Fast Distributed Algorithms for LP-Type Problems of Low Dimension

Kristian Hinnenthal
Paderborn University, Germany
krijan@mail.upb.de

Christian Scheideler
Paderborn University, Germany
scheideler@upb.de

Martijn Struijs
TU Eindhoven, The Netherlands
m.a.c.struijs@tue.nl

Abstract

In this paper we present various distributed algorithms for LP-type problems in the well-known gossip model. LP-type problems include many important classes of problems such as (integer) linear programming, geometric problems like smallest enclosing ball and polytope distance, and set problems like hitting set and set cover. In the gossip model, a node can only push information to or pull information from nodes chosen uniformly at random. Protocols for the gossip model are usually very practical due to their fast convergence, their simplicity, and their stability under stress and disruptions. Our algorithms are very efficient (logarithmic rounds or better with just polylogarithmic communication work per node per round) whenever the combinatorial dimension of the given LP-type problem is constant, even if the size of the given LP-type problem is polynomially large in the number of nodes.

2012 ACM Subject Classification Theory of computation → Distributed algorithms; Theory of computation → Mathematical optimization

Keywords and phrases LP-type problems, linear optimization, distributed algorithms, gossip algorithms

Digital Object Identifier 10.4230/LIPIcs.DISC.2019.23

Funding K. Hinnenthal and C. Scheideler are supported by the DFG SFB 901 (On-The-Fly Computing).

1 Introduction

1.1 LP-type Problems

LP-type problems were defined by Sharir and Welzl [18] as problems characterized by a tuple (H, f) where H is a finite set of constraints and $f : 2^H \to T$ is a function that maps subsets from H to values in a totally ordered set $(T, \leq)$ containing $-\infty$. The function f is required to satisfy two conditions:

- **Monotonicity:** For all sets $F \subseteq G \subseteq H$, $f(F) \leq f(G) \leq f(H)$.
- **Locality:** For all sets $F \subseteq G \subseteq H$ with $f(F) = f(G)$ and every element $h \in H$, if $f(G) < f(G \cup \{h\})$ then $f(F) < f(F \cup \{h\})$.

Given an LP-type problem (H, f), the goal is to determine $f(H)$. In doing so, the following notation has commonly been used. A subset $B \subseteq H$ with $f(B') < f(B)$ for all proper subsets B' of B is called a *basis* of H. An *optimal basis* is a basis B with $f(B) = f(H)$. The maximum cardinality of a basis is called the *(combinatorial) dimension* of (H, f) and denoted by $\dim(H, f)$.

LP-type problems cover many important optimization problems like linear optimization problems (where H is the set of linear inequalities, $f(G)$ represents the optimal solution w.r.t. the given objective function under the constraints $G \subseteq H$, and the dimension is at most the number of variables in the LP) or geometric problems like the smallest enclosing ball problem (where H is the set of points, $f(G)$ is the radius of the smallest enclosing ball for point set $G \subseteq H$, and the dimension is at most $d+1$ for the d-dimensional case).

Clarkson [2] proposed a very elegant randomized algorithm for solving LP-type problems (see Algorithm 1). In this algorithm, each $h \in H$ has a multiplicity of $\mu_h \in \mathbb{N}$, and $H(\mu)$ is a multiset where each $h \in H$ occurs μ_h times in $H(\mu)$. The algorithm requires a subroutine for computing $f(S)$ for sets S of size $O(\dim(H,f)^2)$, but this is usually straightforward if $\dim(H,f)$ is a constant. In the following, let $d = \dim(H,f)$, and we say that an iteration of the repeat-loop is *successful* if $|V| \leq |H(\mu)|/(3d)$, where V is the set of elements violating a solution $f(R)$ for the chosen subset $R \subseteq H(\mu)$ (which might be a multiset), see also the algorithm.

Algorithm 1 Clarkson's Algorithm.

```
 1: **if** $|H| \leq 6\dim(H,f)^2$ **then return** $f(H)$
 2: **else**
 3:     $r := 6\dim(H,f)^2$
 4:     **for all** $h \in H$ **do** $\mu_h := 1$
 5:     **repeat**
 6:         choose a random subset $R$ of size $r$ from $H(\mu)$
 7:         $V := \{h \in H(\mu) \mid f(R) < f(R \cup \{h\})\}$
 8:         **if** $|V| \leq |H(\mu)|/(3\dim(H,f))$ **then**
 9:             **for all** $h \in V$ **do** $\mu_h := 2\mu_h$
10:     **until** $V = \emptyset$
11:     **return** $f(R)$
```

▶ **Lemma 1** ([9])**.** *Let (H,f) be an LP-type problem of dimension d and let μ be any multiplicity function. For any $1 \leq r < m$, where $m = |H(\mu)|$, the expected size of $V = \{h \in H(\mu) \mid f(R) < f(R \cup \{h\})\}$ for a random subset R of size r from $H(\mu)$ is at most $d \cdot \frac{m-r}{r+1}$.*

From this lemma and the Markov inequality it immediately follows that the probability that an iteration of the repeat-loop is successful is at least 1/2. Moreover, it holds:

▶ **Lemma 2** ([15, 20])**.** *Let $k \in \mathbb{N}$ and B be an arbitrary optimal basis of H. After $k \cdot d$ successful iterations, $2^k \leq \mu(B) < |H| \cdot e^{k/3}$.*

Lemma 2 implies that Clarkson's algorithm must terminate after at most $O(d \log |H|)$ successful iterations (as otherwise $2^k > |H| \cdot e^{k/3}$), so Clarkson's algorithm performs at most $O(d \log |H|)$ iterations of the repeat-loop, on expectation. This bound is also best possible in the worst case for any $d \ll |H|$: given that there is a unique optimal basis B of size d, its elements can have a multiplicity of at most $\sqrt{|H|}$ after $(\log |H|)/2$ iterations, so the probability that B is contained in R is polynomially small in $|H|$ up to that point.

Clarkson's algorithm can easily be transformed into a distributed algorithm with expected runtime $O(d \log^2 n)$ if n nodes are available that are interconnected by a hypercube, $n = |H|$, and each node is responsible for one element in H, for example, because in that case every iteration of the repeat-loop can be emulated in $O(\log n)$ communication rounds, w.h.p.[1], using appropriate (weighted random) routing, broadcast, and convergecast approaches. However, it has been open so far whether it is also possible to construct a distributed algorithm for LP-type problems with an expected runtime of just $O(d \log n)$ (either with a variant of Clarkson's algorithm or a different approach). We will show in this paper that this is possible when running certain variants of Clarkson's algorithm in the gossip model, even if H has a size polynomial in n.

1.2 Network Model

We assume that we are given a fixed node set U (instead of the standard notation V to distinguish it from the violator set V) of size n consisting of the nodes $v_1, \ldots, v_n$. In our paper, we do not require the nodes to have IDs. Moreover, we assume the standard synchronous message passing model, i.e., the nodes operate in synchronous *(communication) rounds*, and all messages sent (or requested) in round i will be received at the beginning of round $i+1$.

In the (uniform) gossip model, a node can only send or receive messages via random *push* and *pull* operations. In a push operation, it can send a message to a node chosen uniformly at random while in a pull operation, it can ask a node chosen uniformly at random to send it a message. We will restrict the message size (i.e., its number of bits) to $O(\log n)$. A node may execute multiple push and pull operations in parallel in a round. The number of push and pull operations executed by it in a single round is called its *(communication) work*.

