Distributed Optimization via Local Computation Algorithms

Palma London, Niangjun Chen, Shai Vardi, Adam Wierman
Computing and Mathematical Sciences, *California Institute of Technology*
Pasadena, California
{plondon, ncchen, svardi, adamw}@caltech.edu

***Abstract*—We propose a new approach for distributed optimization based on an emerging area of theoretical computer science – local computation algorithms. The approach is fundamentally different from existing methodologies and requires far less communication to compute the solution to a convex optimization problem than existing techniques. We develop an algorithm, LOCO, that given a convex optimization problem P with m variables and a sparse linear constraint matrix with n constraints, provably finds a solution as good as that of the best online algorithm for P. Each node computes its own part of the solution using $O(\log(n+m))$ messages with high probability. LOCO is robust to link failures and adaptive to dynamic settings. In addition to analytic results, we show numerically that the performance improvements over classical approaches for distributed optimization are significant. LOCO uses orders of magnitude less communication than existing distributed algorithms.**

I. Introduction

The goal of this paper is to introduce a new, fundamentally different approach to distributed optimization based on an emerging area of theoretical computer science – *local computation algorithms* (LCAs) [1]. This work, for the first time, imports techniques from LCAs into the domain of distributed convex optimization. The approach allows a large family of problems to be solved using far less communication than existing techniques require.

There are a wide variety of approaches for distributed optimization, which broadly fall into the categories of dual decomposition [2]–[4], primal decomposition [5], [6], and consensus algorithms [7], [8]. See [9], [10] for a survey.

While these approaches are *distributed*, they are not *local*. A local algorithm is one where a query about a small part of a solution to a problem can be answered by communicating with only a small neighborhood around the part queried.[1] Neither dual decomposition nor consensus methods are local: answering a query about a piece of the solution requires global communication. Despite the significant work on distributed algorithms for optimization, the problem of designing a distributed, *local* optimization algorithm is open.

This work introduces an algorithm, LOCO, (LOcal Convex Optimization) that is both *distributed* and *local*. It requires only a small amount of local communication, in contrast to dual decomposition and consensus methods which require global communication. While global dependencies exist in the problem, our method enables us produce a high quality solution while only observing local dependencies. Despite requiring little communication, LOCO returns a solution that is close to optimal. We provide worst-case guarantees on the performance of LOCO with respect to the relative error and communication complexity.

We consider general convex covering or packing problems with linear constraints. Many problems in networked control fall into this framework. Settings where multiple, distributed, cooperative agents need to solve an optimization problem to control a networked system are numerous and varied. Examples include management of content distribution networks and data centers [11], [12], communication network protocol design [13]–[15], trajectory optimization [16], [17], formation control of vehicles [18], [19], sensor networks [20], [21], control of power systems [22], [23], and management of electric vehicles and distributed storage devices [24], [25].

We demonstrate LOCO's performance on a canonical convex optimization problem: *network utility maximization* (NUM). This problem is a powerful tool for the design of distributed control policies [13], [26], [27]. The NUM framework represents a control problem as a convex optimization problem and seeks to maximize the aggregate utility of the individual agents in the network. The utility functions of the agents can be chosen in order to ensure maximal throughput, proportional fairness, etc., as desired; see Section II for an overview. NUM emerged in the context of the design of source rate control in communication networks [13], [15], but has since been applied in a wide array of applications, including optimization across layers of communication protocol [27], [28], market design and demand response in power systems [29], [30].

The nature of the NUM problem and the applications mentioned above illustrate how well suited local computation methods are for distributed optimization. For example, it would be highly desirable that any local failure in a network effect only part of an algorithm's execution. With LOCO, this is true; failures only affect a small number of nodes in the neighborhood of the failure. In comparison, if a node in a distributed system goes offline while a dual decomposition algorithm is executing, the whole process is brought to a halt. LOCO offers a robustness to network failures on a scale that even decomposable algorithms like *alternating method of multipliers* (ADMM) do not. Similarly, lag in a single edge affects the computation of the entire solution in the former

[1]*Local* is an overloaded term in the literature. We refer to local in the sense of [1]. See Subsection I-B for a more comprehensive definition and example.

setting, while most computations are not affected when the computations are local. For example, consider Figure 1(a). If the solution at source 1 is required, the only other sources involved in the calculation are 2 and 3. If a dual decomposition method was used, all the nodes in the graph would be involved in the calculation of source 1's part of the solution.

Another advantage of local computation is that it makes the system robust to dynamic changes in the network. An arrival of a new node in the network requires recomputing the entire solution if the algorithm is not local, but requires only a few local messages and computations if the algorithm is local. In Figure 1(a), if source 4 is a newly arrived node, LOCO does not need to recompute the solution to source 1, but existing methods would require a recomputation of all the nodes.

A. *Summary of Contributions*

Our results are the following.

1) We develop an algorithm, LOCO, that is both *distributed* and *local*, to solve general convex covering or packing problems with linear constraints in a distributed manner.
2) We provide worst-case guarantees on the performance of LOCO with respect to the relative error and the number of messages it requires. Given a convex optimization problem P with m variables and a sparse linear constraint matrix with n constraints, LOCO provably finds a solution as good as that of the best online algorithm for P. Each node computes its own part of the solution using $O(\log(n+m))$ messages with high probability.
3) We apply LOCO to a canonical optimization problem: network utility maximization, and demonstrate numerically that the performance improvements over classical approaches for distributed optimization are significant.

Due to space restrictions, we only consider the variant of NUM that maximizes throughput, which amounts to solving a distributed linear program. We focus on this case because it is particularly well-studied and, in addition, the objective function is linear, which in many cases is known to produce the worst performance guarantee for online convex optimization problems [31], [32]. In order to compare LOCO with existing methods, we perform a numerical experiment comparing LOCO and ADMM. Other comparison methods are possible, which we discuss in Section II. We show that LOCO requires orders of magnitude less communication than ADMM.

B. *Related literature*

The key idea behind LOCO is an extension of recent results on local computation algorithms. The LCA model was formally introduced by Rubinfeld et al. [1], after many algorithms within the framework had recently appeared in distinct areas, e.g., [33]–[35]. LCAs have received increasing attention in the years that followed as the importance of local, distributed computing has grown with the increasing scale of problems in distributed systems, the Internet of Things, etc.

Much of the focus of LCAs has, to this point, been on graph problems such as matching, maximal independent set, and coloring [36]–[39]. This paper extends the LCA literature by moving from graph problems to general optimization problems, which have not been studied in the LCA community previously.

In particular, a key insight of the field is that online algorithms can be converted into local algorithms in graph problems with bounded degree (e.g., [38], [40]). Mansour et al. [40] showed a general reduction from LCAs to online algorithms on graphs with bounded degree. The key technical contribution of our work is extending this technique to design LCAs for convex programs.

The main idea behind LCAs is to compute a piece of the solution to some algorithmic problem using only information that is close to that piece of the problem (as opposed to the complete global solution). More concretely, an LCA receives a *query* and is expected to output the part of the solution associated with the query. For example, an LCA for maximal matching would receive as a query an edge, and its output would be "yes/no," corresponding to whether or not the edge is part of the required matching. The two requirements are (i) the replies to all queries are consistent with the same solution, and (ii) the reply to each query is "efficient," for some natural notion of efficient.

For graph problems studied within the LCA framework, efficiency criteria are typically the number of probes to the graph, the running time and the amount of memory required [41]. In contrast to previous work whose primary focus was probe, time and space complexities, the efficiency criterion we use is the number of messages or communication required, as this is usually the expensive resource in distributed computing.

Distributed optimization is a field with a long history. Beginning in the 1960s, approaches emerged for solving large scale linear programs in a distributed manner. Early approaches by Benders [3], Dantzig and Wolfe [4], [42], and Everett [2], developed the ideas of dual decomposition. These approaches can be generalized to nonlinear optimization via the subgradient method [6], [43], [44]. Other approaches include primal decomposition [5], [6], and consensus-based schemes [7], [8], [10]. The application of these techniques to NUM problems came with [13], [26], which apply the idea of dual decomposition to NUM.

