Space Hardness of Solving Structured Linear Systems

Xuangui Huang
Northeastern University, Boston, MA, USA
stslxg@ccs.neu.edu

Abstract

Space-efficient Laplacian solvers are closely related to derandomization of space-bound randomized computations. We show that if the probabilistic logarithmic-space solver or the deterministic nearly logarithmic-space solver for undirected Laplacian matrices can be extended to solve slightly larger subclasses of linear systems, then they can be used to solve all linear systems with similar space complexity. Previously Kyng and Zhang [12] proved such results in the time complexity setting using reductions between approximate solvers. We prove that their reductions can be implemented using constant-depth, polynomial-size threshold circuits.

2012 ACM Subject Classification Theory of computation → Problems, reductions and completeness

Keywords and phrases linear system solver, logarithmic space, threshold circuit

Digital Object Identifier 10.4230/LIPIcs.ISAAC.2020.56

Funding Supported by NSF CCF award 1813930.

Acknowledgements The author thanks Emanuele Viola for helpful discussions, and anonymous reviewers for comments.

1 Introduction

One of the oldest mathematical problems is to solve a linear system, that is to find a solution $\mathbf{x}$ satisfying $\mathbf{Ax} = \mathbf{b}$ given an $n \times n$ matrix $\mathbf{A}$ and a n-dimensional vector $\mathbf{b}$ as input. In the RealRAM model this can be done in $O(n^\omega)$ time, where $\omega \leq 2.3728$ [9] is the matrix multiplication constant. A slight generlization is to find an $\mathbf{x}$ that minimizes $\|\mathbf{Ax} - \mathbf{b}\|_2^2$ for $\mathbf{A} \in \mathbb{R}^{m \times n}$ and $b \in \mathbb{R}^m$, which is well-defined even when there is no solution to $\mathbf{Ax} = \mathbf{b}$.

We consider its approximate version, in which the goal of the solver is to find a solution $\mathbf{x}$ such that $\|\mathbf{Ax} - \mathbf{b}\|_2^2$ is close to $\min_{\mathbf{x}} \|\mathbf{Ax} - \mathbf{b}\|_2^2$. Much faster algorithms exist for approximately solving a special class of linear systems, the Laplacian of undirected graphs. Indeed recent breakthroughs showed that it can be done in nearly linear time in terms of the sparsity (number of non-zero entries) of $\mathbf{A}$ [20, 6]. An exciting work by Kyng and Zhang [12], which is a starting point of this paper, further showed that if such solvers can be extended to nearly linear time solvers for some classes slightly larger than undirected Laplacians, we can also solve general linear systems in nearly linear time.

In this paper we are interested in the space complexity of this problem. A probabilistic log-space algorithm for its decision version will solve a long standing open problem in space complexity [2]. A space-efficient algorithm for approximately solving Lapliacians implies space-efficient algorithms for approximating important statistics such as hitting times and escape probabilities on random walks, which is closely related to derandomization of space-bounded randomized computation [18]. Indeed, since approximating stationary probabilities on digraph with polynomial mixing time is complete for RL [17], it might be possible to extend recent space-efficient Laplacian solvers [15, 1] to derandomize log-space computation to prove that $\mathsf{RL} \subseteq \mathsf{TIME}(\widetilde{O}(\log n))$, which will be a significant step towards the RL vs. L problem.

Recently there have been several works on the space complexity of solving linear systems. For general linear systems Ta-shma gave a quantum algorithm using logarithmic space [21]. For undirected Laplacian, Doron et al showed that it has a probabilistic logarithmic-space algorithm [7] and hence a deterministic $O(\log^{3/2} n)$-space algorithm by a well-known space-efficient derandomization result by Saks and Zhou [19]. This was improved to $\widetilde{O}(\log n)$ by Murtagh et al [15] (and then by [8]), and later generalized to Eulerian Laplacian by Ahmadinejad et al [1]. It is natural to ask if these space-efficient solvers can be extended to larger classes of linear systems.

1.1 Our results

For matrix $\mathbf{A}$ and vector $\mathbf{b}$ we define their *entry size* $U(\mathbf{A}, \mathbf{b}) = \max(||\mathbf{A}||_{\max}, ||\mathbf{b}||_{\max}, 1/\min_+(\mathbf{A}), 1/\min_+(\mathbf{b}))$ as in [12], where $||\mathbf{A}||_{\max} = \max_{i,j} |\mathbf{A}_{i,j}|$, $\min_+(\mathbf{A}) = \min_{i,j:\mathbf{A}_{i,j}\neq 0} |\mathbf{A}_{i,j}|$, and similarly for $\mathbf{b}$. Note that their bit complexity w is just $O(\log U(\mathbf{A}, \mathbf{b}))$.

We prove a space hardness version of Kyng and Zhang's results [12], showing space hardness of approximate linear system solvers for some classes slightly larger than undirected Laplacians, namely multi-commodity Laplacians, 2D Truss Stiffness Matrices, and Total Variation Matrices.

▶ **Theorem 1.** *Suppose that for multi-commodity Laplacians, 2-D Truss Stiffness Matrices, or Total Variation Matrices, the linear system* $\mathbf{Ax} = \mathbf{b}$ *can be approximately solved in* $O(\log n + \log m + \log \kappa + \log \varepsilon^{-1} + \log U)$*-space, where* $\mathbf{A} \in \mathbb{R}^{m\times n}$ *and* $\mathbf{b} \in \mathbb{R}^m$*,* $U = U(\mathbf{A}, \mathbf{b})$*,* κ *is the condition number of* $\mathbf{A}$*, and* ε^{-1} *is the desired accuracy, then any linear system with polynomially bounded integers and condition number can be solved in* $O(\log n + \log m)$*-space with high accuracy.*

▶ Remark 2. Theorem 1 also works if we replace both $O(\cdot)$ by $\widetilde{O}(\cdot)$. Moreover, if the linear system solver for any of the three classes is a probabilistic one that only works with high probability or has space guarantees only in expectation over the random bits, then we will still get a probabilistic general linear system solver that works with high probability or has space guarantees in expectation, respectively.

