Distributed Optimization via Local Computation Algorithms

Palma London Niangjun Chen Shai Vardi Adam Wierman

***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 provides a number of important benefits, such as robustness to link failure and adaptivity to dynamic settings. Specifically, we develop an algorithm, LOCO, that given a convex optimization problem P with n variables and a "sparse" linear constraint matrix with m constraints, provably finds a solution as good as that of the best online algorithm for P using only $O(\log(n+m))$ messages with high probability. The approach is not iterative and communication is restricted to a localized neighborhood. In addition to analytic results, we show numerically that the performance improvements over classical approaches for distributed optimization are significant, e.g., it uses orders of magnitude less communication than ADMM.**

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.

Distributed optimization is an area of crucial importance to networked control. 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 [10], [41], communication network protocol design [28], [35], [51], trajectory optimization [25], [29], formation control of vehicles [44], [53], sensor networks [33], [40], control of power systems [19], [43], and management of electric vehicles and distributed storage devices [13], [24].

Distributed optimization is a field with a long history. Beginning in the 1960s approaches emerged for solving large scale linear programs via decomposition into pieces that could be solved in a distributed manner. For example, two early approaches are Bender's decomposition [7] and the Dantzig-Wolfe decomposition [17], [18], which can both be generalized to nonlinear objectives via the subgradient method [8], [38], [50].

Today, there is a wide variety of approaches for distributed optimization, e.g., primal decomposition [8], [30] and dual decomposition [20], [34], [38], [51]. See [42] for a survey. Broadly, these approaches tend to fall into two categories. The first category uses dual decomposition and subgradient methods [28], [34], [51]; the second involves consensus-based schemes which enable decentralized information aggregation, which forms the basis for many first order and second order distributed optimization algorithms [9], [39].

While the algorithms described above 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] (see Subsection I-B for a more comprehensive definition and example). Clearly, neither iterative descent methods nor consensus methods are local: answering a query about a piece of the solution requires global communication.

Local computation provides many important benefits for distributed optimization. For example, any failure in the system only has local effects: if a node in a distributed system goes offline while an iterative distributed algorithm is executing, the whole process is brought to a halt (or at least the system needs to be carefully designed to be able to accommodate such failures); if the computations are all local, the failure will only affect a small number of nodes in the neighborhood of the failure. Similarly, lag in a single edge affects the computation of the entire solution in the iterative setting, while most computations are not be affected at all when the computations are local. Another advantage of local computation is that it allows the system to be more dynamic: an arrival of another node 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.

Despite the benefits of local algorithms for distributed optimization, the problem of designing a local, distributed optimization algorithm is open.

A. Contributions of this paper

This paper introduces an algorithm, LOCO, (LOcal Convex Optimization) that is both *distributed* and *local*. It is not an iterative method and uses far less communication to compute small parts of the solution than iterative descent and consensus methods, e.g., ADMM and dual decomposition, while matching the total communication if the whole solution is queried.

While the technique we propose is general, in this work, we focus on a canonical optimization problem: network utility maximization. Due to space restrictions, we only consider the variant of maximizing 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 [6], [26].

An extended version of the paper can be found at [1]. All authors are with the Department of Computing and Mathematical Sciences, California Institute of Technology. Email:{plondon, ncchen, svardi, adamw}@caltech.edu

[1] 'Local' is an overloaded term in the literature. We mean local in the sense of [46].

In Section III, we provide worst-case guarantees on the performance of LOCO with respect to the relative error and the number of messages it requires. In Section IV, we compare the performance of LOCO with ADMM, and show that LOCO uses orders of magnitude less communication than ADMM if only part of the solution is required, and the same order of magnitude if the entire solution is required. Furthermore, in terms of both the amount of communication required and the relative error, LOCO vastly outperforms its theoretical guarantees.

