Electronic Colloquium on Computational Complexity, Revision 1 of Report No. 142 (2016)

Bounding laconic proof systems by solving CSPs in parallel

Jason Li* Ryan O'Donnell†

September 11, 2016

Abstract

We show that the basic semidefinite programming relaxation value of any constraint satisfaction problem can be computed in NC; that is, in parallel polylogarithmic time and polynomial work. As a complexity-theoretic consequence we get that $\mathsf{MIP1}[k, c, s] \subseteq \mathsf{PSPACE}$ provided $s/c \leq (.62 - o(1))k/2^k$, resolving a question of Austrin, Håstad, and Pass. Here $\mathsf{MIP1}[k, c, s]$ is the class of languages decidable with completeness c and soundness s by an interactive proof system with k provers, each constrained to communicate just 1 bit.

1 Introduction

The famous $\mathsf{IP} = \mathsf{PSPACE}$ and $\mathsf{MIP} = \mathsf{NEXP}$ theorems [LFKN92, Sha92, BFL91] are concerned with the computational power of randomized interactive proof systems. In the multiprover interactive proof (MIP) setup, a randomized (private-coin) polynomial-time verifier is allowed to communicate with k all-powerful, separated provers in an attempt to decide if a given n-bit string x is in a language L. The complexity class $\mathsf{MIP}[k, c, s]$ is the set of languages decidable by such a proof system with completeness c and soundness s; this means that for all $x \in L$, the verifier accepts with probability at least c and for all $x \notin L$, the verifier accepts with probability at most s. It is known that in this definition the precise values of the constants k, c, and s do not really matter: for all $k \geq 2$ and all $0 < s < c \leq 1$ it holds that $\mathsf{MIP}[k, c, s] = \mathsf{NEXP}$ (and also $\mathsf{MIP}[1, c, s] = \mathsf{PSPACE}$).

A significant amount of the original research on interactive proof systems was concerned with the effect of bounding the number of *rounds* of communication. Goldreich and Håstad [GH98] introduced a refined version of this question in which the number of *bits* communicated by the provers is bounded. Such provers are called *laconic*. This line of research is motivated in part by the fact that several interesting proof systems indeed use laconic provers; for example, in the well-known 1-prover proof systems for Quadratic-Nonresiduosity [GMR89] and Graph-Nonisomorphism [GMW91], the prover communicates just a single bit to the verifier (achieving $c = 1$, $s = 1/2$). An example result in this area, due to Goldreich, Vadhan, and Wigderson [GVW02], is that if $1 > c^2 > s > c/2 > 0$, then the class of languages decided by an interactive proof system with 1 prover communicating 1 bit is precisely SZK.

A recent work of Austrin, Håstad, and Pass [AHP13] focused on the class $\mathsf{MIP1}[k, c, s]$, the restriction of $\mathsf{MIP}[k, c, s]$ in which the provers are restricted to sending just 1 bit. (More precisely, given x, the verifier prepares k questions and an acceptance predicate $\phi : \{0,1\}^k \to \{0,1\}$. The questions are sent to the provers simultaneously, and the verifier accepts/rejects according to ϕ's value on the provers' 1-bit responses.) They showed that for any $k \geq 2$ and $c = 1 - \epsilon$ (say),

*Computer Science Department, Carnegie Mellon University. jmli@andrew.cmu.edu

†Computer Science Department, Carnegie Mellon University. Supported in part by NSF grants CCF-0747250, CCF-1116594, CCF-1618679. odonnell@cs.cmu.edu

ISSN 1433-8092

one obtains an intriguing fine-grained hierarchy of complexity classed by varying s. For example, suppose we fix the number of provers to $k = 3$. Austrin et al. show that for any $\epsilon > 0$:

$$\begin{aligned}
&\text{if} \quad 0 < s < 1/8 - \epsilon, && \mathsf{MIP1}[3, 1-\epsilon, s] = \mathsf{BPP};\\
&\text{if} \quad 1/8 + \epsilon < s < 1/4 - \epsilon, && \mathsf{MIP1}[3, 1-\epsilon, s] = \mathsf{SZK};\\
&\text{if} \quad 1/4 + \epsilon < s < 1/2 - \epsilon, && \mathsf{AM} \subseteq \mathsf{MIP1}[3, 1-\epsilon, s] \subseteq \mathsf{EXP};\\
&\text{if} \quad 1/2 + \epsilon < s < \ \ 1 \ \ - \epsilon, && \mathsf{MIP1}[3, 1-\epsilon, s] = \mathsf{NEXP}.
\end{aligned}$$

