\cdot c(\lambda^*).$$

Proof. Observe the first service $\lambda \in \Lambda$ after which $b_q = \text{FALSE}$ for every $q \in R^{\perp}_{1,\ell}$. From the definition of $R^{\perp}_{1,\ell}$, we know that $t_\lambda < t_{\lambda^*}$; as $R^{\perp}_{1,\ell}$ are all served after t_{λ^*}, they are all pending both before and after t_λ. As λ set $b_q \leftarrow \text{FALSE}$ for some $q \in R^{\perp}_{1,\ell}$, we have $\ell_\lambda \ge \ell$. This implies $R^{\perp}_{1,\ell} \subseteq E_\lambda$. Define $z := \left|H(R^{\perp}_{1,\ell})\right|$; since no request from $R^{\perp}_{1,\ell}$ was served in λ, we have

$$c(\lambda^*) \ge \mathrm{ND}(R^{\perp}_{1,\ell}) \ge x_\lambda \cdot z \ge 2^{\ell_\lambda}\cdot\sqrt{\frac{\log(1+k)}{k}}\cdot z \ge 2^{\ell}\cdot\sqrt{\frac{\log(1+k)}{k}}\cdot z$$

Noting that $\sum_{h\in H(R^{\perp}_{1,\ell})}\mathrm{ND}(h) \le z\cdot 2^{\ell}$, this yields $\sum_{h\in H(R^{\perp}_{1,\ell})}\mathrm{ND}(h) \le \sqrt{\frac{k}{\log(1+k)}}\cdot c(\lambda^*)$. ◀

▶ **Proposition 20.** *For every optimal service λ^*, and for every ℓ, it holds that*

$$\sum_{h\in H(R^{\perp}_{2,\ell})} ND(h) \le O\left(\sqrt{\frac{k}{\log k}}\right)\cdot c(\lambda^*).$$

Proof. The proof is similar to that of Proposition 19. Consider the first service $\lambda \in \Lambda$ such that $t_\lambda \ge t_{\lambda^*}$ and $\ell_\lambda \ge \ell$. It must be the case that all requests in $R^{\perp}_{2,\ell}$ are pending before and after λ, and moreover, λ sets $b_q \leftarrow \text{FALSE}$ for every request in $R^{\perp}_{2,\ell}$. Define $z := \left|H(R^{\perp}_{2,\ell})\right|$; since no request from $R^{\perp}_{2,\ell}$ was served in λ, we have

$$c(\lambda^*) \ge \mathrm{ND}(R^{\perp}_{2,\ell}) \ge x_\lambda \cdot z \ge 2^{\ell_\lambda}\cdot\sqrt{\frac{\log(1+k)}{k}}\cdot z \ge 2^{\ell}\cdot\sqrt{\frac{\log(1+k)}{k}}\cdot z$$

Noting that $\sum_{h\in H(R^{\perp}_{2,\ell})}\mathrm{ND}(h) \le z\cdot 2^{\ell}$, the above yields $\sum_{h\in H(R^{\perp}_{2,\ell})}\mathrm{ND}(h) \le \sqrt{\frac{k}{\log(1+k)}}\cdot c(\lambda^*)$, which completes the proof. ◀

Proof of Lemma 13. Define $\gamma := \lceil \log k \rceil$. The following holds:

$$\begin{aligned}
\sum_{\lambda\in\Lambda | t_\lambda > t_{\lambda^*}} D\left(R^{\perp}_{\lambda,\lambda^*}, t_\lambda\right) &\le \sum_{h\in H(R^{\perp})}\mathrm{ND}(h) \le \sum_{\ell=-\infty}^{\infty}\left(\sum_{h\in H(R^{\perp}_{1,\ell})}\mathrm{ND}(h) + \sum_{h\in H(R^{\perp}_{2,\ell})}\mathrm{ND}(h)\right)\\
&= \sum_{\ell=-\infty}^{\ell^*-\gamma}\left(\sum_{h\in H(R^{\perp}_{1,\ell})}\mathrm{ND}(h) + \sum_{h\in H(R^{\perp}_{2,\ell})}\mathrm{ND}(h)\right) + \sum_{\ell=\ell^*-\gamma+1}^{\ell^*}\left(\sum_{h\in H(R^{\perp}_{1,\ell})}\mathrm{ND}(h) + \sum_{h\in H(R^{\perp}_{2,\ell})}\mathrm{ND}(h)\right)
\end{aligned} \tag{7}$$