This shows that if the probabilistic logspace solver for undirected Laplacian in [7], or the deterministic $\widetilde{O}(\log n)$-space one in [15], can be extended to solve any of these three slightly larger subclasses of linear systems, we would have a surprising result that all linear systems can be approximately solved in probabilistic logspace or in deterministic $\widetilde{O}(\log n)$-space, respectively. Pessimistically speaking, the theorem means that it is very hard to get space efficient algorithms for these subclasses, since it is as difficult as solving all linear systems in a space efficient way. On the bright side, we actually prove that any progress on solving these subclasses using less space will immediately translate into similar progress for solving all linear system using less space.

Kyng and Zhang [12] proved their results via reductions from approximate solvers of general linear systems to those of three subclasses. In this paper we prove Theorem 1 by proving that their reductions can be carried out in a space efficient manner. Indeed we prove a much stronger result that their reductions can be implemented in TC^0 circuits, which are constant-depth polynomial-size unbounded-fan in circuits with MAJORITY and $\neg$ gates. It shows that these reductions are actually highly parallelizable.

We denote $\mathcal{G}$ as the class of all matrices with integer valued entries. In the context of solving linear systems, an all-zero row or column can be trivially handled, so we can assume without loss of generality that matrices in $\mathcal{G}$ has at least one non-zero entry in every row and

column.[1] Finding an $\mathbf{x}$ that minimizes $\|\mathbf{Ax}-\mathbf{b}\|_2^2$ is equivalent to solving $\mathbf{A}^\top\mathbf{Ax}=\mathbf{A}^\top\mathbf{b}$, so as in [12] we are dealing with the so-called *incidence matrices* in this paper. For example, the undirected Laplacian can be written as $\mathbf{L}=\mathbf{A}^\top\mathbf{A}$, where $\mathbf{A}$ is an *undirected Laplacian incidence matrix*, so exactly solving $\mathbf{Lx}=\mathbf{c}$ is equivalent to exactly solving $\mathbf{Ax}=\mathbf{b}$ where $\mathbf{A}^\top\mathbf{b}=\mathbf{c}$. More interestingly, $\mathbf{A}$ can be viewed as coefficients of a set of equations of the form $x_i-x_j=0$ for $x_i,x_j\in X$, where X is the set of variables. As a generalization, for *2-commodity incidence matrices* $\mathcal{MC}_2$, we have variable sets X and Y of same size, and the equations are scalings of $x_i-x_j=0$, $y_i-y_j=0$, and $x_i-y_i-(x_j-y_j)=0$, where $x_i,x_j\in X$ and $y_i,y_j\in Y$. (For more details about these incidence matrices and the linear system approximation problem, please refer to [13].)

Our main technical result is the following theorem showing that the reduction from $\mathcal{G}$ to $\mathcal{MC}_2$ in [12] is TC^0-computable. As shown in [12], the reduction produces matrix $\mathbf{A}'$, vector $\mathbf{b}'$, desired accuracy ε'^{-1} such that the new size, entry size, condition number, desired accuracy are polynomial in the original size, entry size, condition number, and desired accuracy. Combining with the well-known result in circuit complexity that TC^0 circuit of size s can be simulated in space $O(\log s)$ [4], we can get the part of Theorem 1 for multi-commodity Laplacians.

▶ **Theorem 3.** *There is a TC^0-reduction from approximately solving $\mathcal{G}$ to approximately solving $\mathcal{MC}_2$. In particular, the sizes of the TC^0 circuits are polynomial in the size and entry size of the matrix $\mathbf{A}$ and vector $\mathbf{b}$ in the original linear system, the condition number of $\mathbf{A}$, and the desired accuracy ε^{-1}.*

In [12] it is shown that the matrix produced by this reduction is also a 2D Truss Stiffness Matrix as well as a Total Variation Matrix, therefore Theorem 3 also works for these classes. Hence we can also get the parts of Theorem 1 for these two classes. Also note that this reduction is a Karp-style reduction [3], i.e. it requires only one query to a linear system solver and will use the result in a black-box way. That is why Theorem 1 still applies if we only have a probabilistic solver that works with high probability or has space guarantees in expectation.

We also show TC^0-computability of the reductions in [12] to some more restrictive subclasses of $\mathcal{MC}_2$, including $\mathcal{MC}_2^{>0}$, the exact class we have to solve when we use Interior Point Methods for 2-commodity problems, as explained in the Kyng and Zhang's full paper [13]. They also showed that the promise problem of deciding whether a vector is in the image of a matrix or ε-far from the image can be directly reduced to approximately solving linear systems. Therefore by combining with the above results, this promise problem can be reduced in TC^0 to approximately solving the above-mentioned subclasses.

2 Simplified reductions in TC^0: the easy parts

Throughout this paper we use the sign-magnitude representation to encode a w-bit signed integer $z\in[-2^{w+1}+1,2^{w+1}-1]$ into a sign bit, with 0 for positive integers and 1 for negative ones, and w bits for $|z|$ in binary. Note that in this way 0 might have two representations. We accept both of them.

[1] It is worth noting that in the time complexity setting, e.g. in [20, 6, 12], the algorithms usually allow for sparse representation so the runtime is measure in terms of the sparsity of the matrix, $\mathsf{nnz}(\mathbf{A})$. However, for logarithm-space algorithms the difference between sparse and dense is not crucial, as the assumption here implies that $\log\mathsf{nnz}(\mathbf{A})=\Theta(\log n+\log m)$.

For simplicity we first prove the special case of Theorem 3, showing that there is a TC^0-reductions from exact solvers of $\mathcal{G}$ to exact solvers of $\mathcal{MC}_2$.

▶ **Theorem 4.** *Given an $m \times n$ w-bit matrix $\mathbf{A} \in \mathcal{G}$ and a vector $\mathbf{b}$ as input, we can compute in TC^0 an $O(nmw \log n) \times O(nmw \log n)$ $O(w \log n)$-bit matrix $\mathbf{A}' \in \mathcal{MC}_2$ and a vector $\mathbf{b}'$ such that:*

- *$\mathbf{Ax} = \mathbf{b}$ has a solution if and only if $\mathbf{A}'\mathbf{x}' = \mathbf{b}'$ has a solution;*
- *if $\mathbf{A}'\mathbf{x}' = \mathbf{b}'$ has a solution $\mathbf{x}'$, then we can compute $\mathbf{x}$ in TC^0 from $\mathbf{x}'$ such that $\mathbf{Ax} = \mathbf{b}$.*

Similar to [12], the proof of Theorem 4 is split into proofs of existence of several TC^0-reductions between exact solvers of the following intermediate classes of matrices.