Protocols for the gossip model are usually very practical due to their fast convergence, their simplicity, and their stability under stress and disruptions. Many gossip-based protocols have already been presented in the past, including protocols for information dissemination, network coding, load-balancing, consensus, and quantile computations (see [3, 11, 12, 13, 14] for some examples). Also, gossip protocols can be used efficiently in the context of population protocols and overlay networks, two important areas of network algorithms. In fact, it is easy to see that any algorithm with runtime T and maximum work W in the gossip model can be emulated by overlay networks in $O(T + \log n)$ time and with maximum work $O(W \log n)$ w.h.p. (since it is easy to set up n (near-)random overlay edges, one per node, in hypercubic networks in $O(\log n)$ time and with $O(\log n)$ work, w.h.p., and this can be pipelined to avoid a $\log n$-overhead in the runtime).

1.3 Related Work

There has already been a significant amount of work on finding efficient sequential and parallel algorithms for linear programs of low dimension (i.e., based on a small number of variables), which are a special case of LP-type problems of low combinatorial dimension (see [5] for a very thorough survey). We just focus here on parallel algorithms. The fastest parallel algorithm known for the CRCW PRAM is due to Alon and Megiddo [1], which has a runtime of $O(d^2 \log^2 d)$. It essentially follows the idea of Clarkson, with the main difference that it replicates elements in V much more aggressively by exploiting the power of the CRCW PRAM. This is achieved by first compressing the violated elements into a small area and then replicating them by a factor of $n^{1/(4d)}$ (instead of just 2). The best work-optimal algorithm for the CRCW PRAM is due to Goodrich [10], which is based on an algorithm by Dyer and Frieze [6] and has a runtime of $O((\log \log n)^d)$. This also implies a work-optimal algorithm

[1] By "with high probability", or short, "w.h.p.", we mean a probability of least $1 - 1/n^c$ for any constant $c > 0$.

for the EREW PRAM, but the runtime increases to $O(\log n(\log\log n)^d)$ in this case. The fastest parallel algorithm known for the EREW PRAM is due to Dyer [4], which achieves a runtime of $O(\log n(\log\log n)^{d-1})$ when using an $O(\log n)$-time parallel sorting algorithm (like Cole's algorithm). Since the runtime of any algorithm for solving a linear program of constant dimension in an EREW PRAM is known to be $\Omega(\log n)$ [5], the upper bound is optimal for $d = 1$.

Due to Ranade's seminal work [16], it is known that any CRCW PRAM step can be emulated in a butterfly network in $O(\log n)$ communication rounds, yielding an $O(d^2 \log^2 d \log n)$-time algorithm for linear programs of constant dimension in the butterfly. However, it is not clear whether any of the parallel algorithms would work for arbitrary LP-type problems. Also, none of the proposed parallel algorithms seem to be easily adaptable to an algorithm that works efficiently (i.e., in time $o(\log^2 n)$ and with *polylog* work) for the gossip model as they require processors to work together in certain groups or on certain memory locations in a coordinated manner, and assuming no (unique) node IDs would further complicate the matter.

As mentioned above, LP-type problems were introduced by Sharir and Welzl [18]. Since then, various results have been shown, but only for sequential algorithms. Combining results by Gärtner [7] with Clarkson's methods, Gärtner and Welzl [8] proposed an algorithm with runtime $O(d^2|H|) + e^{O(\sqrt{d \log d})}$, for any d, which implies that as long as $d = O(\log^2 |H| / \log\log |H|)$, $f(H)$ can be determined in polynomial time. Extensions of LP-type problems were studied by Gärtner [7] (abstract optimization problems) and Skovron [19] (violator spaces).

1.4 Our Results

In all of our results, we assume that initially, H is randomly distributed among the nodes. This is easy to achieve in the gossip model if this is not the case (for example, each node initially represents its own point for the smallest enclosing ball problem) by performing a push operation on each element. The nodes are assumed to know f, and we require the nodes to have a constant factor estimate of $\log n$ for the algorithms to provide a correct output, w.h.p., but they may not have any information about $|H|$. For simplicity, we also assume that the nodes know d. If not, they may perform a binary search on d (by stopping the algorithm and switching to $2d$ if it takes too long for some d), which does not affect our bounds below since they depend at least linearly on d.

In all of our results and proofs we assume that the dimension d of the given LP-type problem is at most logarithmic in $|H|$, to ensure that internal computations only take polynomial time and message sizes satisfy our $O(\log n)$ bound. Section 2 starts with the lightly loaded case (i.e., $|H| = O(n \log n)$, where $n = |U|$) and proves the following theorem.

▶ **Theorem 3.** *For any LP-type problem (H, f) satisfying $|H| = O(n \log n)$, the Low-Load Clarkson Algorithm finds an optimal solution in $O(d \log n)$ rounds with maximum work $O(d^2 + \log n)$ per round, w.h.p.*

At a high level, the Low-Load Clarkson Algorithm is similar to the original Clarkson algorithm, but sampling a random subset and termination detection are more complex now, and a filtering approach is needed to keep $|H(\mu)|$ low at all times so that the work is low. In Section 3, we then consider the highly loaded case and prove the following theorem.

▶ **Theorem 4.** *For any LP-type problem (H, f) with $|H| = \omega(n \log n)$ and $|H| = O(poly(n))$, the High-Load Clarkson Algorithm finds an optimal solution in $O(d \log n)$ rounds with maximum work $O(d \log n)$ per round, w.h.p. If we allow a maximum work of $O(d \log^{1+\epsilon} n)$ per round, for any constant $\epsilon > 0$, the runtime reduces to $O(d \log(n)/ \log\log(n))$, w.h.p.*

Note that as long as we only allow the nodes to spend polylogarithmic work per round, a trivial lower bound on the runtime when using Clarkson's approach is $\Omega(\log(n)/\log\log(n))$ since in $o(\log(n)/\log\log(n))$ rounds an element in H can only be spread to $n^{o(1)}$ nodes, so the probability of fetching it under the gossip model is minute.

The reason why we designed different algorithms for the lightly loaded and highly loaded cases is that the Low-Load Clarkson Algorithm is much more efficient than the High-Load Clarkson Algorithm concerning internal computations. Also, it is better concerning the work for the lightly loaded case, but its work does not scale well with an increasing $|H|$. The main innovation for Theorem 4 is that we come up with a Chernoff-style bound for $|V|$ that holds for all LP-type problems. Gärtner and Welzl [9] also provided a Chernoff-style bound on $|V|$ for LP-type problems, but their proof only works for LP-type problems that are regular (i.e., for all $G \subseteq H$ with $|G| \geq d$, all optimal bases of G have a size of exactly d) and non-degenerate (i.e., every $G \subset H$ with $|G| \geq d$ has a unique optimal basis). While regularity can be enforced in the non-degenerate case, it is not known so far how to make a general LP-type problem non-degenerate without substantially changing its structure (though for most of the applications considered so far for LP-type problems, slight perturbations of the input would solve this problem). Also, since the duplication approach of Clarkson's algorithm generates degenerate instances, their Chernoff-style bound therefore cannot be used here.

2 Low-Load Clarkson Algorithm

Suppose that we have an arbitrary LP-type problem (H, f) of dimension d with $|H| = O(n \log n)$. First, we present and analyze an algorithm for $|H| \geq n$, and then we extend it to any $1 \leq |H| = O(n \log n)$.

Recall that initially the elements of H are assigned to the nodes uniformly and independently at random. Let us denote the set of these elements in node v_i by $H_0(v_i)$ to distinguish them from copies created later by the algorithm, and let $H_0 = \bigcup_i H_0(v_i)$.