where the first inequality uses Equation (6), the second inequality partitions $R^{\perp}$ into $\left\{R^{\perp}_{1,\ell}\right\}_\ell$ and $\left\{R^{\perp}_{2,\ell}\right\}_\ell$, and the equality makes use of the fact that $R^{\perp}$ does not contain requests q such that $\ell(q) > \ell^*$ (since $R^{\perp} \subseteq Q_{\lambda^*}$). From the proof of Proposition 19, we know that for every ℓ the requests $R^{\perp}_{1,\ell}$ were all pending during a single service. Thus, we have $\left|H(R^{\perp}_{1,\ell})\right| \le k$, and therefore $\sum_{h\in H(R^{\perp}_{1,\ell})}\mathrm{ND}(h) \le 2^{\ell}\cdot k$. Similarly, using the proof of Proposition 20, we have $\left|H(R^{\perp}_{2,\ell})\right| \le k$ and thus $\sum_{h\in H(R^{\perp}_{2,\ell})}\mathrm{ND}(h) \le 2^{\ell}\cdot k$. We can therefore conclude that

$$\sum_{\ell=-\infty}^{\ell^*-\gamma}\left(\sum_{h\in H(R^{\perp}_{1,\ell})}\mathrm{ND}(h) + \sum_{h\in H(R^{\perp}_{2,\ell})}\mathrm{ND}(h)\right) \le 4\cdot 2^{\ell^*-\gamma}\cdot k \le 4\cdot 2^{\ell^*} \le 8\cdot c(\lambda^*). \tag{8}$$

Moreover, using Propositions 19 and 20,

$$\sum_{\ell=\ell^*-\gamma+1}^{\ell^*}\left(\sum_{h\in H(R^{\perp}_{1,\ell})}\mathrm{ND}(h)+\sum_{h\in H(R^{\perp}_{2,\ell})}\mathrm{ND}(h)\right)\le\gamma\cdot O\left(\sqrt{\frac{k}{\log k}}\right)\cdot c(\lambda^*)=O(\sqrt{k\log k})\cdot c(\lambda^*). \quad (9)$$

Combining Equations (7) to (9) yields $\sum_{\lambda\in\Lambda|t_\lambda>t_{\lambda^*}} D\left(R^{\perp}_{\lambda,\lambda^*},t_\lambda\right)\le O(\sqrt{k\log k})\cdot c(\lambda^*)$. ◀

B Analysis of Lagrangian Approximation Framework

We focus on proving Theorem 15, following the same lines as the proof of Theorem 8. Following the notation of Section 4, we use the subscript λ to refer to the values of variables in the UPONCRITICAL call that started service λ. For example, this includes S^1_λ, S^2_λ.

▶ **Proposition 21** (Properties of PCSOLVE). *Suppose* PCSOLVE *is called on request types H and penalties π, and outputs H' and solution S. It holds that:*

1. *The cost of solution S to H' is at most $\gamma\pi(H')$.*
2. *For every subset $H''\subseteq H\backslash H'$, it holds that $ND(H'')\ge\pi(H'')$.*

Proof. Let b be the number of iterations of the main loop in PCSOLVE. We use subscript i to refer to the value of a variable in the i'th iteration of the loop; note that $c(S)\le\sum_{i\in[b]}c(S_i)$. For every iteration i, through the Lagrangian approximation guarantee, it holds that $c(S_i)+\gamma\cdot\pi(\hat{H}_i\backslash H'_i)\le\gamma\cdot\pi(\hat{H}_i)$, implying $c(S_i)\le\gamma\cdot\pi(H'_i)$; thus, we have $c(S)\le\sum_i\pi(H'_i)=\gamma\pi(H')$, proving the first claim.

To prove the second claim, observe that in the final iteration no request types from $\hat{H}_b$ were served. Through the Lagrangian guarantee, $\gamma\pi(\hat{H}_b)\le\gamma\cdot(\mathrm{ND}(H'')+\pi(\hat{H}_b\backslash H''))$ for every subset $H''\subseteq\hat{H}_b$, which implies that $\mathrm{ND}(H'')\ge\pi(H'')$. Observing that $\hat{H}_b=H\backslash H'$ completes the proof of the second claim. ◀

▶ **Proposition 22** (Analogue of Proposition 10). *For a service λ, it holds that*

$$c(\lambda)\le O(\gamma)\cdot\sum_{\lambda^*\in\Lambda^*}D\left(R^{\perp}_{\lambda,\lambda^*},t_\lambda\right)+O(\gamma^3\sqrt{k\log k})\cdot\sum_{\lambda^*\in\Lambda^*}D\left(R^{\star}_{\lambda,\lambda^*},t_\lambda\right)$$