Narrowing the gap between AM and EXP for $1/4 < s < 1/2$ is quite an interesting open problem. Austrin et al. asked whether, in this range, the upper bound of EXP can be reduced to PSPACE. In this paper we answer the question in the affirmative. More generally, we reduce the upper bound to PSPACE for all parameter ranges k, c, s for which Austrin et al. had an upper bound of EXP:

Theorem 1.1. *Provided* $s/c \leq (.62 - o(1))k/2^k$ *we have* $\mathsf{MIP1}[k, c, s] \subseteq \mathsf{PSPACE}$. *Also, for all* $\epsilon > 0$ *we have* $\mathsf{MIP1}[3, 1-\epsilon, 1/2-\epsilon] \subseteq \mathsf{PSPACE}$.

Remark 1.2. As discussed in Section 2, we can show similar results for MIP systems in which the provers may communicate any fixed number of bits $c > 1$.

We achieve these new complexity-theoretic upper bounds by showing that the basic semidefinite programming (SDP) value of any constraint satisfaction problem (CSP) can be computed by an efficient deterministic parallel algorithm — polylogarithmic time and polynomial work.

2 Efficient parallel approximation algorithms for CSP SDPs

Here we review the connection between MIP systems and exponential-size CSPs; this connection is standard and appears in Austrin et al.'s work [AHP13, Theorem 5.1]. Suppose $L \in \mathsf{MIP1}[k, c, s]$ and that the associated polynomial-time verifier uses exactly $t(n) = \mathrm{poly}(n)$ random coins on inputs $x \in \{0,1\}^n$. Consider enumerating the actions of the verifier over all its random strings $r \in \{0,1\}^{t(n)}$. For each r, the verifier produces k "questions", along with an acceptance predicate $\phi : \{0,1\}^k \to \{0,1\}$ to be applied to the provers' answers. Thinking of the questions as "CSP variables" to be assigned bits by the provers, we see that on each input $x \in \{0,1\}^n$, the verifier implicitly produces an exponential-size (k-partite) Boolean k-CSP, with a multiset of exactly $2^{t(n)}$ constraints (using various predicates). The optimal value (fraction of constraints satisfied) of this CSP instance is precisely the highest probability with which the provers can cause the verifier to accept. Write $m = 2^{t(n)}$ for the number of constraints in the CSP instance. Now to show that $L \in \mathsf{EXP}$ (say), it suffices to show that for such "implicitly-given" CSPs we can in $\mathrm{poly}(m)$ time distinguish whether the optimal value is at least c or at most s. This is an approximation algorithms problem. In general, the best known polynomial-time approximation algorithms for Boolean k-CSP use semidefinite programming and achieve the approximation ratio $(.62 - o(1))k/2^k$; they are due to Makarychev and Makarychev [MM14]. This is precisely how Austrin et al. achieve Theorem 1.1 with EXP in place of PSPACE. (The factor-$1/2$ approximation when $k = 3$ is due to Zwick [Zwi98].) Regarding Remark 1.2, it is clear that to handle provers communicating c bits, we just need to investigate the best known [MM14] approximation algorithms for k-CSPs over a larger alphabet Σ of cardinality 2^c.

As noted by Austrin et al., to achieve the PSPACE upper bound in Theorem 1.1, what we need to do is show that the Makarychev–Makarychev (and Zwick) SDP-based approximation algorithms for m-constraint CSPs can be implemented in $\mathrm{polylog}(m)$ *space*. Let us make this a little more precise. For any arity-k CSP instance $\mathcal{I}$ with arbitrary predicates $\phi : \Sigma^k \to \{0,1\}$ over a constant-size alphabet Σ, there is a certain "basic SDP" relaxation introduced by Raghavendra [Rag09]. It

is known [RS09] to be at least as strong as the ones used by Makarychev–Makarychev and Zwick; i.e., its value, $\mathrm{SDP}(\mathcal{I})$ is known to be as close or closer to $\mathrm{Opt}(\mathcal{I})$. Thus to prove Theorem 1.1, it suffices to prove the following:

Theorem 2.1. *For any constants*[1] k, q, *and* $\epsilon > 0$, *there is a* $\mathrm{polylog}(m)$*-space Turing Machine that, given as input an* m*-constraint* k*-ary CSP instance* $\mathcal{I}$ *over alphabet* Σ *of size* q, *computes the basic SDP value* $\mathrm{SDP}(\mathcal{I})$ *to within* $\pm\epsilon$

In fact, we prove the following stronger statement:

Theorem 2.2. *The algorithm in Theorem 2.1 can be carried out in (log-space uniform)* NC*; equivalently [SV84], by a (log-space uniform) parallel algorithm using* $\mathrm{poly}(m)$ *deterministic processors and* $\mathrm{polylog}(m)$ *parallel time.*

This is stronger than Theorem 2.1 because (log-space uniform) $\mathsf{NC} \subseteq \mathsf{polyL}$.

Our proof of Theorem 2.2 is similar to prior works (e.g., [JW09, JUW09, JJUW10, Tan11]) that show how to (approximately) solve certain SDPs in NC using the matrix multiplicative weights (MMW) technique of Kale and coauthors [AHK05, AK07, Kal07]. Unfortunately, each specific SDP seems to require its own analysis. Here we use the work of Steurer [Ste10] that shows how the basic $\mathrm{SDP}(\mathcal{I})$ for CSPs can be approximately solved in *serial* quasilinear time, $\widetilde{O}(m)$. We show how to parallelize the algorithm so as to use polylog parallel time, as claimed in Theorem 2.2.

We mention here one important subtlety that arises in translating Steurer's algorithm. Although our complexity-theoretic application Theorem 1.1 is only concerned with space, not time, one may anyway ask whether Steurer's algorithm can be implemented in quasilinear time and polylog space. The subtlety entering into this question has to do with the crucial use of randomness in the algorithm; particularly, the use of the Johnson–Lindenstrauss Lemma while solving the SDP. It seems that it may be possible to perform Steurer's algorithm with $\widetilde{O}(m)$ work and $\mathrm{polylog}(m)$ depth in a standard model of *randomized* parallel computation (for example, this was shown in the special case of approximating Max-Cut by Tangwongsan [Tan11, Chapter 4]). However, in such models, each processor is allowed to store and access any random bits that it uses. When moving to a Turing Machine model, this would seem to necessitate two-way access to a random tape (the "wrong" model of randomized space-bounded computation [Nis90]), as opposed to just a Turing Machine with read-once coin flips. Thus it wouldn't be clear how to get the needed deterministic polylog space algorithm using, say, Nisan's space-derandomization results. Alternatively, one might attempt to directly derandomize the use of the Johnson–Lindenstrauss Lemma using ideas from [EIO02], but it did not seem easy to us to do that in polylog space. See also [Pap13].

Thus in the end, we will only show how to implement Steurer's algorithm in deterministic $\mathrm{polylog}(m)$ parallel time and $\mathrm{poly}(m)$ work, as this is sufficient for Theorem 1.1. (Even $2^{\mathrm{polylog}(m)}$ work would have been sufficient.) By allowing ourselves more than quasilinear work and not insisting on the use of the Johnson–Lindenstrauss Lemma, certain aspects of Steurer's algorithm are simplified. However some effort is still required to prove Theorem 2.1 as parts of Steurer's proof are only sketched, especially the parts concerning arithmetic precision.

3 Outline of Steurer's algorithm

Herein we recall Steurer's algorithm from [Ste10] for approximately solving the basic SDP relaxation for CSPs. As is standard, the algorithm actually solves a decision version of the problem: given $\mathcal{I}$

[1] We could allow k, q, and $1/\epsilon$ to be slightly superconstant; e.g., $\epsilon = \frac{1}{\mathrm{polylog}(m)}$ would be okay. However for simplicity we'll insist they're constant.

and α, the algorithm outputs YES if $\mathrm{SDP}(\mathcal{I}) \geq \alpha$ and outputs NO if $\mathrm{SDP}(\mathcal{I}) \leq \alpha - \epsilon$. Combining this with a binary search allows the algorithm to approximate $\mathrm{SDP}(\mathcal{I})$ to within $\pm\epsilon$ with only a constant multiplicative overhead in parallel time and work.