▶ **Definition 5** ([12])**.**

- *Let $\mathcal{G}_z \subset \mathcal{G}$ denote the class of matrices with integer-valued entries such that every row has zero row sum;*
- *Let $\mathcal{G}_{z,2} \subset \mathcal{G}_z$ denote the class of matrices with integer-valued entries such that every row has zero row sum, and for each row the sum of positive coefficients is a power of 2.*

▶ **Lemma 6.** *There are TC^0-reductions for exact solvers of the following classes:*

(i) *from $\mathcal{G}$ to $\mathcal{G}_z$;*
(ii) *from $\mathcal{G}_z$ to $\mathcal{G}_{z,2}$;*
(iii) *from $\mathcal{G}_{z,2}$ to $\mathcal{MC}_2$.*

Item (iii) is the main reduction in this paper, which will be proved in the next section. In the remaining of this section we prove Lemma 6 (i) and (ii). For convenience the description of Lemma 6 is somewhat informal. Nonetheless, the formal descriptions of each reductions resemble closely to that of Theorem 4, and later in Theorem 15 we will also precisely describe what it means to say "we can compute in TC^0 a matrix $\mathbf{B}$ given a matrix $\mathbf{A}$".

From $\mathcal{G}$ to $\mathcal{G}_z$

Proof. Given a matrix $\mathbf{A} \in \mathcal{G}$, we can define a matrix $\mathbf{A}'$ with one more column by $\mathbf{A}' = \begin{pmatrix}\mathbf{A} & -\mathbf{A1}\end{pmatrix}$. Obviously $\mathbf{A}' \in \mathcal{G}_z$, and $\mathbf{Ax} = \mathbf{b}$ has a solution if and only if $\mathbf{A}'\mathbf{x}' = \mathbf{b}$ has a solution. We can also recover $\mathbf{x} \in \mathbb{R}^n$ from $\mathbf{x}' \in \mathbb{R}^{n+1}$ by taking the first n rows of $\mathbf{x}'$ and then subtracting each of them by $\mathbf{x}'_{n+1}$. The following results about additions imply that $\mathbf{A}'$ can be calculated in TC^0, and we can recover $\mathbf{x}'$ from $\mathbf{x}$ in AC^0. ◀

▷ Claim 7.

- Addition of 2 w-bit numbers has AC^0 circuit of size $\mathrm{poly}(w)$ (c.f. [5]);[2]
- Addition of n w-bit numbers has TC^0 circuit of size $\mathrm{poly}(n, w)$ [11].

From $\mathcal{G}_z$ to $\mathcal{G}_{z,2}$

Proof. Given an $m \times n$ w-bit signed-integer matrix $\mathbf{A}' \in \mathcal{G}_z$, we just need to add two more columns to make the sum of positive (and negative) entries in each row to the closest power of 2. This can be done in TC^0 in the following way. For each row $1 \le i \le m$, we calculate the sum of positive entries s_i by checking the sign bit then do the iterated addition in TC^0 by Claim 7. We then take $s = \max s_i$, which can be computed in AC^0 given s_i's. s has at most $O(w \log n)$ bits, hence by searching brute-forcely in AC^0 we can find the minimum k

[2] AC^0 circuits are constant-depth polynomial-size unbounded-fan in circuits with $\wedge$, $\vee$, and $\neg$ gates.

s.t. $2^k \geq s$. Therefore by Claim 7 we can calculate $a_i = 2^k - s_i$ in AC^0 given s_i. Then for each row i we add two new entries a_i and $-a_i$ to the end of $\mathbf{A}'$ to get $\mathbf{A}''$. Additionally we need to add a new row to $\mathbf{A}''$ and $\mathbf{b}''$ to zero out the last two variables we just added. For the new row of $\mathbf{A}''$, we set the last two entries into 1 and -1, and all other entries 0. For $\mathbf{b}''$ we set the new entry to be 0.

Obviously we have $\mathbf{A}'' \in \mathcal{G}_{z,2}$, and $\mathbf{A}'\mathbf{x}' = \mathbf{b}'$ has a solution iff $\mathbf{A}''\mathbf{x}'' = \mathbf{b}''$ has a solution. We can easily recover $\mathbf{x}'$ from $\mathbf{x}''$ by taking the first n rows. ◀

For the next section we need the following definition.

▶ **Definition 8.** *We say a matrix $\mathbf{A} \in \mathcal{G}_z$ is* w-bounded *if for each row the sum of positive coefficients is at most 2^w. Note that in such matrix every entry is a w-bit signed integer.*

Note that the reduction from $\mathcal{G}$ to $\mathcal{G}_z$ reduces an $m \times n$ w-bit matrix $\mathbf{A}$ into an $m \times (n+1)$ $O(w \log n)$-bounded matrix $\mathbf{A}'$, then the reduction from $\mathcal{G}_z$ to $\mathcal{G}_{z,2}$ reduces $\mathbf{A}'$ into an $(m+1) \times (n+3)$ $O(w \log n)$-bounded matrix $\mathbf{A}''$.

3 Simplified main reduction in TC^0

The main reduction in [12] uses the pair-and-replace scheme to transform a linear system in $\mathcal{G}_{z,2}$ to a 2-commodity equation system. The main idea is to use $\mathcal{MC}_2$`Gadget`s consisting of $\mathcal{MC}_2$ equations to replace pairs of variables in the original linear system round-by-round with respect to the bit representation of their coefficients. A simplified version of the gadget, implicitly given in [12], is as follows.

▶ **Definition 9** (Simplified $\mathcal{MC}_2$`Gadget`)**.** *Define $\mathcal{MC}_2\texttt{Gadget}(t, t', j_1, j_2)$ to be the following set of 2-commodity linear equations representing "$2x_t = x_{j_1} + x_{j_2}$":*

$$
\begin{aligned}
x_t - x_{t'+1} &= 0,\\
x_{t'+2} - x_{j_2} &= 0,\\
y_{t'+1} - y_{t'+3} &= 0,\\
y_{t'+4} - y_{t'+2} &= 0,\\
x_{t'+3} - x_{j_1} &= 0,\\
x_t - x_{t'+4} &= 0,\\
x_{t'+4} - y_{t'+4} - (x_{t'+3} - y_{t'+3}) &= 0,\\
x_{t'+1} - y_{t'+1} - (x_{t'+2} - y_{t'+2}) &= 0.
\end{aligned}
$$

For convenience we use an extra parameter t' to keep track of new variables that are only used in the gadgets.

