Optimizing in the Dark: Learning an Optimal Solution through a Simple Request Interface

Qiao Xiang,[1,2*] **Haitao Yu,**[1] **James Aspnes,**[2] **Franck Le,**[3] **Linghe Kong,**[4] **Y. Richard Yang**[1,2]
[1]Tongji University, [2]Yale University, [3]IBM T.J. Watson Research Center, [4]Shanghai Jiao Tong University
qiao.xiang@cs.yale.edu, haitao.yu@tongji.edu.cn, james.aspnes@yale.edu,
fle@us.ibm.com, linghe.kong@sjtu.edu.cn, yry@cs.yale.edu

Abstract

Network resource reservation systems are being developed and deployed, driven by the demand and substantial benefits of providing performance predictability for modern distributed applications. However, existing systems suffer limitations: They either are inefficient in finding the optimal resource reservation, or cause private information (*e.g.*, from the network infrastructure) to be exposed (*e.g.*, to the user). In this paper, we design BoxOpt, a novel system that leverages efficient oracle construction techniques in optimization and learning theory to automatically, and swiftly learn the optimal resource reservations without exchanging any private information between the network and the user. We implement a prototype of BoxOpt and demonstrate its efficiency and efficacy via extensive experiments using real network topology and trace. Results show that (1) BoxOpt has a 100% correctness ratio, and (2) for 95% of requests, BoxOpt learns the optimal resource reservation within 13 seconds.

1 Introduction

When facing a genie that only tells you whether it can grant a wish or not, how can you find the best wish it can grant?

Although the question may sound like one from fairy tales, people deal with such question in many real world scenarios. For example, modern distributed applications (*e.g.*, (Zaharia et al. 2012; White 2012)) construct complex data flows between end hosts, *e.g.*, in data center networks. The key to supporting these applications is the ability to provide guaranteed network resources (*i.e.*, bandwidth) for performance predictability (Mogul and Popa 2012). As such, many network resource reservation systems are developed and deployed (Campanella et al. 2006; Guok and Robertson 2006; Johnston, Guok, and Chaniotakis 2011; Riddle 2005; Zheng et al. 2005; Sobieski, Lehman, and Jabbari 2004). However, because of the underlying networks' concern of revealing sensitive information, existing reservation systems do not provide applications with an interface to access information of the underlying network infrastructure (*e.g.*, topology, links' available bandwidth). Instead, networks only offer a *simple reservation interface* for applications to submit requests for reserving a specific amount of bandwidths for a set of flows: $\mathrm{request}(\mathrm{flow\_set}, \mathrm{bw\_values})$), and returns either success or failure. A major concern of this design is its inefficiency for the applications/users to find the optimal amount of network resources to reserve. To further illustrate the issues, consider the example in Figure 1, where a user (*e.g.*, application) wants to determine and reserve the maximum achievable bandwidth for two flows from S_1 to D_1, and S_2 to D_2, respectively. Using existing solutions that offer only a *simple reservation interface*, finding the constraint that both flows can collectively get only 100 Mbps of bandwidth is already an instance of the NP-hard membership-query based constraint acquisition problem (Bessiere et al. 2017), letting alone finding the optimal reservation for both flows (*e.g.*, 50 Mbps for each flow).

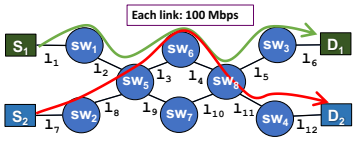


Figure 1: An example network topology: the routes of two flows share bottleneck links, *i.e.*, l_3 and l_4, hence they can only collectively get a 100 Mbps bandwidth.

To address this problem, researchers have proposed several solutions, but all of them suffer limitations, and violate privacy requirements. For example, to determine optimal bandwidth reservations, recent proposals depart from the simple reservation interface, and require either networks to reveal sensitive information to users (Soulé et al. 2014; Subramanian, D'Antoni, and Akella ; Heorhiadi, Reiter, and Sekar 2016; Lee et al.), or vice versa (Gao et al. 2016; Gao et al. 2017; Xiang et al. 2018). These solutions are therefore limited to settings where the level of trust between the applications and the underlying network is high. These solutions cannot be deployed in general settings as malicious parties may use the exposed network information to identify vulnerable links and launch attacks (*e.g.*, DDoS).

In this paper, we explore the feasibility and benefits of learning the optimal network resource reservation for the user without exposing the private information of the network (*i.e.*, bandwidth capacity region) and the user (*e.g.*, resource orchestration policy) to each other. In particular, we tackle the following question: *How can a user learn the optimal*

*The corresponding authors are Q.Xiang and Y. R. Yang.

network resource reservation using only the simple reservation interface? This task is non-trivial due to the extremely limited feedback (*i.e.*, success/failure) provided by simple reservation interface.

Our solution to this problem is BoxOpt, a novel learning system that automatically, and efficiently learns the optimal resource reservations for the user through the simple reservation interface, without exchanging any private information between the network and the user (*e.g.*, bandwidth feasible region of the network and the resource orchestration policy of the user). Specifically, BoxOpt allows users to include their resource reservation objectives as concave utility functions of the requested resources (*e.g.*, bandwidths) in the reservation requests. Upon receiving a reservation request, BoxOpt models the simple reservation interface of network resource reservation systems as a membership oracle over a polytope. It then expands oracle construction techniques (Lovasz, Grotschel, and Schrijver 1993; Lee, Sidford, and Vempala 2017) from optimization and learning theory to construct a separation oracle through invoking the membership oracle in near $O(n)$ iterations (n being the number of flows), which when called upon will accurately infer a search space in which the optimal reservation vector lies. With such a separation oracle, BoxOpt then constructs an optimization oracle based on ellipsoid method, which can learn the optimal reservation vector through $O(n^2)$ calls on the separation oracle.[1] In this way, BoxOpt not only can learn the optimal resource reservation efficiently, but also is privacy-preserving in that no private information is exchanged between the user and the network.

The **main contributions** of this paper are as follows:

• We study the important problem of learning the optimal network resource reservation through the simple reservation interface of network resource reservation systems. In particular, we design BoxOpt, a novel, fast, automatic, privacy-preserving learning system. To the best of our knowledge, BoxOpt is the first working system that solves this problem, and can be extended to other optimization problems.