The key idea behind LOCO is an extension of recent results from the emerging field of *local computation algorithms* (LCA) in theoretical computer science (e.g., [31], [36], [45]). In particular, a key insight of the field is that online algorithms can be converted into local algorithms in graph problems with bounded degree [36]. However, much of the focus of local algorithms has, to this point, been on graph problems (see related literature below). The technical contribution of this work is the extension of these ideas to convex programs.

B. Related literature

This work, for the first time, brings techniques from the field of local computation algorithms into the domain of networked control. The LCA model was formally introduced by Rubinfeld et al. [46], after many algorithms fitting within the framework had recently appeared in distinct areas, e.g., [5], [27], [47]. 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.

The main idea of 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 a global solution, by exchanging information across distributed agents. More concretely, an LCA receives a *query* and is expected to output the part of the solution associated with that 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.

Most of the work on LCAs has focused on graph problems such as matching, maximal independent set, and coloring (e.g., [4], [21], [31], [45]) and the efficiency criteria were the number of probes to the graph, the running time and the amount of memory required. This paper extends the LCA literature by moving from graph problems to optimization problems, which have not been studied in the LCA community previously. Mansour et al. [36] showed a general reduction from LCAs to online algorithms on graphs with bounded degree. The key technical contribution of our the work is extending that technique to design LCAs for convex programs. In contrast to previous work whose primary focus was probe, time and space complexities, the efficiency criterion we use is the number of messages required as this is usually the expensive resource in networked control.

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 [28], [34], [35], [51] to understanding of protocol layering as optimization decomposition [14], [42] and power system demand response [32], [48]. For a recent survey, see [54].

A. Model

The NUM framework considers a network containing a set of links $\mathcal{L} = \{1, \dots, m\}$ of capacity c_j, for $j \in \mathcal{L}$. A set of $\mathcal{N} = \{1, \dots, n\}$ sources shares the network; source $i \in \mathcal{N}$ is characterized by $(L_i, f_i, \underline{x}_i, \bar{x}_i)$: a path $L_i \subseteq \mathcal{L}$ in the network; a (usually) concave utility function $f_i : \mathbb{R}_+ \to \mathbb{R}$; and the minimum and maximum transmission rates of i.

The goal in NUM is to maximize the sources' aggregate utility. Source i attains a concave utility $f_i(x_i)$ when it transmits at rate x_i that satisfies $\underline{x}_i \le x_i \le \bar{x}_i$; the optimization of aggregate utility can be formulated as follows,

$$\begin{aligned} \max_{x} \quad & \sum_{i=1}^{n} f_i(x_i) \\ \text{subject to} \quad & Ax \le c \\ & \underline{x} \le x \le \bar{x}, \end{aligned}$$

where $A \in \mathbb{R}_+^{m \times n}$ is defined as $A_{ji} = \begin{cases} 1, & j \in L(i) \\ 0, & \text{otherwise} \end{cases}$.

The NUM framework is general in that the choice of f_i allows for the representation of different goals of the network operator. For example, using $f_i(x_i) = x_i$, maximizes throughput; setting $f_i(x_i) = \log(x_i)$ achieves proportional fairness among the sources; setting $f_i(x_i) = -1/x_i$ minimizes potential delay; these are common goals in communication network applications [34], [37].

In this paper we focus on the throughput maximization case, i.e., $f_i(x_i) = x_i$; in this case NUM is an LP. Note that the classical dual decomposition approach does not work for throughput maximization since it requires the objective function to be strictly concave. However, ADMM can be applied.

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 n, 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 [11], [51].

There is a large literature across the networked control and communication networks literatures that seeks to design such distributed optimization algorithms, e.g., [14], [28], [35]. Dual decomposition algorithms are particularly prominent for use in this setting. However, many such methods cannot be applied to the case of throughput maximization, i.e., linear f_i. One extremely prominent algorithm that does apply in the case of throughput maximization is Alternating Method of Multipliers (ADMM), which was introduced by [23] and has found broad applications in e.g., denoising images [52], support vector machine [22], and signal processing [15], [16], [49]. 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 [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 $1 - \frac{1}{n^\alpha}$, where where n is the number of vertices in the network and $\alpha > 0$ can be an arbitrarily large constant. We denote this by $1 - \frac{1}{\text{poly}\, n}$. 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 $O(\log n)$.