The statement "$\mathrm{SDP}(\mathcal{I}) \geq \alpha$" corresponds to the statement that there exists a PSD matrix X that satisfies certain inequalities. Steurer's algorithm performs MMW iterations in an attempt to find a PSD matrix that is "δ-close" to satisfying all the inequalities. Steurer's analysis shows two things: that if indeed $\mathrm{SDP}(\mathcal{I}) \geq \alpha$ then a δ-close X will be found after some $T \coloneqq \mathrm{poly}(k, q, 1/\delta) \cdot \log m$ iterations; and, the existence of a δ-close X implies that $\mathrm{SDP}(\mathcal{I}) \geq \alpha - \mathrm{poly}(k, q) \cdot \delta^{1/4}$. Thus the ϵ-approximate decision problem can be solved with $O(\log m)$ MMW iterations by taking δ to be a constant of the form $\epsilon^4/\mathrm{poly}(k, q)$.

More details. Steurer's algorithm works in a slightly more general setting of CSPs than we need, with a *weighted* list of m *payoffs*, rather than an unweighted list of m predicates. Since we are only concerned approximating $\mathrm{SDP}(\mathcal{I})$ up to a constant $\pm\epsilon$, it is easy to see that we may assume the payoffs and weights are rationals expressible with $\log m + O(1)$ bits. The inputs to Steurer's algorithm are then as follows:

- V, the set of CSP variables, and Σ, the alphabet of cardinality q. We will write $n = |V \times \Sigma|$.
- m, the number of payoffs; their positive weights $w_1, \ldots, w_m$, summing to 1; the scopes $S_1, \ldots, S_m \subset V$ of the payoffs; k, the cardinality of each S_j; and, the payoffs themselves $\phi_1, \ldots, \phi_m : \Sigma^{S_j} \to [-1, 1]$. We may assume m and n are polynomially related.
- α, the target SDP value, and $\delta > 0$, a "closeness" parameter. Both are rational constants.

We also need the following notation:

- $d \in \mathbb{R}^V$ has d_i equal to the normalized degree of variable $i \in V$, i.e., $\frac{1}{k}\sum\{w_j : S_j \ni i\}$.
- $\overline{d} \in \mathbb{R}^n$ is the vector whose (i, a)-entry is d_i for all $a \in \Sigma$.
- $D = \mathrm{diag}(\overline{d})$. We may assume D's entries are rational with common denominator $M = O(m)$.
- $\triangle_D$ denotes the set of $n \times n$ PSD matrices X satisfying $D \bullet X = 1$.
- ρ is a certain "width" parameter, of the form $\mathrm{poly}(k, q, 1/\delta)$.

We now give the high-level outline of the MMW method as employed by Steurer (with slightly adjusted notation). It constructs certain $n \times n$ matrices Y_t, X_t, and uses the notation $Y_{<t} = \sum_{i<t} Y_i$:

Main Algorithm:

1. for $t = 1 \ldots T$ (where, recall, $T = \mathrm{poly}(k, q, 1/\delta) \cdot \log m$)
2. Compute $X_t = D^{-1/2} E\left(\frac{\delta}{4\rho^2} D^{-1/2} Y_{<t} D^{-1/2}\right) D^{-1/2}$ where $E(Z) \coloneqq \frac{\exp(Z)}{\mathrm{tr}(\exp(Z))}$.
3. Call $\textsc{Oracle}(X_t)$. If this returns "X_t is δ-close" then halt and output YES.
4. Otherwise it returns matrix A_t and scalar b_t with $A_t \bullet X_t \leq b_t - \delta$.
5. Set $Y_t = A_t - b_t D$.
6. Halt and output NO.

Here the subroutine ORACLE(X) will be described later; we mention that it expects $X \in \triangle_D$.

We remark that Steurer's algorithm is actually more complicated than the above because (in an effort to keep the total serial time $\widetilde{O}(m)$) it does not compute X_t explicitly; rather, it uses the (randomized) Johnson–Lindenstrauss Lemma to compute an approximate, low-dimensional Gram representation of X_t. As mentioned earlier, we will omit this step and instead keep the matrices X_t and $Y_{<t}$ explicitly.

Relying on some theorems concerning the "width" and "separation" of the subroutine ORACLE(X), Steurer proves the above Main Algorithm correct. However there is a complication; the algorithm is idealized and cannot be carried out exactly, due to precision issues involved with approximating $D^{-1/2}$ and the matrix exponential. Steurer sketches some details of how to handles this. Because we need to be very careful about our computational model, we provide full details in the next section.