At any time, $H(v_i)$ denotes the (multi)set of elements in H known to v_i (including the elements in H_0) and $H(U) = \bigcup_i H(v_i)$, where U represents the node set. Let $m = |H(U)|$. At a high level, our distributed algorithm is similar to the original Clarkson algorithm, but sampling a random subset and termination detection are more complex now (which will be explained in dedicated subsections). In fact, the sampling might fail since a node v_i might not be able to collect enough elements for its sample set R_i. Also, a filtering approach is needed to keep $|H(U)|$ low at all times (see Algorithm 2). However, it will never become too low since the algorithm never deletes an element in H_0, so $|H(U)| \geq n$ at any time. Note that never deleting an element in H_0 also guarantees that no element in H will ever be washed out (which would result in incorrect solutions).

For the runtime analysis, we note that sampling R_i can be done in one round (see Section 2.1), spreading the violator set V_i just takes one round (by executing the push operations in parallel), and we just need one more round for processing the received elements h, so each iteration of the repeat loop just takes $O(1)$ rounds. We start with a slight variant of Lemma 1.

▶ **Lemma 5.** *Let (H, f) be an LP-type problem of dimension d. For any $1 \leq r < m$, where $m = |H(U)|$, the expected size of $V_i = \{h \in H(v_i) \mid f(R) < f(R \cup \{h\})\}$ for a random subset R of size r from $H(U)$ is at most $d \cdot \frac{m-r}{n(r+1)}$.*

Proof. According to Lemma 1, the expected size of $V(R) = \{h \in H(V) \mid f(R) < f(R \cup \{h\})\}$ for a random subset R is at most $d \cdot \frac{m-r}{r+1}$. Since every element in $H(U)$ has a probability of $1/n$ to belong to $H(v_i)$, $\mathbb{E}[|V_i|] \leq d \cdot \frac{m-r}{n(r+1)}$. ◀

Algorithm 2 Low-Load Clarkson Algorithm.

```
1: repeat
2:     for all nodes $v_i$ in parallel do
3:         choose a random subset $R_i$ of size $6d^2$ from $H(U)$          ▷ see Section 2.1
4:         if the sampling of $R_i$ succeeds then
5:             $V_i := \{h \in H(v_i) \mid f(R_i) < f(R_i \cup \{h\})\}$
6:             for all $h \in V_i$ do push($h$)                            ▷ randomly spread $V_i$
7:         for all $h$ received by $v_i$ do add $h$ to $H(v_i)$
8:         for all $h \in H(v_i) - H_0(v_i)$ do
9:             keep $h$ with probability $1/(1 + 1/(2d))$
10: until at least one $v_i$ satisfies $f(R_i) = f(H)$                      ▷ see Section 2.2
```

This allows us to prove the following lemma.

▶ **Lemma 6.** *For all i, $|V_i| = O(m/n + \log n)$, w.h.p., and $\sum_{i=1}^{n} |V_i| \le m/(3d)$, w.h.p.*

Proof. Let the random variable X_i be defined as $|V_i|$ and let $X = \sum_i X_i$. If the sampling of R_i fails then, certainly, $X_i = 0$, and otherwise, $\mathbb{E}[X_i] \le d \cdot \frac{m-r}{n(r+1)}$ for all i. Thus, $\mathbb{E}[X] \le d \cdot \frac{m-r}{r+1}$. Also, since the elements in $H(U)$ are distributed uniformly and independently at random among the nodes at all times, the standard Chernoff bounds imply that $|H(v_i)| = O(m/n + \log n)$ w.h.p., and therefore also $X_i \le O(m/n + \log n)$ w.h.p. Unfortunately, the X_i's are not independent since the $H(v_i)$'s are not chosen independently of each other though the R_i's are, but the dependencies are minute: given that we have already determined $H(v_j)$ for k many v_j's, where $k = o(n)$ is sufficiently small, the probability that any one of the remaining elements $h \in H(U)$ is assigned to $H(v_i)$ is $1/(n - o(n)) = (1 + o(1))/n$, so that for any subset $S = \{i_1, \ldots, i_k\} \subseteq \{1, \ldots, n\}$ of size k,

$$\begin{aligned}
\mathbb{E}[X_{i_1} \cdot \ldots \cdot X_{i_k}] &= \sum_{x_{i_1},\ldots,x_{i_k}} x_{i_1} \cdot \ldots \cdot x_{i_k} \cdot \Pr[X_{i_1} = x_{i_1} \ \wedge \ldots \wedge \ X_{i_k} = x_{i_k}] \\
&= \sum_{H(v_{i_1})} \Pr[H(v_{i_1})] \sum_{x_{i_1}} x_{i_1} \Pr[X_{i_1} = x_{i_1} \mid H(v_{i_1})] \cdot \\
&\quad \left(\sum_{H(v_{i_2})} \Pr[H(v_{i_2}) \mid H(v_{i_1})] \sum_{x_{i_2}} x_{i_2} \Pr[X_{i_2} = x_{i_2} \mid H(v_{i_1}) \wedge H(v_{i_2})] \cdot \ldots \right) \\
&= \sum_{H(v_{i_1})} \Pr[H(v_{i_1})] \sum_{x_{i_1}} x_{i_1} \Pr[X_{i_1} = x_{i_1} \mid H(v_{i_1})] \cdot \\
&\quad \left(\sum_{H(v_{i_2})} (1 + o(1)) \Pr[H(v_{i_2})] \sum_{x_{i_2}} x_{i_2} \Pr[X_{i_2} = x_{i_2} \mid H(v_{i_2})] \cdot \ldots \right) \\
&= \ldots \le (1 + o(1))^k \prod_{j=1}^{k} \mathbb{E}[X_{i_j}] \le \left((1 + o(1))\, d \cdot \frac{m-r}{n(r+1)} \right)^k . \qquad (*)
\end{aligned}$$

This allows us to use a Chernoff-Hoeffding-style bound for k-wise negatively correlated random variables, which is a slight extension of Theorem 3 in [17]:

▶ **Theorem 7.** *Let $X_1, \ldots, X_n$ be random variables with $X_i \in [0, C]$ for some $C > 0$. Suppose there is a $k > 1$ and $q > 0$ with $\mathbb{E}[\prod_{i \in S} X_i] \leq q^s$ for all subsets $S \subseteq \{1, \ldots, n\}$ of size $s \leq k$. Let $X = \sum_{i=1}^n X_i$ and $\mu = q \cdot n$. Then it holds for all $\delta > 0$ with $k \geq \lceil \mu\delta \rceil$ that*

$$\Pr[X \geq (1+\delta)\mu] \leq e^{-\min\{\delta^2, \delta\}\mu/(3C)}$$

Setting $C = \Theta(m/n + \log n)$, $\mu = q \cdot n$ with $q = (1 + o(1))d \cdot \frac{m-r}{n(r+1)}$ and $r = 6d^2$, and $\delta > 0$ large enough so that $\delta^2\mu = \omega(C \ln n)$ but $\delta\mu = o(n)$ so that inequality $(*)$ applies, which works for $\delta = \Theta(\sqrt{(Cd \ln n)/m}) = O(\sqrt{(d \ln^2 n)/n})$, $\Pr[X \geq m/(3d)]$ is polynomially small in n. ◀

Next, we show that m will never be too large, so that the communication work of the nodes will never be too large.