• We model the simple reservation interface as a membership oracle over a polytope, and expand oracle construction techniques from optimization and learning theory to develop an efficient optimization oracle in BoxOpt, which learns the optimal resource reservation in near $O(n^3)$ of calls on the membership oracle.

• We implement a prototype of BoxOpt and demonstrate both its efficiency and efficacy through extensive experiments using real topologies and traces. Results show that (1) BoxOpt has a 100% correctness ratio, and (2) for 95% cases, it can learn the optimal reservation within 13 seconds.

The remaining of this paper is organized as follows. We present an overview of BoxOpt in Section 2. We give details on how BoxOpt efficiently learns the optimal network resource reservation only using the simple reservation interface in Section 3. We present the evaluation results of BoxOpt in Section 4. We discuss related work in Section 5 and conclude the paper in Section 6.

[1] We choose the ellipsoid method because it is a classic cutting plane method. However, the design of BoxOpt is modular and other cutting plane methods, *e.g.*, analytic center method (Atkinson and Vaidya 1995) and random walk (Bertsimas and Vempala), can also be used to construct an optimization oracle from a separation oracle.

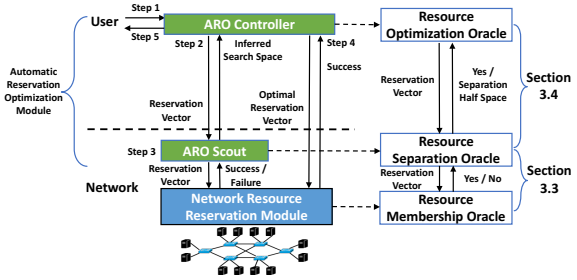


Figure 2: The architecture and workflow of BoxOpt.

2 Overview of BoxOpt

In this section, we first present the architecture and the workflow of BoxOpt. We then give a formal, mathematical formulation of the key technical challenge in BoxOpt: how to find the optimal network resource reservation through the simple reservation interface.

2.1 Architecture

BoxOpt is composed of two components: an automatic reservation optimization (ARO) module for the user, and a network resource reservation (NRR) module for the network (Figure 2). The two components interact with each other through the simple reservation interface commonly used in traditional network resource reservation systems.

Automatic reservation optimization module: The ARO module is a private component belonging to the user, and is composed of two sub-components: an ARO controller, and an ARO scout.

The ARO controller is the main interface for the user to submit the resource reservation requests. A request consists of a set of n flows, $F = \{f_1, f_2, \ldots, f_n\}$, to reserve the resources for, and a concave utility function $util(\mathbf{x})$ to maximize, with $\mathbf{x} = [x_1, x_2, \ldots, x_n]$ and each x_i representing the available bandwidth that can be reserved for flow $f_i \in F$. Example utility functions include total throughput and priority-based total throughput. Given a user resource reservation request, the objective of the ARO controller is to infer the optimal resource reservation to maximize $util(\mathbf{x})$. The ARO controller achieves it with the assistance of the ARO scout: Specifically, the ARO controller iteratively selects a vector $\check{\mathbf{x}}$ of bandwidth values for F (called *reservation vector*) and sends it to the ARO scout. For each reservation vector, the ARO scout returns a search space where the optimal reservation vector lies in. With the inferred search spaces returned by the ARO scout, the ARO controller gradually converges to the optimal reservation vector that maximizes $util(\mathbf{x})$.

The ARO scout is the main user entity interacting with the NNR. For each $\check{\mathbf{x}}$ from the ARO controller, the ARO scout infers a search space where the optimal reservation

vector lies in, and returns the inferred search space back to the ARO controller. To infer the search space where the the optimal reservation vector lies in, the ARO scout sends a sequence of reservation vectors to the NRR through the simple reservation interface. As further described in Section 3 and Section 4, for each reservation vector submitted from the ARO controller, the ARO scout might submit tens or hundreds of reservation vectors to the NNR to get an accurately-inferred search space, potentially, leading to a high overhead. As such, to reduce the total latency to find the optimal reservation vector, the ARO scout is placed with the network instead of the user. This design decision reduces the user-network communication latency by 20x as demonstrated in the evaluation section. More importantly, this design does not expose the private information of the user (*i.e.*, $util(\mathbf{x})$) to the network, as the ARO controller does not send such information to the scout.

Network Resource Reservation Module: The NRR module is a private component belonging to the network. Its primary role is to verify whether the reservation vectors submitted by the ARO scout can be satisfied. Upon receiving a reservation vector from the ARO scout, the NRR extracts the relevant constraints from the network. The constraints include both physical network constraints (*e.g.*, if two flows share a same link, their allocated bandwidths cannot exceed the link's available bandwidth), and network policies (*e.g.*, rate limiting, etc.) The constraints are captured as an abstraction of linear inequalities (Gao et al. 2016; Xiang et al. 2018). For example, to capture the physical network constraints, the NNR first retrieves the routes (i.e., sequence of traversed links) for each flow. Then, for each link l in the network, the NNR generates the following linear inequality to ensure that the allocated bandwidths to the flows do not exceed the link's available bandwidth:

$$\sum x_i \leq w_l, \forall f_i \text{ that uses } l \text{ in this route},$$

where w_l is the available bandwidth on link l. Considering the example in Figure 1, the NRR module generates the following linear inequalities:

$$\begin{array}{ll} x_1 \leq 100 & \forall l_u \in \{l_1, l_2, l_5, l_6\}, \\ x_2 \leq 100 & \forall l_u \in \{l_7, l_8, l_{11}, l_{12}\}, \\ x_1 + x_2 \leq 100 & \forall l_u \in \{l_3, l_4\}, \\ x_1, x_2, x_3 \geq 0. & \end{array} \quad (1)$$

Then, the NRR generates additional linear inequalities to represent the network's internal traffic engineering policies, such as load-balancing and bandwidth limiting. For example, suppose the network wants to limit the total bandwidth of flows f_1 and f_2 to be no more than 80 Mbps even if there is no common link in their routes. Then a linear inequality $x_1 + x_2 + x_3 \leq 80$ is generated to represent this policy. Geometrically, the abstraction of linear inequalities represents the *bandwidth feasible region* of the network for providing bandwidths to a set of flows.

Finally, for each generated linear inequality, the NRR checks if it is satisfied by the bandwidth values specified in the reservation vector. If any inequality is violated, it returns a FAILURE signal. Otherwise, it returns SUCCESS.