The oracle. We now recall Steurer's ORACLE subroutine, which takes as input a matrix $X \in \triangle_D$. At the heart of this subroutine is a certain family of small linear programs, one for each payoff $j \in [m]$. Given such a j, let X_j denote the $kq \times kq$ PSD submatrix of X formed by the rows and columns associated to $S_j \times \Sigma$. Steurer considers the following dual linear programs:

$$\begin{aligned} \alpha_j &= \max_{\substack{\mu \text{ a distribution} \\ \text{on } \Sigma^{S_j}}} \left\{ \mathop{\mathbf{E}}_{\boldsymbol{x} \sim \mu} [\phi_j(\boldsymbol{x})] - C \cdot \|X_j - \mathfrak{M}(\mu)\|_\infty \right\} && \text{(primal)} \\ &= \min_{\Upsilon : \|\Upsilon\|_1 \le C} \{X_j \bullet \Upsilon + P(\Upsilon)\} && \text{(dual).} \end{aligned}$$

Here $C = 3/\delta$, $\|A\|_\infty$ (respectively $\|A\|_1$) denotes the maximum (respectively, sum) of the absolute values of A's entries, $\mathfrak{M}(\mu)$ is the matrix whose $((i,a),(i',a'))$-entry is $\sum\{\mu(x) : x_i = a, x_{i'} = a'\}$, and

$$P(\Upsilon) = \max_{x \in \Sigma^{S_j}} \left\{ \phi_j(x) - \sum_{i,i' \in S_j} \Upsilon_{(i,x_i),(i',x_{i'})} \right\}.$$

We may now give the ORACLE algorithm. (The reader may check that the below is equivalent to [Ste10, Figure 2].)

ORACLE(X):

1. Solve the dual LP above for α_j and the optimizing Υ_j, for each $j \in [m]$,
2. If $\sum_j w_j \alpha_j \le \alpha - \delta$, return $A \coloneqq \sum_j w_j \Upsilon_j$, $b \coloneqq \alpha - \sum_j w_t P(\Upsilon_j)$.
3. Let $s_0 = \overline{d}\,\overline{d}^\top \bullet X - 1$. If $|s_0| \ge \delta$, return $A \coloneqq -\mathrm{sgn}(s_0)\,\overline{d}\,\overline{d}^\top$, $b \coloneqq -\mathrm{sgn}(s_0)$.
4. Define $s \in \mathbb{R}^n$ by $s_{i,a} = \sum_{(i',a')} d_{i'} X_{(i,a),(i',a')} - X_{(i,a),(i,a)}$.
5. If $\sum_{(i,a)} |\overline{d}_{i,a} s_{i,a}| \ge \delta$, return $A \coloneqq -(\overline{d}\,\overline{d}^\top \circ S) - D \cdot \mathrm{diag}(\mathrm{sgn}(s))$, $b \coloneqq 0$, where $S \in \mathbb{R}^{n \times n}$ is defined by $S_{(i,a),(i',a')} = \frac{1}{2}(\mathrm{sgn}(s_{i,a}) + \mathrm{sgn}(s_{i',a'}))$.
6. Otherwise, return "X is δ-close".

4 An efficient parallel implementation

In this section we establish Theorem 2.2 by showing that Steurer's algorithm — as described in the previous section — can effectively be carried out by a parallel algorithm using $\mathrm{poly}(m)$ processors and $\mathrm{polylog}(m)$ parallel time.

Some precision issues. As mentioned, we do not carry out the Main Algorithm literally as written, due to the precision issues arising in Line 2. Instead, in each step of the main loop, our parallel algorithm computes a very good rational approximation $\widehat{X}_t$ to X_t. We will ensure this $\widehat{X}_t$ is always in $\triangle_D$ (precisely); thus it always serves as an acceptable argument for Oracle. If Oracle$(\widehat{X}_t)$ returns "$\widehat{X}_t$ is δ-close" then it is correct to output YES (as Steurer's analysis shows). Otherwise, it returns some A_t and b_t with $A_t \bullet \widehat{X}_t \leq b_t - \delta$. We will show that this implies $A_t \bullet X_t \leq b_t - (2/3)\delta$. It is easy to check that this is sufficient (after slightly adjusting the constant in the definition of T) for the algorithm to still find a δ-close X presuming that $\mathrm{SDP}(\mathcal{I}) \geq \alpha$.