The second is the *approximation ratio*, which measures the quality of the solution provided by the algorithm. Specifically, an algorithm is said to α-approximate a maximization problem if its solution is guaranteed to be at least $\frac{OPT}{\alpha}$, 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.

[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 [3], β is small compared to m.

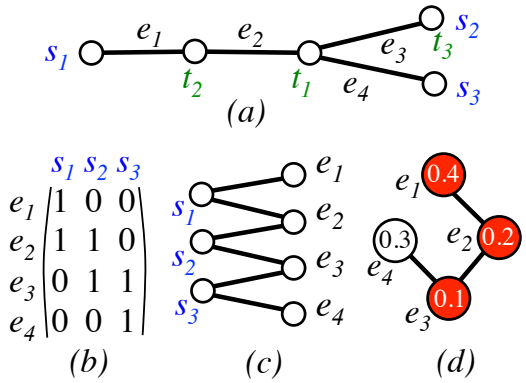


Fig. 1: An illustration of LOCO on a toy graph with five nodes and four edges, $e_1, \ldots, e_4$. There are three sources, s_1, s_2, s_3, with paths ending in destinations t_1, t_2, t_3 respectively. The graph is depicted in (a); the constraint matrix for NUM is given in (b); the bipartite graph representation of the matrix in (c); and the dependency graph in (d). The rank of each constraint (edge) is written in the node representing the constraint in the dependency graph. The shaded nodes represent the query set for source s_1.

III. Local Convex Optimization

In this section, we introduce a 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 or iteration. This is in contrast to iterative descent methods, e.g. ADMM, which are *global*, i.e., they spread the information necessary to find an optimal solution throughout the whole network over a series of rounds. LOCO has provable worst-case guarantees on both its approximation ratio and message complexity, and improves on the communication overhead of iterative descent methods by orders of magnitude in practice when asked to compute a piece of the optimal solution.

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.

[3]Strictly speaking, 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 [45].

More formally, to reduce a static optimization problem to an online optimization problem, we do the following. Let Y be the set of constraints of an optimization problem P. Let $r : Y \to [0, 1]$ be a ranking function that assigns each constraint y_j a real number between 0 and 1, uniformly at random. We call $r(y_j)$ y_j's *rank*. Suppose that there is some online algorithm ALG that receives the constraints sequentially and must augment the variables immediately and irrevocably so as to satisfy each arriving constraint. Suppose furthermore that for each constraint y_j, we can pinpoint a small set of constraints $S(y_j)$ (which we call y_j's *query set*) that arrived before it so that restricting the set of constraints of P to $S(y_j)$ results in ALG producing (exactly) the same solution for the variables that are present in y_j. Then simulating ALG only on $S(y_j)$ would suffice to obtain the solution for the variables in y_j. This is precisely what our algorithm does: it generates a random order of arrival for the constraints, and for each constraint y_j, it constructs such a set $S(y_j)$ and simulates the online algorithm on it. An arbitrary ordering could mean that these dependency sets are very large for some constraints; to bound the size of these sets, we require that (i) the constraint matrix of P is sparse and (ii) the order generated is random.[4]

Concretely, there are two main steps in LOCO. In the first, LOCO generates a localized neighborhood for each vertex. In the second, LOCO simulates an online algorithm on the localized neighborhood. Importantly, the first step is independent of the precise nature 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.