2.2 Workflow

Having presented the basic components of BoxOpt, we now briefly present its workflow to automatically compute and reserve the optimal network resources for a set of flows as follows (Figure 2):

• Step 1: The user submits a resource reservation request for a set of flows F to the ARO controller. The request also includes a concave utility function $util(\mathbf{x})$ of the bandwidths of F.

• Step 2: In an outer loop, the ARO controller iteratively selects reservation vectors to send to the ARO scout. The selection of the reservation vectors is described in Section 3.4, Algorithm 3. In return, for each reservation vector, the ARO scout determines and replies with an inferred search space.

• Step 3: In an inner loop, upon receiving a reservation vector from the controller, the ARO scout interacts with the NRR, according to Algorithm 1 from Section 3.3, to infer the next search space and send it back to the ARO controller.

The nested iteration of Step 2 and 3 stops when the ARO controller converges to the optimal reservation vector maximizing $util(\mathbf{x})$.

• Step 4: The ARO controller sends the optimal reservation vector to the NRR module to reserve the optimal resources for the user.

• Step 5: The ARO controller confirms with the user that the optimal network resource reservation has been successful.

2.3 Key Challenge

Through the introduction of its architecture and workflow, we show that BoxOpt is *privacy-preserving* by design: neither the user nor the network exposes the private information (*i.e.*, internal optimization objective of the user and the bandwidth capacity region of the network) to the other party. As such, the remaining key challenge for BoxOpt lies in Step 2 and 3: *how can the ARO module interact with the NRR module through the simple reservation interface to compute the optimal network resource reservation?*. To address this challenge, we first give a formal, mathematical formulation.

Specifically, we first model the NRR module as a *resource membership oracle*. Without loss of generality, we use $\mathbf{Ax} \leq \mathbf{b}$ to denote the set of linear inequalities generated by the NRR module, and use $K : \{\mathbf{x} | \mathbf{Ax} \leq \mathbf{b}, \mathbf{x} \geq \mathbf{0}\}$ to represent the bandwidth feasible region for a set of flows F. In this way, we give the definition of resource membership oracle:

Definition 1. *[Reservation Membership Oracle (ReMEM)] Given a reservation vector $\check{\mathbf{x}}$, return YES if $\check{\mathbf{x}} \in K$, and return NO otherwise.*

$ReMEM(\check{\mathbf{x}})$ accurately captures the interaction between the ARO scout and the NRR module. Next, we formally define the problem of network resource reservation optimization via simple reservation interface.

Problem 1 (Optimization via Membership Oracle)**.** *Find the optimal solution to the following optimization problem*

$$\textit{maximize} \quad util(\mathbf{x}), \quad (2)$$

subject to,

$$\mathbf{A}\mathbf{x} \leq \mathbf{b}, \tag{3}$$

$$\mathbf{x} \geq \mathbf{0}, \tag{4}$$

without the knowledge of $\mathbf{A}$ *and* $\mathbf{b}$*, but only using ReMEM defined in Definition 1.*

Maximizing $util(\mathbf{x})$ subject to $K : \{\mathbf{x}|\mathbf{A}\mathbf{x} \leq \mathbf{b}, \mathbf{x} \geq \mathbf{0}\}$ is a classic convex optimization problem. There has been a rich body of literature on how to efficiently solve such problems (Boyd and Vandenberghe 2004). However, most of the existing algorithms require the knowledge of the feasible region (in our case $\mathbf{A}\mathbf{x} \leq \mathbf{b}$). One may think of a strawman to learn K through the ReMEM oracle, and apply the standard optimization techniques to find the optimal $\mathbf{x}$. However, finding the feasible region through a membership oracle is NP-hard (Bessiere et al. 2017), making this strawman impractical. In contrast, as we will present next, BoxOpt resorts to efficient oracle transformation techniques in optimization and learning theory (Lee, Sidford, and Vempala 2017; Lovasz, Grotschel, and Schrijver 1993) to solve this problem (*i.e.*, efficiently learn the optimal resource reservation via membership oracle).

3 Optimizing Network Resource Reservation via Simple Reservation Interface

Having formally defined the key challenge for BoxOpt as a problem of optimization via membership oracle, this section discusses how we solve this problem. For presentation clarity, this section starts by reviewing some concepts in optimization theory. Then, it presents the basic idea of our solution, followed by its details.

3.1 Notations

Unless explicitly noted, we use v to denote a scalar and $\mathbf{v}$ to denote a vector of n dimensions, where n is the number of flows the user wants to reserve bandwidth for (see Section 2.1). We use $\|\mathbf{v}\|_2 = \sqrt{\sum v_i^2}$ to denote the Euclidean norm of $\mathbf{v}$, and use $\|\mathbf{v}\|_\infty = \max|v_i|$ to denote the maximum norm of $\mathbf{v}$. We use $B_2^+(\mathbf{m}, \eta) = \{\mathbf{x}|\|\mathbf{x} - \mathbf{m}\|_2 \leq \eta, \mathbf{x} \geq \mathbf{0}\}$ to denote the set of all positive vectors whose Euclidean distance to $\mathbf{m}$ is at most η, and use $B_\infty^+(\mathbf{m}, \eta) = \{\mathbf{x}|\|\mathbf{x} - \mathbf{m}\|_\infty \leq \eta, \mathbf{x} \geq \mathbf{0}\}$ to denote the set of all positive vectors whose maximum norm distance to $\mathbf{m}$ is at most η.