II. Network Utility Maximization

In order to illustrate the application of local computation algorithms to distributed optimization, we focus on the classic setting of network utility maximization (NUM). The NUM framework is a general class of optimization problems that has seen wide-spread application to distributed control in domains from the design of TCP congestion control [13]–[15], [26] to understanding of protocol layering as optimization decomposition [27], [28] and power system demand response [29], [30]. For a recent survey, see [45].

A. *Model*

The NUM framework considers a network containing a set of sources $\mathcal{S} = \{1, \ldots, m\}$ and links $\mathcal{L} = \{1, \ldots, n\}$ of capacity c_j, for $j \in \mathcal{L}$. Source $i \in \mathcal{S}$ is characterized by

$(L_i, f_i, \underline{x}_i, \bar{x}_i)$: a path $L_i \subseteq \mathcal{L}$ in the network; a concave utility function $f_i : \mathbb{R}_+ \to \mathbb{R}$; and the minimum and maximum transmission rates of source i.

The goal is to maximize the sources' aggregate utility. Source i attains a concave utility $f_i(x_i)$ when it transmits at rate x_i along path L_i, within the minimum and maximum rates allowed. The maximization of aggregate utility is formulated as follows,

$$\begin{aligned} \max_x \quad & \sum_{i=1}^{m} f_i(x_i) \\ \text{subject to} \quad & A^T x \le c \\ & \underline{x} \le x \le \bar{x}, \end{aligned}$$

where $A \in \mathbb{R}_+^{m\times n}$ is defined as $A_{ij} = 1$ if $j \in L_i$ and 0 otherwise. Choices of f_i correspond to different network goals. For example, setting $f_i(x_i) = x_i$ maximizes throughput; $f_i(x_i) = \log(x_i)$ achieves proportional fairness; $f_i(x_i) = -1/x_i$ minimizes potential delay; these are common goals in communication network applications [26], [46].

Our complexity results hinge on the assumption that the constraint matrix A is sparse. The *sparsity* of A is defined as $\max\{\alpha, \beta\}$, where α and β denote the maximum number of non-zero entries in a row and column of A respectively. Formally, we say that A is *sparse* if the sparsity of A is bounded by a constant. This assumption usually holds in network control applications since α is the maximum number of sources sharing a link, which is typically small compared to m, and β is the maximum number of links each source uses, which is typically small compared to m.[2]

B. Distributed Algorithms for Network Utility Maximization

Given the NUM formulation above, the algorithmic goal is to design a protocol that efficiently finds an approximately optimal solution. If the network is huge, it is often beneficial to distribute the solution, as performing the entire computation on a single machine is too costly [15], [48].

There is a large body of work within the networked control and communication networks literature that seeks to design distributed optimization algorithms [13], [14], [27]. Dual decomposition algorithms are particularly prominent in this setting. However, many such methods require a strictly concave objective function [26], and cannot be applied to the case of throughput maximization, i.e., linear f_i, when NUM is a linear program. In this paper we focus on the case of throughput maximization because in many cases linear objectives produce the worst performance guarantee for online convex optimization problems [31], [32]. One prominent algorithm that does apply in the case of throughput maximization is ADMM, which was introduced by [49] and has found broad applications in e.g., denoising images [50], support vector machine [51], and signal processing [52]–[54]. As a result, we use ADMM as a benchmark for comparison in this paper. For completeness, the application of ADMM to NUM is described in the extended version of this paper [55].

[2]When α is large, many links will be congested and all sources will experience greater delay, the routing protocol (IP) will start using different links; also, due to the small diameter of the Internet graph [47], β is small compared to m.

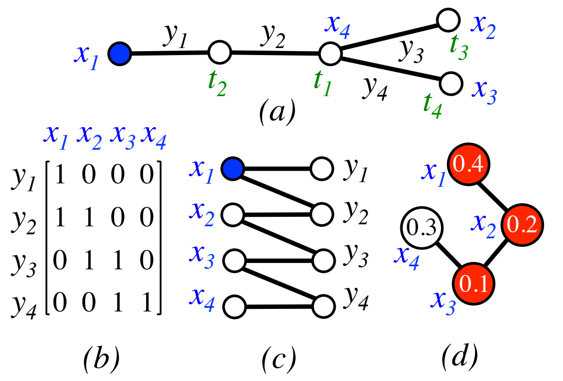

Fig. 1: An illustration of LOCO on a toy graph with five nodes. There are four sources, associated with primal variables x_1, x_2, x_3, x_4, whose paths end in destinations t_1, t_2, t_3, t_4 respectively. There are four links, associated with dual variables y_1, y_2, y_3, y_4. The graph is depicted in (a); the constraint matrix for NUM is given in (b); the bipartite graph representation of the matrix in (c); and the dependency graph in (d). The rank of each primal variable (source) is written in the node representing the variable in the dependency graph. Shaded nodes represent the query set $S(x_1)$ for source 1.

C. Performance metrics

Distributed algorithms for NUM should perform well on two measures. The first is *message complexity*: the number of messages that are sent across links of the network in order to compute the solution. When the algorithm uses randomization, we want the message complexity to hold with probability at least $1 - \frac{1}{m^\gamma}$, where where m is the number of vertices in the network and $\gamma > 0$ can be an arbitrarily large constant. We denote this by $1 - \frac{1}{\operatorname{poly} m}$. We do not bound the size of the messages, but note that in both our algorithm and ADMM the message length will be of order $O(\log m)$.

The second is the *approximation ratio*, which measures the quality of the solution provided by the algorithm. Specifically, an algorithm is said to γ-approximate a maximization problem if its solution is guaranteed to be at least $\frac{OPT}{\gamma}$, where OPT is the value of the optimal solution. If the algorithm is randomized, the approximation ratio is with respect to the expected size of the solution. We will compare the performance of LOCO with iterative algorithms such as ADMM, for which approximation ratio is not a standard measure. Thus in our empirical results, comparison with the optimal solution is made using *relative error*, defined in Section IV-A, which is related to but slightly different from the approximation ratio.

III. Local Convex Optimization

In this section, we introduce our local algorithm for distributed convex optimization, *LOcal Convex Optimization* (LOCO). In LOCO, every source in the network computes its portion of a near optimal solution using a small number of messages, without needing global communication. This is in contrast to decomposition methods, e.g. ADMM, which are *global*; they spread information necessary to find an optimal solution throughout the whole network over a series of iterations. LOCO has provable worst-case guarantees on both its approximation ratio and message complexity, and improves on the communication overhead of global decomposition methods by orders of magnitude in practice.

A. An overview of LOCO

The key insight in the design and analysis of LOCO is that any natural[3] online optimization algorithm can be converted into a local, distributed optimization algorithm. Note that the resulting distributed algorithm is for a static problem, *not* an online one. Further, after this conversion, the distributed optimization algorithm has the same approximation ratio as the original online optimization algorithm. Thus, given an optimization problem for which there exist effective online algorithms, these online algorithms can be converted into effective local, distributed algorithms.

More formally, to reduce a static optimization problem to an online optimization problem, we do the following. Let V be the set of variables of an optimization problem P. Let $r : V \rightarrow [0, 1]$ be a ranking function that assigns each variable x_i a real number between 0 and 1, uniformly at random. We call $r(x_i)$ x_i's *rank*. Suppose that there is some online algorithm ALG that receives variables sequentially and must augment the current set of variables so as to satisfy all constraints. Suppose furthermore that for each variable x_i, we find a small set of variables $S(x_i)$ (which we call x_i's *query set*) that arrived before it. We can think of $S(x_i)$ as a localized neighborhood around x_i, or source i. Then, restricting the set of variables of P to $S(x_i)$ results in ALG producing exactly the same solution for the variables that are present in the constraints involving the variables in $S(x_i)$. This is precisely what our algorithm does: it generates a random order of arrival for the variables, and for each variable x_i, it constructs a set $S(x_i)$ and simulates the online algorithm on it. An arbitrary ordering could mean that these sets are very large for some variables; to bound the size of these sets, we require that the constraint matrix of P is sparse and that the order generated is random. Pseudo-random orders suffice [38].

