A Strongly Polynomial Algorithm for Linear Programs with At Most Two Nonzero Entries per Row or Column*

Daniel Dadush
Centrum Wiskunde & Informatica
Amsterdam, Netherlands
dadush@cwi.nl

Zhuan Khye Koh
Centrum Wiskunde & Informatica
Amsterdam, Netherlands
zhuan.koh@cwi.nl

Bento Natura
Georgia Institute of Technology
Atlanta, USA
bento.natura@isye.gatech.edu

Neil Olver
London School of Economics and Political Science
London, United Kingdom
n.olver@lse.ac.uk

László A. Végh
London School of Economics and Political Science
London, United Kingdom
l.vegh@lse.ac.uk

ABSTRACT

We give a strongly polynomial algorithm for minimum cost generalized flow, and hence for optimizing any linear program with at most two non-zero entries per row, or at most two non-zero entries per column. Primal and dual feasibility were shown by Végh (MOR '17) and Megiddo (SICOMP '83), respectively. Our result can be viewed as progress towards understanding whether all linear programs can be solved in strongly polynomial time, also referred to as Smale's 9th problem.

Our approach is based on the recent primal-dual interior point method (IPM) by Allamigeon, Dadush, Loho, Natura, and Végh (FOCS '22). The number of iterations needed by the IPM is bounded, up to a polynomial factor in the number of inequalities, by the *straight line complexity* of the central path. Roughly speaking, this is the minimum number of pieces of any piecewise linear curve that multiplicatively approximates the central path.

As our main contribution, we show that the straight line complexity of any minimum cost generalized flow instance is polynomial in the number of arcs and vertices. By applying a reduction of Hochbaum (ORL '04), the same bound applies to any linear program with at most two non-zeros per column or per row.

To be able to run the IPM, one requires a suitable initial point. For this purpose, we develop a novel multistage approach, where each stage can be solved in strongly polynomial time given the result of the previous stage. Beyond this, substantial work is needed to ensure that the bit complexity of each iterate remains bounded during the execution of the algorithm. For this purpose, we show that one can maintain a representation of the iterates as a low complexity convex combination of vertices and extreme rays. Our approach is black-box and can be applied to any log-barrier path-following method.

*Full version available at https://bentonatura.com/genflow.

STOC '24, June 24–28, 2024, Vancouver, BC, Canada

ACM ISBN 979-8-4007-0383-6/24/06
https://doi.org/10.1145/3618260.3649764

CCS CONCEPTS

• **Theory of computation** → **Linear programming**; • **Mathematics of computing** → *Combinatorial optimization*; Network optimization; *Network flows.*

KEYWORDS

strongly polynomial, generalized flow, linear programming, interior point method, circuits

ACM Reference Format:
Daniel Dadush, Zhuan Khye Koh, Bento Natura, Neil Olver, and László A. Végh. 2024. A Strongly Polynomial Algorithm for Linear Programs with At Most Two Nonzero Entries per Row or Column. In *Proceedings of the 56th Annual ACM Symposium on Theory of Computing (STOC '24), June 24–28, 2024, Vancouver, BC, Canada.* ACM, New York, NY, USA, 12 pages. https://doi.org/10.1145/3618260.3649764

1 INTRODUCTION

We consider linear programming (LP) in the following primal-dual form:

$$\begin{array}{lll} \min\ \langle c, x\rangle & & \max\ \langle b, y\rangle \\ \mathbf{A}x = b & & \mathbf{A}^\top y + s = c \\ x \geq \mathbf{0}_n\,, & & s \geq \mathbf{0}_n\,, \end{array} \tag{LP}$$

where $\mathbf{A} \in \mathbb{R}^{m\times n}$, $b \in \mathbb{R}^m$, $c \in \mathbb{R}^n$, and $\mathbf{A}$ has rank m. Our focus is on LP algorithms that find exact primal and dual optimal solutions, or conclude infeasibility or unboundedness. We say that the dual progam is a *two variable per inequality (2VPI) linear program* if every row of $\mathbf{A}^\top$ includes at most two nonzero entries. In such a case, we refer to the pair of LPs as a *2VPI primal-dual pair.*

The first polynomial-time LP algorithms were the ellipsoid method by Khachiyan in 1979 [29] and interior point methods, introduced by Karmarkar in 1984 [28]. However, it remains an outstanding open question to find a *strongly polynomial algorithm* for linear programming. The question was listed by the Fields medalist Smale as one of the most prominent mathematical challenges for the 21st century [41]. In such an algorithm, only poly(n) basic arithmetic operations and comparisons are allowed, and the algorithm uses polynomial space.

The notion of strongly polynomial algorithms was first formally introduced by Megiddo [32], under the term 'genuinely polynomial'.

The same paper gave an algorithm for two variable per inequality feasibility systems, that is, for the dual feasibility problem in (LP) when all rows of $\mathbf{A}^\top$ have at most two nonzero entries. The corresponding primal feasibility problem can be reduced to the *maximum generalized flow problem*. For this, the first strongly polynomial algorithm was given by Végh [50], followed by a faster and simpler algorithm by Olver and Végh [37]. The *minimum-cost generalized flow problem* is the dual of a 2VPI LP, where the two nonzero entries in each column of $\mathbf{A}$ are a -1 entry and a positive entry. As discussed below, this naturally corresponds to a network flow model with multipliers on the arcs. As shown in [23], all 2VPI LPs are reducible to the dual of a minimum-cost generalized flow problem. The existence of a strongly polynomial algorithm for this problem has been a longstanding open question, mentioned e.g. in [1, 10, 11, 20, 24, 36, 37, 50, 51]. Our main result resolves this question.

Theorem 1.1. *There is a strongly polynomial algorithm for the minimum-cost generalized flow problem, and for two variable per inequality primal-dual pairs.*

1.1 Background and Previous Work

Strongly polynomial algorithms for well-conditioned LPs. In a seminal, Fulkerson-prize winning paper [42], Tardos obtained the first strongly polynomial algorithm for minimum-cost circulations. A particularly important technique in this paper was *variable fixing*: by solving an approximate version of the LP with rounded costs, one can deduce that a certain variable is at the lower or upper capacity bound in an optimal solution.

Towards general LP, Tardos [43] extended this approach to obtain a strongly polynomial algorithm for 'combinatorial LPs'. More precisely, for (LP) with an integer constraint matrix $\mathbf{A} \in \mathbb{Z}^{m\times n}$, this algorithm runs in $\mathrm{poly}(n, \log \Delta_{\mathbf{A}})$ iterations, where $\Delta_{\mathbf{A}}$ is the maximum subdeterminant of $\mathbf{A}$. The running time is independent of b and c. In particular, this bound is strongly polynomial if all entries of $\mathbf{A}$ are at most $\mathrm{poly}(n)$, such as for multicommodity flows and other combinatorial problems. Using an interior point approach discussed below, Vavasis and Ye [49] obtained an algorithm with $\mathrm{poly}(n, \log \bar{\chi}_{\mathbf{A}})$ arithmetic operations, where $\bar{\chi}_{\mathbf{A}}$ is the Dikin–Stewart–Todd condition number of the matrix $\mathbf{A}$. For integer matrices, $\bar{\chi}_{\mathbf{A}} = O(n\Delta_{\mathbf{A}})$, thus, this strengthens Tardos's result. A similar dependence, using a black-box approach extending Tardos's work [43] was obtained by Dadush, Natura and Végh [15]. Further, Dadush, Huiberts, Natura and Végh [13] strengthened this dependence to $\mathrm{poly}(n, \log \bar{\chi}^*_{\mathbf{A}})$, where $\bar{\chi}^*_{\mathbf{A}}$ is the optimized value of $\bar{\chi}_{\mathbf{A}}$ under column rescalings.

Prior results on 2VPI and generalized flows. 2VPI LPs are a natural class of LP that does not fall into the above 'well-conditioned' classes: even $\bar{\chi}^*_{\mathbf{A}}$ may be unbounded for the constraint matrix. At the same time, they form an interesting intermediate class, as it is easy to see that solving an arbitrary LP is reducible to solving one with at most three nonzero entries per row in $\mathbf{A}^\top$.

For finding a feasible solution to a 2VPI system, Megiddo's [32] approach relied on parametric search. A faster parametric search algorithm was given by Cohen and Megiddo [10]. Hochbaum and Naor [24] used an efficient Fourier–Motzkin elimination to obtain what is still the fastest deterministic approach. Dadush, Koh, Natura and Végh [14] used a variant of the discrete Newton method. Recently, Karczmarz [27] gave an improved randomized strongly polynomial algorithm, also using parametric search.

Consider now *monotone* 2VPI (M2VPI) systems, where each inequality has at most one positive and at most one negative entry. If such an LP is bounded, then there exists a unique pointwise minimal solution and a unique pointwise maximal solution. Already the algorithm in [32] can be used to find these solutions. As noted by Adler and Cosares [1], an M2VPI linear program is strongly polynomially solvable if $b \geq \mathbf{0}$ or $b \leq \mathbf{0}$. Norton, Plotkin and Tardos [36] gave a strongly polynomial algorithm for a constant number of nonzero demands.

The generalized flow problem is (after normalization) the dual of the M2VPI problem. In this problem, we are given a directed graph $G = (V, E)$ with node demands b_i, $i \in V$ and arc costs c_e and gain factors $\gamma_e > 0$ for $e \in E$. While traversing the arc $e = (i, j)$, the flow value x_e gets multiplied to $\gamma_e x_e$. In the *minimum-cost generalized flow problem*, we need to exactly satisfy all node demands at a minimum cost. The *maximum generalized flow problem* is the special case when the objective is to maximize the net flow reaching a special sink node t.