▶ **Lemma 8.** *For up to polynomially many iterations of the Low-Load Clarkson Algorithm, $|H(U)| = O(|H_0|)$, w.h.p.*

Proof. Let $q = |H(U) - H_0|$ and suppose that $|H(U)| \geq c|H_0|$ for some $c \geq 4$, which implies that $q \geq (c-1)/c \cdot m$. Then it holds for the size q' of $H(U) - H_0$ at the end of a repeat-round that

$$\begin{aligned}\mathbb{E}[q'] &\leq \left(q + \frac{m}{3d}\right) \cdot 1/\left(1 + \frac{1}{2d}\right) \leq \left(q + \frac{cq}{(c-1) \cdot 3d}\right) / \left(1 + \frac{1}{2d}\right) \\ &= q\left(\left(1 + \frac{1}{2d}\right) - \left(\frac{1}{6d} - \frac{1}{(c-1) \cdot 3d}\right)\right) / \left(1 + \frac{1}{2d}\right) \\ &= q(1 - \Theta(1/d))\end{aligned}$$

Since the decision to keep elements $h \in H(U) - H_0$ is done independently for each h, it follows from the Chernoff bounds that $\Pr[q' > q]$ is polynomially small in n for $|H(U)| \geq 4|H_0|$. Moreover, Lemma 6 implies that $|H(U)|$ can increase by at most $|H(U)|/3$ in each iteration, w.h.p., so for polynomially many iterations of the algorithm, $|H(U)| \leq 5|H_0|$ w.h.p. ◀

Thus, combining Lemma 6 and Lemma 8, the maximum work per round for pushing out some V_i is bounded by $O(\log n)$ w.h.p. Next we prove a lemma that adapts Lemma 2 to our setting.

▶ **Lemma 9.** *Let B be an arbitrary optimal basis of H. If, for T many iterations of the Low-Load Clarkson Algorithm, every node was successful in sampling a random subset and no v_i satisfies $f(R_i) = f(H)$, then $\mathbb{E}[|\{h \in H(U) \mid h \in B\}|] \geq (2/\sqrt{e})^{T/d}$ after these T iterations.*

Proof. Let $B = \{h_1(B), \ldots, h_b(B)\}$, $b \leq d$, and let $p_{i,j}$ be the probability that $f(R_i) < f(R_i \cup \{h_j(B)\})$. If node v_i has chosen some R_i with $f(R_i) < f(H)$, then there must exist an $h_j(B)$ with $f(R_i) < f(R_i \cup \{h_j(B)\})$, which implies that under the condition that $f(R_i) < f(H)$, $\sum_j p_{i,j} \geq 1$. The $p_{i,j}$'s are the same for each v_i since each v_i has the same probability of picking some subset R of $H(U)$ of size $6d^2$. Hence, we can simplify $p_{i,j}$ to p_j and state that $\sum_j p_j \geq 1$. Now, let $p_{j,t}$ be the probability that $f(R) < f(R \cup \{h_j(B)\})$ for a randomly chosen subset R in iteration t, and fix any values for the $p_{j,t}$ so that $\sum_j p_{j,t} \geq 1$ for all j and t. Let $\mu_{j,t}$ be the multiplicity of $h_j(B)$ at the end of round t. Then, for all j, $\mu_{j,0} \geq 1$, and

$$\mathbb{E}[\mu_{j,t+1}] \geq \frac{1 + p_{j,t}}{1 + 1/(2d)} \cdot \mu_{j,t}$$

Hence, $\mathbb{E}[\mu_{j,T}] \geq (\prod_{t=1}^{T}(1+p_{j,t}))/(1+1/(2d))^T$. Since $1+x \geq 2^x$ for all $x \in [0,1]$, it follows that $\prod_t(1+p_{j,t}) \geq \prod_t 2^{p_{j,t}} = 2^{\sum_t p_{j,t}}$. Also, since $\sum_{t=1}^{T}\sum_j p_{j,t} \geq T$, there must be a j^* with $\sum_{t=1}^{T} p_{j^*,t} \geq T/d$. Therefore, there must be a j^* with $\mathbb{E}[\mu_{j^*,T}] \geq 2^{T/d}/(1+1/(2d))^T \geq 2^{T/d}/e^{T/(2d)}$, which completes the proof. ◀

Since $|H(U)|$ is bounded by $O(|H_0|)$ w.h.p., the expected number of $h \in H$ with $h \in B$ should be in $O(|H_0|)$ as well. Due to Lemma 8, this cannot be the case if $T = \Omega(d \log |H_0|)$ is sufficiently large. Thus, the algorithm must terminate within $O(d \log |H_0|) = O(d \log n)$ rounds w.h.p. In order to complete the description of our algorithm, we need distributed algorithms satisfying the following claims:

1. The nodes v_i succeed in sampling subsets R_i uniformly at random in a round, w.h.p., with maximum work $O(d^2 + \log n)$.
2. Once a node v_i has chosen an R_i with $f(R_i) = f(H)$, all nodes are aware of that within $O(\log n)$ communication rounds, w.h.p., so that the Low-Load Clarkson Algorithm can terminate. The maximum work for the termination detection is $O(\log n)$ per round.

The next two subsections are dedicated to these algorithms.

2.1 Sampling Random Subsets

For simplicity, we assume here that every node knows the exact value of $\log n$, but it is easy to see that the sampling algorithm also works if the nodes just know a constant factor estimate of $\log n$, if the constant c used below is sufficiently large.

Each node v_i samples a subset R_i in a way that is as simple as it can possibly get: v_i asks (via pull requests) $s = c(6d^2 + \log n)$ many nodes v_j, selected uniformly and independently at random, to send it a random element in $H(v_j)$, where c is a sufficiently large constant. If v_i doesn't receive at least $6d^2$ distinct elements (where two elements are distinct if they do not represent the *same* copy of some $h \in H$), the sampling fails. Otherwise, v_i selects the first $6d^2$ distinct elements (based on an arbitrary order of the pull requests) for its subset R_i. Certainly, the work for each node is just $O(d^2 + \log n)$.

▶ **Lemma 10.** *For any i, node v_i succeeds in sampling a subset R_i uniformly at random, w.h.p.*

Proof. Suppose that v_i succeeds in receiving k distinct elements in the sampling procedure above. Since the elements in $H(U)$ are distributed uniformly and independently at random among the nodes, every subset R of size k in $H(U)$ has the same probability of representing these k elements. Hence, it remains to show that v_i succeeds in receiving at least $6d^2$ distinct elements w.h.p.

Consider any numbering of the pull requests from 1 to s. For the jth pull request of v_i, two bad events can occur. First of all, the pull request might be sent to a node that does not have any elements. Since $|H(U)| \geq n$, the probability for that is at most $(1-1/n)^n \leq 1/e$. Second, the pull request might return an element that was already returned by one of the prior $j-1$ pull requests. Since this is definitely avoided if the jth pull request selects a node that is different from the nodes selected by the prior $j-1$ pull requests, the probability for that is at most $(j-1)/n$. So altogether, the probability that a pull request fails is at most $1/e + s/n \leq 1/2$.

Now, let the binary random variable X_j be 1 if and only if the jth pull request fails. Since the upper bound of $1/2$ for the failure holds independently of the other pull requests, it holds for any subset $S \subseteq \{1, \ldots, s\}$ that $\mathbb{E}[\prod_{j \in S} X_j] \leq (1/2)^{|S|}$. Hence, Theorem 7 implies that $\sum_j X_j \leq 3s/4$ w.h.p. If c is sufficiently large, then $s/4 \geq 6d^2$, which completes the proof. ◀

Note that our sampling strategy does not reveal any information about which elements are stored in $H(v_i)$, so each element still has a probability of $1/n$ to be stored in $H(v_i)$, which implies that Lemma 5 still holds.