3.2 Basic Idea

Our approach to solve Problem 1 utilizes the equivalence and polar relationships between different oracles in optimization theory (Lovasz, Grotschel, and Schrijver 1993). In particular, we focus on the relationships between ReMEM with the following two oracles:

Definition 2. *[Resource Separation Oracle (ReSEP)] Given a reservation vector* $\check{\mathbf{x}}$*, return YES if* $\check{\mathbf{x}} \in K$*, and otherwise return a half space* $\{\mathbf{y}|\mathbf{p}^T(\mathbf{y} - \check{\mathbf{x}}) \leq \delta\}$ *that contains* K *but not* $\check{\mathbf{x}}$*.*

Definition 3. *[Resource Optimization Oracle (ReOPT)] Given a reservation request for a set of flows* F *and the utility function* $util(\mathbf{x})$*, find* $\mathbf{x}^* \in K$ *that maximizes* $util(\mathbf{x})$*.*

Algorithm 1: Resource Separation Oracle $ReSEP(\check{\mathbf{x}})$

```
Select ε ∈ (0, r], ρ ∈ (0, 1);
κ ← R/r;
if ReMEM returns YES for x̌ then
    return x̌ ∈ K ;
else if x̌ ∉ B_2(0, R) then
    return the half space {y|x̌^T(y − x̌) ≤ 0};
else
    r_1 ← rR^{−1/3} ε^{1/3};
    g̃ ← Subgradient(0, r_1, 4ε, 3κ);
    return the half space
      {y|g̃^T(y − x̌) ≤ (40n + 1)ρ^{−1} R^{2/3} κε^{1/3}};
```

Given that there exist efficient algorithms (*e.g.*, ellipsoid method) that can construct an optimization oracle through invoking a separation oracle with a polynomial number of iterations, if we can construct a separation oracle through a polynomial number of calls to a membership oracle, we will be able to solve Problem 1.

One may think classic half space learning techniques can achieve such a construction of separation oracle via membership oracle. However, the problems are different. In half space learning, the goal is to compute a hyperplane to separate a set of given samples (in our case, the reservation vectors) from two predefined classes. In contrast, the goal of ReMEM to ReSEP construction is to compute a hyperplane separating K and a reservation vector not belonging to K by strategically choosing a minimal number of reservation vectors to send to ReMEM.

Specifically, we develop our solution to Problem 1 in two phases. First, we leverage recent progress on geometric algorithms (Lee, Sidford, and Vempala 2017) to develop an efficient algorithm that constructs ReSEP through invoking ReMEM for a polynomial number of times. Specifically, our algorithm expands the weak membership/separation oracle construction in (Lee, Sidford, and Vempala 2017) to strong membership/separation oracle construction. Second, we develop an ellipsoid-method-based algorithm that constructs ReOPT through local feasibility checks and invoking ReSEP for a polynomial number of times. Mapping these two phases to Step 2 and 3 in the workflow of BoxOpt, we see that the ARO scout is essentially the separation oracle ReSEP, and the ARO controller is the optimization oracle ReOPT (Figure 2). Next, we give the details of each phase.

3.3 From Resource Membership Oracle to Resource Separation Oracle

To construct ReSEP from ReMEM, we first define two auxiliary functions on vector $\mathbf{d} \in K$ given a reservation vector $\check{\mathbf{x}}$:

$$\alpha_{\check{\mathbf{x}}}(\mathbf{d}) \leftarrow \max_{\mathbf{d}+\alpha\check{\mathbf{x}} \in K} \alpha, \tag{5}$$

$$h_{\check{\mathbf{x}}}(\mathbf{d}) \leftarrow -\alpha_{\check{\mathbf{x}}}(\mathbf{d})\|\check{\mathbf{x}}\|_2 \tag{6}$$

We see that given a vector $\mathbf{d} \in K$, $\mathbf{d} + \alpha_{\check{\mathbf{x}}}(\mathbf{d})$ is the last vector on the line from $\mathbf{d}$ to $\mathbf{d} + \check{\mathbf{x}}$ that is in K, and that $-h_{\check{\mathbf{x}}}(\mathbf{d})$

Algorithm 2: Computing the subgradient of $h_{\check{\mathbf{x}}}(\mathbf{d})$ $Subgradient(\mathbf{d}, r_1, \tau, L)$

1 $r_2 \leftarrow \sqrt{\frac{\tau r_1}{\sqrt{n}L}}$;
2 Randomly select $\mathbf{y}$ from $B_\infty(\mathbf{d}, r_1)$ following a uniform distribution;
3 Randomly select $\mathbf{z}$ from $B_\infty(\mathbf{y}, r_2)$ following a uniform distribution;
4 **for** $i \leftarrow 1, \ldots, n$ **do**
5 Define line segment $B_\infty(\mathbf{y}, r_2) \cap \mathbf{z} + s\mathbf{e_i}$, where $s \in R$ and $\mathbf{e_i}$ is a vector whose elements are all zeros except the ith one;
6 Denote the endpoints of this line segment as $\mathbf{s_i}$ and $\mathbf{t_i}$, respectively;
7 Evaluate $h_{\check{\mathbf{x}}}(\mathbf{t_i})$ and $h_{\check{\mathbf{x}}}(\mathbf{s_i})$ using binary search and ReMEM;
8 $\tilde{g}_i = \frac{h_{\check{\mathbf{x}}}(\mathbf{t_i}) - h_{\check{\mathbf{x}}}(\mathbf{s_i})}{2r_2}$;
9 **return** $\tilde{\mathbf{g}}$;

is the Euclidean distance from d to this point. Without loss of generality, we assume that $B_2^+(\mathbf{0}, r) \subset K \subset B_2^+(\mathbf{0}, R)$ for some positive numbers r, R and such an assumption can be trivially satisfied in practice. Extending the proof technique in (Lee, Sidford, and Vempala 2017) for an n-dimensional ball to only a partial ball on the first orthant, we get

Lemma 1. *Given $\check{\mathbf{x}}$, $h_{\check{\mathbf{x}}}(\mathbf{d})$ is convex on K, and is $\frac{R+\theta}{R-\theta}$ Lipschitz in $B_2^+(\mathbf{0}, \theta)$ for $0 < \theta < r$.*

With these auxiliary functions and a theorem that for any Lipschitz function, it is linear on a small ball (Bubeck and Eldan 2016), we can construct ReSEP by computing the subgradient of $h_{\check{\mathbf{x}}}(\mathbf{d})$ at $\mathbf{d} = \mathbf{0}$, which can be computed by binary search and invoking ReMEM. The constructed separation oracle is presented in Algorithm 1, and the computation of the subgradient of $h_{\check{\mathbf{x}}}(\mathbf{d})$ is presented in Algorithm 2.