This is a fundamental network optimization model that traces back to Kantorovich's 1939 paper [26] introducing linear programming. Generalized flow networks can be used to model transportation of a commodity through a network with leakages, or conversions between various equities in financial networks, as well as generalized assignment problems. We refer the reader to [2, Chapter 15] for further applications.

Goldberg, Plotkin, and Tardos [20] gave the first weakly polynomial combinatorial algorithm for the maximum generalized flow problem. This was followed by a significant number of further such algorithms, such as [11, 21, 38, 39, 44, 51], see further references in [37]. In particular, [11] gave a strongly polynomial approximation scheme, i.e., a strongly polynomial algorithm that achieves a fixed fraction of the optimum flow value in a capacitated generalized flow network. The strongly polynomial algorithms by Végh [50] and Olver and Végh [37] rely on the variable fixing technique, however, in a new, 'continuous' scaling framework. While the original LP can be ill-conditioned, variable fixing is still possible, since the dual solutions can be used to 'relabel' the flow to make it 'locally' amenable to classical network flow arguments.

However, relabelling heavily relies on the special cost function of the flow maximization problem, and does not seem to be extendable to the minimum-cost version. For solving the minimum-cost generalized flow problem, the only known (weakly polynomial) combinatorial approach is the ratio-circuit cancelling algorithm by Wayne [51]. The fastest previous weakly polynomial algorithms can be obtained using interior point methods; an early such example is by Vaidya [45]. Daitch and Spielman [16], and Lee and Sidford [31] gave fast algorithms for obtaining an additive ε-approximation; however, such approximation cannot be used to obtain exact optimal solutions. We also note that the latter results only apply for lossy flows, i.e., with gain factors $\gamma_e \leq 1$.

Interior Point Methods and their limitations. Interior point methods (IPMs) give the fastest current weakly polynomial algorithms for general LP, see [12, 25, 46, 47] as well as for special classes such as minimum-cost circulations [8, 9] and multicommodity flows [48].

They are also a potent approach in the context of strongly polynomial computability, and form the basis of our result.

The algorithms discussed next fall into the class of primal-dual path-following algorithms. A key concept here is the *central path*, the algebraic curve formed by minimizers of $\langle c, x\rangle - \mu \sum_{i=1}^{n} \log(x_i)$ for $\mu > 0$. As $\mu \to 0$, the limit of the central path is an optimal solution. Path-following methods maintain iterates in a certain neighborhood of the central path while geometrically decreasing μ, and thus, the optimality gap. The logarithmic barrier function above can be replaced by more general barrier functions. The affine scaling step is a standard way to find a movement direction. This can be interpreted as a least square computation in the local norm induced by the Hessian of the logarithmic barrier function.

Layered least squares (LLS) IPMs were introduced in the influential work of Vavasis and Ye [49]. The LLS step in the Vavasis–Ye algorithm decomposes the variables into different layers based on the values of the current iterate. The step direction is determined as a sequence of least squares computations that prioritizes decreasing variables at lower layers. Roughly speaking, such steps enable to traverse arbitrarily long but relatively straight segments of the central path in a single iteration. Combinatorial progress is measured by *crossover events*, where two variables get reordered consistently with their order in the limit optimal solution. This is very different from the variable fixing technique prevalent in the combinatorial approaches discussed above. In particular, while we can infer the occurrence of a new crossover event within a certain number of iterations, the argument only shows existence, and we cannot identify the participating variables. The condition number $\bar{\chi}_{\mathbf{A}}$ appearing in the running time is a bound on the norms of oblique projections.

This led to a line of research on improved combinatorial IPMs [30, 33–35]. The paper [13] revealed that $\bar{\chi}_{\mathbf{A}}$ is closely related to the *circuit imbalance measure* $\kappa_{\mathbf{A}}$ that bounds the maximum ratio of two nonzero entries of an elementary vector in the kernel of $\mathbf{A}$. Moreover, they obtained an LLS algorithm invariant under column rescaling, thus improving the dependence to the best $\kappa^*_{\mathbf{A}}$ value achievable under column rescalings.

The above results may raise hopes to finding a strongly polynomial IPM. However, the papers by Allamigeon, Benchimol, Gaubert and Joswig [3], Allamigeon, Gaubert and Vandame [6], and Zong, Lee and Yue [53] yield a surprising negative answer. By analyzing the tropical limits of linear programs, these papers exhibit parametric families of LPs such that for suitably large parameter values, no path-following method can be strongly polynomial. This was first shown for the standard logarithmic barrier [3], and later for arbitrary self-concordant barriers [6].

1.2 The Subspace Layered Least Squares Interior Point Method and Straight Line Complexity

The primal central path has a natural dual counterpart. The primal-dual central path point $(x^{\mathrm{cp}}(\mu), s^{\mathrm{cp}}(\mu))$ is the unique pair of primal and dual feasible solutions to (LP) such that $x_i^{\mathrm{cp}}(\mu) s_i^{\mathrm{cp}}(\mu) = \mu$ for all $1 \le i \le n$. Thus, the duality gap between $x^{\mathrm{cp}}(\mu)$ and $s^{\mathrm{cp}}(\mu)$ is $n\mu$.

The lower bounds in [3, 6] are ultimately based on the following insight. The trajectory of any path-following IPM is a piecewise linear curve in the neighborhood of the central path; the number of pieces correspond to the number of iterations. Thus, a lower bound on the number of *any* piecewise linear curve in the neighborhood provides a lower bound on the number of iterations. For the examples in these papers, exponential lower bounds are shown.

The recent algorithm designed by Allamigeon, Dadush, Loho, Natura and Végh [4] complements these negative results by a positive algorithmic bound. Namely, they provide an IPM whose number of iterations matches such a lower bound within a strongly polynomial factor. Let us elaborate on the lower bound.

Assume (LP) is feasible and bounded with optimum value $v^\star$. Given $g \ge 0$, we denote by

$$\begin{aligned} \mathcal{P}_g &:= \{x \in \mathbb{R}^n_+ \mid \mathbf{A}x = b\,,\ \langle c, x\rangle \le v^\star + g\}\,, \\ \mathcal{D}_g &:= \{s \in \mathbb{R}^n_+ \mid \exists y \in \mathbb{R}^m : \mathbf{A}^\top y + s = c\,,\ \langle b, y\rangle \ge v^\star - g\} \end{aligned} \tag{1}$$

the feasible sublevel sets. They correspond to the sets of the primal and dual feasible points (x, s) with objective value within g from the optimum $v^\star$, respectively.

Assuming that (LP) has strictly feasible primal and dual solutions, the *max central path* is defined as the parametric curve $g \mapsto z^{\mathfrak{m}}(g) := (x^{\mathfrak{m}}(g), s^{\mathfrak{m}}(g)) \in \mathbb{R}^{2n}_+$, where

$$x_i^{\mathfrak{m}}(g) := \max\{x_i : x \in \mathcal{P}_g\}\,, \qquad s_i^{\mathfrak{m}}(g) := \max\{s_i : s \in \mathcal{D}_g\} \tag{2}$$

for all $i \in [n]$. The max central path can be seen as a combinatorial proxy to the central path. In particular, for $g = n\mu$, $x^{\mathrm{cp}}(\mu) \in \mathcal{P}_g$ and $s^{\mathrm{cp}}(\mu) \in \mathcal{D}_g$, and it is easy to see that $x^{\mathfrak{m}}(g)/n \le x^{\mathrm{cp}}(\mu) \le x^{\mathfrak{m}}(g)$ and $s^{\mathfrak{m}}(g)/n \le s^{\mathrm{cp}}(\mu) \le s^{\mathfrak{m}}(g)$.

For each $1 \le i \le n$, the function $g \mapsto x_i^{\mathfrak{m}}(g)$ is a piecewise linear concave function. It corresponds to a trajectory of the shadow simplex algorithm that interpolates between the objective functions $\langle c, x\rangle$ and $-x_i$. The breakpoints correspond to basic feasible solutions, and therefore the number of linear pieces is at most the number of vertices, that is, at most 2^n. We will use the following definition.

Definition 1.2. Let $f : \mathbb{R}_+ \to \mathbb{R}_+$ be a function and $\eta \in [0, 1]$. The *straight line complexity* of f with respect to η, denoted $\mathrm{SLC}_\eta(f)$, is the infimum number of pieces of a continuous piecewise linear function $h : \mathbb{R}_+ \to \mathbb{R}_+$ where $\eta f \le h \le f$. [1]

The number of iterations taken by the *Subspace Layered Least Squares (SLLS) IPM* in [4] can be bounded by the sum of straight line complexities of each coordinate of the (primal) max central path. We now state the guarantees of SLLS IPM as given in [5], which strengthens the main result of the conference version [4] by ensuring that each iteration of the IPM can be implemented in strongly polynomial time.

Theorem 1.3 ([4, 5]). *There is an interior-point method that given an instance of (LP) and strictly feasible solutions $x, s > \mathbf{0}$ such that $\| \frac{nx \circ s}{\langle x, s\rangle} - \mathbf{1}_n \| \le \beta$, $0 < \beta \le 1/6$, finds a pair of primal and dual optimal solutions in*

$$O\left(\min_{\eta \in (0,1]} \frac{\sqrt{n}}{\beta} \log\left(\frac{n}{\beta\eta}\right) \sum_{i=1}^{n} \mathrm{SLC}_\eta(x_i^{\mathfrak{m}}) \right)$$

many iterations.[2] The algorithm can be implemented in the real RAM model, moreover, each iteration runs in strongly polynomial time in the Turing model.

[1] We remark that our convention on the parameter η differs from [4, 5], which uses $(1-\theta)f \le h \le f$ instead.

[2] The condition on the starting point asserts that it is near the central path; $x \circ s$ denotes the Hadamard product.