2.2 Termination

For efficiency reasons, the termination check is done *concurrently* with the iterations of the repeat-loop, i.e., instead of waiting for the termination check to complete, the nodes already start a new iteration. We use the following strategy for each node v_i:

Suppose that in iteration t of the repeat loop, $|V_i| = 0$, i.e., $f(R_i) = f(R_i \cup H(v_i))$. Then v_i determines an optimal basis B of R_i, stores the entry $(t, B, 1)$ in a dedicated set S_i, and performs a push operation on $(t, B, 1)$. At the beginning of iteration t_i of the repeat loop, v_i works as described in Algorithm 3. In the comparison between $f(B')$ and $f(B)$ we assume w.l.o.g. that $f(B') = f(B)$ if and only if $B' = B$ (otherwise, we use a lexicographic ordering of the elements as a tie breaker). The parameter c in the algorithm is assumed to be a sufficiently large constant known to all nodes.

Algorithm 3 One iteration of the Termination Algorithm for v_i.

```
1:  for all $(t, B, x)$ received by $v_i$ do                ▷ update best seen base w.r.t. $t$
2:      if there is some $(t, B', x')$ in $S_i$ then
3:          if $f(B') > f(B)$ then discard $(t, B, x)$
4:          if $f(B') < f(B)$ then replace $(t, B', x')$ by $(t, B, x)$
5:          if $f(B') = f(B)$ then
6:              replace $(t, B', x')$ by $(t, B, \min\{x, x'\})$
7:      else
8:          add $(t, B, x)$ to $S_i$
9:  for all $(t, B, x)$ in $S_i$ do                      ▷ check optimality and maturity
10:     if $\exists h \in H(v_i) : f(B) < f(B \cup \{h\})$ then $x := 0$        ▷ not optimal
11:     if $t < t_i - c \log n$ then               ▷ $B$ is mature ($t_i$: current iteration)
12:         remove $(t, B, x)$ from $S_i$
13:         if $x = 1$ then output $f(B)$, stop
14:     else
15:         push$(t, B, x)$
```

▶ **Lemma 11.** *If the constant c in the termination algorithm is large enough, it holds w.h.p.: Once a node v_i satisfies $f(R_i) = f(H)$, then all nodes v_j output a value $f(B)$ with $f(B) = f(H)$ after $c \log n$ iterations, and if a node v_i outputs a value $f(B)$, then $f(B) = f(H)$.*

Proof. Using standard arguments, it can be shown that if the constant c is large enough, then for every iteration t, it takes at most $(c/2) \log n$ further iterations, w.h.p., until the basis B with maximum $f(B)$ injected into some S_i at iteration t (which we assume to be unique by using some tie breaking mechanism) is contained in all S_i's. At this point, we have two cases. If $f(B) = f(H)$, then for all v_i, there is no $h \in H(v_i)$ with $f(B) < f(B \cup \{h\})$ at any point from iteration t to $t + (c/2) \log n$, and otherwise, there must be at least one v_i at iteration $t + (c/2) \log n$ with $f(B) < f(B \cup \{h\})$ for some $h \in H(v_i)$. In the first case, no v_i will ever set x in the entry (t, B, x) to 0, so after an additional $(c/2) \log n$ iterations, every v_i

still stores $(t, B, 1)$ and therefore outputs B. In the second case, there is at least one entry of the form $(t, B, 0)$ at iteration $s + (c/2)\log n$. For this entry, it takes at most $(c/2)\log n$ further iterations, w.h.p., to spread to all nodes so that at the end, no node outputs B. ◀

Since the age of an entry is at most $c \log n$ and for each age a node performs at most one push per iteration, every node executes just $O(\log n)$ pushes per iteration.

2.3 Extension to Any $|H| \geq 1$

If $|H| < n$, the probability that our sampling strategy might fail will get too large. Hence, we need to extend the Low-Load Clarkson algorithm so that we quickly reach a point where $|H(U)| \geq n$ at any time afterwards. We do this by integrating a so-called *pull phase* into the algorithm.

Initially, a node v_i sets its Boolean variable *pull* to *true* if and only if $H_0(v) = \emptyset$ (which would happen if none of the elements in H has been assigned to it). Afterwards, it executes the algorithm shown in Algorithm 4. As long as *pull* = *true* (i.e., v_i is still in its pull phase), v_i keeps executing a pull operation in each iteration of the algorithm, which asks the contacted node v_j to send it a copy of a random element in $H_0(v_j)$, until it successfully receives an element h that way. Once this is the case, v_i pushes the successfully pulled element to a random node v_j (so that all elements are distributed uniformly and independently at random among the nodes), which will store it in $H_0(v_j)$, and starts executing the Low-Load Clarkson algorithm from above.

Algorithm 4 Extended Low-Load Clarkson Algorithm for v_i.

```
1:  repeat
2:      if $pull = true$ then                                  ▷ $v_i$ is still in its pull phase
3:          pull($h$)                                           ▷ $v_i$ expects some $h \in H_0$
4:          if $h \neq NULL$ then
5:              push($h, 0$)                                    ▷ 0: flag that $h$ belongs to $H_0$
6:              $pull := false$                                 ▷ pull phase is over
7:      else
8:          choose a random subset $R_i$ of size $6d^2$ from $H(U)$
9:          if the sampling of $R_i$ succeeds then
10:             $V_i := \{h \in H(v_i) \mid f(R_i) < f(R_i \cup \{h\})\}$
11:             for all $h \in V_i$ do push($h$)
12:     for all $(h, 0)$ received by $v_i$ do add $h$ to $H_0(v_i)$
13:     for all $h$ received by $v_i$ do add $h$ to $H(v_i)$
14:     for all $h \in H(v_i) - H_0(v_i)$ do
15:         keep $h$ with probability $1/(1 + 1/(2d))$
16: until $v_i$ outputs a solution
```

▶ **Lemma 12.** *After $O(\log n)$ rounds, all nodes have completed their pull phase, w.h.p.*

Proof of Lemma 11. Note that no node will ever delete an element in H_0, and pull requests only generate elements for H_0, so the filtering approach of the Low-Load Clarkson algorithm cannot interfere with the pull phase. Thus, it follows from a slight adaptation of proofs in previous work on gossip algorithms (e.g., [13]) that for any $|H| \geq 1$, all nodes have completed their pull phase after at most $O(\log n)$ rounds, w.h.p. ◀

Certainly, $|H_0| \leq n + |H| = O(n \log n)$ and $H \subseteq H_0$ at any time, and once all nodes have finished their pull phase, $|H_0| \geq n$, so we are back to the situation of the original Low-Load Clarkson Algorithm.

During the time when some nodes are still in their pull phase, some nodes might already be executing Algorithm 2, which may cause the sampling of R_i to fail for some nodes v_i. However, the analyses of Lemma 6 and Lemma 8 already take that into account. Once all nodes have finished their pull phase, Lemma 9 applies, which means that after an additional $O(d \log n)$ rounds at least one node has found the optimal solution, w.h.p. Thus, after an additional $O(\log n)$ rounds, all nodes will know the optimal solution and terminate. Altogether, we therefore still get the same runtime and work bounds as before, completing the proof of Theorem 3.

3 High-Load Clarkson Algorithm

If $|H| = \omega(n \log n)$, then our LP-type algorithm in the previous section will become too expensive since, on expectation, $|V_i|$ would be in the order of $m/(dn)$, which is now $\omega(\log n)$. In this section, we present an alternative distributed LP-type algorithm that just causes $O(d \log n)$ work for any $|H| = \mathit{poly}(n)$, but the internal computations are more expensive than in the algorithm presented in the previous section because now $f(S)$-values for sets of size $\Theta(m/n)$ have to be computed instead of just $O(d^2)$. Again, we assume that initially the elements in H are randomly distributed among the nodes in U. Let the initial $H(v_i)$ be all elements of H assigned that way to v_i. As before, $H(U) = \bigcup_i H(v_i)$.

