# 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 benefits, such as robustness to link failure and adaptivity in 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.

## 1. INTRODUCTION

The goal of this work is to introduce a new, fundamentally different approach to distributed optimization based on an emerging area of theoretical computer science – *local computation algorithms* (LCAs) [5, 8, 10, 11].

There are a wide variety of approaches for distributed optimization, which fall into the categories of dual decomposition and subgradient methods, and consensus-based schemes. 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. Neither iterative descent methods nor consensus methods are local because answering a query about a piece of the solution requires global communication.

This work 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 the solution than iterative descent and consensus methods. While the technique we propose is general, in this work we focus on a canonical optimization problem: network utility maximization. We provide worst-case guarantees on the performance of LOCO with respect to the relative error and the number of messages it requires.

The key idea behind LOCO is an extension of recent results on LCAs; this work, for the first time, imports techniques from the field of local computation algorithms into the domain of networked control. In particular, a key insight is that online algorithms can be converted into local algorithms in graph problems with bounded degree [8]. The technical contribution of this work is the extension of these ideas to convex programs.

Local computation methods are well suited 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; 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 affected 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.

## 2. NETWORK UTILITY MAXIMIZATION

To illustrate the application of local computation algorithms to distributed optimization, we focus on the classic setting of network utility maximization (NUM). Consider 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 share 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 concave utility function $f_i : \mathbb{R}_+ \to \mathbb{R}$; and the minimum and maximum transmission rates of $i$. The optimization of aggregate utility can be formulated as follows,

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

where $A \in \mathbb{R}_+^{m \times n}$ is defined as $A_{ji} = 1$ if $j \in L_i$ and 0 otherwise. 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 NUM framework is general in that the choice of $f_i$ allows for the representation of different goals of the net-

*This is an extended abstract of [6]. All authors are with the Department of Computing and Mathematical Sciences, California Institute of Technology. Email:{plondon, ncchen, svardi, adamw}@caltech.edu

work 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 [7, 9].

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 [1,12]. There is a large literature across the networked control and communication networks literatures that seeks to design such distributed optimization algorithms, e.g., [3]. We compare LOCO to the Alternating Method of Multipliers (ADMM) [1, 4].

In this work we focus on the throughput maximization case, i.e., $f_i(x_i) = x_i$; in this case NUM is an LP. The classical dual decomposition approach does not work for throughput maximization since it requires the objective function to be strictly concave [7]. 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 $\alpha$ and $\beta$ 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 $\alpha$ is the maximum number of sources sharing a link, which is typically small compared to $n$, and $\beta$ is the maximum number of links each source uses, which is typically small compared to $m$.

# 3. 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.

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. For 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 $\alpha$-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. 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.

## 3.1 An overview of LOCO

The key insight in the design and analysis of LOCO is that any natural 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 $Y$ 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 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 the algorithm producing exactly the same solution for the variables that are present in $y_j$. Then simulating the algorithm on only $S(y_j)$ would suffice to obtain the solution for the variables in constraint $y_j$. More specifically, the steps of LOCO are as follows.

*Step 1, Generating a localized neighborhood.* For clarity, we break the first step into three sub-steps.

*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 $(v_\ell, v_r)$ 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 $v$'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 $r(u) = r(v)$), tie-break by ID.

*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 [2, chapter 14]. We provide the pseudocode in the extended version of the paper [6], 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 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 $x_i$. The constraints arrive in the order defined by $r$. The resulting $x_i$ 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 $r$.

## 3.2 Performance 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, while 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.

**Theorem 1.** *Let $P$ be a problem with a concave objective function and linear inequality constraints, with $n$ variables and $m$ constraints, whose constraint matrix has sparsity $\sigma$. 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 $1 - 1/poly(n, m)$.*

We also have the following result for NUM with a linear objective function, which corresponds to maximizing throughput in NUM.

**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 $1 - 1/\operatorname{poly}(n, m)$.*

In a simulation study, we focus on the case of maximizing throughput, 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. For a description of our experimental setup, please see the full version [6].

Figure 1(a) highlights that LOCO requires considerably fewer messages than ADMM, across both small and large $n$. 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. Even answering *every* query (LOCO Tot) requires only the same order of magnitude as ADMM. The figure includes ADMM with a tolerance $\epsilon^{rel}$ of $10^{-4}$ (ADMM 1) and $10^{-3}$ (ADMM 2) [1]. Even with suboptimal tolerance, which results in fewer iterations, ADMM still requires orders of magnitude more communication than LOCO.

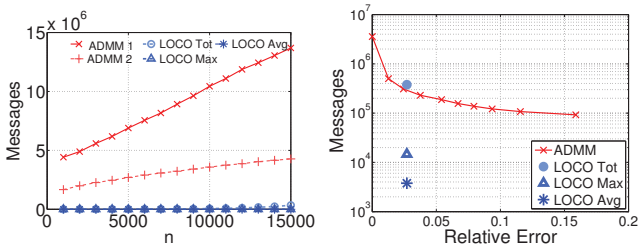


Figure 1: Illustration of the number of messages required by ADMM and LOCO for the synthetic data set with results averaged over 50 trials, for (a) various $n$ and the (b) Pareto optimal curve for ADMM with a range of relative tolerances.

Figure 1(b) illustrates the Pareto optimal frontier for ADMM: the minimal messages needed in order to obtain a particular relative error. Interestingly, the total number of messages used by LOCO is only slightly outside the Pareto frontier of ADMM, indicating that the local advantages that LOCO offers do not incur much global overhead.

# 4. REFERENCES


[1] 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.

[2] 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.

[3] 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.

[4] 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.

[5] 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.

[6] P. London, N. Chen, S. Vardi, and A. Wierman. Distributed optimization via local computation algorithms. `http://users.cms.caltech.edu/~plondon/loco.pdf`, 2017.

[7] 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.

[8] 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.

[9] 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.

[10] 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.

[11] 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.

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