We explain the real RAM and strongly polynomial computational models in Section 1.3 below. The SLLS IPM requires at most a $\tilde{O}(n^{1.5})$ factor more iterations than *any* path-following IPM for any self-concordant barrier function. This is because it can be shown that each $\mathrm{SLC}_\eta(x_i^{\mathfrak{m}})$ gives a lower bound on the number of piecewise linear segments traversing a corresponding wide neighborhood. Moreover, as noted above, $\mathrm{SLC}_1(x_i^{\mathfrak{m}}) \le 2^n$, thus, the number of iterations is always at most singly exponential.

We note that the theorem could be equivalently written in terms of the dual straight line complexities (see [5, Lemma 4.5]). Further, we note that the neighborhood parameter η is not important for the overall bound. It is not difficult to show that for $0 < \eta < \eta' < 1$, $\mathrm{SLC}_{\eta'}(f) = O(\log(1/\eta)/\log(1/\eta'))\,\mathrm{SLC}_\eta(f)$.

According to Theorem 1.3, analyzing the number of iterations of SLLS IPM boils down to upper-bounding the straight line complexities of the variables. Note that this is a purely geometric question about understanding the structure of univariate piecewise linear functions $x_i^{\mathfrak{m}}(g)$.

1.3 Computational Models

There are multiple related, yet distinct notions of a *strongly polynomial* computational model. Smale's question was posed in the *Blum–Shub–Smale (BSS) real model of computation* [7]. In this model, the input can be given by arbitrary real numbers, and one step may compute a rational polynomial function of the previously computed quantities with real coefficients, or make comparisons between two quantities. In the more restrictive *real RAM* model, one can perform a sequence of elementary arithmetic operations $(+, -, \times, /)$ and comparisons $(\ge)$ on real numbers. In this paper, we say that an algorithm is *polynomial in the real RAM model* if the number of elementary arithmetic operations and comparisons is bounded polynomially in the dimension of the input; in the case of LP, this is $K = n \times m + n + m$.

We now turn to the Turing model. Consider a problem where the input is given by K integers; for LP, the input $(\mathbf{A}, b, c)$ is described by $K = 2(n \times m + n + m)$ integers representing the rational entries. An algorithm is *strongly polynomial in the Turing model* (see [22]), if it only performs $\mathrm{poly}(K)$ (in the LP case, this means $\mathrm{poly}(m, n)$) elementary arithmetic operations and comparisons as in the real model. Additionally, the bit-complexity of the numbers during the computations must remain polynomially bounded in the encoding length of the input. Equivalently, the algorithm must be PSPACE. The model has an ambiguity regarding how divisions can be implemented, see discussion of variants in [22, Section 1.3]. The results of this paper work with the most restricted setting: we maintain rational representations (p, q) of all numbers during the computation, and division $\frac{p}{q} / \frac{p'}{q'}$ corresponds to computing the representation $(pq', p'q)$.

While a strongly polynomial algorithm in the Turing model implies a polynomial algorithm in the real RAM model, the converse is not necessarily the case: enforcing PSPACE may be challenging. For example, Gaussian elimination needs to be done carefully to keep the sizes of numbers under control, see [17] and [22, Section 1.4]. The LLS interior point methods [4, 13, 33–35, 49] are polynomial in the real RAM model[3] whenever $\log(\bar{\chi}_{\mathbf{A}}) = \mathrm{poly}(n)$. However, we are not aware of any strongly polynomial implementation of such algorithms in the Turing model. The principal difficulty in this regard is keeping the bit-complexity of the iterates produced by the IPM uniformly bounded using only the allowed operations $(+, -, \times, /)$. In particular, truncating the bit representation of the current iteration to L-bits of precision, where L depends on the bit-length of the input cannot be achieved using $O(1)$ basic operations.

In the weakly polynomial model, i.e., when running time dependence on the total encoding length is allowed, the bit complexity of the algorithms can be controlled by approximately solving linear systems and roundings. Recent work by Ghadiri, Peng, and Vempala [19] developed general tools that enable to keep the bit complexity of recent fast IPMs under control. However, these techniques are not applicable in the strongly polynomial model. In particular, they require estimates on parameters such as the total bit length of the input or the condition number of the matrix. They also require rounding to a fixed number of bits depending on such numerical parameters. As mentioned above, in the most stringent definition of strongly polynomial time, this cannot be done.

The implementation of the SLLS IPM given in [5], which we rely on, guarantees that each iteration of the IPM is strongly polynomial in the Turing model. More precisely, given the constraint matrix A and the current iterate (x, s) as input, the IPM computes the next iterate (x', s') in time strongly polynomial in the input (A, x, s). This in particular implies that the bit complexity of the iterates grows at most by a polynomial factor in each iteration. This is however insufficient for controlling the bit complexity over many iterations. We resolve this issue by providing a combinatorial rounding scheme, which maintains a representation of the current iterate as a low complexity convex combination of vertices and extreme rays. We describe this in further detail in Section 1.4.3.

We note that the algorithms presented in this paper are fully deterministic.

1.4 Our Contributions

We prove Theorem 1.1, i.e., give a strongly polynomial algorithm for the minimum-cost generalized flow problem by showing that the total number of iterations of the SLLS IPM by [4] is strongly polynomially bounded, and that the SLLS IPM can be implemented in strongly polynomial time in the Turing model. Our result has three main ingredients:

(1) *Straight line complexity bound:* We establish in Theorem 3.1 a strongly polynomial bound $\mathrm{SLC}_\eta(x_e^{\mathfrak{m}}) = O(mn)$ on the straight line complexities of the variables in the minimum-cost generalized flow problem, with $\eta = \Omega(1/(m^2 n))$, where m is the number of arcs and n is the number of nodes of the graph. This bound applies for the uncapacitated version of the problem as described above; if in addition arcs have capacities, the bound becomes $O(m^2)$.
(2) *Initialization:* IPMs require a strictly feasible and well centered starting point (x^0, s^0), even though a strictly feasible (or even a feasible) solution may not exist. We present a careful initialization scheme that solves linear programs in three stages, and preserves the straight line complexity bounds.

[3] Some of these IPMs make use of square-root computations, and hence rely on the extended real model $(+, -, \times, /, \sqrt{\cdot})$.

(3) *Implementation in the Turing model:* We show that the bit-length of the computations can be controlled in a model using only basic arithmetic operations and comparisons.

The straight line complexity is established via a combinatorial argument using structural properties of generalized flows. In contrast, the initialization and implementation tasks are applicable for general LP, and can be seen as a direct strengthening of the result in [4, 5]. We now elaborate on each of these parts, and highlight the main technical ideas.

1.4.1 Straight Line Complexity Bound for Generalized Flows. Theorem 1.3 enables to bound the number of iterations in SLLS IPM by bounding the straight line complexities $\mathrm{SLC}_\eta(x_i^{\mathrm{m}})$ for a suitable $\eta > 0$. In the first step, we reduce this to an even more concrete combinatorial question of *circuit covers* as explained next.

Circuit covers. For the purposes of analyzing straight line complexities, we can assume that a pair of primal and dual optimal solutions $(\bar{x}, \bar{s})$ to (LP) is provided.

For any vector $h \in \ker(\mathbf{A})$, with $\langle c, h\rangle \geq 0$ we can define the function $\bar{x}^h(g) : \mathbb{R}_+ \to (\mathbb{R}_+ \cup \{\infty\})^n$ by moving from $\bar{x}$ in the direction of h; this is called the *h-curve from* $\bar{x}$. Namely, we define $\bar{x}^h(g) = \bar{x} + \alpha(g)h$, where $\alpha(g)$ is chosen maximally so that $\bar{x}^h(g)$ is feasible, and has cost at most g larger than the cost of $\bar{x}$. For every $i \in [n]$, the i-th coordinate $\bar{x}_i^h(g)$ can easily be seen to be a piecewise linear concave function with two pieces, the first with slope $h_i/\langle c, h\rangle$ and the second constant.Note that $\bar{x}^h$ is constant if $\langle c, h\rangle = 0$.

Given $h, h' \in \ker(\mathbf{A})$, $\langle c, h\rangle, \langle c, h'\rangle \geq 0$, $\alpha > 0$ and $i \in [n]$, we say that h *α-dominates h' on $i \in [n]$* if $\bar{x}_i^h(g) \geq \alpha \bar{x}_i^{h'}(g)$ for all $g \geq 0$.

In the linear space $\ker(\mathbf{A})$, an *elementary vector* is a support minimal nonzero vector, and the support of an elementary vector is called a *circuit*. Note that the latter coincides with the notion of circuits of the linear matroid of $\mathbf{A}$; each circuit corresponds to a one-dimensional subspace of elementary vectors. We let $\mathcal{E}(\mathbf{A})$ denote the set of all elementary vectors.

Given an optimal solution $\bar{x}$, let us consider the augmentations from $\bar{x}$ by an elementary vector h. Noting that $\bar{x}^h(g)$ is invariant under rescaling h to αh for $\alpha > 0$, this gives one function per circuit. For any coordinate $i \in [n]$, let us now consider the pointwise maximum at the i-th coordinate $\hat{x}_i(g) = \max\{\bar{x}_i^h(g) : h \in \mathcal{E}(\mathbf{A})\}$. This is a piecewise linear function, but is not concave. Note also that the number of pieces can be exponential. Nevertheless, using standard circuit decomposition techniques, it is not difficult to show that $\hat{x}_i(g)$ approximates $x_i^{\mathrm{m}}(g)$ up to a factor n: $x_i^{\mathrm{m}}(g)/n \leq \hat{x}_i(g) \leq x_i^{\mathrm{m}}(g)$.