A key insight in Algorithm 1 is that it expands the applicability of similar construction process from weak membership/separation oracles to strong membership/separation oracles (*i.e.*, ReMEM and ReSEP). In particular, we have

Lemma 2. *There is a random variable ϕ with expectation $E(\phi) \leq 2n\sqrt{\frac{\tau L}{r_1}}$ such that $\forall \mathbf{q} \in K$,*

$$h_{\check{\mathbf{x}}}(\mathbf{q}) \geq h_{\check{\mathbf{x}}}(\mathbf{d}) + \tilde{\mathbf{g}}^T(\mathbf{q} - \mathbf{d}) - \phi\|\mathbf{q} - \mathbf{d}\|_\infty - 4nr_1L.$$

With this lemma, we show the correctness of Algorithm 1 in the following theorem.

Theorem 1. *If $\check{\mathbf{x}} \notin K$, Algorithm 1 yields a half space containing K but not $\check{\mathbf{x}}$ with probability $1 - \rho$.*

Proof. When $\check{\mathbf{x}} \notin B_2^+(\mathbf{0}, R)$, it is easy to see that the returned half space $\{\mathbf{y}|\check{\mathbf{x}}(\mathbf{y}) - \check{\mathbf{x}} \leq \mathbf{0}\}$ (Line 6 of Algorithm 1) contains K but not $\check{\mathbf{x}}$. When $\check{\mathbf{x}} \notin K$ but $\check{\mathbf{x}} \in B_2^+(\mathbf{0}, R)$, from Lemma 1, we know that $h_{\check{\mathbf{x}}}(\mathbf{d})$ has a Lipschitz constant of 3κ on $B_2^+(\mathbf{0}, \frac{r}{2})$. By selecting $\epsilon \in (0, r]$ and setting $r_1 = rR^{-\frac{1}{3}}\epsilon^{\frac{1}{3}}$ (Line 1 and 8 of Algorithm 1, respectively), we can get $B_\infty^+(\mathbf{0}, 2r_1) \subset B_2^+(\mathbf{0}, \frac{r}{2})$. As such, we can apply Lemma 2 and get that $\forall \mathbf{q} \in K$

$$h_{\check{\mathbf{x}}}(\mathbf{q}) \geq h_{\check{\mathbf{x}}}(\mathbf{0}) + \tilde{\mathbf{g}}^T \cdot \mathbf{q} - \phi\|\mathbf{q}\|_\infty - 12nr_1\kappa. \quad (7)$$

Next, because $\check{\mathbf{x}} \in B_2^+(\mathbf{0}, R)$, we have $-\frac{1}{\kappa}\check{\mathbf{x}} \in K$ and $h_{\check{\mathbf{x}}}(-\frac{1}{\kappa}\check{\mathbf{x}}) = h_{\check{\mathbf{x}}}(\mathbf{0}) - \frac{\|\check{\mathbf{x}}\|_2}{\kappa}$. Then we use Lemma 2 to get

$$h_{\check{\mathbf{x}}}(-\frac{1}{\kappa}\check{\mathbf{x}}) \geq h_{\check{\mathbf{x}}}(\mathbf{0}) + \tilde{\mathbf{g}}^T \cdot -\frac{1}{\kappa}\check{\mathbf{x}} - \frac{\phi}{\kappa}\|\check{\mathbf{x}}\|_\infty - 12nr_1\kappa, \quad (8)$$

As such, we then get

$$\tilde{\mathbf{g}}^T \cdot \check{\mathbf{x}} \geq \|\check{\mathbf{x}}\|_2 - \phi\|\check{\mathbf{x}}\|_\infty - 12nr_1\kappa^2. \quad (9)$$

Next, because $\epsilon \in (0, r]$, $\check{\mathbf{x}} \notin K$ and $B_2^+(\mathbf{0}, r) \subset K$, we have $(1 - \frac{\epsilon}{r})K \subset K$. By definition of $h_{\check{\mathbf{x}}}(\mathbf{d})$, we have

$$h_{\check{\mathbf{x}}}(\mathbf{0}) \geq -(1 - \frac{\epsilon}{r})\|\check{\mathbf{x}}\|_2 \geq -\|\check{\mathbf{x}}\|_2 + \epsilon\kappa. \quad (10)$$

Adding Equations (9) and (10) and then subtracting $2\epsilon\kappa$ on the right hand side, we get

$$h_{\check{\mathbf{x}}}(\mathbf{0}) + \tilde{\mathbf{g}}^T \cdot \check{\mathbf{x}} \geq -\phi\|\check{\mathbf{x}}\|_\infty - 12nr_1\kappa^2 - \epsilon\kappa. \quad (11)$$

Next, we add Equation (11) to Equation (7) and get that $\forall \mathbf{q} \in K$,

$$\begin{aligned} h_{\check{\mathbf{x}}}(\mathbf{q}) &\geq \tilde{\mathbf{g}}^T(\mathbf{q} - \check{\mathbf{x}}) - \phi\|\mathbf{q}\|_\infty - \phi\|\check{\mathbf{x}}\|_\infty \\ &\quad -12nr_1\kappa - 12nr_1\kappa^2 - \epsilon\kappa \\ &\geq \tilde{\mathbf{g}}^T(\mathbf{q} - \check{\mathbf{x}}) - 2\phi R - 24nr_1\kappa^2 - \epsilon\kappa. \end{aligned} \quad (12)$$

$\forall \mathbf{q} \in K$, we have $h_{\check{\mathbf{x}}}(\mathbf{q}) \leq 0$, and then we can have $\tilde{\phi} \geq \tilde{\mathbf{g}}(\mathbf{q} - \check{\mathbf{x}})$, where $\tilde{\phi}$ is a random scalar independent of $\mathbf{q}$ that satisfies

$$E(\tilde{\phi}) \leq 4\sqrt{\frac{12\epsilon\kappa}{r_1}}nR + 24nr_1\kappa^2 - \epsilon\kappa. \quad (13)$$

Putting $r_1 = rR^{-\frac{1}{3}}\epsilon^{\frac{1}{3}}$ into Equation (13), we get

$$E(\tilde{\phi}) \leq 40n\epsilon^{\frac{1}{3}}R^{\frac{2}{3}}\kappa + \epsilon\kappa. \quad (14)$$

Further leveraging $\epsilon \leq r \leq R$, we get

$$E(\tilde{\phi}) \leq (40n + 1)\epsilon^{\frac{1}{3}}R^{\frac{2}{3}}\kappa. \quad (15)$$

Then we can finish the proof using Equation (15) and Markov inequality. □

In addition, observing Algorithm 1 and Algorithm 2, we see that the bottleneck to construct ReSEP is to compute $h_{\check{\mathbf{x}}}$ using binary search and ReMEM (Line 4-8 in Algorithm 2). As such, we give the following theorem on the complexity of Algorithm 1.