Correctness of this gadget can be easily verified by summing up all the equations in it.

We now present a simplified reduction from exactly solving $\mathcal{G}_{z,2}$ to exactly solving $\mathcal{MC}_2$ in Algorithm 1. We use $\mathbf{A}_i$ to denote the i-th row of $\mathbf{A}$.

Note that in $\mathcal{MC}_2$ we have two input variable sets X, Y of the same size. That is why we have to multiply n' by 2 at last in the reduction. But here we will only use some variables in Y in the $\mathcal{MC}_2$`Gadget`s, and all the other Y variables are unused. For convenience we arrange the variables in the following way.

▶ Remark 10 (Arrangement of variables). In $\mathbf{B}$ we put all the X variables before all the Y variables. More importantly, we put those X variables that are only used in the gadgets behind all those $x_{n'+1}$ in Line 8. Equivalently it can be viewed as the following process. We

Algorithm 1 Simplified REDUCE$\mathcal{G}_{z,2}$TO$\mathcal{MC}_2$.

Input : a w-bounded $m \times n$ matrix $\mathbf{A} \in \mathcal{G}_{z,2}$ and $\mathbf{c} \in \mathbb{R}^n$.
Output : a w-bit $m' \times n'$ matrix $\mathbf{B} \in \mathcal{MC}_2$ and $\mathbf{d} \in \mathbb{R}^n$.

1 $m' \leftarrow m$, $n' \leftarrow n$
2 $n_g \leftarrow 0$ `// # of new variables used only in the` $\mathcal{MC}_2$`Gadgets`
3 **for** $i \leftarrow 1$ **to** m **do**
4 **for** $s \in \{+, -\}$ **do**
5 **for** $k \leftarrow 1$ **to** w **do**
6 **while** *strictly more than 1 entry in* $\mathbf{A}_i$ *has sign* s *and the* k*-th bit being* 1 **do**
7 Let j_1, j_2 be the first and second indices of such entries in $\mathbf{A}_i$
8 In $\mathbf{A}_i$, replace $2^k(x_{j_1} + x_{j_2})$ by $2^{k+1}x_{n'+1}$ (thus adding one column to the right of $\mathbf{A}$)
9 Add the coefficients of $\mathcal{MC}_2$`Gadget(`$n' + n_g + 1$, $n' + n_g + 1$, j_1, j_2`)` to $\mathbf{C}$
10 $n' \leftarrow n' + 1$
11 $n_g \leftarrow n_g + 4$
12 $m' \leftarrow m' + 8$
13 **end**
14 **end**
15 **end**
16 **end**
17 $n' \leftarrow 2 \times (n' + n_g)$
18 Stack $\mathbf{C}$ in the bottom of $\mathbf{A}$ and fill in 0's to get $\mathbf{B}$
19 Add $m' - m$ 0's under $\mathbf{c}$ to get $\mathbf{d}$

first run Algorithm 1 virtually before Line 17 to get n', which is the number of X variables ignoring those only used in the gadgets. Then we run it again on the original input, starting with n_g being this value, and in Line 9 we use $\mathcal{MC}_2$`Gadget`$(n' + 1, n_g, j_1, j_2)$ instead.

We give an example showing how the reduction works under this arrangement.

▶ **Example 11.** We show how the reduction runs on $3x_1 + 5x_2 + x_3 + 7x_4 - 16x_5 = 0$:

$$
\begin{aligned}
& 00011x_1 + 00101x_2 + 00001x_3 + 00111x_4 - 10000x_5 = 0 \\
\xrightarrow{x_1+x_2-2x_6=0}\ & 00010x_1 + 00100x_2 + 00001x_3 + 00111x_4 + 00010x_6 - 10000x_5 = 0 \\
\xrightarrow{x_3+x_4-2x_7=0}\ & 00010x_1 + 00100x_2 + 00110x_4 + 00010x_6 + 00010x_7 - 10000x_5 = 0 \\
\xrightarrow{x_1+x_4-2x_8=0}\ & 00100x_2 + 00100x_4 + 00010x_6 + 00010x_7 + 00100x_8 - 10000x_5 = 0 \\
\xrightarrow{x_6+x_7-2x_9=0}\ & 00100x_2 + 00100x_4 + 00100x_8 + 00100x_9 - 10000x_5 = 0 \\
\xrightarrow{x_2+x_4-2x_{10}=0}\ & 01000x_{10} + 00100x_8 + 00100x_9 - 10000x_5 = 0 \\
\xrightarrow{x_8+x_9-2x_{11}=0}\ & 01000x_{10} + 01000x_{11} - 10000x_5 = 0 \\
\xrightarrow{x_{10}+x_{11}-2x_{12}=0}\ & 10000x_{12} - 10000x_5 = 0.
\end{aligned}
$$

For simplicity in this example we are only eliminating the positive coefficient variables. In the first round we use new variables (and the corresponding gadgets) x_6 and x_7 to eliminate the first bit, getting the equation $2x_1 + 4x_2 + 6x_4 + 2x_6 + 2x_7 = 0$. These two generated

variables are then eliminated in the second round by x_9, in addition to x_8 for the second bit. In the third round we use x_{10} and x_{11}. Finally in the fourth round we use x_{12} and get the equation after the reduction $16x_{12} - 16x_5 = 0$, with $\mathcal{MC}_2$Gadgets representing $2x_6 = x_1 + x_2$, $2x_7 = x_3 + x_4$, $2x_8 = x_1 + x_4$, $2x_9 = x_6 + x_7$, $2x_{10} = x_2 + x_4$, $2x_{11} = x_8 + x_9$, and $2x_{12} = x_{10} + x_{11}$.

Correctness of this simplified reduction follows easily by correctness of the gadget, since our transformation preserves the original solution. Moreover, given a solution $\mathbf{x}^*$ such that $\mathbf{B}\mathbf{x}^* = \mathbf{d}$ where $\mathbf{B}$ and $\mathbf{d}$ are obtained from running Algorithm 1 on input $\mathbf{A}$ and $\mathbf{c}$, we can easily get the solution to the original equation system $\mathbf{A}\mathbf{x} = \mathbf{c}$ by taking the first n elements in $\mathbf{x}^*$. It is also easy to get $\mathbf{d}$ from $\mathbf{c}$ if we can calculate m' in TC^0.