Our strategy to obtain SLC bounds is by constructing *circuit covers*. Given a primal optimal solution $\bar{x}$, an index $i \in [n]$ and $\alpha > 0$, we say that a set of vectors $S \subseteq \ker(\mathbf{A})$ is an *α-circuit cover of i with respect to $\bar{x}$* if for every $h' \in \mathcal{E}(\mathbf{A})$, there is a $h \in S$ that α-dominates h' on i. Then, the piecewise-linear function $\bar{x}_i^S(g) = \max\{\bar{x}_i^h(g) : h \in S\}$ satisfies $\alpha x_i^{\mathrm{m}}(g)/n \leq \bar{x}_i^S(g) \leq x_i^{\mathrm{m}}(g)$. The upper convex envelope of this function has at most $|S| + 1$ linear pieces, and consequently, $\mathrm{SLC}_{\alpha/n}(x_i^{\mathrm{m}}) \leq |S| + 1$.

Circuit covers for generalized flows. We work with minimum-cost generalized flow in its capacitated form, where arcs may have capacities, and all demands are zero; this reduction yields only n capacitated arcs when starting from the demand form. The circuits in this version of the generalized flow problem correspond to simple combinatorial structures: namely, a circuit either corresponds to a *conservative* directed cycle, where the product of the gain factors is one, or a *'bicycle'*, namely, a flow generating cycle connected by a path to a flow absorbing cycle. The latter are cycles where the product of the gain factors is greater and less than one, respectively. These structures played a fundamental role in all prior works on generalized flows, see e.g., [20, 37, 50, 51], as well as for 2VPI algorithms, e.g., [10, 14, 27, 32].

We construct a circuit cover of size $O(mn)$ to bound the straight line complexity of the variables in the generalized flow problem. The basis of our construction is *path domination*. Let us fix two nodes s and t in the graph. We demonstrate a small collection of s–t walks that "dominate" the collection of all s–t paths in a certain sense. The general cover will be constructed by combining such walks. We now highlight the main ideas of path domination.

Consider an s–t walk W; this induces an s–t flow $\bar{x}^W$ on the walk. We define the function $\vec{f}_W : \mathbb{R}_+^2 \to \mathbb{R}_+$ such that $\vec{f}_W(\lambda, r)$ denotes the maximum amount of flow that can be sent from s to t if there are r units available at s, each step of the walk satisfies the capacity bound, and the cost incurred in any step of the walk is at most λ. This corresponds to a certain maximal scaling of $\bar{x}^W$. This scaling may have total cost larger than λ, and moreover since an arc of the graph can be used multiple times, it may also violate arc capacities. But, as long as the walk is *n-recurrent*, meaning it uses each edge at most n times, scaling down by mn will yield a feasible flow with total cost at most λ. There are two possible bottleneck arcs that prevent a larger scaling of $\bar{x}^W$ from being used: a *cost bottleneck arc* e_{c} where $c_{e_{\mathrm{c}}}\bar{x}_{e_{\mathrm{c}}}^W$ is maximal, and a *flow bottleneck arc* e_{f} where $\bar{x}_{e_{\mathrm{f}}}^W/u_{e_{\mathrm{f}}}$ is maximal, where c_e and u_e denote the cost and capacity of arc e, respectively. We associate the combinatorial signature $(e_{\mathrm{c}}, e_{\mathrm{f}}, \preceq)$ or $(e_{\mathrm{c}}, e_{\mathrm{f}}, \succ)$ with W, where $\preceq$ means that e_{c} precedes or equals e_{f} on the walk W, and $\succ$ means that e_{f} precedes e_{c}.

We say that the s–t walk W' dominates W if $\vec{f}_{W'} \geq \vec{f}_W$. Denoting the number of finite capacity arcs by $\bar{m}$, we are able to show the existence of an $O(m\bar{m})$-sized family of n-recurrent s–t walks that dominate all s–t paths. The family is constructed by fixing the signature, and from each signature, selecting the best walk for the three segments defined by s, t, and the two bottleneck arcs, such that each segment is highest gain subject to recurrence bounds and not having other bottlenecks.

Once we have this path domination result, we can use this to demonstrate domination of more complicated collections of objects with small dominating sets, and eventually all circuits. It easily follows, for instance, that there is a small collection of s–s walks with the property that for any flow-generating cycle C containing s, there exists a walk R in this collection that dominates C, in the sense that for every choice of cost bound $\lambda \in \mathbb{R}_+$, at least as much excess can be created using R than C. From this, by 'composing' dominating sets for cycles and paths, we obtain a small dominating set for the collection of all bicycles in the graph.

1.4.2 Initialization. A strongly polynomial straight line complexity bound implies a strongly polynomial iteration bound of the SLLS IPM; however, it requires an initial point $(x^0, s^0) \in \mathcal{P}_{++} \times \mathcal{D}_{++}$ near the central path. Such a point may not even exist; in fact, the primal or dual programs in (LP) may be infeasible. Whereas one could use

the combinatorial algorithms to decide primal [37] and dual [24] feasibility, these algorithms do not directly yield strictly feasible solutions (which may again not exist).

The situation is analogous to Simplex, where Stage I can be used to find a feasible solution by solving an auxiliary LP. Various initialization methods have been developed for IPMs, but none of these is directly applicable for our purposes: only solving auxiliary systems with small straight line complexity, while remaining in the strongly polynomial model.

A common initialization technique is the self-dual homogenous formulation [52]. However, writing the self-dual formulation of a generalized flow LP results in a more complicated problem and it is not clear if the straight line complexity admits a similar bound. (Note also that in the simpler case of (standard) network flows, the constraint matrix is totally unimodular, while the combined matrix does not have this property.)

We present two initialization methods. Our first approach uses a 'big-M' method, as in [49]. Let us create a negative copy of each variable with a large penalty cost. That is, one can replace the primal system using variables $(x, x', x'') \in \mathbb{R}^{3n}$ in the form

$$\begin{aligned} &\min \langle c, x\rangle + M\left\langle \mathbf{1}_n, x'\right\rangle \\ &\text{s.t.} \quad \mathbf{A}x - \mathbf{A}x' = b\,, \quad x + x'' = 2M\mathbf{1}_n\,, \quad x, x', x'' \geq \mathbf{0}\,. \end{aligned} \tag{3}$$

Here, x' represents a negative copy of each variable and x'' corresponds to a slack variable for the box constraint $0 \leq x \leq 2M\mathbf{1}_n$. Such a system, along with its dual, is easy to initialize for sufficiently large M. Moreover, the constraint matrix remains 'nice', e.g., it can be still interpreted as a (capacitated) generalized flow problem, where the x' variables correspond to expensive reverse arcs. As long as there exists a pair of primal and dual optimal solutions $(x^\star, s^\star)$ to (LP) with $\|x^\star\|_\infty, \|s^\star\|_\infty < M\mathbf{1}_n$, these will also be optimal solutions to the extended formulation.

However, finding a suitable large M becomes challenging. In [49], such a bound is derived based on $\bar{\chi}_{\mathbf{A}}$. This is hard to compute in general; one could use a repeated guessing of this condition number, but this would lead to a $\log\log \bar{\chi}_{\mathbf{A}}$ running time dependence. Bounds on the norms of optimal solutions are routinely derived using bit-complexity arguments, see e.g. [22]; however, this is also not possible in the strongly polynomial model.

To address this, we use the existing strongly polynomial algorithms of e.g., [37] and [24] to solve up to n primal and dual feasibility problems to first obtain maximum support primal and dual solutions. We then reduce the problem to a system with a pair of strictly positive primal and dual solutions. The reduction is achieved by deleting some variables and projecting out some others. In the generalized flow problem, these amount to graphical operations of deletions and contractions, and thus preserve the generalized flow structure. Given the strictly positive primal and dual solutions $(\hat{x}, \hat{s})$, choosing M larger than $\langle \hat{x}, \hat{s}\rangle$ divided by the smallest entry of $(\hat{x}, \hat{s})$ guarantees that $\|x^\star\|_\infty, \|s^\star\|_\infty < M\mathbf{1}_n$ for any pair of primal and dual optimal solutions.

Whereas the above approach can implement the big-M method, it is only applicable to the particular minimum-cost generalized flow setting as it requires feasibility solvers. Also, it needs to solve $2n$ systems as preprocessing. We also develop a more principled, multistage initialization strategy that is applicable to general LP, preserves straight line complexity, and only requires solving four IPM problems. Since we will need to solve different LPs derived from (LP), one needs to clarify what 'preserving straight line complexity' means. We define $\mathrm{SLC}_\eta(\mathbf{A})$ as the maximum value of $\mathrm{SLC}_\eta(x_i^{\mathrm{m}})$ for any variable $i \in [n]$ in any LP of the form (LP) with constraint matrix $\mathbf{A}$, but taking any possible right-hand side $b \in \mathbb{R}^m$ and cost $c \in \mathbb{R}^n$. All our auxiliary LPs will have SLC bounded by $\mathrm{SLC}_\eta(\mathbf{B})$, where $\mathbf{B} = \begin{pmatrix} \mathbf{A} & -\mathbf{A} & \mathbf{0}_{m\times n} \\ \mathbf{I}_n & \mathbf{0}_{n\times n} & \mathbf{I}_n \end{pmatrix}$ is the matrix also used in the big-M formulation (3).

Our strategy can be interpreted as a facial reduction strategy, where we carefully fix or project out variables that yield an equivalent LP to the original one, and where strictly feasible primal and dual solutions in fact exist. Throughout the process, the solutions from a previous stage provide a starting point near the central path.

1.4.3 Implementation in the Turing model. As discussed in Section 1.3, to obtain a strongly polynomial algorithm in the Turing model one needs to devise a new rounding approach, as the previous ones rely on bit-complexity information and rounding that are not implementable in the strongly polynomial model.

The main ingredient is a general strongly polynomial technique to keep the bit-complexity of all iterations polynomially bounded in the input encoding length. This technique is not particular to the SLLS algorithm but can be used for any path-following method. The main subroutine takes an iterate (x, s) in the central path neighborhood, and computes $(\tilde{x}, \tilde{s})$ that is in a slightly larger neighborhood, may have slightly worse optimality gap, but its encoding length is polynomially bounded in the length of the input.