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 row of A is represented by a vertex $v_\ell \in L$; each column by a vertex $v_r \in R$. The edge (v_ℓ, v_r) is in E' if and only if $A_{\ell,r} = 1$. A more intuitive way to interpret G is the following: L represents the variables, R the constraints. 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 R; 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 constraints: changing the value of any variable immediately affects all constraints in which it appears, hence these constraints can be thought of as directly dependent. 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 the query set, we generate a random *ranking function* on the vertices of H, $r : V \to [0, 1]$. Given the dependency graph H, an initial node $y \in V$ and the ranking function r, we build the *query set* of y, denoted $S(y)$, using a variation of BFS, as follows. Initialize $S(y)$ to contain y. For every vertex $v \in S(y)$, scan all of v's neighbors, denoted $N(v)$. For each $u \in N(v)$, if $r(u) \leq r(v)$, add u to $S(y)$. Continue iteratively until no more vertices can be added to $S(y)$ (that is, for every vertex $v \in S(y)$ all of its neighbors that are not themselves in $S(y)$ have lower rank than v). If there are ties (i.e., two neighbors u, v such that $r(u) = r(v)$), we tie-break by ID.[5]

Step 2, Simulating the online algorithm: Assume that we have an online algorithm for the problem that we would like LOCO to solve (in this paper we use the online packing Algorithm of Buchbinder and Naor [12, chapter 14]. We provide the pseudocode in the extended version of the paper [1], for completeness). The specific setting that the online algorithm must apply to is the following: the variables of the convex program are known in advance, as are the univariate constraints. The (rest of the) constraints arrive one at a time; the online algorithm is expected to satisfy each constraint as it arrives, by increasing the value of some of the variables. It is never allowed to decrease the value of any variable. We simulate the online algorithm as follows:

In order to compute its own value in the solution, source i applies r to the set of constraints in which it is contained, $Y(i)$. For $y = \arg\max_{z \in Y(i)}\{r(z)\}$, it simulates the online algorithm on $S(y)$. That is, it executes the online algorithm on the neighborhood constructed in Step 1 for the "last arriving" constraint that contains i. i's value is the value of i at the end of the simulation. Claim 4 below shows that i's value is identical to its value if the online algorithm was executed on the entire program, with the constraints arriving in the order defined by r.

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 theorem (and its proof) apply 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 n*

[4]Pseudo-random orders suffice, see [45].

[5]Any consistent tie breaking rule suffices.

variables and m constraints, whose constraint matrix has sparsity σ. Given an online algorithm[6] 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 n variables, m constraints, and a sparse constraint matrix. LOCO computes an $O(\log m)$ – approximation to the optimal solution of P using $O(\log(n+m))$ messages with probability $1 - 1/\operatorname{poly}(n,m)$.*

The approximation ratio in Theorem 2 comes from the online algorithm presented and analyzed in [12] (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 $\lambda = 4(d+1)$,*

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

The proof of Lemma 3 uses ideas from a proof in [45], and employs a *quantization* of the rank function. Its proof is deferred to the extended version of the paper [1]. 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 i in the output of LOCO is identical to its value in the output of the online algorithm.*

Proof. Let constraint y_j be the last constraint containing i that arrives in the order defined by r; its arrival is the last time i will be updated. Therefore it is sufficient to only consider constraints arriving before y_j. Further, by design, $S(y_j)$ is the set of constraints at whose arrival there is possibly some change that may affect the value of i. □

The following lemma is a restatement of Theorem 14.1 in [12], adapted to throughput maximization. See [1] for the pseudocode of the algorithm.

[6]See footnote [3].

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

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

C. Contrasting LOCO and ADMM

LOCO fundamentally differs from iterative descent and consensus style approaches to distributed optimization. While iterative descent and consensus style approaches are inherently iterative, LOCO is not. Under LOCO, a node can compute its value in one shot, once it gets information about its query set.

Additionally, while iterative descent and consensus style approaches are *global*, LOCO is local. Under LOCO, communication stays within the query set and so the computation only needs to be updated if changes happen within the query set. This means that LOCO is robust to churn, failures, and communication problems outside of that set of nodes.