In the remaining of this section we are going to prove that Algorithm 1 can be implemented in TC^0. For $1 \le i \le m$ and $1 \le k \le w$, we define

$$\begin{aligned}
\mathtt{Len}_i^+(\mathbf{A}) &= \text{binary length of the sum of positive coefficients in } \mathbf{A}_i,\\
\mathtt{CountBit}_{i,k}^+(\mathbf{A}) &= \#\text{ of positive coefficient variables in the original } \mathbf{A}_i\\
&\qquad \text{with the } k\text{-th bit being 1},\\
\mathtt{NumGadget}_{i,k}^+(\mathbf{A}) &= \#\text{ of } \mathcal{MC}_2\text{GADGETs used for } \mathbf{A}_i \text{ to eliminate the } k\text{-th bit}\\
&\qquad \text{of positive coefficient variables},
\end{aligned}$$

and similarly $\mathtt{Len}_i^-(\mathbf{A})$, $\mathtt{CountBit}_{i,k}^-(\mathbf{A})$, $\mathtt{NumGadget}_{i,k}^-(\mathbf{A})$ for the negative coefficient variables.

▶ **Example 12.** For the above example, we have $\mathtt{Len}_i^+(\mathbf{A}) = 5$, and the values of $\mathtt{CountBit}_{i,k}^+(\mathbf{A})$ and $\mathtt{NumGadget}_{i,k}^+(\mathbf{A})$ for each k are shown in Table 1.

Table 1 Values of $\mathtt{CountBit}_{i,k}^+(\mathbf{A})$ and $\mathtt{NumGadget}_{i,k}^+(\mathbf{A})$ for Example 11.

k	1	2	3	4	5
$\mathtt{CountBit}_{i,k}^+(\mathbf{A})$	4	2	2	0	0
$\mathtt{NumGadget}_{i,k}^+(\mathbf{A})$	2	2	2	1	0

We have the following simple but crucial properties for these values.

▷ Claim 13.

(i) $\mathtt{Len}_i^s(\mathbf{A}) \le w$ for all $s \in \{+,-\}$ and $1 \le i \le m$;

(ii) $\mathtt{CountBit}_{i,k}^s(\mathbf{A}) \le n$ for all $s \in \{+,-\}$, $1 \le i \le m$, and $1 \le k \le w$;

(iii) For all $s \in \{+,-\}$ and $1 \le i \le m$, we have

$$\mathtt{NumGadget}_{i,k}^s(\mathbf{A}) = \begin{cases} 2^{-(k+1)} \sum_{k'=1}^{k} 2^{k'} \mathtt{CountBit}_{i,k'}^s(\mathbf{A}) & \text{for } 1 \le k \le \mathtt{Len}_i^s(\mathbf{A}) - 1,\\ 0 & \text{for } \mathtt{Len}_i^s(\mathbf{A}) \le k \le w, \end{cases}$$

thus $\mathtt{NumGadget}_{i,k}^s(\mathbf{A}) \le O(n)$;

(iv) $m' = m + 8\sum_{i=1}^{m}\sum_{s\in\{+,-\}}\sum_{k=1}^{w} \mathtt{NumGadget}_{i,k}^s(\mathbf{A}) \le O(nmw)$;

(v) $n' = 2n + 10\sum_{i=1}^{m}\sum_{s\in\{+,-\}}\sum_{k=1}^{w} \mathtt{NumGadget}_{i,k}^s(\mathbf{A}) \le O(nmw)$.

Proof. (i) and (ii) are trivial by definition. (iv) and (v) are straightforward from Algorithm 1 and (iii). For (iii), note that in each round for k we will eliminate all the variables generated in the previous round for $k-1$ by construction (ignoring those variables that are only used in the gadgets), therefore we have

$$\texttt{NumGadget}^s_{i,k}(\mathbf{A}) = \begin{cases} \frac{1}{2}\texttt{CountBit}^s_{i,1}(\mathbf{A}) & \text{for } k = 1, \\ \frac{1}{2}(\texttt{CountBit}^s_{i,k}(\mathbf{A}) + \texttt{NumGadget}^s_{i,k-1}(\mathbf{A})) & \text{for } 2 \le k \le \texttt{Len}^s_i(\mathbf{A}) - 1. \end{cases}$$

Here we rely on the property that the sum of positive (and negative) coefficients in each row is a power of 2 to ensure that $\texttt{NumGadget}^s_{i,k}(\mathbf{A})$ as calculated in this way are always integers. Then by induction we get the formula. ◁

Note that in (iii) multiplying by $2^{-(k+1)}$ and $2^{k'}$ are just right and left shifts, which can be easily implemented in the circuit model. Combining Claim 13 and Claim 7, we can see that all of these values can be computed in TC^0 for all i, k, s, i.e. the depths of the TC circuits are absolute constants independent of i, k, and s.

▶ **Lemma 14.**
- $\texttt{Len}^s_i$, $\texttt{CountBit}^s_{i,k}$, *and* $\texttt{NumGadget}^s_{i,k}$ *have* TC^0 *circuits of size* $\mathrm{poly}(n, w)$ *for all* i, k, s*;*
- n' *and* m' *have* TC^0 *circuits of size* $\mathrm{poly}(n, m, w)$*.*

Now we will prove that Algorithm 1 (with the changes specified in Remark 10) can be implemented in TC^0 by the following theorem. Combining with correctness of Algorithm 1, we get Lemma 6 (iii). We represent the input matrix $\mathbf{A}$ explicitly by giving all its entries in sign-magnitude form, and the output matrix $\mathbf{B}$ implicitly by outputting the entry of $\mathbf{B}$ in row i column j with indices i,j as additional inputs.