Proof. According to Proposition 21, the cost of the first transmission of solution S_λ (Line 17) is at most $\gamma\cdot D(R_\lambda,t_\lambda)$. However, Proposition 21 also implies that $2\gamma D\left(R^{\top}_\lambda\backslash R^{\star}_\lambda,t_\lambda\right)\le\mathrm{ND}(R^{\top}_\lambda\backslash R^{\star}_\lambda)$; together with the fact that S_λ serves $R^{\top}_\lambda\backslash R^{\star}_\lambda$ implies that $\gamma D\left(R^{\top}_\lambda\backslash R^{\star}_\lambda,t_\lambda\right)\le c(S_\lambda)/2$. Combining, we have the following:

$$c(S_\lambda)\le\gamma D(R_\lambda,t_\lambda)=\gamma(D\left(R^{\perp}_\lambda,t_\lambda\right)+D\left(R^{\star}_\lambda,t_\lambda\right))+c(S_\lambda)/2$$

Simplifying, $c(S_\lambda)\le 2\gamma(D\left(R^{\perp}_\lambda,t_\lambda\right)+D\left(R^{\star}_\lambda,t_\lambda\right))$, yielding the bound for the first transmission.

For the second transmission, note that $2^{\hat{\ell}_\lambda}\le 2\cdot c\left(S^1_\lambda\right)\le 4\gamma^2 D(R^\star,t_\lambda)$, where the second inequality uses Proposition 21 for PCSOLVE. Applying Proposition 21 again for Line 20, we obtain the following bound for the cost of the solution S^2_λ used for the second transmission:

$$\begin{aligned}c\left(S^2_\lambda\right)&\le\gamma\cdot|H_\lambda|\cdot x_\lambda=\gamma\cdot|H_\lambda|\cdot 2^{\hat{\ell}_\lambda}\cdot\sqrt{\log(1+|H_\lambda|)/|H_\lambda|}\le\gamma\sqrt{k\log(1+k)}\cdot 2^{\hat{\ell}_\lambda}\\&\le 4\gamma^3\cdot\sqrt{k\log(1+k)}\cdot D\left(R^\star,t_\lambda\right)\end{aligned}$$

Combining this with the previous bound for the first transmission completes the proof. ◀

We henceforth define joint costs $c(\lambda,\lambda^*)$ as in Section 4. Note that Proposition 10 holds for Algorithm 4 without modification.

▶ **Proposition 23** (Analogue of Proposition 12). *For every optimal service λ^*, it holds that*

$$\sum_{\lambda\in\Lambda|t_\lambda>t_{\lambda^*}} D\left(R^\star_{\lambda,\lambda^*},t_\lambda\right) \le O(1)\cdot c(\lambda^*).$$

Proof. Let $\Lambda'\subseteq\Lambda$ be the subset of services that occurred after t_{λ^*}. First, we claim that for every integer ℓ, there exists at most one service in Λ' with $\ell_{\lambda,\lambda^*}=\ell$. To prove the claim, assume for contradiction that there are two services $\lambda_1,\lambda_2\in\Lambda'$ such that $\ell_{\lambda_1,\lambda^*}=\ell_{\lambda_2,\lambda^*}=\ell$, and assume without loss of generality that $t_{\lambda_2}>t_{\lambda_1}>t_{\lambda^*}$. As $t_{\lambda_1}>t_{\lambda^*}$, it must be that all requests in $R^\star_{\lambda_2,\lambda^*}$ were pending before and after t_{λ_1}.

Moreover, every request $q\in R^\star_{\lambda_2,\lambda^*}$ must have $\check{\ell}(q)\le\ell(q)\le\ell\left(R^\star_{\lambda_2,\lambda^*}\right)=\ell$. However, $\hat{\ell}_{\lambda_1}\ge\ell_{\lambda_1,\lambda^*}=\ell$, and thus $R^\star_{\lambda_2,\lambda^*}\subseteq E_{\lambda_1}$. But, Line 15 in λ_1 sets $b_q=$ FALSE for every $q\in R^\star_{\lambda_2,\lambda^*}$, in contradiction to having $b_q=$ TRUE at t_{λ_2}. We conclude that $R^\star_{\lambda_2,\lambda^*}=\emptyset$; however, this implies that $\ell_{\lambda_2,\lambda^*}=-\infty$, in contradiction to $\ell_{\lambda_2,\lambda^*}=\ell$.