Concretely, there are two main steps in LOCO. In the first, LOCO generates a localized neighborhood for each variable or source. In the second, LOCO simulates an online algorithm on the localized neighborhood. Importantly, the first step is independent of the online algorithm, and the second is independent of the method used to generate the localized neighborhoods. Therefore, we can think of LOCO as a general methodology that can yield a variety of algorithms. For example, we can use different online algorithms for the second step of LOCO depending on whether we consider a linear NUM problem or a strictly convex NUM problem. More specifically, the two steps work as follows and are described in Algorithm 1 below.

Step 1) Generating a localized neighborhood: For clarity, we break the first step into three sub-steps, see also Figure 1.

Step 1a) Representing the constraint matrix as a bipartite graph: A boolean matrix A can be represented as a bipartite graph $G = (L, R, E')$ as follows. Each primal packing variable x_i, or equivalently each column of A^T which correspond to the sources in NUM, is represented by a vertex $v_i \in L$. Each dual covering variable y_j, which correspond to the links in NUM, by a vertex $v_j \in R$. An edge (v_i, v_j) is in E' if and only if $A_{ij} = 1$. Intuitively, edges represent which variables appear in which constraints. Note that the maximum degree of G is exactly the sparsity of A.

Step 1b) Constructing the dependency graph: We construct the *dependency graph* $H = (V, E)$ as follows. The vertices of the dependency graph are the vertices of L; an edge exists between two vertices in H if the corresponding vertices in G share a neighbor. Intuitively, H represents the "direct dependencies" between the primal variables; changing the value of any variable immediately affects all constraints in which it appears. The maximum degree of H is upper bounded by the square of the sparsity of A.

Step 1c) Constructing the query set: In order to build $S(x_i)$, the *query set of* x_i, we generate a random *ranking function* on the vertices of H, $r : V \rightarrow [0, 1]$. Given the dependency graph H and the ranking function r, we build the query set using a variation of BFS on the vertices in H as follows.

Initialize sets $S = T = \{x_i\}$. Scan all of x_i's neighbors and add them to both S and T. Now repeat the following iteratively: for every vertex $v \in T$, scan all of v's neighbors, denoted $N(v)$. For each $w \in N(v)$, if $r(w) \leq r(v)$, add w to S and to T. Once all of v's neighbors have been checked, remove v from T. Set T is used to keep track of vertices in the BFS that have been added to the query set, but whose neighbors have yet to been checked. Continue iteratively until T is empty; no more vertices can be added to S (that is, for every vertex $v \in S$ all of its neighbors that are not themselves in S have lower rank than v). If there are ties (i.e., two neighbors w, v such that $r(w) = r(v)$), we tie-break by ID. Any consistent tie breaking rule suffices [38].

Step 2) Simulating the online algorithm: Assume that we have an online algorithm for the problem that we would like to solve. We will internally simulate the execution of an online algorithm, where we assume that the columns of A^T and components of the cost function, or equivalently the variables x_i, arrive online, in the order determined by the ranking function r. The NUM problem is a concave packing maximization

[3]We require that the online algorithm have the following characteristic: knowing the output of the algorithm for the "neighbors" of a query q that arrived before q is sufficient to determine the output for q. We omit this technicality from the theorem statements as the online algorithm we use, and indeed all online algorithms for convex optimization that we are aware of, have this property. For a more in-depth discussion, we refer the reader to [38].

problem, so we require an online packing algorithm.[4] In this paper we use the online packing algorithm of Buchbinder and Naor [56, chapter 14], for which we provide the pseudocode in Appendix A. Depending on the type of optimization problem, various other online algorithms are well suited [57], [58].

In order to compute the value of x_i, source i considers its query set, $S(x_i)$. The online algorithm solves a problem that consists of only the variables contained in $S(x_i)$ along with all the constraints in which those variables appear. For clarity, call matrix $\hat{A}^T \in \mathbb{R}^{n \times |S(x_i)|}$ the matrix containing only the columns of A^T that correspond to the variables in $S(x_i)$. The algorithm will have $|S(x_i)|$ iterations in total. Note that the univariate non-negativity constraints do not arrive online and are known initially.

At each step of the algorithm, a new variable x_k, or equivalently the k-th new column of $\hat{A}^T$ and the k-th component of the cost function, arrive online. The result will be a solution for all the variables contained in $S(x_i)$. We return only the component corresponding to x_i. Claim 4 below shows that x_i's value is identical to its value if the online algorithm was executed on the entire program, with the variables arriving in the order defined by r. In order to compute the entire solution $x \in \mathbb{R}^m$, the above steps are taken for all sources.

B. Analysis of LOCO

Our main theoretical result shows that LOCO can compute solutions to convex optimization problems that are as good as those of the best online algorithms for the problems using very little communication. We then specialize this case to throughput maximization in NUM. While we focus on NUM in this paper, the following theorem and its proof applies to a wider family of problems as well. Specifically, the conversion from online to local outlined below can be used more broadly for any class of optimization problems for which effective online algorithms exist. Thus, improvements to online optimization problems immediately yield improved local optimization algorithms.

Theorem 1. *Let P be a problem with a concave objective function and linear inequality constraints, with m variables and n constraints, whose constraint matrix has sparsity σ. Given an online algorithm for P with competitive ratio $h(n, m)$, there exists a local computation algorithm for P with approximation ratio $h(n, m)$ that uses $2^{O(\sigma^2)} \log(n + m)$ messages with probability $1 - 1/poly(n, m)$.*

In particular, we have the following result, for NUM with a linear objective function.

Theorem 2. *Let P be a throughput maximization problem with m variables, n constraints, and a sparse constraint matrix. LOCO computes an $O(\log n)$ – approximation to the optimal solution of P using $O(\log(n + m))$ messages with probability $1 - 1/\operatorname{poly}(n, m)$.*

[4]Our framework applies to both packing maximization problems and the dual covering minimization problem. We require an online algorithm that can solve either problem. In the case of covering problems, constraints arrive online, and in the case of packing, variables arrive online.

Proof of Theorem 2. Theorem 1, the the following Claim 4 and Lemma 5, setting $B = 2\log(1 + n)$, imply Theorem 2. □

Algorithm 1: LOCO (LOcal Convex Optimization)

```
Input: $A^T \in \mathbb{R}^{n\times m}, f \in \mathbb{R}^m, c \in \mathbb{R}^n$
Step 1a) Construct bipartite graph $G = (L, R, E')$:
• vertices $v_i \in L$ for $i = 1...m$ correspond to primal packing
  variables $x_i$ (sources; columns of $A^T$)
• vertices $v_j \in R$ for $j = 1...n$ correspond to dual covering
  variables $y_j$ (links)
• an edge $(v_i, v_j)$ is in $E'$ if and only if $A_{ij} = 1$
Step 1b) Construct dependency graph $H = (V, E)$:
• vertices $V = L$
• add edges as follows:
for $\forall v_i, v_k \in L$ do
    $E \leftarrow (v_i, v_k)$ if $\exists\{w : w \in N(v_i)$ and $w \in N(v_k)\}$
Step 1c) Construct $S(x_i)$, the query set of $x_i$:
• Generate a random ranking function on the vertices of $H$,
  $r : V \to [0, 1]$
• Initialize $S = T = \{x_i\}$ and in the dependency graph $H$,
  $\forall v \in N(x_i)$, if $r(v) < r(x_i)$, then $S \leftarrow v$, $T \leftarrow v$
while $T$ is nonempty do
    for $\forall v \in T$ do
        for $\forall w \in N(v)$ do
            if $r(w) < r(v)$ then
                $S \leftarrow w$, $T \leftarrow w$, $T \leftarrow T \setminus v$
Step 2) Run an online algorithm to solve for $x_i$.
• Let matrix $\hat{A}^T \in \mathbb{R}^{n\times|S(x_i)|}$ be the matrix containing only
  the columns of $A^T$ that correspond to variables in $S(x_i)$.
for $k = 1...|S(x_i)|$ do
    $k$th column of $\hat{A}^T$ and $f_k$ arrives
    $x_k \leftarrow$ onlineAlgorithm($\hat{A}^T$, $f_k$)
```