▶ **Theorem 15.** *There is a* TC^0 *circuit family* $\{C_{n,m,w}\}_{n,m,w\in\mathbb{N}}$ *of size* $\mathrm{poly}(n, m, w)$ *with* $(O(nmw) + O(\log nmw))$*-bit input and* $O(w)$*-bit output such that:*
- *the first* $O(nmw)$ *bits of input represent a* w*-bounded* $m \times n$ *matrix* $\mathbf{A} \in \mathcal{G}_{z,2}$ *in sign-magnitude form, the next* $O(\log nmw)$ *bits of input represent the row index* i*, and the last* $O(\log nmw)$ *bits represent the column index* j*;*
- *for* $i \le m'$ *and* $j \le n'$ *where* $m', n' \le O(nmw)$ *is calculated as in Algorithm 1 on* $\mathbf{A}$*, the output represents the entry of* $\mathbf{B}$ *(in sign-magnitude form) in row* i *column* j *as calculated by Algorithm 1 on* $\mathbf{A}$*, otherwise it represents* 0*.*

Proof. Consider the i-th row of $\mathbf{B}$, we need to know the equation which it corresponds to. Because we put those equations of $\mathcal{MC}_2\texttt{Gadget}$s behind the modified equations of $\mathbf{A}$, we have two cases:
- $i \le m$: in this case, the equation is $\mathbf{A}_i$ after the transformation, which has the form $C(x_{j_+} - x_{j_-}) = \mathbf{c}_i$ with $C = \underbrace{100\cdots0}_{\texttt{Len}^+_i(\mathbf{A})\text{ bits}}$ in binary since our transformation does not change the sum of positive (and negative) coefficients in $\mathbf{A}$. What remains is to calculate the indices j_+ and j_- in TC^0.
  For all $s \in \{+, -\}$ and $1 \le i \le m$, let

  $$\texttt{SumNumG}^s_i(\mathbf{A}) = \sum_{k=1}^{w} \texttt{NumGadget}^s_{i,k}.$$

  For each $s \in \{+, -\}$, there are two cases:
  - No new variable is added, i.e. $\texttt{SumNumG}^s_i(\mathbf{A}) = 0$: there is only one entry in the original $\mathbf{A}_i$ with the corresponding sign thus j_s is the index of this entry. We search for this entry to get its index, which can be implemented in AC^0 because there are n possibilities.

- Some new variables are added: then j_s is the index of the last added new variable, ignoring those only in $\mathcal{MC}_2\texttt{Gadget}$s. It can be calculated by

$$j_s = \begin{cases} n + \texttt{SumNumG}_i^+(\mathbf{A}) + \sum_{i'=1}^{i-1} \sum_{s' \in \{+,-\}} \texttt{SumNumG}_{i'}^{s'}(\mathbf{A}) & \text{if } s = +, \\ n + \sum_{i'=1}^{i} \sum_{s' \in \{+,-\}} \texttt{SumNumG}_{i'}^{s'}(\mathbf{A}) & \text{if } s = -. \end{cases}$$

$\texttt{SumNumG}_i^s \in \mathsf{TC}^0$ by Claim 7 and Lemma 14 so both j_+, j_- can be calculated in TC^0 by Claim 7. Therefore we can calculate the entries in row i in TC^0 for $i \le m$.

- $i > m$: in this case, this equation is in an $\mathcal{MC}_2\texttt{Gadget}(t, t', j_1, j_2)$ gadget for some t, t', j_1, j_2. We need to calculate t, t', j_1, and j_2 in TC^0 then it is easy to recover the equation from the offset in the gadget.
We can first calculate in AC^0 the gadget's index

$$\mathsf{ind} = \left\lfloor \frac{i - m - 1}{8} \right\rfloor + 1,$$

as $\lfloor \frac{\cdot}{8} \rfloor$ is just ignoring the three least significant bits. By Algorithm 1 with the changes to the way we use $\mathcal{MC}_2\texttt{Gadget}(t, t', j_1, j_2)$ as specified by Remark 10, we know $t = n + \mathsf{ind}$ thus it is AC^0-computable, and by Claim 7 we can also calculate in TC^0

$$t' = n + \sum_{i'=1}^{m} \sum_{s' \in \{+,-\}} \texttt{SumNumG}_{i'}^{s'}(\mathbf{A}) + 4(\mathsf{ind} - 1).$$

Then we can calculate the number i', k', sign $s' \in \{+, -\}$, and number ℓ' such that this gadget is the ℓ'-th gadget we created to eliminate the k'-th bit of $\mathbf{A}_{i'}$ for variables with sign s'. More specifically, we want to find the minimum i', k', s' (where we treat "+" < "−") such that

$$\texttt{PrefixSum}_{i',k'}^+(\mathbf{A}) < \mathsf{ind} \le \texttt{PrefixSum}_{i',k'}^+(\mathbf{A}) + \texttt{NumGadget}_{i',k'}^+ \text{ if } s' = +,$$
$$\texttt{PrefixSum}_{i',k'}^-(\mathbf{A}) < \mathsf{ind} \le \texttt{PrefixSum}_{i',k'}^-(\mathbf{A}) + \texttt{NumGadget}_{i',k'}^- \text{ if } s' = -,$$

where

$$\texttt{PrefixSum}_{i',k'}^+(\mathbf{A}) = \sum_{i''=1}^{i'-1} \sum_{s'' \in \{+,-\}} \texttt{SumNumG}_{i''}^{s''}(\mathbf{A}) + \sum_{k''=1}^{k'-1} \sum_{s'' \in \{+,-\}} \texttt{NumGadget}_{i',k''}^{s''}(\mathbf{A}),$$
$$\texttt{PrefixSum}_{i',k'}^-(\mathbf{A}) = \texttt{PrefixSum}_{i',k'}^+(\mathbf{A}) + \texttt{NumGadget}_{i',k'}^+.$$

There are $O(mw)$ possible choices for i', k', s' and each condition is a prefix sum of $\texttt{NumGadget}_{i,k}^s$'s, therefore this can be done in TC^0 by a parallel comparison of ind to the prefix sums of $\texttt{NumGadget}_{i,k}^s$'s. After getting i', k', and s', by Claim 7 we can calculate

$$\ell' = \mathsf{ind} - \texttt{PrefixSum}_{i',k'}^{s'}.$$

What remains is to calculate j_1 and j_2 from i', k', s', and ℓ'. Note that when eliminating the k'-th bit of $\mathbf{A}_{i'}$ for variables with sign s', we first eliminate in pairs those variables in the original $\mathbf{A}_{i'}$ that have 1 in the k'-th bit before the reduction (we call them *original pairs*), then eliminate in pairs those generated in the previous round for i', $k' - 1$, and s'. The number of original pairs is given by $p = \lfloor \texttt{CountBit}_{i',k'}^{s'}(\mathbf{A})/2 \rfloor$, computable in TC^0. There are two cases:

- $\ell' \le p$: j_1 and j_2 is the indices of variables in the ℓ'-th original pairs, which are the indices of the $(2\ell'-1)$-th and $2\ell'$-th variables in $\mathbf{A}_{i'}$ that have 1 in the k'-th bit. Similarly as above, this can be done by a simple parallel comparison to prefix sums of the k'-th bits of variables in $\mathbf{A}_{i'}$.
- $\ell' > p$: then j_1 and j_2 are the $(2(\ell'-p)-1)$-th and $2(\ell'-p)$-th new variables generated in the previous round, therefore

$$j_1 = \texttt{PrefixSum}^{s'}_{i',k'-1}(\mathbf{A}) + 2(\ell'-p) - 1 \text{ and } j_2 = j_1 + 1.$$

However the second case above only works for even $\texttt{CountBit}^{s'}_{i',k'}(\mathbf{A})$. When it is odd, $\ell' = p+1$ corresponds to a pair with the last original variable in this round and the first generated variables in the previous round, thus j_1 can be calculated as in the first case and $j_2 = \texttt{PrefixSum}^{s'}_{i',k'-1}(\mathbf{A}) + 1$. Then for $\ell' > p+1$, we have

$$j_1 = \texttt{PrefixSum}^{s'}_{i',k'-1}(\mathbf{A}) + 2(\ell'-p) - 2 \text{ and } j_2 = j_1 + 1.$$

In conclusion, given row index i as input, we can first check in TC^0 if $i \le m'$ by Lemma 14. If so, we can use the above procedure to compute in TC^0 coefficients of $\mathbf{B}_i$. Then with column index j as input, we can check in TC^0 if $j \le n'$ and if so compute in TC^0 the entry of $\mathbf{B}$ in row i column j. ◀

4 Genearlization to approximate solvers, and more restrictive classes

Now we are going to generalize the above results of reductions between exact solvers into those between approximate solvers, thus proving Theorem 3. First we need the following result showing the power of TC^0.

▷ Claim 16. Division and iterated multiplication are in DLOGTIME-uniform TC^0 [10]. Moreover, we can approximate in TC^0 functions represented by sufficiently nice power series, such as log, exp, and $x^{1/k}$ [16, 14, 10].

Proof of Theorem 3. Based on our proof of Theorem 4, we are going to prove that the original reductions from $\mathcal{G}$ to $\mathcal{MC}_2$ in [12] can be computed in TC^0. As shown in [12], each step of the reduction produces matrix $\mathbf{A}'$, vector $\mathbf{b}'$, desired accuracy ε'^{-1} such that the new size, entry size, condition number, desired accuracy are polynomial in the original size, entry size, condition number, and desired accuracy.

- **From $\mathcal{G}$ to $\mathcal{G}_z$**: this reduction remains the same, but now knowing the accuracy we can recover $\mathbf{x}'$ from $\mathbf{x}$ using fix-point arithmetic in TC^0.
- **From $\mathcal{G}_z$ to $\mathcal{G}_{z,2}$**: the only difference (besides the calculation of accuracy) is that in the original reduction in [13, Section 7.2], in the last row of the reduced matrix $\mathbf{A}'$ the last two entries are set to be w and $-w$ for some value w, instead of 1 and -1 as in our simplified version. Nevertheless, the value w is also computable in TC^0 by Claim 16 so the reduction is still computable in TC^0.
- **From $\mathcal{G}_{z,2}$ to $\mathcal{MC}_2$**: for approximate solvers we will have to use the original $\mathcal{MC}_2\texttt{Gadget}$ in [13, Section 4] consisting of ten 2-commodity equations and with additional variables. So we need to modify the corresponding numbers in our calculation. In particular the gadget's index becomes

$$\mathsf{ind} = \left\lfloor \frac{i-m-1}{10} \right\rfloor + 1,$$

which is still TC^0-computable. Besides, the equations in the $\mathcal{MC}_2\texttt{Gadget}$ for eliminating the k-th bit of variables with sign s in $\mathbf{A}_i$ will be multiplied by $-s \cdot 2^k \cdot w_i$, where

$$w_i = \sqrt{10 \sum_{s' \in \{+,-\}} \texttt{SumNumG}_i^s(\mathbf{A})},$$

as specified in Algorithm 1 of [13]. These multiplicative factors can be calculated within desired accuracy in TC^0 by Claim 16. Therefore the reduction is still computable in TC^0. Additionally in the second and third reduction, when recovering $\mathbf{x}'$ from $\mathbf{x}$ we need to check if the original matrix $\mathbf{A}$ and vector $\mathbf{b}$ satisfy $\mathbf{A}^\top \mathbf{b} = \mathbf{0}$ and simply return $\mathbf{0}$ if so. This can be done in TC^0 by Claim 7.

Therefore by composing these three steps, we can reduce the problem of approximately solving equations in $\mathcal{G}$ to approximately solving equations in $\mathcal{MC}_2$ in TC^0 with size polynomial in the original size, entry size, condition number, and desired accuracy. ◀

In [12] they also considered some more restrictive subclasses of $\mathcal{MC}_2$. Intuitively, the set of *strict 2-commodity matrices* $\mathcal{MC}_2^{>0} \subset \mathcal{MC}_2$ is the class of 2-commodity equations $\mathbf{A}$ such that for every pair i, j, equation $x_i - x_j = 0 \in \mathbf{A}$ iff equation $y_i - y_j = 0 \in \mathbf{A}$ iff equation $x_i - y_i - (x_j - y_j) = 0 \in \mathbf{A}$. The set of strict 2-commodity matrices with integer entries is denoted by $\mathcal{MC}_{2,\mathbb{Z}}^{>0}$. They showed that reductions from approximately solving $\mathcal{MC}_2$ to approximately solving $\mathcal{MC}_2^{>0}$, and from $\mathcal{MC}_2^{>0}$ to $\mathcal{MC}_{2,\mathbb{Z}}^{>0}$. We are going to show that these reductions can be computed in TC^0.