Another important difference is that LOCO does not compute the optimal solution, while iterative descent and consensus style approaches will eventually converge to the true optimal. 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 in part because the ranking is done randomly rather than in an adversarial fashion (we elaborate on this in Section IV).

Finally, note that there is an important difference in the form of the theoretical guarantees for LOCO and iterative descent and consensus style algorithms. LOCO has guarantees in terms of the approximation ratio, while iterative descent and consensus style algorithms have convergence rate guarantees. For example, ADMM has guarantees on convergence of the norms of the primal and dual residuals [11, Chapter 3.3].

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 [6], [26].

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 $A \in \mathbb{R}^{(m \times n)}$ as follows. Set $\tilde{A}_{j,i} = 1$ with probability p and $\tilde{A}_{j,i} = 0$ otherwise.

Let $A = \tilde{A} + I_n$ to ensure each row of A has at least one non zero entry.[7] 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.[8]

For our second set of experiments, we use the real network from the graph of Autonomous System (AS) relationships in [2]. 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 node 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 [11]. Unless otherwise specified, we set the relative and absolute tolerances to be $\epsilon^{rel} = 10^{-4}$ and $\epsilon^{abs} = 10^{-2}$, 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 [11]. 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 $B = 2\ln(1 + m)$ 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.

[7]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 [45].

[8]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 [45].

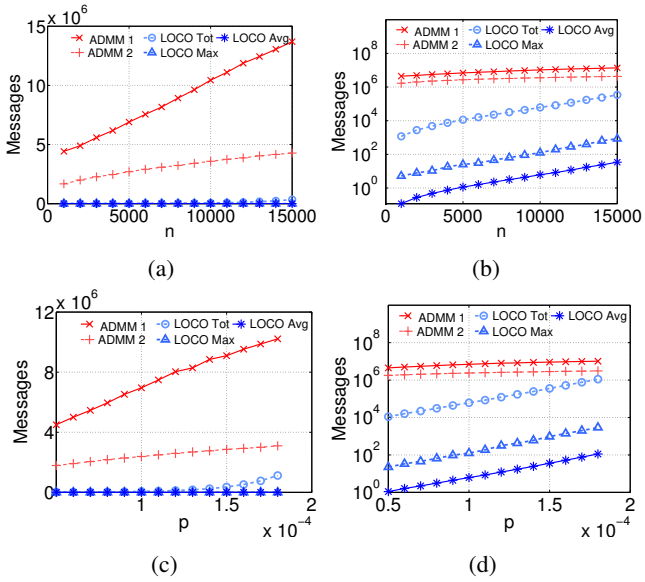


Fig. 2: Illustration of the number of messages required by ADMM and LOCO for the synthetic data set with results averaged over 50 trials. Plots (a) and (b) vary n while fixing sparsity $p = 10^{-4}$, showing the results in linear-scale and log-scale respectively. Plots (c) and (d) fix $n = 10^3$ and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

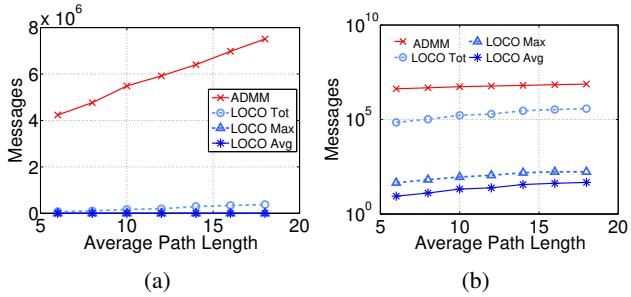
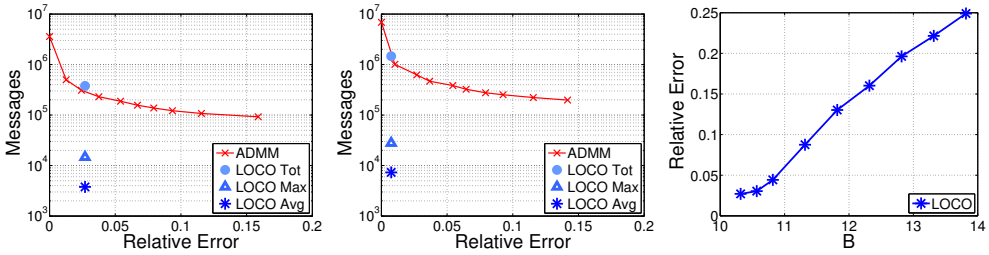