To argue about the encoding length, we 'anchor' the point $(\tilde{x}, \tilde{s})$ to vertices of the primal and dual polytopes. In strongly polynomial time, we can write a Minkowski–Weyl decomposition of x and s using vertices and extreme rays. However, we cannot simply round the coefficients. In particular, it is possible that (x, s) is written as a 'highly unstable' convex combination of primal and dual vertices such that either $\langle v, u\rangle < \langle x, s\rangle / 2^n$ or $\langle v, u\rangle < 2^n \langle x, s\rangle$ for each pair of primal and dual vertices (v, u). We proceed in two stages. First, we try to find a value $\mu^\star \approx \langle x, s\rangle / n$ such that $\mu^\star$ has small encoding length. This is easy as long as the combination contains primal and dual vertices (v, u) with $\langle x, s\rangle / 2^n \leq \langle v, u\rangle \leq 2^n \langle x, s\rangle$. In the 'highly unstable' situation as above, it turns out that the direction from (x, s) pointing towards a pair of primal and dual vertices (v, u) with much better gap is a very good movement direction of the IPM. Hence, we can replace (x, s) during the rounding step by a much better iterate that is also numerically more stable. In the second stage, we add a cost bound to our feasible region according to $\mu^\star$. On this bounded polytope, we can now find a Minkowski–Weyl decomposition and simply round the coefficients. The guarantees of this rounding are based on the near-monotonocity property of the central path.

1.5 Organization

The rest of the paper is structured as follows. Section 2 introduces some necessary background, in particular, regarding straight line complexity and circuits. Section 3 analyzes the straight line complexity of minimum cost generalized flows. It contains an overview of the proof strategy for obtaining a weaker bound of $O(nm^4)$. The stronger bound of $O(m^2)$ and all the proofs can be found in the full version.

The initialization procedure for the IPM, as well as the rounding procedure needed to control the bit-complexity, are also deferred to the full version.

2 PRELIMINARIES

Notation. We let $\mathbb{R}_{++}$ denote the set of positive reals, and $\mathbb{R}_{+}$ the set of nonnegative reals; similarly for $\mathbb{Z}_{++}, \mathbb{Z}_{+}, \mathbb{Q}_{++}$ and $\mathbb{Q}_{+}$. For $n \in \mathbb{N}$, we let $[n] := \{1, 2, \ldots, n\}$. We let $\mathbf{0}_n, \mathbf{1}_n \in \mathbb{R}^n$ denote the all 0s and all 1s vectors, respectively. For $x \in \mathbb{R}^n$, we let $\operatorname{supp}(x) \subseteq [n]$ denote its support. For $\alpha \in \mathbb{R}$, we let $\alpha^+ = \max\{\alpha, 0\}$ and $\alpha^- = \max\{-\alpha, 0\}$; for a vector $x \in \mathbb{R}^n$, we use x^+ and x^- coordinatewise. We let $\operatorname{supp}^+(x) = \operatorname{supp}(x^+)$ and $\operatorname{supp}^-(x) = \operatorname{supp}(x^-)$. For functions $f_i : \mathbb{R}^n \to \mathbb{R}^m$, $i \in \mathcal{I}$, we let $\bigvee_{i\in\mathcal{I}} f_i$ denote the pointwise maximum.

We let $\ker(\mathbf{A})$ denote the kernel of the matrix $\mathbf{A} \in \mathbb{R}^{m\times n}$. The standard inner product of the two vectors $x, y \in \mathbb{R}^n$ is denoted as $\langle x, y\rangle = x^\top y$. For $x, y \in \mathbb{R}^n$, we let $x \circ y = (x_1 y_1, \ldots, x_n y_n)$ denote the Hadamard-product, and if $y \in \mathbb{R}^n_{++}$, we let $x/y = (x_1/y_1, \ldots, x_n/y_n)$.

We denote the primal and dual feasible regions of (LP) by

$$\mathcal{P} := \{x \in \mathbb{R}^n_+ \mid \mathbf{A}x = b\} \text{ and } \mathcal{D} := \{s \in \mathbb{R}^n_+ \mid \exists y : \mathbf{A}^\top y + s = c\}$$

respectively. We let $\mathcal{P}_{++} := \mathcal{P} \cap \mathbb{R}^n_{++}$ and $\mathcal{D}_{++} := \mathcal{D} \cap \mathbb{R}^n_{++}$ denote the strictly feasible regions. Interior point methods require $\mathcal{P}_{++}, \mathcal{D}_{++} \neq \emptyset$. We do not make this assumption in general; in the full version, we show how one can use a sequence of reductions to simpler IPM problems to first either find a suitable initial point (x^0, s^0), or conclude infeasibility or unboundedness of the input LP.

The sublevel sets. Assume $\mathcal{P}, \mathcal{D} \neq \emptyset$, in which case (LP) admits a pair of primal and dual optimal solutions $(\bar{x}, \bar{y}, \bar{s})$ with optimum value $\langle c, \bar{x}\rangle = \langle b, \bar{y}\rangle = v^\star$. Recall that this holds precisely if these two solutions are complementary: $\langle \bar{x}, \bar{s}\rangle = 0$; in particular, $\bar{x}_i \bar{s}_i = 0$ for all $i \in [n]$.

Recall the definitions of the sublevel sets $\mathcal{P}_g, \mathcal{D}_g$ in (1) as the set of primal and dual solutions with objective value within g from the optimum value $v^\star$.

The duality gap of any pair (x, y, s) of primal-dual feasible points of (LP) fulfills $\langle c, x\rangle - \langle b, y\rangle = \langle x, s\rangle$. In particular, we have $\langle x, \bar{s}\rangle = \langle c, x\rangle - v^\star$ and $\langle \bar{x}, s\rangle = v^\star - \langle b, y\rangle$. Thus, the two sets $\mathcal{P}_g$ and $\mathcal{D}_g$ are equivalently given by

$$\mathcal{P}_g = \{x \in \mathcal{P} : \langle x, \bar{s}\rangle \leq g\} \quad \text{and} \quad \mathcal{D}_g = \{s \in \mathcal{D} : \langle \bar{x}, s\rangle \leq g\}.$$

These expressions are in fact independent of the choice of optimal solutions $(\bar{x}, \bar{y}, \bar{s})$. The following is immediate.

Proposition 2.1. *Assume that $\mathcal{P}_{++}$ and $\mathcal{D}_{++}$ are nonempty. Then, for all $g \geq 0$, the sets $\mathcal{P}_g$ and $\mathcal{D}_g$ are bounded.*

Our main tool for analyzing SLC are circuits.

Definition 2.2 (Elementary vectors and circuits)**.** Let $\mathbf{A} \in \mathbb{R}^{m\times n}$ and assume $\ker(\mathbf{A}) \neq \{\mathbf{0}_n\}$. A vector $z \in \ker(\mathbf{A})$ is an *elementary vector in* $\ker(\mathbf{A})$ if z is a support-minimal nonzero vector in $\ker(\mathbf{A})$. We let $\mathcal{E}(\mathbf{A})$ denote the set of all elementary vectors. A set $C \subseteq [n]$ is a *circuit of* $\mathbf{A}$ if it is the support of some elementary vector; we let $\mathcal{C}(\mathbf{A}) \subseteq 2^{[n]}$ denote the set of circuits.

We say that a vector $y \in \mathbb{R}^n$ *conforms to* $x \in \mathbb{R}^n$ if $x_i y_i > 0$ whenever $y_i \neq 0$. A *conformal circuit decomposition* of a vector $z \in \ker(\mathbf{A})$ is a decomposition of the form $z = \sum_{i=1}^{\ell} g^{(i)}$, where $g^{(1)}, \ldots, g^{(\ell)} \in \mathcal{E}(\mathbf{A})$, $\ell \leq n$, and each $g^{(i)}$ conforms to z. This notion can be seen as a generalization of the cycle decomposition of circulations for networks flows. The existence of such a decomposition is well-known, see e.g., [18, 40].

Proposition 2.3. *For every $\mathbf{A} \in \mathbb{R}^{m\times n}$, every vector $z \in \ker(\mathbf{A})$ admits a conformal circuit decomposition.*

2.1 Straight Line Complexity and Circuits

In this section, we establish an intimate connection between the SLC of an LP and its circuits. Recall the definition (2) of the max central path $(x^{\mathrm{m}}, s^{\mathrm{m}})$ from the introduction.

Definition 2.4 (h-curve)**.** Let $\bar{x}$ be a primal optimal solution to (LP). Given a vector $h \in \ker(\mathbf{A})$ where $\langle c, h\rangle \geq 0$, the *$h$-curve from $\bar{x}$* is the function $\bar{x}^h : \mathbb{R}_+ \to (\mathbb{R}_+ \cup \{\infty\})^n$ that maps $\bar{x}^h(g)$ to $\bar{x} + \alpha h$, for $\alpha \in \mathbb{R}_+ \cup \{\infty\}$ chosen maximally such that $\bar{x} + \alpha h \geq \mathbf{0}$ and $\langle c, \alpha h\rangle \leq g$.

Note that $\bar{x}^h = \bar{x}^{\beta h}$ for all $\beta > 0$. It is easy to see that

$$\bar{x}^h(g) = \bar{x} + \min\left(\frac{g}{\langle c, h\rangle}, \min_{j\in\operatorname{supp}^-(h)} \frac{\bar{x}_j}{|h_j|}\right) h, \tag{4}$$

with the convention that we omit the first term from the minimum if $\langle c, h\rangle = 0$ ($\bar{x}^h$ is a constant function in this case). The next lemma shows that for every $i \in [n]$ and $g \geq 0$, the ith coordinate of the max central path at g is upper bounded by a circuit augmentation from an optimal solution, up to a factor n.