The approximation ratio in Theorem 2 comes from the online algorithm presented and analyzed in [56] (see Lemma 5). The analysis of the online algorithm is for adversarial input; therefore it is natural to expect LOCO to achieve a much better approximation ratio in practice, as LOCO randomizes the order in which the constraints "arrive". It is an open question to give better theoretical bounds for stochastic inputs, and if such results are obtained they would immediately improve the bounds in Theorem 2.

The core technical lemma required for the proof of Theorem 1 is the following.

Lemma 3. *Let $G = (V, E)$ be a graph whose degree is bounded by d and let $r : V \to [0, 1]$ be a function that assigns to each vertex $v \in V$ a number between 0 and 1 independently and uniformly at random. Let T_{max} be the size* *of the largest query set of G: $T_{max} = \max\{|T_v| : v \in V\}$. Then, for $\lambda = 4(d+1)$,*

$$\Pr[|T_{max}| > 2^{\lambda} \cdot 15\lambda \log m] \leq \frac{1}{m^2}.$$

The proof of Lemma 3 uses ideas from a proof in [38], and employs a *quantization* of the rank function. Its proof is deferred to the extended version of the paper [55]. The following simple claim implies that the approximation ratio of LOCO is the same as that of the online algorithm.

In addition to Lemma 3, the following claim and technical lemma are needed to complete the proof of Theorem 2.

Claim 4. *For any source i, the value of x_i in LOCO's output is identical to its value in the output of the online algorithm.*

Proof. We present a proof by induction. Variables arrive in the order determined by r. For the base case, the variable with the smallest rank clearly has the same value in LOCO and in the online algorithm. For the inductive step, call x_i's rank r_i. Assume that the values of all of the variables whose rank is at most $r_i - 1$ is the same under LOCO and the online algorithm. Then by the inductive hypothesis, all of the neighbors of x_i in the dependency graph whose ranks are less than r_i have the same value under LOCO and the online algorithm. Variables with ranks less than r_i are the only values that the online algorithm uses to set the value of x_i. Hence, x_i will be set to have the same value as it would have if the online algorithm were used to solve the original problem with all m variables. □

The following lemma is a restatement of Theorem 14.1 in [56], adapted to throughput maximization.

Lemma 5. *For any $B > 0$, there exists is a B-competitive online algorithm to linearly-constrained NUM with n constraints; each constraint is violated by a factor at most $\frac{2\log(1+n)}{B}$.*

C. Contrasting LOCO and ADMM

LOCO fundamentally differs from dual ascent and dual decomposition methods. Dual ascent methods iterate until *global* optimality conditions are met. In order to check for global optimality, dual decomposition methods such as ADMM require communication amongst all nodes in the distributed network at each iteration.

LOCO operates in a completely different way. Once a node gets information about its query set, the node performs a *local* computation, only interacting with nodes in its query set. LOCO executes for a predetermined number of iterations, which is the size of the query set. This is in contrast to ADMM; the number of iterations required is unknown *a priori*.

While LOCO does not compute the optimal solution, dual decomposition style approaches will eventually converge to the true optimal. However, LOCO will be near-optimal, as we prove analytically above. The proven analytical bounds for LOCO are based on worst-case *adversarial* input. We show in Section IV-B that our empirical results outperform the theoretical guarantees by a considerable margin. This is partly because the ranking is done randomly rather than in an adversarial fashion. We elaborate on this in Section IV.

Note that there is a difference in the form of the theoretical guarantees for LOCO and dual ascent algorithms. Dual ascent algorithms have global convergence rate guarantees. There is no notion of global convergence in LOCO because each node computes its own solution independently of the others. Instead, LOCO has guarantees in terms of the approximation ratio.

IV. Case Study

Here we present the results of a simulation study demonstrating the empirical performance of LOCO on both synthetic and real networks. The results highlight that an orders-of-magnitude reduction in communication is possible with LOCO as compared to ADMM, which we choose as a prominent example of current approaches for distributed optimization. For concreteness, our experiments focus our numeric results on distributed linear programming, i.e., the case of linear NUM. This is the NUM setting where one could expect LOCO to perform the worst, given that linear functions are typically the worst-case examples for online convex optimization algorithms [31], [32].

A. Experimental setup

1) Problem Instances: For our first set of experiments, we generate random synthetic instances of linear NUM. Let $n = m$ and define the constraint matrix $A \in \mathbb{R}^{(n \times n)}$ as follows. Set $\tilde{A}_{ij} = 1$ with probability p and $\tilde{A}_{ij} = 0$ otherwise. Let $A = \tilde{A} + I_n$ to ensure each row of A has at least one non zero entry.[5] The vector $c \in \mathbb{R}^n$ is drawn *i.i.d.* from Unif$[0, 1]$. We set the minimum and maximum transmission rates to be $\underline{x}_i = 0$ and $\bar{x}_i = 1$. Finally, for the rank function used by LOCO we use a random permutation of the vertex IDs.[6]

For our second set of experiments, we use the real network from the graph of Autonomous System (AS) relationships in [59]. The graph has 8020 nodes and 36406 edges. In order to interpret the graph in a NUM framework, we associate each source with a path of links, ending at a destination node. To do this, for each source i in the graph, we randomly select a destination node t_i which is at distance ℓ_i, sampled *i.i.d.* from Unif$[\ell - 0.5\ell, \ell + 0.5\ell]$. We repeat this for several values of ℓ. (The distance between two nodes is the length of the shortest path between them.) Then, we designate the path L_i to be the set of links comprising the shortest path between the source and the destination. The vectors c, $\underline{x}$, and $\bar{x}$ are chosen in the same manner as for the synthetic networks.

2) Algorithm tuning: Our results focus on comparing LOCO and ADMM. Running ADMM requires tuning four parameters [48]. Unless otherwise specified, we set the relative and absolute tolerances to be $\epsilon^{rel} = 10^{-4}$ and $\epsilon^{abs} = 10^{-2}$,

[5]Note that this matrix does not have constant sparsity; however this can only increase the message complexity. Irregardless, it is possible to adapt the theoretical results to hold for this data as well, using techniques from [38].

[6]For the purposes of our simulations, such a permutation can be efficiently sampled, and guarantees perfect randomness. For larger n and m, it is possible to use pseudo-randomness with almost no loss in message complexity [38].