By inspection, we see that the Oracle subroutine always returns a matrix A satisfying $\|A\|_1 \leq \mathrm{poly}(q, 1/\delta)$. (This uses $\|\Upsilon_j\|_1 \leq C = O(1/\delta)$, $\sum_j w_j = 1$, and $\sum_{(i,a)} \overline{d}_{i,a} = q$.) Therefore $|A_t \bullet X_t - A_t \bullet \widehat{X}_t| \leq \mathrm{poly}(q, 1/\delta) \cdot \|X_t - \widehat{X}_t\|_\infty$, which in turn is at most $\delta/3$ provided $\|X_t - \widehat{X}_t\|_\infty \leq 1/p(q, 1/\delta)$, where p denotes some sufficiently large polynomial function. It follows that we can retain correctness by replacing Line 2 of the Main Algorithm by

$$2'.\ \text{Compute a matrix } \widehat{X}_t \in \triangle_D \text{ satisfying } \|X_t - \widehat{X}_t\|_\infty \leq 1/p(q, 1/\delta).$$

In fact, we need to address another precision issue here concerning the rational matrix $Y_{<t}$. In order to keep the denominator sizes from increasingly too rapidly, it will be convenient to approximate the A_t and b_t returned by the oracle. We claim that if $\widehat{A}_t$ and $\widehat{b}_t$ agree with A_t and b_t to within $\pm 1/\mathrm{poly}(m)$, entrywise, then the resulting $\widehat{A}_t \bullet X_t - \widehat{b}_t$ will still be accurate to within $\delta/3$. As before, this is sufficient for the overall correctness of the algorithm (after slightly adjusting T). Since δ is a constant, it's clear that rounding b_t to within $\pm 1/\mathrm{poly}(m)$ is harmless. Regarding $|\widehat{A}_t \bullet X_t - A_t \bullet X_t|$, to see that $\pm 1/\mathrm{poly}(m)$ accuracy for A_t is sufficient we use the fact that X_t is PSD with $D \bullet X_t = 1$. Since D's (diagonal) entries are at least $1/O(m)$, we conclude that $\mathrm{tr}(X) \leq O(m)$ and hence $\|X_t\|_1 \leq \mathrm{poly}(m)$ (using the fact that X_t is PSD). Thus indeed it suffices to have $\|\widehat{A}_t - A_t\|_\infty \leq 1/\mathrm{poly}(m)$. As a consequence, for some fixed $M' = \mathrm{poly}(m)$ we can replace Line 5 of the Main Algorithm by

$$5'.\ \text{Set } Y_t = \widehat{A}_t - \widehat{b}_t D, \text{ where } \widehat{A}_t, \widehat{b}_t \text{ are rational approximations of } A_t, b_t \text{ with denominator } M'.$$

In particular, this means that the rational entries of Y_t and $Y_{<t}$ will always have common denominator MM', which has bit-length $O(\log m)$.

4.1 Parallel implementation with $\mathrm{poly}(m)$ processors and $\mathrm{polylog}(m)$ parallel time.

Preliminaries. We describe the algorithm in nonuniform fashion; we trust the reader to verify that it can be constructed by an $O(\log m)$-space machine. (This only requires various very simple things; e.g., memory management, simple calculations involving $O(\log m)$-bit integers, etc.) We will also frequently rely on the fact that NC is closed under composition. We remind the reader that the processors are treated as simple RAMs. Since integer multiplication is in NC, we may allow the RAMs to do basic arithmetic on registers containing rational numbers expressible with $\mathrm{polylog}(m)$ bits [SV84].

For storing the matrices Y_t, $Y_{<t}$, and D we will have a processor for each entry, as is standard. As discussed earlier, all of these will have a common denominator of $O(\log m)$ bits. The entries of D need to be computed at the outset of the algorithm; this is easily done with m processors in polylog(m) time using the parallel iterated-addition algorithm.

The Main Algorithm. The loop in Line 1 of the Main Algorithm only gives a multiplicative running time overhead of $O(\log m)$.