Lemma 2.5. *Let $\bar{x}$ be a primal optimal solution to (LP) and $i \in [n]$. For every $g \geq 0$ where $x^{\mathrm{m}}_i(g) > \bar{x}_i$, there exists an elementary vector $h \in \mathcal{E}(\mathbf{A})$ such that $\langle c, h\rangle \geq 0$, $h_i > 0$, $h_j \geq 0$ whenever $\bar{x}_j = 0$, and $\bar{x}^h_i(g) \geq x^{\mathrm{m}}_i(g)/n$.*

Definition 2.6 (Dominance)**.** Let $\bar{x}$ be a primal optimal solution to (LP). Let $i \in [n]$ and $\alpha \geq 0$. Given vectors $h, h' \in \ker(\mathbf{A})$ where $\langle c, h\rangle, \langle c, h'\rangle \geq 0$, we say that *$h$ α-dominates h' on i with respect to* $\bar{x}$ if $\bar{x}^h_i \geq \alpha \bar{x}^{h'}_i$. More generally, given sets $S, S' \subseteq \ker(\mathbf{A})$, we say that *$S$ α-dominates S' on i with respect to* $\bar{x}$ if $\langle c, h\rangle \geq 0$ for all $h \in S$, and for every $h' \in S'$ with $\langle c, h'\rangle \geq 0$, there exists $h \in S$ such that h α-dominates h' on i with respect to $\bar{x}$.

Definition 2.7 (Circuit cover)**.** Let $\bar{x}$ be a primal optimal solution to (LP). Let $i \in [n]$ and $\alpha \geq 0$. An *α-primal circuit cover of i with respect to* $\bar{x}$ is a set $S \subseteq \ker(\mathbf{A})$ which α-dominates $\mathcal{E}(\mathbf{A})$ on i with respect to $\bar{x}$.

The utility of a circuit cover is illustrated by the following lemma. Note that $x^{\mathrm{m}}_i(0)$ is the maximum value of the i-th coordinate in an optimal solution. Assuming $x^{\mathrm{m}}_i(0) < \infty$, there exists a (basic) optimal solution $\bar{x}$ such that $x^{\mathrm{m}}_i(0) = \bar{x}_i$.

Lemma 2.8. *Fix $i \in [n]$ such that $x^{\mathrm{m}}_i(0) < \infty$, and let $\bar{x}$ be a primal optimal solution to (LP) such that $\bar{x}_i = x^{\mathrm{m}}_i(0)$. If S is an α-primal circuit cover of i with respect to $\bar{x}$, then $\mathrm{SLC}_{\alpha/n}(x^{\mathrm{m}}_i) \leq |S| + 1$.*

3 MINIMUM-COST GENERALIZED FLOW

Let $G = (V, E)$ be a directed multigraph with arc capacities $u \in (\mathbb{R}_{++} \cup \{\infty\})^E$ and gain factors $\gamma \in \mathbb{R}^E_{++}$. A *flow* in G is any nonnegative vector $x \in \mathbb{R}^E_+$. Note that a flow is allowed to violate arc capacities. For a node $i \in V$, we denote $\delta^{\mathrm{in}}(i)$ and $\delta^{\mathrm{out}}(i)$ as the set

of incoming and outgoing arcs of i respectively. The *net flow* of x at node i is defined as

$$\nabla_i x := \sum_{e \in \delta^{\text{in}}(i)} \gamma_e x_e - \sum_{e \in \delta^{\text{out}}(i)} x_e \,.$$

Let $\nabla x \in \mathbb{R}^V$ denote the vector of net flows at every node in V. A flow x is a *circulation* if $\nabla x = \mathbf{0}$. For $i, j \in V$, we denote by $E_{i,j} \subseteq E$ the subset of arcs with tail i and head j.

An instance of the *minimum-cost generalized flow* problem is given by a directed multigraph $G = (V, E)$ with node demands $b \in \mathbb{R}^V$, arc costs $c \in \mathbb{R}^E$, capacities $u \in (\mathbb{R}_{++} \cup \{\infty\})^E$ and gain factors $\gamma \in \mathbb{R}^E_{++}$. It can be formulated as the following LP:

$$\min\{\langle c, x\rangle : \nabla x = b\,,\ \mathbf{0} \le x \le u\}\,. \tag{MGF}$$

Throughout this section, we will use n for the number of nodes of G and m for the number of arcs; *note that applied to* (MGF)*, this is the reverse of the convention used for general LPs.* Let $E_c \subseteq E$ denote the subset of arcs with finite capacities. We define $m_c := |E_c|$ for the number of finite capacity arcs.

We assume that (MGF) has a finite optimum, since otherwise the max central path does not exist; our initialization procedure will ensure that we only consider such instances. For an arc $e \in E$, we denote $x^{\mathfrak{m}}_e$ as the coordinate of the primal max central path which corresponds to the flow variable x_e. For a capacitated arc $e \in E_c$, we also denote $x^{\mathfrak{m}}_{\overleftarrow{e}}$ as the coordinate of the primal max central path which corresponds to the slack variable $u_e - x_e$. Our goal in this section is to prove the following bound on the SLC of each coordinate of the primal max central path.

Theorem 3.1. *Given an instance of minimum-cost generalized flow with a finite optimum, we have*

(i) $\text{SLC}_\eta(x^{\mathfrak{m}}_e) = O(nm^2(m_c + n)^2)$ *for every arc* $e \in E$*, and*
(ii) $\text{SLC}_\eta(x^{\mathfrak{m}}_{\overleftarrow{e}}) = O(nm^2(m_c + n)^2)$ *for every arc* $e \in E_c$

for some $\eta = \Omega(1/(m^2 n))$.

In the full version, we prove a stronger bound of $O(m(m_c + n))$ on the SLC.

It will be more convenient to work with the special case of (MGF) where $b = \mathbf{0}$ and $c \ge \mathbf{0}$. Note that $\mathbf{0}$ is trivially an optimal solution. One can show that it suffices to bound $\text{SLC}_\eta(x^{\mathfrak{m}}_e)$ for every arc e in this instance in order to prove Theorem 3.1. This is achieved by replacing the cost c with any optimal reduced cost, and considering the residual graph with respect to any optimal solution x^* to (MGF). Let F denote the set of finite capacity arcs in the reduced instance, and define $\bar{m} := |F|$. By picking x^* to be basic, one can further ensure that $\bar{m} \le m_c + n$. See the full version for more details.

3.1 SLC Bounds via Domination

We will follow exactly the general plan discussed in Section 2.1: we demonstrate the existence of a small primal circuit cover. Recall the notion of an elementary vector from Definition 2.2. The following is precisely this same notion, in the context of generalized circulations.

Definition 3.2 (Elementary circulation)**.** A nonzero circulation f in G is *elementary* if $\text{supp}(f)$ is inclusion-wise minimal, i.e., there is no nonzero circulation f' in G with $\text{supp}(f') \subsetneq \text{supp}(f)$.

In order to characterize elementary circulations, we need the following concepts.

Definition 3.3 (Walk, trail, path and cycle)**.** A *walk* is a sequence $W = (v_0, e_1, \ldots, e_\ell, v_\ell)$ where e_i is an arc from v_{i-1} to v_i for all $i \in [\ell]$. It is *closed* if $v_0 = v_\ell$, and *open* otherwise. If $e_i \ne e_j$ for all $i \ne j$, then it is called a *trail*. If $v_0 = v_\ell$ and $v_i \ne v_j$ for all $0 \le i < j \le \ell$, then it is called a *cycle at* v_0.

The *gain* of W is $\gamma(W) := \prod_{i=1}^{\ell} \gamma_{e_i}$, with the convention $\gamma(W) := 1$ if $\ell = 0$. We call W *flow-generating* if $\gamma(W) > 1$, *conservative* if $\gamma(W) = 1$, and *flow-absorbing* if $\gamma(W) < 1$.

For an r–s walk Q and an s–t walk W, we denote $Q \oplus W$ as the concatenated r–t walk. We use $V(W)$ and $E(W)$ to refer to the node set and arc set of a walk W.

Definition 3.4 (Objects)**.** A *flow-generating object at* $t \in V$ is a pair (C, W) where C is a flow-generating s–s walk for some $s \in V$ and W is an s–t walk. It is *simple* if C is a cycle, W is a path, and $V(C) \cap V(W) = \{s\}$.

A *flow-absorbing object at* $s \in V$ is a pair (W, D) where W is an s–t walk for some $t \in V$ and D is a flow-absorbing t–t walk. It is *simple* if W is a path, D is a cycle, and $V(W) \cap V(D) = \{t\}$.

A *conservative object* is a triple (C, W, D) where either

(i) for some $s, t \in V$, C is a flow-generating s–s walk, W is an s–t walk, and D is a flow-absorbing t–t walk; or
(ii) C is a conservative closed s–s walk for some s, W is the trivial path at s, and $D = C$.[4]

The object is *simple* in case (ii) if C is a cycle, and in case (i) if (C, W) and (W, D) are simple, and

- $V(C) \cap V(D) = \emptyset$ in the case that $E(W) \ne \emptyset$; or
- the intersection of C and D is a path, in the case that $E(W) = \emptyset$.

Given a flow-generating object $U_1 = (C_1, W_1)$ at s, an s-t walk Q, and a flow-absorbing object $U_2 = (C_2, W_2)$ at t, we also use $U_1 \oplus Q \oplus U_2$ to refer to the conservative object $(C_1, W_1 \oplus Q \oplus W_2, C_2)$.

Since an arc can be used multiple times in a walk or an object, we introduce the notion of recurrence to keep track of its multiplicity.