From $\mathcal{MC}_2$ to $\mathcal{MC}_2^{>0}$

Proof. The reduction, as defined in [13, Section 5.1], runs by checking for each pair i, j that are involved in some equation in $\mathbf{A}$ if any of the three types of equations is missing and add it if so. The added equations will be multiplied by a factor that is computable in TC^0. Obviously the resulting equation systems is in $\mathcal{MC}_2^{>0}$. It is easy to see that the number of added equations for each pair i, j can be computed in AC^0, thus all the prefix sums of these numbers can be calculated in TC^0 simultaneously, and so we can determine the equations in the reduced instance in TC^0. ◀

From $\mathcal{MC}_2^{>0}$ to $\mathcal{MC}_{2,\mathbb{Z}}^{>0}$

Proof. In [13, Section 6] it is done by scaling up all the numbers in the matrix and the vector by a factor of 2^k, where k is computable in TC^0 by Claim 16, then take the ceiling function on entries of the matrix to convert them into integer entries, which also can be done in TC^0. ◀

References

1 AmirMahdi Ahmadinejad, Jonathan A. Kelner, Jack Murtagh, John Peebles, Aaron Sidford, and Salil P. Vadhan. High-precision estimation of random walks in small space. *CoRR*, abs/1912.04524, 2019. arXiv:1912.04524.

2 Eric Allender, Robert Beals, and Mitsunori Ogihara. The complexity of matrix rank and feasible systems of linear equations. *Comput. Complex.*, 8(2):99–126, 1999. doi:10.1007/s000370050023.

3 Sanjeev Arora and Boaz Barak. *Computational Complexity - A Modern Approach*. Cambridge University Press, 2009. URL: http://www.cambridge.org/catalogue/catalogue.asp?isbn=9780521424264.

4 Allan Borodin. On relating time and space to size and depth. *SIAM Journal on Computing*, 6(4):733–744, 1977. doi:10.1137/0206054.

5 Peter Clote and Evangelos Kranakis. *Boolean Functions and Computation Models*. Texts in Theoretical Computer Science. An EATCS Series. Springer, 2002. doi:10.1007/978-3-662-04943-3.

6 Michael B. Cohen, Rasmus Kyng, Gary L. Miller, Jakub W. Pachocki, Richard Peng, Anup B. Rao, and Shen Chen Xu. Solving SDD linear systems in nearly $m\log^{1/2}n$ time. In *Symposium on Theory of Computing, STOC 2014, New York, NY, USA, May 31 - June 03, 2014*, pages 343–352, 2014. doi:10.1145/2591796.2591833.

7 Dean Doron, François Le Gall, and Amnon Ta-Shma. Probabilistic logarithmic-space algorithms for laplacian solvers. In *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques, APPROX/RANDOM 2017, August 16-18, 2017, Berkeley, CA, USA*, pages 41:1–41:20, 2017. doi:10.4230/LIPIcs.APPROX-RANDOM.2017.41.

8 Dean Doron, Jack Murtagh, Salil P. Vadhan, and David Zuckerman. Spectral sparsification via bounded-independence sampling. *Electronic Colloquium on Computational Complexity (ECCC)*, 27:26, 2020. URL: https://eccc.weizmann.ac.il/report/2020/026.

9 François Le Gall. Powers of tensors and fast matrix multiplication. In *International Symposium on Symbolic and Algebraic Computation, ISSAC '14, Kobe, Japan, July 23-25, 2014*, pages 296–303, 2014. doi:10.1145/2608628.2608664.

10 William Hesse, Eric Allender, and David A. Mix Barrington. Uniform constant-depth threshold circuits for division and iterated multiplication. *J. Comput. Syst. Sci.*, 65(4):695–716, 2002. doi:10.1016/S0022-0000(02)00025-9.

11 Neil Immerman and Susan Landau. The complexity of iterated multiplication. In *Proceedings: Fourth Annual Structure in Complexity Theory Conference, University of Oregon, Eugene, Oregon, USA, June 19-22, 1989*, pages 104–111, 1989. doi:10.1109/SCT.1989.41816.

12 Rasmus Kyng and Peng Zhang. Hardness results for structured linear systems. In *58th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2017, Berkeley, CA, USA, October 15-17, 2017*, pages 684–695, 2017.

13 Rasmus Kyng and Peng Zhang. Hardness results for structured linear systems. *CoRR*, abs/1705.02944, 2017. arXiv:1705.02944.

14 Alexis Maciel and Denis Thérien. Efficient threshold circuits for power series. *Inf. Comput.*, 152(1):62–73, 1999. doi:10.1006/inco.1998.2783.

15 Jack Murtagh, Omer Reingold, Aaron Sidford, and Salil P. Vadhan. Derandomization beyond connectivity: Undirected laplacian systems in nearly logarithmic space. In *58th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2017, Berkeley, CA, USA, October 15-17, 2017*, pages 801–812, 2017. doi:10.1109/FOCS.2017.79.

16 John H. Reif and Stephen R. Tate. On threshold circuits and polynomial computation. *SIAM J. Comput.*, 21(5):896–908, 1992. doi:10.1137/0221053.

17 Omer Reingold, Luca Trevisan, and Salil P. Vadhan. Pseudorandom walks on regular digraphs and the RL vs. L problem. In Jon M. Kleinberg, editor, *Proceedings of the 38th Annual ACM Symposium on Theory of Computing, Seattle, WA, USA, May 21-23, 2006*, pages 457–466. ACM, 2006. doi:10.1145/1132516.1132583.

18 Michael E. Saks. Randomization and derandomization in space_bounded computation. In Steven Homer and Jin-Yi Cai, editors, *Proceedings of the Eleveth Annual IEEE Conference on Computational Complexity, Philadelphia, Pennsylvania, USA, May 24-27, 1996*, pages 128–149. IEEE Computer Society, 1996. doi:10.1109/CCC.1996.507676.

19 Michael E. Saks and Shiyu Zhou. RSPACE(S) \subseteq dspace($s^{3/2}$). In *36th Annual Symposium on Foundations of Computer Science, Milwaukee, Wisconsin, USA, 23-25 October 1995*, pages 344–353, 1995. doi:10.1109/SFCS.1995.492490.

20 Daniel A. Spielman and Shang-Hua Teng. Nearly linear time algorithms for preconditioning and solving symmetric, diagonally dominant linear systems. *SIAM J. Matrix Analysis Applications*, 35(3):835–885, 2014. doi:10.1137/090771430.

21 Amnon Ta-Shma. Inverting well conditioned matrices in quantum logspace. In *Symposium on Theory of Computing Conference, STOC'13, Palo Alto, CA, USA, June 1-4, 2013*, pages 881–890, 2013. doi:10.1145/2488608.2488720.