Theorem 2. *Algorithm 1 constructs the reservation separation oracle (ReSEP) through an $O(n \log R)$ calls on the reservation membership oracle (ReMEM).*

3.4 From Resource Separation Oracle to Resource Optimization Oracle

Having constructed ReSEP from ReMEM, we next develop an ellipsoid-method algorithm to construct ReOPT from ReSEP, which is summarized in Algorithm 3.

This algorithm adopts a binary search strategy to find the largest $util(\mathbf{x})$ that is feasible on K. In each main iteration (Line 3-24), it constructs a feasible problem

$$\begin{aligned} util(\mathbf{x}) &\geq u_{next}, \\ K : \mathbf{A}\mathbf{x} &\leq \mathbf{b}, \mathbf{x} \geq \mathbf{0}, \end{aligned} \quad (16)$$

and uses ellipsoid method to test if this problem is feasible. One difference from the classic ellipsoid method is: when evaluating if the center $\mathbf{p}$ of an ellipsoid E_i is a feasible solution, we first evaluate if $\mathbf{p}$ is feasible for $util(\mathbf{x}) \geq u_{next}$, and only invoke ReSEP if $util(\mathbf{p}) \geq u_{next}$. This is because $util(\mathbf{x})$ is kept at the ARO controller, where ReOPT runs, but not shared to ARO scout, where ReSEP resides. This would not affect the correctness of the ellipsoid method for verifying the feasibility of Equation (16) because a half space containing $util(\mathbf{x}) \geq u_{next}$ will also contain the feasible region defined in Equation (16). In the meantime, the privacy of user is also preserved.

Algorithm 3: Reservation Optimization Oracle $ReOPT(util(\mathbf{x}))$.

```
Compute the maximum and minimum of util(x) subject to
  x_i ∈ [0, R] where i = 1, ..., n and denote the value as
  u_max and u_min;
u_l ← u_min, u_r ← u_max;
while u_l < u_r do
    u_next ← (u_l + u_r)/2;
    Build an ellipsoid E_0 bounding B_2^+(0, R);
    V_l ← Vol(B_2^+(0, r)), i ← 0;
    feasible ← false;
    while Vol(E_i) ≥ V_l do
        p ← center of E_i;
        if util(p) < u_next then
            feasible ← false;
            H ← {y|(∇util(p))^T (y − p) >= 0};
        else if ReSEP(p) returns a half space H then
            feasible ← false;
        else
            feasible ← true;
            x* ← p;
            break;
        E_{i+1} ← the minimum-volume ellipsoid containing
          E_i ∩ H;
        i ← i + 1;
    if feasible == false then
        u_r ← u_next;
    else
        u_l ← u_next;
return x*;
```

We present the following theorem on the optimality and efficiency of Algorithm 3.

Theorem 3. *Algorithm 3 finds $\mathbf{x}^*$ that maximizes $util(\mathbf{x})$ subject to K through an $O(n^2)$ calls on the reservation separation oracle (ReMEM).*

Proof. The complexity result in this theorem follows the classic ellipsoid method. For the optimality claim, we observe that even if $util(\mathbf{x})$ is concave, K is still a polytope. As such, $\mathbf{x}^*$ will be a vertex of this polytope. In this way, the optimality of Algorithm 3 can be proved in the same way as the ellipsoid method optimally solves linear programming. □

Putting Theorems 2 and 3 together, we get the following theorem on the optimality and efficiency of BoxOpt.

Theorem 4. *BoxOpt finds $\mathbf{x}^*$ that maximizes $util(\mathbf{x})$ subject to K through an $O(n^3 \log R)$ calls on the reservation membership oracle (ReMEM).*

4 Performance Evaluation

We implement a prototype of BoxOpt and evaluate its performance on an operational federation network supporting large-scale distributed science collaborations, and using real traffic traces from recent science experiments. We first describe our setup, followed by the detailed results.

4.1 Methodology

We evaluate the performance of BoxOpt on the topology from LHC Open Network Environment (LHCONE), a global science network consisting of 62 institutes (Martelli and Stancu 2015). We randomly select a topology for each institute from the Topology Zoo (Knight et al. 2011), and then assemble the connections and topologies from previous steps into a unified large network. We replay the actual trace from the CMS experiment (cms-dashb), a main source of traffic in LHCONE. We focus on a 48-hour trace starting from December 14, 2017, consisting of 716 resource reservation requests. The number of flows in each request varies between 1 and 7. Because the CMS experiment is one of the largest ongoing distributed scientific experiments with complex, distributed analytics across tens of geographically distributed locations, we believe the trace is representative of complex data flow of modern distributed applications.

4.2 Results

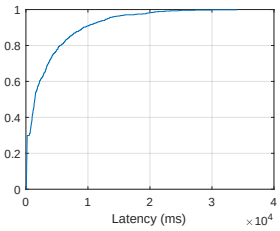
In our experiments, we set R and r in Algorithm 1 to be the maximum and minimum of link bandwidth in the network topology. We run extensive experiments by choosing different values of ϵ and ρ and different utility functions. In what follows, we present the results of one setting: maximizing total throughput when $\rho = 0.001$ and $\epsilon = \frac{1}{5}r$. Results of other settings are highly similar as this setting, hence are omitted due to page limit.

Correctness of BoxOpt: For each reservation request, we compare the optimal resource reservation computed by BoxOpt with the optimal solution to the problem $util(\mathbf{x})$ subject to K computed by a state-of-the-art optimization solver (*e.g.*, CPLEX (CPLEX 2018)). We find that in all 716 requests, BoxOpt outputs the same optimal solution as the solver does, *i.e.*, BoxOpt has a 100% correctness ratio.

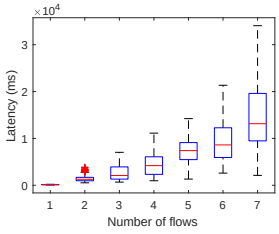
Efficiency of BoxOpt: As illustrated in Figure 2 (Section 2), the main bottleneck of BoxOpt is the communication latency between the optimization oracle at the ARO controller invoking the separation oracle at the ARO scout as the computation latency is ignorable. As such, we use the total communication latency to find the optimal resource reservation for each request to represent the efficiency of BoxOpt. Specifically, we assume the user is located at New York and the network is in Los Angeles. For each invocation of ReSEP at ARO scout, we assign it a round trip time (RTT) randomly chosen from the statistic RTT data collected in (global-ping

). Then the communication latency to find the optimal resource reservation for a given request is the sum of all RTTs incurred by corresponding ReSEP invocations.