Definition 3.5 (Recurrence)**.** A walk W is called *k-recurrent* if every arc appears at most k times as a step in W. Similarly, an object U is *k-recurrent* if every arc appears at most k times in total as a step in some constituent walk of U.

Note that k-recurrent only upper bounds the number of repetitions of an arc; for $k \le \ell$, any k-recurrent walk is also ℓ-recurrent.

When considering flows supported on the arc set of a walk, it will be important to be able distinguish between flow on different "steps" of the walk that involve the same arc of the graph. We do this formally by defining the "splitting" of a walk, which simply makes parallel copies of arcs to turn the walk into a corresponding trail.

Definition 3.6 (Splitting)**.** Let $\tilde{G} = (V, \tilde{E})$ be the directed multigraph with the same node set as G, but with $10n$ parallel copies of each arc, each with the same gain factor, cost and capacity as the corresponding arc in G. (The choice of $10n$ is just to be sufficiently large for our purposes.) For each $e \in E$, we use $e^1, e^2, \ldots, e^{10n}$ to index the corresponding copies in $\tilde{E}$.

[4] This may look somewhat strange, but it essentially views a conservative cycle as a degenerate bicycle; this will be convenient in covering all cases with a single argument.

Given a $10n$-recurrent walk $W = (v_0, e_1, v_1, e_2, \dots, e_r, v_r)$ in G, we define a *splitting* of W to be a trail $\tilde{W} = (v_0, e_1^{\sigma_1}, v_2, e_2^{\sigma_2}, \dots, e_r^{\sigma_r}, v_r)$ in $\tilde{G}$, where for each $i \neq j$ with $e_i = e_j$, $\sigma_i \neq \sigma_j$.

Given an object U, a splitting of U is a tuple of trails in $\tilde{G}$, each being a splitting of the corresponding walk in U, and where in addition the trails are arc-disjoint.

Note that up to trivial relabelling of copies of arcs, the splitting of a walk or object is unique.

For an object or walk U, we use $V(U)$ and $E(U)$ to denote its node set and arc set respectively, and also use $E(\tilde{U}) \subseteq \tilde{E}$ to denote the arc set of a splitting $\tilde{U}$.

Definition 3.7 (Induced flows)**.** Given an s–t walk W with splitting $\tilde{W}$, we say that $\tilde{x} \in \mathbb{R}_+^{\tilde{E}}$ is a *flow induced by* $\tilde{W}$ if $\tilde{x}$ is nonzero, supported on $E(\tilde{W})$, and $\gamma_e \tilde{x}_e = \tilde{x}_f$ for any pair of consecutive arcs $e, f \in E(\tilde{W})$ where e comes before f in $\tilde{W}$. We say that $\bar{x} \in \mathbb{R}_+^E$ is a *flow induced by* W if $\bar{x}$ is the projection onto G of a flow $\tilde{x}$ induced by a splitting of W, that is, $\bar{x}_e = \sum_j \tilde{x}_{e^j}$.

Given a flow-generating object $U = (C, W)$ at t, a flow induced by a splitting $\tilde{U} = (\tilde{C}, \tilde{W})$ is a vector $\tilde{x} \in \mathbb{R}_+^{\tilde{E}}$ that can be written as a sum of a flow induced by $\tilde{C}$ and a flow induced by $\tilde{W}$, and where in addition $\nabla_v \tilde{x} = 0$ for all $v \neq t$. The definition for flow-absorbing objects is completely analogous; and for a conservative object $U = (C, W, D)$, $\tilde{x}$ should satisfy $\nabla \tilde{x} = 0$, and be a sum of flows induced by the components of a splitting of U.

We are ready to characterize elementary circulations.

Lemma 3.8. *A flow is an elementary circulation if and only if it is induced by a simple conservative object.*

We remark that all flows induced by an object are the same up to scaling. The following will be a crucial notion: it is the largest possible flow induced by a walk (or object), with the property that on each step of the walk or object, the flow does not exceed the capacity of the arc, and the cost of that step (flow times arc cost) does not exceed a given bound λ.

Definition 3.9. Let W be a walk, with $\tilde{W}$ a splitting of W and $\tilde{x}$ a flow induced by $\tilde{W}$. Define $x^{\tilde{W}} : \mathbb{R}_+ \to \mathbb{R}_+^{\tilde{E}}$ to be the function that maps $x^{\tilde{W}}(\lambda)$ to the largest scaling of $\tilde{x}$ so that $\tilde{x}_a \leq u_a$ and $c_a \tilde{x}_a \leq \lambda$ for each $a \in E(\tilde{W})$. Then let $x^W : \mathbb{R}_+ \to \mathbb{R}_+^E$ be the projection of $x^{\tilde{W}}$ onto G, i.e., $x_e^W(\lambda) = \sum_j x_{e^j}^{\tilde{W}}(\lambda)$.

We define $x^{\tilde{U}}(\lambda)$ and $x^U(\lambda)$ for an object U with splitting $\tilde{U}$ in identical fashion.

Note that $x^W(\lambda)$ and $x^U(\lambda)$ do not depend on the choice of splitting, and so are well-defined.

Remark 3.10. This definition is closely related to the definition of h-curves for general LPs provided in Definition 2.4. It is more general, in that we define x^U for objects that are not conservative, and hence which do not lie in the kernel. If we consider a conservative object U, and take h to be a flow induced by U, then x^U and the h-curve $\mathbf{0}^h$ are "close": if U is k-recurrent, then $\frac{1}{km} x^U(\lambda) \leq \mathbf{0}^h(\lambda) \leq x^U(\lambda)$. The reason that they are not identical, only within a factor km, is simply because of the per-step nature of the capacity bounds (meaning $x^U(\lambda)$ might overload an arc by a factor k) and cost bounds (meaning $x^U(\lambda)$ could have total cost $km\lambda$, given each arc could in principle contribute a cost of $k\lambda$).

Let $\mathcal{E}$ denote the collection of simple conservative objects. For any $e \in E$ and collection $\mathcal{U}$ of conservative objects, we define $x_e^{\mathcal{U}} := \bigvee_{U \in \mathcal{U}} x_e^U$. The following is essentially Lemma 2.8 for this setting, taking into account the scaling necessary to make $x^U(\lambda)$ feasible for cost bound λ.

Lemma 3.11. *Fix any edge $e \in E$. Suppose that $\mathcal{D}$ is a collection of k-recurrent conservative objects that α-dominates $\mathcal{E}$ on e, in that $x_e^{\mathcal{D}} \geq \alpha x_e^{\mathcal{E}}$ for some constant α. Then $\mathrm{SLC}_{\alpha/(m^2k)}(x_e^{\mathfrak{m}}) \leq |\mathcal{D}|$.*

As such, our goal is now to demonstrate such a dominating collection $\mathcal{D}$; we will do this with $\alpha = 1$, $k = O(n)$ and $|\mathcal{D}| = O(nm^2\bar{m}^2)$.

3.2 Path Domination

While our goal is to dominate simple conservative objects, we build up to this in stages. Our first step is to build a small collection of s–t walks that dominate all s–t paths; these will become building blocks in the next section.

Definition 3.12. Given two distinct nodes s and t, and an s–t walk W, let

$$\vec{f}_W(\lambda, r) := \min\big(\nabla_t x^W(\lambda), \gamma(W) r\big). \tag{5}$$

In words, $\vec{f}_W(\lambda, r)$ is the maximum amount of flow that can be sent to t with a flow induced by W, given that each step respects the cost and capacity bounds, and that there are only r units available at s to be sent.

The function $\vec{f}_W$ for a given s–t walk W has a very simple form. We can write it as

$$\vec{f}_W(\lambda, r) = \min\{\lambda/\mathrm{cost}(W), \gamma(W) r, \mathrm{limit}(W)\},$$

where $\mathrm{cost}(W)$ is the largest cost of a step of the walk per unit of flow measured at t; $\gamma(W)$ is the gain of the walk; and $\mathrm{limit}(W)$ is the maximum amount of flow that can arrive at t given that each step respects the capacity.

For walks, we use the following stronger "bivariate" notion of domination. This will be crucial when we come to use this to demonstrate domination, in the usual sense, for flow-generating/flow-absorbing objects and eventually conservative objects in the following sections.

Definition 3.13. We say that an s–t walk W' *dominates* an s–t walk W if $\gamma(W') \geq \gamma(W)$ and $\nabla_t x^{W'} \geq \nabla_t x^W$.

If W' dominates W, then by (5), clearly $\vec{f}_{W'} \geq \vec{f}_W$ (i.e., $\vec{f}_{W'}(\lambda, r) \geq \vec{f}_W(\lambda, r)$ for all λ and r); indeed this is easily seen to be equivalent.

Write $\mathcal{P}(s, t)$ for the set of all s–t paths for any distinct $s, t \in V$. Given a collection $\mathcal{W}$ of s–t walks, define

$$\vec{f}_{\mathcal{W}} := \bigvee_{W \in \mathcal{W}} \vec{f}_W.$$

The main theorem of this section is the following. It shows that the collection of s–t paths can be dominated by a small collection of n-recurrent s–t walks: for any s–t path P, any cost bound λ, and any amount of flow r available at s, there is a walk in the collection that does a better job at sending flow to t under the same cost and flow restrictions.

Theorem 3.14. *Fix distinct nodes s and t. Then there is an $O(m\bar{m})$-sized collection $\mathcal{W}$ of n-recurrent s–t walks such that for every $P \in \mathcal{P}(s,t)$, there is a walk $W \in \mathcal{W}$ that dominates P. Hence $\vec{f}_{\mathcal{W}} \geq \vec{f}_{\mathcal{P}(s,t)}$.*

The remainder of this section will be devoted to sketching the proof of Theorem 3.14. First, we need a few definitions.