We now analyze Line 2′. Recall we have $Y_{<t}$ and D stored with a processor for each entry. We need to compute a $\pm 1/p(q, 1/\delta)$ approximation $\widehat{X}_t$ to $X_t = D^{-1/2} E\left(\frac{\delta}{4\rho^2} D^{-1/2} Y_{<t} D^{-1/2}\right) D^{-1/2}$, and it should also satisfy $\widehat{X}_t \in \triangle_D$. The techniques for this are standard and straightforward. The trace needed for E can be computed efficiently in parallel by iterated-addition. As noted in, e.g., [JJUW10], rational approximations square-roots and matrix exponentials (for matrices of polylogarithmic norm) can be computed even to poly(m) bits in NC, much more than we need. We need to check that the matrix exponential is "sufficiently continuous" that a good approximation in the input leads to a good approximation in the output. This follows from the method used to approximately compute the matrix exponential; namely, truncating the expansion $\exp(Z) = I + Z + Z^2/2 + Z^3/6 + \cdots$ to polylog(m) terms.

There is one more detail to take care of. We can get an initial approximation $\widehat{X}_t$ to X_t that is highly accurate; say, having $\|X_t - \widehat{X}_t\|_\infty \leq 1/m^c$ for any large constant c of our choice. This is much more accurate than the constant accuracy for $\widehat{X}_t$ we need. However, whereas $X_t \in \triangle_D$ by construction, we also wish to have $\widehat{X}_t \in \triangle_D$ precisely. This issue is straightforward to fix. We can first replace $\widehat{X}_t$ by $\frac{1}{2}(\widehat{X}_t + \widehat{X}_t^\top)$ to make it symmetric; this doesn't change its entrywise error. The matrix $\widehat{X}_t$ may still fail to be positive semidefinite; however since $\|\widehat{X}_t - X_t\|_\infty \leq 1/m^c$, we can fix this by adding some $1/\text{poly}(m)$ multiple of the identity matrix to $\widehat{X}_t$. This $1/\text{poly}(m)$ can be as small as we like, by taking c large. Thus again we only lose $\pm 1/\text{poly}(m)$ in entrywise accuracy. Finally, we can ensure $D \bullet \widehat{X}_t = 1$ exactly by dividing $\widehat{X}_t$ by $D \bullet \widehat{X}_t$; again, since D's entries are in $[1/O(m), 1]$, this will again only cost us $\pm 1/\text{poly}(m)$ in accuracy, taking the initial c large enough.

Line 3–4 of the Main Algorithm, the call to subroutine ORACLE, will be analyzed below. As we will see, the A_t and b_t it returns will involves rationals with $m \cdot \text{polylog}(m)$ bits; this is okay, however, by closure of FNC under composition.

Finally, the rounding in Line 5′ can certainly be done in NC, as can the updating of $Y_{<t}$.

The Oracle. We now show that ORACLE is in NC. The input is a matrix $\widehat{X}$, stored by n^2 processors, each entry being a rational number expressible with $O(\log m)$ bits. Thus each linear program that the oracle needs to solve has bit-complexity $O(\log m)$. Therefore we can devote m processors to these LPs, one for each constraint, and each processor can use a standard serial algorithm to solve its LP in polylog(m) time. Actually, we need to allow each processor polylog(m) registers for this, but that is not a problem. The resulting solutions $\alpha_j, \Upsilon_j, P(\Upsilon_j)$ will be rationals with polylog(m) bit-complexity.

In Line 2 of ORACLE$(\widehat{X})$ we first need to compute $\sum_j w_j \alpha_j$. This can be done in NC using iterated-addition and integer multiplication algorithms. Due to a potential lack of common denominators for the α_j's, the resulting quantity may require as many as $m \cdot \text{polylog}(m)$ bits; as mentioned before, however, this is not a problem. If we need to return A, b at this stage, note that again they can be computed via iterated-addition in polylog(m) time. Finally, lines 3, 4, 5 are similar; it's easy to see that $\overline{d}\,\overline{d}^\top$, s_0, s, and S can be computed in NC.