Algorithm 5 High-Load Clarkson Algorithm.

```
1: repeat
2:     for all nodes $v_i$ in parallel do
3:         compute an optimal basis $B_i$ of $H(v_i)$
4:         push($B_i$)
5:         for all $B_j$ received by $v_i$ do
6:             $V_j := \{h \in H(v_i) \mid f(B_j) < f(B_j \cup \{h\})\}$
7:             for all $h \in V_j$ do push($h$)
8:         for all $h$ received by $v_i$ do add $h$ to $H(v_i)$
9: until at least one $v_i$ satisfies $f(H(v_i)) = f(H)$
```

Irrespective of which elements get selected for the V_i's in each round, $H(v_i)$ is a random subset of $H(U)$ because the elements in H are assumed to be randomly distributed among the nodes and every element in V_i is sent to a random node in U. Hence, if follows from $|H(U)| = \omega(n \log n)$ and the standard Chernoff bounds that $|H(v_i)|$ is within $(1 \pm \epsilon)|H(U)|/n$, w.h.p., for any constant $\epsilon > 0$. Thus, we are computing bases of random subsets R of size r within $(1 \pm \epsilon)|H(U)|/n$, w.h.p. This, in turn, implies with $\mathbb{E}[|V_i|] \leq d \cdot \frac{m-r}{n(r+1)}$, where $m = |H(U)|$, that $\mathbb{E}[|V_i|] \leq (1 + \epsilon)d$. In the worst case, however, $|V_i|$ could be very large, so just bounding the expectation of $|V_i|$ does not suffice to show that our algorithm has a low work. Therefore, we need a proper extension of Lemma 5 that exploits higher moments. Note that it works for arbitrary LP-type problems, i.e., also problems that are non-regular and/or degenerate.

▶ **Lemma 13.** *Let (H, f) be an LP-type problem of dimension d and let μ be any multiplicity function. For any $k \geq 1$ and any $1 \leq r < m/2 - k$, where $m = |H(\mu)|$, it holds for $V = \{h \in H(\mu) \mid f(R) < f(R \cup \{h\})\}$ for a random subset R of size r from $H(\mu)$ that $\mathbb{E}[|V|^k] \leq 2(k \cdot d \cdot (m-r)/(r+1))^k$.*

Proof of Lemma 12. By definition of the expected value it holds that

$$\mathbb{E}[|V|^k] = \frac{1}{\binom{m}{r}} \sum_{R \in \binom{H(\mu)}{r}} |V_R|^k$$

For $R \in \binom{H(\mu)}{r}$ and $h \in H(\mu)$ let $X(R,h)$ be the indicator variable for the event that $f(R) < f(R \cup \{h\})$. Then we have

$$\begin{aligned}
\binom{m}{r}\mathbb{E}[|V|^k] &= \sum_{R \in \binom{H(\mu)}{r}} |V_R|^k = \sum_{R \in \binom{H(\mu)}{r}} \left(\sum_{h \in H(\mu)-R} X(R,h) \right)^k \\
&\overset{(1)}{\leq} \sum_{R \in \binom{H(\mu)}{r}} \Bigg(\sum_{h \in H(\mu)-R} X(R,h) + 2^k \sum_{\{h_1,h_2\} \subseteq H(\mu)-R} X(R,h_1) \cdot X(R,h_2) + \dots \\
&\qquad + k^k \sum_{\{h_1,\dots,h_k\} \subseteq H(\mu)-R} X(R,h_1) \cdot \ldots \cdot X(R,h_k) \Bigg)
\end{aligned}$$

(1) holds because $X(R,h)^i = X(R,h)$ for any $i \geq 1$ and there are at most i^k ways of assigning k $X(R,h)$'s, one from each sum in $(\sum_{h \in H(\mu)-R} X(R,h))^k$, to the i $X(R,h)$'s in some $X(R,h_1) \cdot \ldots \cdot X(R,h_i)$. Moreover, for any $k > 1$,

$$\begin{aligned}
&\sum_{R \in \binom{H(\mu)}{r}} \sum_{\{h_1,\dots,h_k\} \subseteq H(\mu)-R} X(R,h_1) \cdot \ldots \cdot X(R,h_k) \\
&= \sum_{Q \in \binom{H(\mu)}{r+k}} \sum_{\{h_1,\dots,h_k\} \subseteq Q} X(Q - \{h_1,\dots,h_k\}, h_1) \cdot \ldots \cdot X(Q - \{h_1,\dots,h_k\}, h_k) \\
&= \sum_{Q \in \binom{H(\mu)}{r+k}} \sum_{\{h_1,\dots,h_{k-1}\} \subseteq Q} \sum_{h_k \in Q - \{h_1,\dots,h_{k-1}\}} X(Q - \{h_1,\dots,h_k\}, h_1) \cdot \ldots \\
&\qquad \cdot X(Q - \{h_1,\dots,h_k\}, h_{k-1}) \cdot X((Q - \{h_1,\dots,h_{k-1}\}) - h_k, h_k) \\
&\overset{(2)}{\leq} \sum_{Q \in \binom{H(\mu)}{r+k}} \sum_{\{h_1,\dots,h_{k-1}\} \subseteq Q} \\
&\qquad \sum_{h_k \in B(Q - \{h_1,\dots,h_{k-1}\})} X((Q - h_k) - \{h_1,\dots,h_{k-1}\}, h_1) \cdot \ldots \\
&\qquad\qquad \cdot X((Q - h_k) - \{h_1,\dots,h_{k-1}\}, h_{k-1}) \\
&\leq \sum_{Q \in \binom{H(\mu)}{r+k}} d \cdot \max_{h_k \in Q} \Bigg(\sum_{\{h_1,\dots,h_{k-1}\} \subseteq Q - h_k} X((Q - h_k) - \{h_1,\dots,h_{k-1}\}, h_1) \cdot \ldots \\
&\qquad \cdot X((Q - h_k) - \{h_1,\dots,h_{k-1}\}, h_{k-1}) \Bigg) \\
&\leq \ \dots \ \leq \sum_{Q \in \binom{H(\mu)}{r+k}} d^k
\end{aligned}$$

where $B(S)$ is an optimal basis of S. (2) holds because $X((Q-\{h_1,\ldots,h_{k-1}\})-h_k,h_k)=0$ for every $h_k \notin B(Q-\{h_1,\ldots,h_{k-1}\})$. The skipped calculations apply the same idea for h_k to $h_{k-1},\ldots,h_2$. Hence, as long as $r+k<|H(\mu)|/2$,

$$\begin{aligned}\binom{m}{r}\mathbb{E}[|V|^k] &= \sum_{R\in\binom{H(\mu)}{r}} |V_R|^k \le \sum_{Q\in\binom{H(\mu)}{r+1}} d + 2^k \sum_{Q\in\binom{H(\mu)}{r+2}} d^2 + \ldots + k^k \sum_{Q\in\binom{H(\mu)}{r+k}} d^k \\ &\le 2k^k \sum_{Q\in\binom{H(\mu)}{r+k}} d^k = 2(dk)^k\binom{m}{r+k}\end{aligned}$$

Resolving that to $\mathbb{E}[|V|^k]$ results in the lemma. ◀

Lemma 13 allows us to prove the following probability bound, which is essentially best possible for constant d by a lower bound in [9].