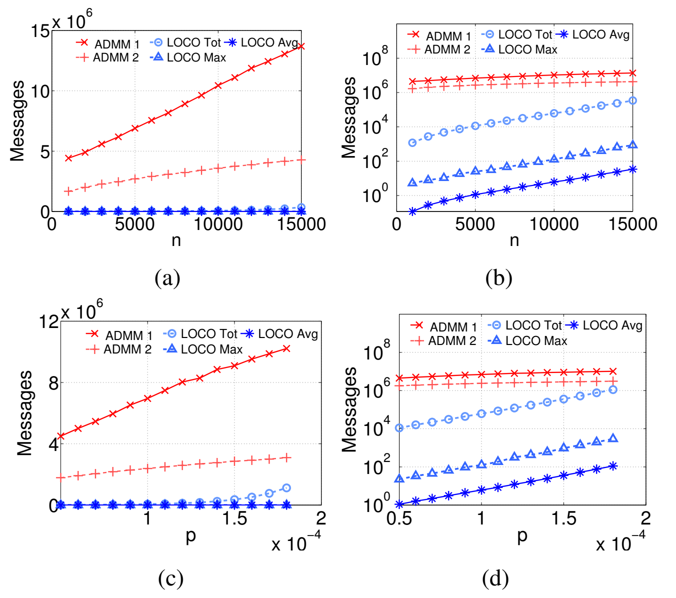

Fig. 2: Illustration of the number of messages required by ADMM and LOCO for the synthetic data set with results averaged over 50 trials. Plots (a) and (b) vary n while fixing sparsity $p = 10^{-4}$, showing the results in linear-scale and log-scale respectively. Plots (c) and (d) fix $n = 10^3$ and vary the sparsity p, showing the results in linear-scale and log-scale respectively.

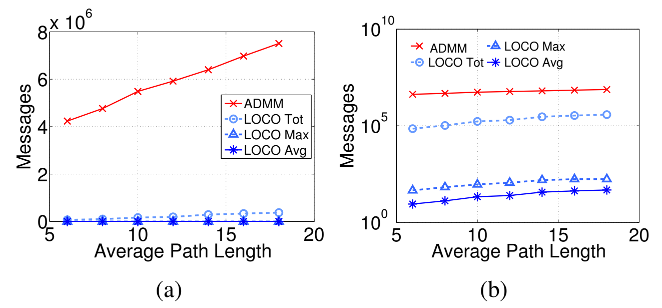

Fig. 3: Illustration of the number of messages required by ADMM and LOCO for the real network data with $n = 8020$ and various average path lengths $L(i)$.

the penalty parameter to be $\rho = 1$, and the maximum number of allowed iterations to be $t_{max} = 10000$. This is done to provide the best performance for ADMM: the parameters are tuned in the typical fashion to optimize ADMM [48]. Running LOCO requires tuning only one parameter: B, which governs the worst-case guarantee for the online algorithm used in step 2. A smaller B gives a "better guarantee", however some constraints may be violated. Setting $B = 2\ln(1+n)$ provides the best worst-case guarantee, and is our choice in the experiments unless stated otherwise. In fact, it is possible to tune B (akin to tuning ADMM) to specific data, as the constraints are often still satisfied for smaller B. In Figure 4 (c), we show the improvement in performance guarantee by tuning B, while keeping the dual solution feasible.

3) Metrics: For our numeric results, we evaluate ADMM and LOCO with respect to the quality of the solution provided and the number of messages sent.

To assess the quality of the solution we measure the *relative error*, which is defined as $\frac{|p^*-p^{LOCO}|}{|p^*|}$, where p^* is the optimal solution. For problem instances of small dimension, one can run an interior point method to check the optimal solution, but this is too tedious for large problem sizes. In the large dimension cases we consider, we regard p^* to be ADMM's solution with small tolerances, such that the maximum number of allowed iterations is never needed. Note that the relative error is an empirical, normalized version of the approximation ratio for a given instance.

To measure the number of *messages* used by each of the algorithms, we consider the following. For a distributed implementation of ADMM, two sets of n variables are updated on separate processors and reported to a central controller which updates another variable (see [48, Chapter 7.1]). The number of *messages* for a run of ADMM is twice the number of sources in the NUM problem, multiplied by the number of iterations required by ADMM. LOCO needs to use communication only to construct the query set; running the online algorithm does not require any communication. Therefore, the number of messages is proportional to the number of edges with at least one endpoint in the query set (this is the number of edges we need to send information over in order to construct the query set, see e.g., [38] for more details). We note that the number of messages depends both on the network topology and the realization of the ranking function.

B. Experimental Results

We now describe our empirical comparison of the performance of LOCO with ADMM.

Our first set of experiments investigates the communication used by ADMM and LOCO, i.e., the number of messages required. Figure 2 highlights that LOCO requires considerably fewer messages than ADMM, across both small and large n and varying levels of sparsity. More specifically, the figure shows that both the average and maximum amount of communication needed to answer a query about a specific piece of the solution under LOCO (LOCO Avg and LOCO Max respectively) are substantially lower than for ADMM. Further, even answering *every* query (LOCO Tot) requires only the same order of magnitude as ADMM. The figure includes ADMM with a tolerance ϵ^{rel} of 10^{-4} (ADMM 1) and 10^{-3} (ADMM 2). Even with suboptimal tolerance, which results in fewer iterations, ADMM still requires orders of magnitude more communication than LOCO.

Figure 3 shows the same qualitative behavior in the case of the real network data. In particular, the number of messages used is shown as a function of the average length of paths in the AS topology. We see that LOCO greatly outperforms ADMM for all tested average path lengths.

The improvement achieved by LOCO is possible because the size of the query sets used are small compared to the

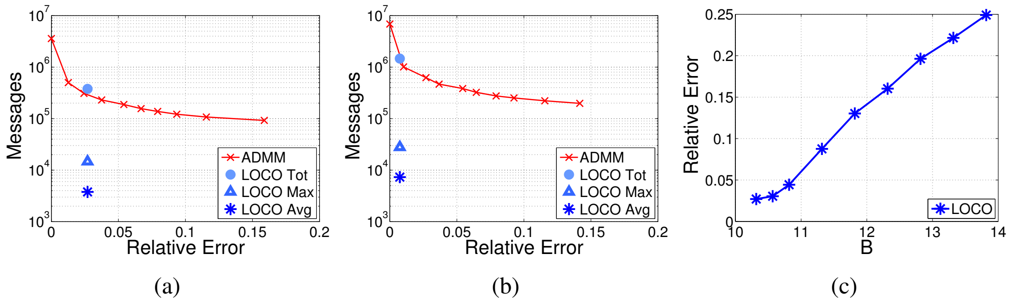

Fig. 4: Comparison of the relative error and the number of messages required by LOCO and ADMM. Plots (a) and (b) show the Pareto optimal curve for ADMM with a range of relative tolerances $\epsilon^{rel} \in \{10^{-4}, 10^{-1}\}$. Plot (c) depicts how tuning B effects the relative error. The right most point corresponds to $B = 2\ln(1+n)$.

number of sources. When $n = 10^3$, as in Figure 2, the number of nodes in the largest query set (over all trials) was 60.

We note that the improvement in the amount of communication is achieved at a cost: LOCO does not precisely solve the optimization, it only approximates the solution. When B is set to its worst-case guarantee (Figure 2), the relative error of LOCO ranges from 0.29 to 0.34.

Next we explore the tradeoff between message complexity and relative error. Figures 4 (a) and (b) illustrate the Pareto optimal frontier for ADMM: the minimal messages needed in order to obtain a particular relative error. We tune the parameters of ADMM and LOCO such that the algorithms have comparable relative error while LOCO Tot and ADMM require about the same number of messages. Unlike ADMM, LOCO cannot trade off the number of messages used with the relative error; LOCO corresponds to a single point in the figures. This point is outside the Pareto frontier of ADMM. Figure 4 (c) illustrates the impact of tuning B. Similarly to ADMM, tuning B can significantly improve the relative error; unlike ADMM, tuning B does not affect the communication complexity.

V. Concluding Remarks

We introduced a new, fundamentally different approach for distributed optimization based on techniques from the field of local computation algorithms. In particular, we designed a generic algorithm, LOCO, that constructs small neighborhoods and simulates an online algorithm on them. Due to the fact that LOCO is local, it has several advantages over existing methods for distributed optimization. In particular, it is more robust to network failures, communication lag, and changes in the system. To illustrate the benefits of LOCO we considered throughput maximization. The improvements of LOCO over ADMM in terms of communication in this setting are significant.