Fig. 3: Illustration of the number of messages required by ADMM and LOCO for the real network data with $n = 8020$ and various average path lengths $L(i)$.

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 [11, 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



(a) (b) (c)

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 $B = 2\ln(1+m)$.

in order to construct the query set, see e.g., [45] 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 by LOCO are small compared to the 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.

It may seem somewhat unfair to compare the message complexity of LOCO and ADMM when they have differing 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. Figures 4 (a) and (b) illustrate the Pareto optimal frontier for ADMM: the minimal messages needed in order to obtain a particular relative error. Unlike ADMM, LOCO cannot trade off the number of messages used with the relative error, thus LOCO corresponds to a single point on 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 investigate 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] An extended version of this paper can be found at:. `http://users.cms.caltech.edu/~plondon/loco.pdf`.

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

[3] R. Albert, H. Jeong, and A.-L. Barabási. Internet: Diameter of the world-wide web. *Nature*, 401(6749):130–131, 1999.

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

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

[6] 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*, pages 741–763, 2013.

[7] J. F. Benders. Partitioning procedures for solving mixed-variables programming problems. *Numerische mathematik*, 4(1):238–252, 1962.

[8] D. P. Bertsekas. *Nonlinear programming*. Athena scientific Belmont, 1999.

[9] V. D. Blondel, J. M. Hendrickx, A. Olshevsky, and J. N. Tsitsiklis. Convergence in multiagent coordination, consensus, and flocking. In *Proceedings of IEEE Conference on Decision and Control*, pages 2996–3000. IEEE, 2005.

[10] S. Borst, V. Gupta, and A. Walid. Distributed caching algorithms for content distribution networks. In *Proceedings of IEEE INFOCOM*, pages 1–9. IEEE, 2010.

[11] 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*, 3(1):1–122, 2011.

[12] N. Buchbinder and J. Naor. The design of competitive online algorithms via a primal-dual approach. *Foundations and Trends in Theoretical Computer Science*, 3(2-3):93–263, 2009.

[13] 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*, 3(1):388–393, 2012.

[14] 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*, 95(1):255–312, 2007.

[15] 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*, 1(4):564–574, 2007.

[16] P. L. Combettes and V. R. Wajs. Signal recovery by proximal forward-backward splitting. *Multiscale Modeling & Simulation*, 4(4):1168–1200, 2005.

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

[18] G. B. Dantzig and P. Wolfe. Decomposition principle for linear programs. *Operations research*, 8(1):101–111, 1960.

[19] T. Erseghe. Distributed optimal power flow using admm. *IEEE transactions on power systems*, 29(5):2370–2380, 2014.

[20] H. Everett III. Generalized lagrange multiplier method for solving problems of optimum allocation of resources. *Operations research*, 11(3):399–417, 1963.

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

[22] P. A. Forero, A. Cano, and G. B. Giannakis. Consensus-based distributed support vector machines. *Journal of Machine Learning Research*, 11(May):1663–1707, 2010.

[23] D. Gabay and B. Mercier. A dual algorithm for the solution of nonlinear variational problems via finite element approximation. *Computers & Mathematics with Applications*, 2(1):17–40, 1976.

[24] L. Gan, U. Topcu, and S. H. Low. Optimal decentralized protocol for electric vehicle charging. *IEEE Transactions on Power Systems*, 28(2):940–951, May 2013.