Figure 3a plots the CDF of communication latency of all requests in the experiment. We observe that for 95% of the requests, BoxOpt is able to learn the optimal resource reservation within 13 seconds. This demonstrates the efficiency of BoxOpt to swiftly learn the optimal resource reservation via the simple reservation interface. Figure 3b plots the statistics of the communication latency for different sizes of reservation requests. The nonlinear increase of the latency is consistent with conclusion in Theorem 3. However, compared with the lasting time of network resource reservation (*e.g.*, hours and days) and the amount of data being transmitted (*e.g.*, TBs), BoxOpt is highly efficient.



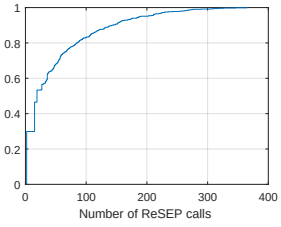
(a) CDF of the communication latency of BoxOpt.



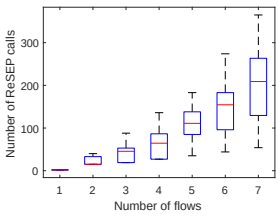
(b) Communication latency of BoxOpt with different sizes of request.

Figure 3: Efficiency of BoxOpt: total communication latency to compute the optimal resource reservation.

Efficiency of ReOPT: We next study the efficiency of the ReOPT oracle to learn the optimal resource reservation. To this end, we count the number of ReSEP invocations (*i.e.*, the bottleneck operation of the ReOPT oracle) for each request. Figure 4a gives the CDF of the number of ReSEP invocations. We observe that for 95% requests, the ReOPT learns the optimal reservation within 200 ReSEP calls. Figure 4b further breaks down the statistics based on the size of requests. We observe that the nonlinear increase of ReSEP invocations is consistent with Theorem 3 and Figure 3b.



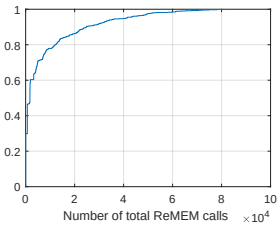
(a) CDF of the number of ReSEP invocations.



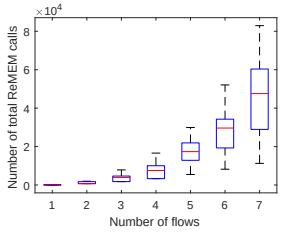
(b) Number of ReSEP invocations with different sizes of request.

Figure 4: Efficiency of ReOPT: number of ReSEP invocations to learn the optimal resource reservation.

Efficiency of ReSEP: In the end, we study the efficiency of the ReSEP oracle to infer the search space for ReOPT. To this end, we count the number of ReMEM invocations (*i.e.*, the bottleneck operation of the ReSEP oracle) for each request. Figure 5a gives the CDF of the number of ReMEM invocations. We observe that for 95% requests, the total ReMEM invocations required is within 30000. This large number demonstrates the necessity and benefits of putting ReSEP (*i.e.*, the ARO scout) with the network. Integrating this observation from Figure 4a and Figure 3a, we can conclude that this design improves the efficiency of BoxOpt (*i.e.*, the communication latency) by 20 times. Figure 5b further breaks down the statistics based on the size of requests. We observe that the almost linear increase of ReMEM invocations is consistent with Theorem 2.



(a) CDF of the number of ReMEM invocations.



(b) Number of ReMEM invocations with different sizes of request.

Figure 5: Efficiency of ReSEP: number of ReMEM invocations to learn the optimal resource reservation.

5 Related Work

Many network resource reservation systems have been developed and deployed (Campanella et al. 2006; Guok and Robertson 2006; Johnston, Guok, and Chaniotakis 2011; Riddle 2005; Zheng et al. 2005; Sobieski, Lehman, and Jabbari 2004). However, existing systems either are inefficient, or cause private information to be exposed. In contrast, BoxOpt adopts a novel approach to efficiently learn the optimal resource reservation through the limited feedback from the simple interface provided by reservation systems.

One area closely related to our problem is *constraint learning* (De Raedt, Passerini, and Teso 2018; Bessiere et al. 2017; Ruggieri 2012; Bessiere et al. 2004; Ruggieri 2013). We refer readers to (De Raedt, Passerini, and Teso 2018) for a comprehensive survey. Instead of learning all linear inequalities that compose the bandwidth feasible region, in BoxOpt, we show that the optimal solution to an optimization problem can be learnt efficiently and accurately without knowing any constraints. One future direction is to integrate constraint learning into BoxOpt to further accelerate the learning of the optimal resource reservation.

BoxOpt leverages several powerful tools from optimization theory (Lovasz, Grotschel, and Schrijver 1993; Boyd and Vandenberghe 2004; Lee, Sidford, and Vempala 2017). We expand the recent theoretical progress on efficient oracle constructions to a broader scenario. To the best of our knowledge, BoxOpt is the first working system that demonstrates the feasibility and benefits of learning the optimal solution of an optimization problem with only membership oracle. In addition to network resource reservation, it also sheds light for other areas such as multi-domain traffic engineering and collaborative data analytics.

6 Conclusion

We design BoxOpt, a novel, automatic learning system to efficiently learn the optimal resource reservations through the simple reservation interface of network resource reservation systems, without exposing the private information of network or user. We demonstrate its efficiency and efficacy through extensive evaluation using real network topology and trace.

Acknowledgment

The authors thank Christian Bessiere, Haizhou Du, Kai Gao, Chin Guok, Max Del Giudice, Chris Harshaw, Amin Karbasi, Yin Tat Lee, Geng Li, Yang Liu, John MacAuley, Harvey Newman, Lam M. Nguyen, Salvatore Ruggieri, Mudhakar Srivatsa, Xin Wang, Jingxuan Zhang and Rui Zhang for their help during the preparation of this paper. The authors also thank the anonymous reviewers for their valuable comments. This research is supported in part by NSFC grants 61702373, 61672385, 61701347, and 61672349; China Postdoctoral Science Foundation 2017-M611618; NSF awards CCF-1637385, CCF-1650596, OAC-1440745; Google Research Award; and the U.S. Army Research Laboratory and the U.K. Ministry of Defence under Agreement Number W911NF-16-3-0001.