▶ **Lemma 14.** *Let (H,f) be an LP-type problem of dimension d and let μ be any multiplicity function. For any $\gamma\ge 1$ and $1\le r<m/2-\gamma$, where $m=|H(\mu)|$, it holds for $V=\{h\in H(\mu)\mid f(R)<f(R\cup\{h\})\}$ for a random subset R of size r from $H(\mu)$ that*

$$\Pr[|V|\ge 4\gamma\cdot\frac{d\cdot m}{r+1}]\le 1/2^\gamma$$

Proof. From Lemma 13 and the Markov inequality it follows that, for any $c\ge 1$ and $k\ge 1$, $\Pr[|V|^k\ge c^k\cdot 2(k\cdot d\cdot(m-r)/(r+1))^k]\le 1/c^k$ and therefore,

$$\Pr[|V|\ge c\cdot(1+1/k)(k\cdot d\cdot(m-r)/(r+1))]\le 1/c^k$$

Setting $c=2$ and $k=\gamma$ results in the lemma. ◀

Since, for every element $h\in V$, the probability that $h\in H(v_i)$ is equal to $1/n$, it follows that $|V_i|=O(d\log n)$ for every i, w.h.p., so the maximum work needed for pushing some V_i is $O(d\log n)$. Moreover, the size of $H(U)$ after T iterations is at most $|H|+O(Tdn\log n)$, w.h.p. On the other hand, we will show the following variant of Lemma 9.

▶ **Lemma 15.** *Let B be an arbitrary optimal basis of H. As long as no v_i has satisfied $f(H(v_i))=f(H)$ so far, $\mathbb{E}[|\{h\in H(U)\mid h\in B\}|]\ge 2^{T/d}$ after T iterations of the High-Load Clarkson Algorithm.*

Proof. Let $B=\{h_1(B),\ldots,h_b(B)\}$, $b\le d$, and let $p_{i,j}$ be the probability that $f(B_i)<f(B_i\cup\{h_j(B)\})$. If $f(B_i)<f(H)$, then there must exist an $h_j(B)$ with $f(B_i)<f(B_i\cup\{h_j(B)\})$, which implies that under the condition that $f(B_i)<f(H)$, $\sum_j p_{i,j}\ge 1$. Let ρ_j be the expected number of duplicates created for some copy of $h_j(B)$. Since the B_i's are sent to nodes chosen uniformly at random, $\rho_j=(1/n)\sum_i p_{i,j}$. Certainly, since $p_{i,j}\in[0,1]$ for all i, also $\rho_j\in[0,1]$. Moreover,

$$\sum_j \rho_j=\sum_j(1/n)\sum_i p_{i,j}=(1/n)\sum_i\sum_j p_{i,j}\ge 1$$

Hence, we can use the same arguments as in the proof of Lemma 9, with p_j replaced by ρ_j and without the term $(1+1/(2d))$ in the denominator since we do not perform filtering, to complete the proof. ◀

Thus, because $|H(U)|\le|H|+O(Tdn\log n)$ after T iterations, w.h.p., our algorithm must terminate within $O(d\log|H|)=O(d\log n)$ rounds, w.h.p. For the termination detection, we can again use the algorithm proposed in Section 2.2, which results in an additional work of $O(\log n)$ per round.

3.1 Accelerated High-Load Clarkson Algorithm

If we are willing to spend more work, we can accelerate the High-Load Clarkson Algorithm. Suppose that in Algorithm 5 node v_i does not just push B_i once but C many times. Then the work for that goes up from $O(d)$ to $O(C \cdot d)$, and the maximum work for pushing out the elements of the W_i's is now bounded by $O(C \cdot d \log n)$, w.h.p., which means that after T rounds, $|H(U)|$ is now bounded by $|H| + O(TC \cdot dn \log n)$, w.h.p. Furthermore, we obtain the following result, which replaces Lemma 15.

▶ **Lemma 16.** *Let B be an arbitrary optimal basis of H. As long as no v_i has satisfied $f(H(v_i)) = f(H)$ so far, $\mathbb{E}[|\{h \in H(U) \mid h \in B\}|] \geq (C+1)^{\lfloor T/d \rfloor}$ after T rounds of the High-Load Clarkson Algorithm with parameter C.*

Proof. Recall the definition of ρ_j in the proof of Lemma 15. It now holds that $\rho_j = (C/n)\sum_i p_{i,j}$, which implies that $\rho_j \in [0, C]$ for all j and $\sum_j \rho_j \geq C$. Now, let $\rho_{j,t}$ be the expected number of duplicates created for some copy of $h_j(B)$ in round t, and fix any values of $\rho_{j,t}$ so that $\sum_j \rho_{j,t} \geq C$ and $\rho_j \in [0, C]$ for all j and t. Let $\mu_{j,t}$ be the multiplicity of $h_j(B)$ at the end of round t. Then, for all j, $\mu_{j,0} \geq 1$, and

$$\mathbb{E}[\mu_{j,t+1}] \geq (1 + \rho_{j,t}) \cdot \mu_{j,t}$$

Hence, $\mathbb{E}[\mu_{j,T}] \geq \prod_{t=1}^{T}(1+\rho_{j,t})$. Suppose that $\sum_{t=1}^{T} \rho_{j,t} = M$. Since $\prod_{t=1}^{T}(1+\rho_{j,t})$ is a convex function (i.e., it attains its maximum when $\rho_{j,t} = \rho_{j,t'}$ for all t, t' under the constraint that $\sum_{t=1}^{T} \rho_{j,t}$ is fixed, which can be seen from the fact that $((1+r)+\epsilon)((1+r)-\epsilon) = (1+r)^2 - \epsilon^2$), it gets lowest if as many of the $\rho_{j,t}$'s are as large as possible and the rest is 0. Thus, $\prod_{t=1}^{T}(1+\rho_{j,t}) \geq (C+1)^{\lfloor M/C \rfloor}$.

Since $\sum_{t=1}^{T}\sum_j p_{j,t} \geq C \cdot T$, there must be a j^* with $\sum_{t=1}^{T} p_{j^*,t} \geq C \cdot T/d$. Therefore, there must be a j^* with $\mathbb{E}[\mu_{j^*,T}] \geq (C+1)^{\lfloor T/d \rfloor}$, which completes the proof. ◀

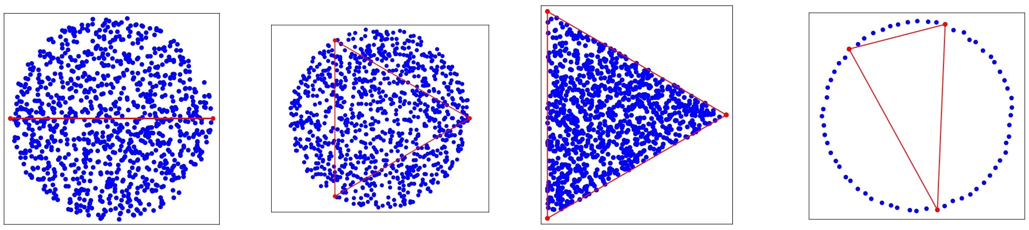
Setting $C = \log^\epsilon n$ for any constant $\epsilon > 0$, it follows that our algorithm must terminate in $O((d/\epsilon)\log(|H|)/\log\log(n)) = O(d\log(n)/\log\log(n))$ rounds, w.h.p. This completes the proof of Theorem 4.