Using this claim, for every level ℓ, there exists at most one service $\lambda\in\Lambda'$ such that $\ell_{\lambda,\lambda^*}=\ell$. For such λ, it holds that $D\left(R^\star_{\lambda,\lambda^*},t_\lambda\right)\le\mathrm{ND}(R^\star_{\lambda,\lambda^*})\le 2^\ell$. Also note that $\max_{\lambda\in\Lambda'}\ell_{\lambda,\lambda^*}\le\ell(Q_{\lambda^*})$. Summing over the possible levels, we get that

$$\sum_{\lambda\in\Lambda|t_\lambda>t_{\lambda^*}} D\left(R^\star_{\lambda,\lambda^*},t_\lambda\right)\le 2\cdot 2^{\max_{\lambda\in\Lambda'}\ell_{\lambda,\lambda^*}}\le 2\cdot 2^{\ell(Q_{\lambda^*})}\le 4\cdot c(\lambda^*).$$ ◀

▶ **Lemma 24** (Analogue of Lemma 13). *For every optimal service $\lambda^*\in\Lambda$, it holds that* $\sum_{\lambda\in\Lambda|t_\lambda>t_{\lambda^*}} D\left(R^\perp_{\lambda,\lambda^*},t_\lambda\right)\le O(\log\gamma\cdot\sqrt{k\log k})\cdot c(\lambda^*)$.

Proof sketch. The proof follows the same main lines as that of Lemma 13. First, define $R^\perp,R^\perp_1,R^\perp_2$ as in the proof of Lemma 13. Now, define $R^\perp_{1,\ell}=\left\{q\in R^\perp_1|\hat{\ell}_q=\ell\right\}$; define $R^\perp_{2,\ell}$ analogously. Note that the $\hat{\ell}_q$ is used for these definitions, rather than ℓ_q.

We can prove analogues to Proposition 19 and Proposition 20, using identical proofs. Specifically, for every optimal service λ^*, for every ℓ and for every $b\in\{1,2\}$, it holds that

$$\sum_{h\in H(R^\perp_{b,\ell})}\mathrm{ND}(h)\le O\left(\sqrt{k/\log k}\right)\cdot c(\lambda^*)\tag{10}$$

Now, following the proof of Lemma 13, define $\delta:=\lceil\log k\rceil+\lceil\log\gamma\rceil$.

$$\begin{aligned}\sum_{\lambda\in\Lambda|t_\lambda>t_{\lambda^*}} D\left(R^\perp_{\lambda,\lambda^*},t_\lambda\right)&\le\sum_{h\in H(R^\perp)}\mathrm{ND}(h)\le\sum_{\ell=-\infty}^{\infty}\left(\sum_{h\in H(R^\perp_{1,\ell})}\mathrm{ND}(h)+\sum_{h\in H(R^\perp_{2,\ell})}\mathrm{ND}(h)\right)\\&=\sum_{\ell=-\infty}^{\ell^*-\delta}\left(\sum_{h\in H(R^\perp_{1,\ell})}\mathrm{ND}(h)+\sum_{h\in H(R^\perp_{2,\ell})}\mathrm{ND}(h)\right)+\sum_{\ell=\ell^*-\delta+1}^{\ell^*}\left(\sum_{h\in H(R^\perp_{1,\ell})}\mathrm{ND}(h)+\sum_{h\in H(R^\perp_{2,\ell})}\mathrm{ND}(h)\right)\end{aligned}\tag{11}$$

Using a similar argument to that in Lemma 13, we note that $\sum_{h\in H(R^\perp_{1,\ell})}\mathrm{ND}(h)\le k\cdot\gamma\cdot 2^\ell$; a similar bound applies to $\sum_{h\in H(R^\perp_{2,\ell})}\mathrm{ND}(h)$. Combining with the definition of δ, the first term in the RHS of Equation (11) can be bounded by $O(1)\cdot c(\lambda^*)$. Using Equation (10), the second term can be bounded by $O(\delta)\cdot c(\lambda^*)$, which is $O((\log k+\log\gamma)\cdot\sqrt{k/\log k})\cdot c(\lambda^*)$; this is at most $O(\log\gamma\cdot\sqrt{k\log k})\cdot c(\lambda^*)$. This completes the proof. ◀

Proof of Theorem 15. Results from combining Propositions 22 and 23 and Lemma 24. ◀