We view this paper as a first step toward the investigation of local computation algorithms for distributed optimization. In future work, we intend to continue to study the performance of LOCO in more general network optimization problems. Further, it would be interesting to apply other techniques from the field of local computation algorithms to develop algorithms for other settings in which distributed computing is useful, such as power systems and machine learning.

References

[1] R. Rubinfeld, G. Tamir, S. Vardi, and N. Xie, “Fast local computation algorithms,” in *Proc. 2nd Symposium on Innovations in Computer Science (ICS)*, 2011, pp. 223–238.

[2] H. Everett III, “Generalized lagrange multiplier method for solving problems of optimum allocation of resources,” *Operations research*, vol. 11, no. 3, pp. 399–417, 1963.

[3] J. F. Benders, “Partitioning procedures for solving mixed-variables programming problems,” *Numerische mathematik*, vol. 4, no. 1, pp. 238–252, 1962.

[4] G. B. Dantzig and P. Wolfe, “Decomposition principle for linear programs,” *Operations Research*, vol. 8, no. 1, pp. 101–111, 1960.

[5] L. S. Lasdon, *Optimization theory for large systems*. Courier Corporation, 1970.

[6] D. P. Bertsekas, *Nonlinear programming*. Athena Scientific, 1999.

[7] V. D. Blondel, J. M. Hendrickx, A. Olshevsky, and J. N. Tsitsiklis, “Convergence in multiagent coordination, consensus, and flocking,” in *Proc. of IEEE Conference on Decision and Control*, 2005, pp. 2996–3000.

[8] A. Nedić and A. Ozdaglar, “Convergence rate for consensus with delays,” *Journal of Global Optimization*, vol. 47, no. 3, pp. 437–456, 2010.

[9] D. P. Bertsekas and J. N. Tsitsiklis, *Parallel and Distributed Computation: Numerical Methods*. Prentice Hall, 1989.

[10] A. Nedic and A. Ozdaglar, “Cooperative distributed multi-agent optimization,” in *Convex Optimization in Signal Processing and Communications*, D. P. Palomar and Y. C. Eldar, Eds. Cambridge University Press, 2010.

[11] S. Borst, V. Gupta, and A. Walid, “Distributed caching algorithms for content distribution networks,” in *Proceedings of IEEE INFOCOM*, 2010, pp. 1–9.

[12] V. N. Padmanabhan, H. J. Wang, P. A. Chou, and K. Sripanidkulchai, “Distributing streaming media content using cooperative networking,” in *Proceedings of workshop on Network and operating systems support for digital audio and video*. ACM, 2002, pp. 177–186.

[13] F. P. Kelly, A. K. Maulloo, and D. K. Tan, “Rate control for communication networks: shadow prices, proportional fairness and stability,” *Journal of the Operational Research Society*, vol. 49, no. 3, pp. 237–252, 1998.

[14] S. H. Low, F. Paganini, and J. C. Doyle, “Internet congestion control,” *IEEE Control Systems*, vol. 22, no. 1, pp. 28–43, 2002.

[15] R. Srikant, *The mathematics of Internet congestion control*. Springer Science & Business Media, 2012.

[16] Y. Guo and L. E. Parker, “A distributed and optimal motion planning approach for multiple mobile robots,” in *Robotics and Automation, 2002. Proceedings. ICRA'02. IEEE International Conference on*, vol. 3, 2002, pp. 2612–2619.

[17] Y. Kuwata and J. P. How, “Cooperative distributed robust trajectory optimization using receding horizon milp,” *IEEE Transactions on Control Systems Technology*, vol. 19, no. 2, pp. 423–431, 2011.

[18] I. Suzuki and M. Yamashita, "Distributed anonymous mobile robots: Formation of geometric patterns," *SIAM Journal on Computing*, vol. 28, no. 4, pp. 1347–1363, 1999.

[19] R. L. Raffard, C. J. Tomlin, and S. P. Boyd, "Distributed optimization for cooperative agents: Application to formation flight," in *Proc. of IEEE Conference on Decision and Control*, vol. 3, 2004, pp. 2453–2459.

[20] R. Olfati-Saber, "Distributed Kalman filtering for sensor networks," in *Proc. of IEEE Conference on Decision and Control*, 2007, pp. 5492–5498.

[21] Y. Liao, H. Qi, and W. Li, "Load-balanced clustering algorithm with distributed self-organization for wireless sensor networks," *IEEE Sensors Journal*, vol. 13, no. 5, pp. 1498–1506, 2013.

[22] T. Erseghe, "Distributed optimal power flow using admm," *IEEE Transactions on Power Systems*, vol. 29, no. 5, pp. 2370–2380, 2014.

[23] Q. Peng and S. H. Low, "Distributed optimal power flow algorithm for radial networks, i: Balanced single phase case," *IEEE Transactions on Smart Grid*, 2016.

[24] Y. Cao, S. Tang, C. Li, P. Zhang, Y. Tan, Z. Zhang, and J. Li, "An optimized ev charging model considering tou price and soc curve," *IEEE Transactions on Smart Grid*, vol. 3, no. 1, pp. 388–393, 2012.

[25] L. Gan, U. Topcu, and S. H. Low, "Optimal decentralized protocol for electric vehicle charging," *IEEE Transactions on Power Systems*, vol. 28, no. 2, pp. 940–951, May 2013.

[26] S. H. Low and D. E. Lapsley, "Optimization flow control. I. Basic algorithm and convergence," *IEEE/ACM Transactions on Networking*, vol. 7, no. 6, pp. 861–874, Dec 1999.

[27] M. Chiang, S. H. Low, A. R. Calderbank, and J. C. Doyle, "Layering as optimization decomposition: A mathematical theory of network architectures," *Proceedings of the IEEE*, vol. 95, no. 1, pp. 255–312, 2007.

[28] D. P. Palomar and M. Chiang, "Alternative distributed algorithms for network utility maximization: Framework and applications," *IEEE Transactions on Automatic Control*, vol. 52, no. 12, pp. 2254–2269, 2007.

[29] P. Samadi, A.-H. Mohsenian-Rad, R. Schober, V. W. Wong, and J. Jatskevich, "Optimal real-time pricing algorithm based on utility maximization for smart grid," in *Proc. of IEEE Smart Grid Communications (SmartGridComm)*, 2010, pp. 415–420.

[30] N. Li, L. Chen, and S. H. Low, "Optimal demand response based on utility maximization in power networks," in *IEEE Power and Energy Society General Meeting*, 2011, pp. 1–8.

[31] L. L. Andrew, S. Barman, K. Ligett, M. Lin, A. Meyerson, A. Roytman, and A. Wierman, "A tale of two metrics: Simultaneous bounds on competitiveness and regret." in *COLT*, 2013, pp. 741–763.

[32] E. Hazan, "Introduction to online convex optimization," *Foundations and Trends in Optimization*, vol. 2, no. 3-4, pp. 157–325, 2016.

[33] M. E. Saks and C. Seshadhri, "Local monotonicity reconstruction," *SIAM Journal on Computing*, vol. 39, no. 7, pp. 2897–2926, 2010.

[34] R. Andersen, C. Borgs, J. Chayes, J. Hopcroft, V. Mirrokni, and S. Teng, "Local computation of pagerank contributions," *Internet Mathematics*, vol. 5(1–2), pp. 23–45, 2008.

[35] J. Katz and L. Trevisan, "On the efficiency of local decoding procedures for error-correcting codes," in *Proc. 32nd Annual ACM Symposium on the Theory of Computing (STOC)*, 2000, pp. 80–86.

[36] N. Alon, R. Rubinfeld, S. Vardi, and N. Xie, "Space-efficient local computation algorithms," in *Proc. 22ndACM-SIAM Symposium on Discrete Algorithms (SODA)*, 2012, pp. 1132–1139.