4 Experimental Results

While we have obtained the theoretical bound of $O(d \log n)$ rounds w.h.p. for Algorithms 2 and 5, we are also interested in their practical performance. In particular, we would like to estimate the constant factor hidden in our asymptotic bound. To achieve this, we will look at the specific LP-type problem of finding the minimum enclosing disk, i.e., the two-dimensional version of the minimum enclosing ball problem mentioned in the introduction.

Note that the running time for the termination phase (Algorithm 3) of these algorithms is predictable and independent of the actual input, so we will measure the number of rounds until at least one node found the solution. We consider the four different test cases shown in Figure 1. For each test case, we take the average result of 10 runs of our algorithms with n nodes on n data-points, where $n = 2^i$ ranges over $i = 1, \ldots 14$, (this is extended to 17 for the duo-disk case for the Low-Load Clarkson Algorithm), see Figures 2a and 2b for the results.

For the Low-Load Algorithm, note that the small test cases finish within one round, because there is a high probability that there is a node v_i where $H(v_i)$ contains an optimal basis. For the duo-disk test case the number of rounds is $1.2 \log n$, while it is $1.7 \log n$ for



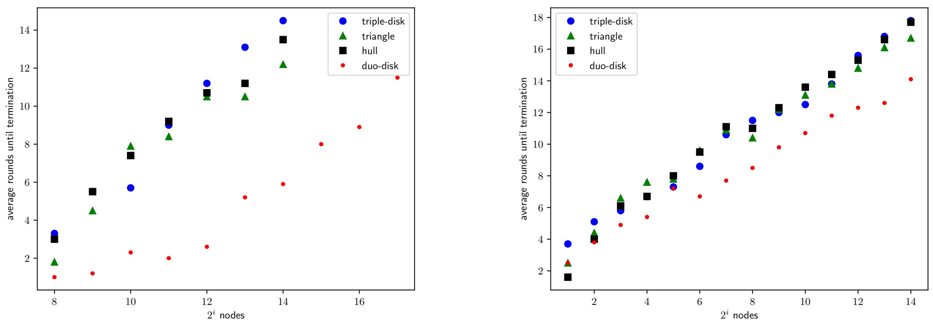
(a) duo-disk: The points are uniformly distributed over a disk defined by 2 points.

(b) triple-disk: The points are uniformly distributed over a disk defined by 3 points.

(c) triangle: The points are uniformly distributed over a triangle.

(d) hull: The points are perturbed vertices of a regular polygon.

Figure 1 The 4 types of data-sets of the minimum enclosing disk problem used in our experimental evaluation: duo-disk, triple-disk, triangle, and hull.



(a) The average number of rounds until a node finds the minimum enclosing disk over 10 runs of the Low-Load Clarkson Algorithm. Test instances of size $< 2^8$ finish in one round.

(b) The average number of rounds until a node finds the minimum enclosing disk over 10 runs of the High-Load Clarkson Algorithm.

Figure 2 The result of the experimental analysis.

the other test cases. For the High-Load Algorithm, the runtime of the duo-disk test cases is around $0.9 \log n$, while it is $1.1 \log n$ for the other test cases. So these experiments show that when applied to the minimum enclosing disk problem, where $d \leq 3$, the constants hidden in our asymptotic bounds are small. Note that the three test cases other than duo-disk behave similarly, while duo-disk runs a bit faster. The difference between the duo-disk case and the other test cases is the size of the optimal basis, which is 2 for duo-disk and 3 for the others. This confirms the runtime bound implied by Lemma 2, since we can define d there as the minimum size of an optimal basis, and suggests that other features of the problem do not influence the number of rounds much.

5 Conclusion

In this paper we presented various efficient distributed algorithms for LP-type problems in the gossip model. Of course, it would be interesting to find out which other problems can be efficiently solved within Clarkson's framework, and whether some of our bounds can be improved.

References

1 Noga Alon and Nimrod Megiddo. Parallel linear programming in fixed dimension almost surely in constant time. *Journal of the ACM*, 41(2):422–434, 1994.

2 Kenneth L. Clarkson. Las Vegas algorithms for linear and integer programming when the dimension is small. *Journal of the ACM*, 42(2):488–499, 1995.

3 Benjamin Doerr, Leslie Ann Goldberg, Lorenz Minder, Thomas Sauerwald, and Christian Scheideler. Stabilizing Consensus with the Power of Two Choices. In *Proc. 23rd ACM Symposium on Parallelism in Algorithms and Architectures (SPAA)*, pages 149–158, 2011.

4 Martin Dyer. A parallel algorithm for linear programming in fixed dimension. In *Proc. 11th Symposium on Computational Geometry (SoCG)*, pages 345–349, 1995.

5 Martin Dyer, Bernd Gärtner, Nimrod Megiddo, and Emo Welzl. Linear Programming. In Jacob E. Goodman and Joseph O'Rourke, editors, *Handbook of Discrete and Computational Geometry, 3rd edition*, chapter 49, pages 49:1–49:19. Chapman and Hall/CRC, 2017.

6 Martin E. Dyer and Alan M. Frieze. A randomized algorithm for fixed-dimensional linear programming. *Mathematical Programming*, 44(1–3):203–212, 1989.

7 Bernd Gärtner. A subexponential algorithm for abstract optimization problems. *SIAM Journal on Computing*, 24(5):1018–1035, 1995.

8 Bernd Gärtner and Emo Welzl. Linear programming – randomization and abstract frameworks. In *Proc. 13th Symposium on Theoretical Aspects of Computer Science (STACS)*, pages 669–687, 1996.

9 Bernd Gärtner and Emo Welzl. A simple sampling lemma: Analysis and applications in geometric optimization. *Discrete Computational Geometry*, 25:569–590, 2001.

10 Michael T. Goodrich. Geometric partitioning made easier, even in parallel. In *Proc. 9th ACM Symposium on Computational Geometry (SoCG)*, pages 73–82, 1993.

11 Bernhard Häupler. Analyzing network coding (gossip) made easy. *Journal of the ACM*, 63(3):26:1–26:22, 2016.

12 Bernhard Häupler, Jeet Mohapatra, and Hsin-Hao Su. Optimal gossip algorithms for exact and approximate quantile computations. In *Proc. ACM Symposium on Principles of Distributed Computing (PODC)*, pages 179–188, 2018.

13 Richard M. Karp, Christian Schindelhauer, Scott Shenker, and Berthold Vöcking. Randomized rumor spreading. In *Proc. 41st IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 565–574, 2000.

14 David Kempe, Alin Dobra, and Johannes Gehrke. Gossip-Based Computation of Aggregate Information. In *Proc. 44th IEEE Symposium on Foundations of Computer Science (FOCS 2003)*, pages 482–491, 2011.

15 Nick Littlestone. Learning quickly when irrelevant attributes abound: A new linear-threshold algorithm. In *Proc. 28th IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 68–77, 1987.

16 Abhiram G. Ranade. How to emulate shared memory. *Journal of Computer and System Sciences*, 42(3):307–326, 1991.

17 Jeanette Schmidt, Alan Siegel, and Aravind Srinivasan. Chernoff-Hoeffding bounds for applications with limited independence. *SIAM Journal on Discrete Mathematics*, 8(2):223–250, 1995.

18 Micha Sharir and Emo Welzl. A combinatorial bound for linear programming and related problems. In *Proc. 9th Symposium on Theoretical Aspects of Computer Science (STACS)*, pages 569–579, 1992.

19 Petr Škovroň. *Abstract Models of Optimization Problems*. PhD thesis, Charles University, Prague, 2007.

20 Emo Welzl. Partition trees for triangle counting and other range searching problems. In *Proc. 4th ACM Symposium on Computational Geometry (SoCG)*, pages 23–33, 1988.