[25] 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*, volume 3, pages 2612–2619. IEEE, 2002.

[26] E. Hazan. Introduction to online convex optimization. *Foundations and Trends® in Optimization*, 2(3-4):157–325, 2016.

[27] 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)*, pages 80–86, 2000.

[28] 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*, 49(3):237–252, 1998.

[29] Y. Kuwata and J. P. How. Cooperative distributed robust trajectory optimization using receding horizon milp. *IEEE Transactions on Control Systems Technology*, 19(2):423–431, 2011.

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

[31] R. Levi, R. Rubinfeld, and A. Yodpinyanee. Brief announcement: Local computation algorithms for graphs of non-constant degrees. In *Proceedings of the 27th ACM on Symposium on Parallelism in Algorithms and Architectures, SPAA*, pages 59–61, 2015.

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

[33] Y. Liao, H. Qi, and W. Li. Load-balanced clustering algorithm with distributed self-organization for wireless sensor networks. *IEEE Sensors Journal*, 13(5):1498–1506, 2013.

[34] S. H. Low and D. E. Lapsley. Optimization flow control. I. Basic algorithm and convergence. *IEEE/ACM Transactions on Networking*, 7(6):861–874, Dec 1999.

[35] S. H. Low, F. Paganini, and J. C. Doyle. Internet congestion control. *IEEE control systems*, 22(1):28–43, 2002.

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

[37] L. Massoulié and J. Roberts. Bandwidth sharing: objectives and algorithms. In *INFOCOM'99. Eighteenth Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings. IEEE*, volume 3, pages 1395–1403. IEEE, 1999.

[38] A. Nedic and A. Ozdaglar. Distributed subgradient methods for multi-agent optimization. *IEEE Transactions on Automatic Control*, 54(1):48–61, 2009.

[39] A. Nedić and A. Ozdaglar. Convergence rate for consensus with delays. *Journal of Global Optimization*, 47(3):437–456, 2010.

[40] R. Olfati-Saber. Distributed Kalman filtering for sensor networks. In *Decision and Control, 2007 46th IEEE Conference on*, pages 5492–5498. IEEE, 2007.

[41] 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*, pages 177–186. ACM, 2002.

[42] D. P. Palomar and M. Chiang. Alternative distributed algorithms for network utility maximization: Framework and applications. *IEEE Transactions on Automatic Control*, 52(12):2254–2269, 2007.

[43] 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.

[44] R. L. Raffard, C. J. Tomlin, and S. P. Boyd. Distributed optimization for cooperative agents: Application to formation flight. In *Decision and Control, 2004. CDC. 43rd IEEE Conference on*, volume 3, pages 2453–2459. IEEE, 2004.

[45] O. Reingold and S. Vardi. New techniques and tighter bounds for local computation algorithms. *Journal of Computer and System Science*, 82(7):1180–1200, 2016.

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

[47] M. E. Saks and C. Seshadhri. Local monotonicity reconstruction. *SIAM Journal on Computing*, 39(7):2897–2926, 2010.

[48] 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 *Proceedings of Smart Grid Communications (SmartGridComm)*, pages 415–420. IEEE, 2010.

[49] 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*, 56(1):350–364, 2008.

[50] N. Z. Shor. *Minimization methods for non-differentiable functions*, volume 3. Springer Science & Business Media, 2012.

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

[52] G. Steidl and T. Teuber. Removing multiplicative noise by douglas-rachford splitting methods. *Journal of Mathematical Imaging and Vision*, 36(2):168–184, 2010.

[53] I. Suzuki and M. Yamashita. Distributed anonymous mobile robots: Formation of geometric patterns. *SIAM Journal on Computing*, 28(4):1347–1363, 1999.

[54] 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*, 19(4):421–442, 2008.

APPENDIX

A. Pseudocode for General Online Fractional Packing

The following pseudocode is replicated from [12]. Constraints arrive in some order. During the jth round, the dual variable $y(j)$ and all the primal variables are increased. The minimum $y(j)$ is found, such that the primal constraints are satisfied.