References

[AHK05] Sanjeev Arora, Elad Hazan, and Satyen Kale. Fast algorithms for approximate semidefinite programming using the multiplicative weights update method. In *Proceedings of the 46th Annual IEEE Symposium on Foundations of Computer Science*, pages 339–348, 2005. 2

[AHP13] Per Austrin, Johan Håstad, and Rafael Pass. On the power of many one-bit provers. pages 215–220. ACM, 2013. 1, 2

[AK07] Sanjeev Arora and Satyen Kale. A combinatorial, primal-dual approach to semidefinite programs. In *Proceedings of the 39th Annual ACM Symposium on Theory of Computing*, pages 227–236, 2007. 2

[BFL91] László Babai, Lance Fortnow, and Carsten Lund. Non-deterministic exponential time has two-prover interactive protocols. *Computational Complexity*, 1:3–40, 1991. 1

[EIO02] Lars Engebretsen, Piotr Indyk, and Ryan O'Donnell. Derandomized dimensionality reduction with applications. In *Proc. 13th ACM-SIAM Symposium on Discrete Algorithms*, pages 705–712, 2002. 2

[GH98] Oded Goldreich and Johan Håstad. On the complexity of interactive proofs with bounded communication. *Information Processing Letters*, 67:205–214, 1998. 1

[GMR89] Shafi Goldwasser, Silvio Micali, and Charles Rackoff. The knowledge complexity of interactive proofs. *SIAM Journal of Computing*, 18(1):186–208, 1989. 1

[GMW91] Oded Goldreich, Silvio Micali, and Avi Wigderson. Proofs that yield nothing but their validity or all languages in NP have zero-knowledge proof systems. *Journal of the ACM*, 38(3):691–729, 1991. 1

[GVW02] Oded Goldreich, Salil Vadhan, and Avi Wigderson. On interactive proofs with a laconic prover. *Computational Complexity*, 11(1–2):1–53, 2002. 1

[JJUW10] Rahul Jain, Zhengfeng Ji, Sarvagya Upadhyay, and John Watrous. QIP = PSPACE. *Communications of the ACM*, 53(12):102–109, 2010. 2, 4.1

[JUW09] Rahul Jain, Sarvagya Upadhyay, and John Watrous. Two-message quantum interactive proofs are in PSPACE. In *Proceedings of the 50th Annual IEEE Symposium on Foundations of Computer Science*, pages 534–543, 2009. 2

[JW09] Rahul Jain and John Watrous. Parallel approximation of non-interactive zero-sum quantum games. In *Proceedings of the 24th Annual Computational Complexity Conference*, pages 243–253, 2009. 2

[Kal07] Satyen Kale. *Efficient algorithms using the multiplicative weights update method.* PhD thesis, Princeton University, 2007. 2

[LFKN92] Carsten Lund, Lance Fortnow, Howard Karloff, and Noam Nisan. Algebraic methods for interactive proof systems. *Journal of the ACM*, 39(4):859–868, 1992. 1

[MM14] Konstantin Makarychev and Yury Makarychev. Approximation algorithm for non-Boolean Max-k-CSP. *Theory of Computing*, 10(13):341–358, 2014. 2

[Nis90] Noam Nisan. On read-once vs. multiple access to randomness in logspace. pages 179–184, 1990. 2

[Pap13] Periklis Papakonstantinou. Is uniform RNC contained in polylog space? Theoretical Computer Science Stack Exchange, 2013. http://cstheory.stackexchange.com/q/18538. 2

[Rag09] Prasad Raghavendra. *Approximating NP-hard problems: efficient algorithms and their limits*. PhD thesis, University of Washington, 2009. 2

[RS09] Prasad Raghavendra and David Steurer. How to round any CSP. In *Proceedings of the 50th Annual IEEE Symposium on Foundations of Computer Science*, pages 586–594. IEEE, 2009. 2

[Sha92] Adi Shamir. IP = PSPACE. *Journal of the ACM*, 39(4):869–877, 1992. 1

[Ste10] David Steurer. Fast SDP algorithms for constraint satisfaction problems. In *Proceedings of the 21st Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 684–697, 2010. 2, 3, 3

[SV84] Larry Stockmeyer and Uzi Vishkin. Simulation of parallel random access machines by circuits. *SIAM Journal on Computing*, 13(2):409–422, 1984. 2.2, 4.1

[Tan11] Kanat Tangwongsan. *Efficient Parallel Approximation Algorithms*. PhD thesis, Carnegie Mellon University, 2011. 2

[Zwi98] Uri Zwick. Approximation algorithms for constraint satisfaction problems involving at most three variables per constraint. In *Proceedings of the 9th Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 201–210, 1998. 2

ECCC ISSN 1433-8092

http://eccc.hpi-web.de