[37] R. Levi, R. Rubinfeld, and A. Yodpinyanee, "Brief announcement: Local computation algorithms for graphs of non-constant degrees," in *Proc. of the 27th ACM on Symposium on Parallelism in Algorithms and Architectures, (SPAA)*, 2015, pp. 59–61.

[38] O. Reingold and S. Vardi, "New techniques and tighter bounds for local computation algorithms," *Journal of Computer and System Science*, vol. 82, no. 7, pp. 1180–1200, 2016.

[39] U. Feige, B. Patt-Shamir, and S. Vardi, "On the probe complexity of local computation algorithms," 2017, under submission.

[40] Y. Mansour, A. Rubinstein, S. Vardi, and N. Xie, "Converting online algorithms to local computation algorithms," in *Proc. of 39th International Colloquium on Automata, Languages and Programming (ICALP)*, 2012, pp. 653–664.

[41] Y. Mansour, B. Patt-Shamir, and S. Vardi, "Constant-time local computation algorithms," in *Approximation and Online Algorithms - 13th International Workshop, WAOA*, 2015, pp. 110–121.

[42] G. Dantzig, *Linear programming and extensions*. Princeton university press, 2016.

[43] A. Nedic and A. Ozdaglar, "Distributed subgradient methods for multi-agent optimization," *IEEE Transactions on Automatic Control*, vol. 54, no. 1, pp. 48–61, 2009.

[44] N. Z. Shor, *Minimization methods for non-differentiable functions*. Springer - Verlag, 1985.

[45] Y. Yi and M. Chiang, "Stochastic network utility maximization - a tribute to Kelly's paper published in this journal a decade ago," *European Transactions on Telecommunications*, vol. 19, no. 4, pp. 421–442, 2008.

[46] L. Massoulié and J. Roberts, "Bandwidth sharing: objectives and algorithms," in *IEEE INFOCOM'99*, vol. 3, 1999, pp. 1395–1403.

[47] R. Albert, H. Jeong, and A.-L. Barabási, "Internet: Diameter of the world-wide web," *Nature*, vol. 401, no. 6749, pp. 130–131, 1999.

[48] S. Boyd, N. Parikh, E. Chu, B. Peleato, and J. Eckstein, "Distributed optimization and statistical learning via the alternating direction method of multipliers," *Foundations and Trends in Machine Learning*, vol. 3, no. 1, pp. 1–122, 2011.

[49] D. Gabay and B. Mercier, "A dual algorithm for the solution of nonlinear variational problems via finite element approximation," *Computers & Mathematics with Applications*, vol. 2, no. 1, pp. 17–40, 1976.

[50] G. Steidl and T. Teuber, "Removing multiplicative noise by douglas-rachford splitting methods," *Journal of Mathematical Imaging and Vision*, vol. 36, no. 2, pp. 168–184, 2010.

[51] P. A. Forero, A. Cano, and G. B. Giannakis, "Consensus-based distributed support vector machines," *Journal of Machine Learning Research*, vol. 11, pp. 1663–1707, 2010.

[52] P. L. Combettes and V. R. Wajs, "Signal recovery by proximal forward-backward splitting," *Multiscale Modeling & Simulation*, vol. 4, no. 4, pp. 1168–1200, 2005.

[53] P. L. Combettes and J.-C. Pesquet, "A douglas–rachford splitting approach to nonsmooth convex variational signal recovery," *IEEE Journal of Selected Topics in Signal Processing*, vol. 1, no. 4, pp. 564–574, 2007.

[54] I. D. Schizas, A. Ribeiro, and G. B. Giannakis, "Consensus in ad hoc WSNs with noisy linkspart I: Distributed estimation of deterministic signals," *IEEE Trans. on Signal Processing*, vol. 56, no. 1, pp. 350–364, 2008.

[55] "An extended version of this paper can be found at," https://1drv.ms/b/s!AjNk3vqq1VPdaqICDufe6tnSWGs.

[56] N. Buchbinder and J. Naor, "The design of competitive online algorithms via a primal-dual approach," *Foundations and Trends in Theoretical Computer Science*, vol. 3, no. 2-3, pp. 93–263, 2009.

[57] R. Eghbali and M. Fazel, "Designing smoothing functions for improved worst-case competitive ratio in online optimization," in *Neural Information Processing Systems*, 2016, accepted.

[58] Y. Azar, N. Buchbinder, T. H. Chan, S. Chen, I. R. Cohen, A. Gupta, Z. Huang, N. Kang, V. Nagarajan, J. Naor, and D. Panigrahi, "Online algorithms for covering and packing problems with convex objectives," in *IEEE 57th Annual Symposium on Foundations of Computer Science, (FOCS)*, 2016, pp. 148–157.

[59] "The CAIDA UCSD AS Relationship Dataset, September 17, 2007," http://www.caida.org/data/as-relationships/.

Appendix

A. Pseudocode for General Online Fractional Packing

The following pseudocode is replicated from [56]. Constraints arrive in some order. During the ith round, the packing variable x_i and all the covering variables are increased. The minimum x_i is found such that the covering constraints are satisfied.

Instead of increasing x_i continuously, one can perform a binary search over possible values of x_i. For each candidate x_i, a corresponding new value of $y \in \mathbb{R}^n$ is computed and the covering constraints are checked for feasibility. If feasible, y is accepted, and x_i will be increased in the next round of the binary search. If infeasible, y is rejected, and the value of x_i will be decreased in the next round of the search.

Algorithm 2: General Online Fractional Packing

```
Input: $A \in \mathbb{R}^{m\times n}$, $f \in \mathbb{R}^n$
Output: $x$, $y$
Initialize $x = \mathbf{0_m}$, $y = \mathbf{0_n}$
for $i = 1...m$ do
    for $j = 1...n$ do
        $a^j(\max) \leftarrow \max_{k=1}^{i}\{a_{kj}\}$
    while $\sum_{j=1}^{n} a_{ij}y_j < 1$ do
        Increase $x_i$ continuously
        for $j = 1...n$ do
            $\delta = \exp(\frac{B}{2f_i}\sum_{k=1}^{i} a_{kj}x_k) - 1$
            $y_j = \max\Big\{y_j, \frac{1}{na^j(\max)}\delta\Big\}$
```

B. ADMM

In our numerical results we compare LOCO to ADMM in the case of linear NUM. For completeness, we describe the application of ADMM to that setting here.

To apply ADMM, we first absorb the inequality constraint $x \le \bar{x}$ into the inequality $A''x \le c'$ by letting $A'' = \big[A, I\big]^T$ and $c' = \big[c, \bar{x}\big]^T$, where this notation indicates a stack of vectors. We introduce a slack variable $s \ge 0$ such that the inequality constraint becomes $A''x + s = c'$. Let $x' = \big[x, s\big]^T$, $A' = [A'' \quad I]$ and $b = \big[\mathbf{1}_m, \mathbf{0}_m\big]^T$. We can now write the problem in standard ADMM form,

$$\begin{aligned} \min_{x',z} \quad & g(x') + h(z) \\ \text{s.t.} \quad & x' - z = 0 \end{aligned}$$

where $g = (x - \underline{x})_+$ is the indicator function associated with the constraints $\underline{x} \le x$ and $h(z') = -b^T z$ where $\operatorname{dom} h = \{z | A'z = c'\}$.

Writing down the scaled augmented Lagrangian $L_\rho(x', z, u) = g(x') + h(z) + u^T(z - x') + \frac{\rho}{2}\|x' - z\|^2$, we can see that all the update steps have closed form solution (see [48, Chapter 5.2]). The updates become:

$$\begin{aligned} x'^{k+1} &= (z^{k+1} + u^k - \underline{x})_+ \\ z^{k+1} &= \begin{bmatrix} \rho I & A'^T \\ A' & 0 \end{bmatrix}^{-1} \begin{bmatrix} \rho(x'^k - u^k) - b \\ c' \end{bmatrix} \\ u^{k+1} &= u^k + (x'^{k+1} - z^{k+1}) \end{aligned}$$

The solution to the NUM problem is recovered from the first n entries of x'.