Algorithm 1: General Online Fractional Packing

Input: $A \in \mathbb{R}^{m\times n}$, $c \in \mathbb{R}^n$
Output: x, y
Initialize $x = \mathbf{0_n}$, $y = \mathbf{0_m}$

for $j = 1...m$ **do**
  for $i = 1...n$ **do**
    $a_i(\max) \leftarrow \max_{k=1}^{j}\{a(i,k)\}$
  while $\sum_{i=1}^{n} a(i,j)x(i) < 1$ **do**
    Increase $y(j)$ continuously
    for $i = 1...n$ **do**
      $\delta = \exp(\frac{B}{2c(i)}\sum_{k=1}^{j} a(i,k)y(k)) - 1$
      $x(i) = \max\left\{x(i), \frac{1}{na_i(\max)}\delta\right\}$

Instead of increasing $y(j)$ continuously, one can perform a binary search over possible values of $y(j)$. For each candidate $y(j)$, a corresponding new value of x is computed and the primal constraints are checked for feasibility. If feasible, the new x is accepted, and $y(j)$ will be increased in the next round of the binary search. If infeasible, the new x is rejected, and the value of $y(j)$ will be decreased in the next round of the search.

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 $A'' = \left[A, I\right]^T$ and $c' = \left[c, \bar{x}\right]^T$, 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 $x' = \left[x, s\right]^T$, $A' = [A'' \quad I]$ and $b = \left[\mathbf{1}_n, \mathbf{0}_n\right]^T$. 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 $\mathrm{dom}\ h = \{z | A'z = c'\}$.

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 [11, Chapter 5.2]). The updates become:

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

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

C. Proof of Lemma 3

We denote the set $\{0, 1, \ldots, m\}$ by $[m]$. 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, $\gamma \log n \le t \le n$ for which $|S^t \cap f^{-1}(q)| > \frac{2|S^t|}{Q}$ is at most $\frac{1}{n^4}$.*

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 $t \le n$, $\mathbb{E}\left[\sum_{j=1}^{t} X_j\right] = \frac{t}{Q}$. As X_i and X_j are independent for all $i \ne j$, by the Chernoff bound, for $\gamma \log n \le t \le n$,

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

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

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" - $T_{\leq 0}, \ldots, T_{\leq Q}$: $T_{\leq q} = T \cap \bigcup_{i=0}^{q} f^{-1}(i)$. That is, $T_{\leq q}$ is the set of vertices in T whose rank is at most q. (The range of f is $[Q]$, hence $T_{\leq 0}$ 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 $0 \leq i \leq Q-1$,*

$$\Pr[|T_{\leq i}| \leq 2^i \gamma \log n \wedge |T_{\leq i+1}| \geq 2^{i+1}\gamma \log n] \leq \frac{1}{n^4}.$$

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

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

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

Note that $|T_{\leq i}| \leq 2^i \gamma \log n \wedge |T_{\leq i+1}| \geq 2^{i+1}\gamma \log n$ implies that

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

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

Given $|T_{\leq i+1}| > 2^{i+1}\gamma \log n$, it holds that $|R_{\leq i+1}| > 2^{i+1}\gamma \log n$ because $T_{\leq i+1} \subseteq R_{\leq i+1}$. Furthermore, $R_{\leq i+1}$ was constructed by an adaptive vertex exposure procedure and so the conditions of Lemma 6 hold for $R_{\leq i+1}$. From Equations (1) and (2) we get

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

where the second inequality is because $|R_{\leq i+1}| \leq (d+1)|T_{\leq i+1}|$, as G's degree is at most d; the last inequality is due to Lemma 6. □

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

Proof. To prove Lemma 8, we need to show that, for $\gamma = 15Q$,

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

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

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

From the inductive step and Claim 7, using the union bound, the lemma follows. □

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