References

Atkinson, D. S., and Vaidya, P. M. 1995. A cutting plane algorithm for convex programming that uses analytic centers. *Mathematical Programming* 69(1-3):1–43.

Bertsimas, D., and Vempala, S. Solving convex programs by random walks. In *Proceedings of the 34th ACM STOC*.

Bessiere, C.; Hebrard, E.; Hnich, B.; and Walsh, T. 2004. Disjoint, partition and intersection constraints for set and multiset variables. In *International Conference on Principles and Practice of Constraint Programming*, 138–152. Springer.

Bessiere, C.; Koriche, F.; Lazaar, N.; and O'Sullivan, B. 2017. Constraint acquisition. *Artificial Intelligence* 244:315 – 342.

Boyd, S., and Vandenberghe, L. 2004. *Convex optimization*. Cambridge university press.

Bubeck, S., and Eldan, R. 2016. Multi-scale exploration of convex functions and bandit convex optimization. In *Conference on Learning Theory*, 583–589.

Campanella, M.; Krzywania, R.; Reijs, V.; Wilson, D.; Sevasti, A.; Stamos, K.; and Tziouvaras, C. 2006. Bandwidth on demand services for european research and education networks. In *IEEE Bandwidth on Demand 06*, 65–72. IEEE.

CMS Task Monitoring. http://dashb-cms-job.cern.ch/.

2018. Ilog cplex.

De Raedt, L.; Passerini, A.; and Teso, S. 2018. Learning constraints from examples. In *Proceedings in Thirty-Second AAAI Conference on Artificial Intelligence, AAAI, New Orleans, USA*, 02–07.

Gao, K.; Gu, C.; Xiang, Q.; Wang, X.; Yang, Y. R.; and Bi, J. 2016. ORSAP: abstracting routing state on demand. In *24th IEEE International Conference on Network Protocols, ICNP 2016, Singapore, November 8-11, 2016*, 1–2.

Gao, K.; Xiang, Q.; Wang, X.; Yang, Y. R.; and Bi, J. 2017. Nova: Towards on-demand equivalent network view abstraction for network optimization. In *IWQoS'17*, 1–10.

Global Ping Statistics - WonderNetwork, 2018. https://wondernetwork.com/pings/.

Guok, C., and Robertson, D. 2006. Esnet on-demand secure circuits and advance reservation system (oscars). *Internet2 Joint* 92.

Heorhiadi, V.; Reiter, M. K.; and Sekar, V. 2016. Simplifying software-defined network optimization using sol. In *NSDI*, 223–237.

Johnston, W.; Guok, C.; and Chaniotakis, E. 2011. Motivation, design, deployment and evolution of a guaranteed bandwidth network service. In *Proceedings of the TERENA Networking Conference*.

Knight, S.; Nguyen, H. X.; Falkner, N.; Bowden, R.; and Roughan, M. 2011. The internet topology zoo. 29(9):1765–1775.

Lee, J.; Turner, Y.; Lee, M.; Popa, L.; Banerjee, S.; Kang, J.-M.; and Sharma, P. Application-driven bandwidth guarantees in datacenters. In *SIGCOMM'14*.

Lee, Y. T.; Sidford, A.; and Vempala, S. S. 2017. Efficient convex optimization with membership oracles. *arXiv preprint arXiv:1706.07357*.

Lovasz, L.; Grotschel, M.; and Schrijver, A. 1993. *Geometric algorithms and combinatorial optimization - 2nd corrected edition*. Springer.

Martelli, E., and Stancu, S. 2015. Lhcopn and lhcone: status and future evolution. In *Journal of Physics: Conference Series*, volume 664, 052025. IOP Publishing.

Mogul, J. C., and Popa, L. 2012. What we talk about when we talk about cloud network performance. *SIGCOMM CCR 12* 42(5):44–48.

Riddle, B. 2005. Bruw: A bandwidth reservation system to support end-user work. In *TERENA Networking Conference, Poznan, Poland*.

Ruggieri, S. 2012. Deciding membership in a class of polyhedra. In *ECAI*, 702–707.

Ruggieri, S. 2013. Learning from polyhedral sets. In *23rd Int. Joint Conference on Artificial Intelligence (IJCAI 2013)*, 1069–1075. AAAI Press.

Sobieski, J.; Lehman, T.; and Jabbari, B. 2004. Dragon: Dynamic resource allocation via gmpls optical networks. In *MCNC Optical Control Planes Workshop, Chicago, Illinois*.

Soulé, R.; Basu, S.; Marandi, P. J.; Pedone, F.; Kleinberg, R.; Sirer, E. G.; and Foster, N. 2014. Merlin: A language for provisioning network resources. In *CoNEXT'14*, 213–226. ACM.

Subramanian, K.; D'Antoni, L.; and Akella, A. Genesis: Synthesizing forwarding tables in multi-tenant networks. *ACM POPL'17*.

White, T. 2012. *Hadoop: The definitive guide*. O'Reilly Media, Inc.

Xiang, Q.; Zhang, J. J.; Wang, X. T.; Liu, Y. J.; Guok, C.; Le, F.; MacAuley, J.; Newman, H.; and Yang, Y. R. 2018. Fine-grained, multi-domain network resource abstraction as a fundamental primitive to enable high-performance, collaborative data sciences. In *Proceedings of the ACM SIGCOMM 2018 Conference on Posters and Demos*, 27–29. ACM.

Zaharia, M.; Chowdhury, M.; Das, T.; Dave, A.; Ma, J.; McCauley, M.; Franklin, M. J.; Shenker, S.; and Stoica, I. 2012. Resilient distributed datasets: A fault-tolerant abstraction for in-memory cluster computing. In *NSDI'12*, 2–2. USENIX Association.

Zheng, X.; Veeraraghavan, M.; Rao, N. S.; Wu, Q.; and Zhu, M. 2005. Cheetah: Circuit-switched high-speed end-to-end transport architecture testbed. *IEEE Communications Magazine* 43(8):S11–S17.