C. Proof of Lemma 3

We denote the set $\{0, 1, \ldots, n\}$ by $[n]$. Logarithms are base e. Let $G = (V, E)$ be a graph. For any vertex set $S \subseteq V$, denote by $N(S)$ the set of vertices that are not in S but are neighbors of some vertex in S: $N(S) = \{N(v) : v \in S\} \setminus S$. The *length* of a path is the number of edges it contains. For a set $S \subseteq V$ and a function $f : V \to \mathbb{N}$, we use $S \cap f^{-1}(i)$ to denote the set $\{v \in S : f(v) = i\}$.

Let $G = (V, E)$ be a graph, and let $f : V \to \mathbb{N}$ be some function on the vertices. An *adaptive vertex exposure procedure* A is one that does not know f a priori. A is given a vertex $v \in V$ and $f(v)$; A iteratively adds vertices from $V \setminus S$ to S: for every vertex u that A adds to S, $f(u)$ is revealed immediately after u is added. Let S^t denote S after the addition of the t^{th} vertex. The following is a simple concentration bound whose proof is given for completeness.

Lemma 6. *Let $G = (V, E)$ be a graph, let $Q > 0$ be some constant, let $\gamma = 15Q$, and let $f : V \to [Q]$ be a function chosen uniformly at random from all such possible functions. Let A be an adaptive vertex exposure procedure that is given a vertex $v \in V$. Then, for any $q \in [Q]$, the probability that there is some t, $\gamma \log m \le t \le m$ for which $|S^t \cap f^{-1}(q)| > \frac{2|S^t|}{Q}$ is at most $\frac{1}{m^4}$.*

Proof. Let v_j be the j^{th} vertex added to S by A, and let X_j be the indicator variable whose value is 1 iff $f(v_j) = q$. For any $t \le m$, $\mathbb{E}\left[\sum_{j=1}^{t} X_j\right] = \frac{t}{Q}$. As X_i and X_j are independent for all $i \ne j$, by the Chernoff bound, for $\gamma \log m \le t \le m$,

$$\Pr\left[\sum_{j=1}^{t} X_j > \frac{2t}{Q}\right] \le e^{\frac{-t}{3Q}} \le e^{-5\log m}.$$

A union bound over all possible values of $t : \gamma \log m \le t \le m$ completes the proof. $\square$

Let $r : V \to [0, 1]$ be a function chosen uniformly at random from all such possible functions. Partition $[0, 1]$ into $Q = 4(d + 1)$ segments of equal measure, $I_1, \ldots, I_Q$. For every $v \in V$, set $f(v) = q$ if $r(v) \in I_q$ (f is a quantization of r).

Consider the following method of generating two sets of vertices: T and R, where $T \subseteq R$. For some vertex v, set $T = R = \{v\}$. Continue inductively: choose some vertex $w \in T$, add all $N(w)$ to R and compute $f(u)$ for all $u \in N(w)$. Add the vertices u such that $u \in N(w)$ and $f(u) \ge f(w)$ to T. The process ends when no more vertices can be added to T. T is the query set with respect to f, hence $|T|$ is an upper bound on the size of the actual query set (i.e., the query set with respect to r). However, it is difficult to reason about the size of T directly, as the ranks of its vertices are not independent. The ranks of the vertices in R, though, *are* independent, as R is generated by an adaptive vertex exposure procedure. R is a superset of T that includes T and its boundary, hence $|R|$ is also an upper bound on the size of the query set.

We now define $Q + 1$ "layers" - $T_{\le 0}, \ldots, T_{\le Q}$: $T_{\le q} = T \cap \bigcup_{i=0}^{q} f^{-1}(i)$. That is, $T_{\le q}$ is the set of vertices in T whose rank is at most q. (The range of f is $[Q]$, hence $T_{\le 0}$ will be empty, but we include it to simplify the proof.)

Claim 7. *Set $Q = 4(d+1)$, $\gamma = 15Q$. Assume without loss of generality that $f(v) = 0$. Then for all $0 \le i \le Q-1$,*

$$\Pr[|T_{\le i}| \le 2^i \gamma \log m \wedge |T_{\le i+1}| \ge 2^{i+1}\gamma \log m] \le \frac{1}{m^4}.$$

Proof. For all $0 \le i \le Q$, let $R_{\le i} = T_{\le i} \cup N(T_{\le i})$. Note that

$$R_{\le i} \cap f^{-1}(i) = T_{\le i} \cap f^{-1}(i), \tag{1}$$

because if there had been some $u \in N(T_{\le i}), f(u) = i$, u would have been added to $T_{\le i}$.

Note that $|T_{\le i}| \le 2^i \gamma \log m \wedge |T_{\le i+1}| \ge 2^{i+1}\gamma \log m$ implies that

$$|T_{\le i+1} \cap f^{-1}(i+1)| > \frac{|T_{\le i+1}|}{2}. \tag{2}$$

In other words, the majority of vertices $v \in T_{\le i+1}$ must have $f(v) = i+1$.

Given $|T_{\le i+1}| > 2^{i+1}\gamma \log m$, it holds that $|R_{\le i+1}| > 2^{i+1}\gamma \log m$ because $T_{\le i+1} \subseteq R_{\le i+1}$. Furthermore, $R_{\le i+1}$ was constructed by an adaptive vertex exposure procedure and so the conditions of Lemma **??** hold for $R_{\le i+1}$. From Equations (**??**) and (**??**) we get

$$\begin{aligned}
&\Pr[|T_{\le i}| \le 2^i \gamma \log m \wedge |T_{\le i+1}| \ge 2^{i+1}\gamma \log m] \\
&\le \Pr\left[\left|R_{\le i+1} \cap f^{-1}(i+1)\right| > \frac{|T_{\le i+1}|}{2}\right] \\
&\le \Pr\left[\left|R_{\le i+1} \cap f^{-1}(i+1)\right| > \frac{2\,|R_{\le i+1}|}{Q}\right] \\
&\le \frac{1}{m^4},
\end{aligned}$$

where the second inequality is because $|R_{\le i+1}| \le (d+1)|T_{\le i+1}|$, as G's degree is at most d; the last inequality is due to Lemma **??**. □

Lemma 8. *Set $Q = 4(d+1)$. Let $G = (V, E)$ be a graph with degree bounded by d, where $|V| = m$. For any vertex $v \in G$, $\Pr\left[T_v > 2^Q \cdot 15Q \log m\right] < \frac{1}{m^3}$.*

Proof. To prove Lemma **??**, we need to show that, for $\gamma = 15Q$,

$$\Pr[|T_{\le Q}| > 2^L \gamma \log m] < \frac{1}{m^3}.$$

We show that for $0 \le i \le Q, \Pr[|T_{\le i}| > 2^i \gamma \log m] < \frac{i}{m^4}$, by induction. For the base of the induction, $|S_0| = 1$, and the claim holds. For the inductive step, assume that $\Pr[|T_{\le i}| > 2^i \gamma \log m] < \frac{i}{m^4}$. Then, denoting by X the event $|T_{\le i}| > 2^i \gamma \log m$ and by $\bar{X}$ the event $|T_{\le i}| \le 2^i \gamma \log m$,

$$\begin{aligned}
&\Pr[|T_{\le i+1}| > 2^{i+1}\gamma \log m] \\
&= \Pr[|T_{\le i+1}| > 2^{i+1}\gamma \log m : X] \Pr[X] \\
&+ \Pr[|T_{\le i+1}| > 2^{i+1}\gamma \log m : \bar{X}] \Pr[\bar{X}].
\end{aligned}$$

From the inductive step and Claim **??**, using the union bound, the lemma follows. □

Applying a union bound over all the vertices gives the size of *each* query set is $O(\log m)$ with probability at least $1 - 1/m^2$, completing the proof of Lemma 3.