Definition 3.15 (Bottlenecks, signature and backbone of a walk)**.** Consider an s–t walk W with at least one step, and let $\tilde{x}$ be a flow induced by a splitting $\tilde{W}$.

Define the *cost bottleneck step* of $\tilde{W}$ to be the arc $a_c \in E(\tilde{W})$ for which $c_{a_c}\tilde{x}_{a_c}$ is maximal, breaking ties towards steps closer to t. The *cost bottleneck* of W is then the arc $e_c \in E$ that corresponds to a_c. Similarly, define the *flow bottleneck step* of $\tilde{W}$ to be the arc $a_f \in E(\tilde{W})$ for which $\tilde{x}_{a_f}/u_{a_f}$ is maximal, breaking ties towards arcs closer to t; *exceptionally*, if all arcs of W have infinite capacity, set $a_f = a_c$. Again the *flow bottleneck* of W is the arc $e_f \in E$ that corresponds to a_f.

The *signature* of W is

$$\sigma(W) := \begin{cases} (e_c, e_f, \preceq), & \text{if } a_c \text{ is earlier in the walk than } a_f, \text{ or } a_c = a_f \\ (e_c, e_f, \succ), & \text{otherwise.} \end{cases}$$

The *backbone* of W, denoted $\beta(W)$, is the subwalk of W that starts and ends with the bottleneck steps (including the bottleneck steps). We also write $\tau(W)$ for the subwalk of W before $\beta(W)$, and $\eta(W)$ for the subwalk after $\beta(W)$; that is,

$$W = \tau(W) \oplus \beta(W) \oplus \eta(W).$$

If $\tilde{W}$ is a splitting of W, we also define the precisely corresponding partition into subtrails,

$$\tilde{W} = \tau(\tilde{W}) \oplus \beta(\tilde{W}) \oplus \eta(\tilde{W}).$$

For any walk W, it is easy to verify that $\sigma(\beta(W)) = \sigma(W)$.

Definition 3.16. We say that a path P is *σ-capped* if $\sigma(P) = \sigma$ and $\beta(P) = P$.

For each signature σ, let $S(\sigma)$ be any highest gain path amongst all σ-capped paths. For any signature σ, say that a walk $\bar{L}$ is a *left σ-extension* if it starts from s, contains $S(\sigma)$ as a suffix, has signature σ, and $\bar{L} = \tau(\bar{L}) \oplus S(\sigma)$. In other words, if we consider a splitting $\tilde{L}$ of $\bar{L}$, it contains a splitting $\tilde{S}$ of $S(\sigma)$ as its suffix, and the cost and flow bottleneck steps are the first and last arcs of $\tilde{S}$ (in the order specified by the signature). We similarly define a walk $\bar{R}$ to be a *right σ-extension* if it ends at t, contains $S(\sigma)$ as a prefix, has signature σ, and $\bar{R} = S(\sigma) \oplus \eta(\bar{R})$.

We are now ready to define the dominating collection $\mathcal{W}$ of s–t walks. For any signature σ, define $L(\sigma)$ to be a highest-gain $(n-2)$-recurrent walk such that $L(\sigma) \oplus S(\sigma)$ is a left σ-extension, as long as at least one such walk exists; if not, $L(\sigma)$ is undefined. Similarly, let $R(\sigma)$ be a highest-gain *path* such that $S(\sigma) \oplus R(\sigma)$ is a right σ-extension, or undefined if there are none such. Let Σ be the collection of all signatures for which $L(\sigma)$ and $R(\sigma)$ are both defined; note that

$$\Sigma \subseteq (E \times F \times \{\preceq, \succ\}) \cup \{(e, e, \preceq) : e \in E \setminus F\},$$

taking into account the possibility of walks consisting only of infinite capacity arcs.

Now define $W(\sigma) := L(\sigma) \oplus S(\sigma) \oplus R(\sigma)$ for each $\sigma \in \Sigma$, and $\mathcal{W} := \{W(\sigma) : \sigma \in \Sigma\}$. It is easy to see that $\sigma(W(\sigma)) = \sigma$. Clearly, $|\mathcal{W}| \leq |\Sigma| \leq m(2\bar{m}+1)$.

It is left to prove that $\mathcal{W}$ dominates the set of s–t paths. We show this using the following patching operation. Given an s–t walk W with signature σ, define patch(W) to be the s–t walk obtained from W by replacing $\beta(W)$ with $S(\sigma)$. (Note that we do not care about computing patch(W); all of this is purely existential.)

Lemma 3.17 (Patching a walk)**.** *Suppose W is an s–t walk whose backbone $\beta(W)$ is a path, and let $W' =$ patch(W). Then W' dominates W. Furthermore, if $\sigma(W') \neq \sigma(W)$, then $\eta(W')$ is a* strict *suffix of $\eta(W)$.*

We briefly sketch the key idea of the proof. Let σ be the signature of W, and assume that it has the form $\sigma = (e_c, e_f, \preceq)$; the other case is analogous. Consider $W' =$ patch(W), obtained by replacing $\beta(W)$ with $S(\sigma)$, the highest gain path starting with e_c and ending in e_f whose cost bottleneck is e_c and flow bottleneck is e_f. For convenience, let us assume that W and W' are paths (there are no additional conceptual difficulties when they are not paths, but more notational care is needed). Fix some λ, and consider $x := x^W(\lambda)$, the largest flow induced by W that doesn't violate the per-step capacity and cost bounds. Let x' be the flow induced by W' for which the same amount of flow arrives at t, i.e., $\nabla_t x' = \nabla_t x$. Since W and W' agree after the flow bottleneck e_f, x' and x are the same on these arcs; in particular, $x'_{e_f} = x_{e_f}$. Given that $\gamma(S(\sigma)) \geq \gamma(\beta(W))$ by the definition of $S(\sigma)$, $x'_{e_c} \leq x_{e_c}$. Since $S(\sigma)$ has signature σ, and x' satisfies the cost and capacity constraints on the bottleneck arcs, it satisfies these constraints on all arcs of $S(\sigma)$. Finally, since W and W' agree on the segment from s to e_c, x' is no larger than x on all of these arcs. So we can feasibly have as much flow arriving using W' instead of W, while sending the same or less flow from s. This shows that $\vec{f}_{W'}(\lambda, r) \geq \vec{f}_W(\lambda, r)$ for all λ and r.

The signature of W' can differ from W, but the reason for this will be that e_c is no longer the cost bottleneck (since it can receive proportionally less flow compared to what arrives at the sink). In this case the new cost bottleneck will necessarily be later in the walk. It cannot be in $S(\sigma)$, since $S(\sigma)$ has signature σ, and so must lie in $\eta(W)$, ensuring that $\eta(W')$ is a strict suffix of $\eta(W)$.

Now, let P be an arbitrary s–t path. Starting from $W^{(1)} := P$, we construct a sequence $W^{(1)}, W^{(2)}, \ldots, W^{(\ell)}$ of s–t walks by setting $W^{(i+1)} =$ patch$(W^{(i)})$, stopping once $\sigma(W^{(i+1)}) = \sigma(W^{(i)})$. By Lemma 3.17, $W^{(i+1)}$ dominates $W^{(i)}$. Furthermore, as long as $\sigma(W^{(i+1)}) \neq \sigma(W^{(i)})$, $\eta(W^{(i+1)})$ is a strict subwalk of $\eta(W^{(i)})$. Since we started with a path, it follows that $\ell \leq n$. Consequently, $W^{(\ell)}$ is n-recurrent because each patching operation increases the recurrence by at most 1. The proof of Theorem 3.14 is completed by showing that $W^{(\ell)}$ is dominated by $W(\sigma') \in \mathcal{W}$, where $\sigma' = \sigma(W^{(\ell)})$.

3.3 Dominating Simple Flow-Generating and Flow-Absorbing Objects

In this section, we use the domination result for paths in the previous section to dominate the set of flow-generating objects and flow-absorbing objects at a node t.

Given a flow-generating object U at t, we will be interested in $\nabla x_t^U(\lambda)$, the maximum amount of flow that can be generated at t using U, subject to the cost and capacity bounds. So define $f_U^+(\lambda) := \nabla x_t^U(\lambda)$ for all $\lambda \geq 0$, and as usual, $f_{\mathcal{U}}^+ := \bigvee_{U\in\mathcal{U}} f_U^+$ for a collection of flow-generating objects at t. Given two flow-generating objects at t, U and U', we will say that U' dominates U if $f_{U'}^+ \geq f_U^+$; and similarly for collections of objects.

For a flow-absorbing object U at s and a collection $\mathcal{U}$ of such objects, we define the analogous notions $f_U^-(\lambda) := -\nabla x_s^U(\lambda)$ and $f_{\mathcal{U}}^- := \bigvee_{U\in\mathcal{U}} f_U^-$.

Flow-generating cycles at s. Given an s–s walk R, define $f_R^+(\lambda) := \nabla x_s^R(\lambda)$; that is, $f_R^+ = f_{(R,\{s\})}^+$. Define $f_{\mathcal{R}}^+$ for a collection of s–s walks $\mathcal{R}$ in the usual way as $f_{\mathcal{R}}^+ := \bigvee_{R\in\mathcal{R}} f_R^+$. Let $\mathcal{C}^+(s)$ denote the collection of flow-generating cycles at s.

Theorem 3.18. *There is an $O(m\bar{m})$-sized collection $\mathcal{R}$ of $(n+1)$-recurrent flow-generating s–s walks such that $f_{\mathcal{R}}^+ \geq f_{\mathcal{C}^+(s)}^+$.*

The collection $\mathcal{R}$ is obtained by applying Theorem 3.14 to an auxiliary graph G', obtained from the original graph G by adding a new node s' and redirecting all the incoming arcs of s to s', turning cycles at s to s–s' paths. Given such a path, and an s–s' walk dominating this cycle, domination means that we can send the same amount of flow from s along the walk and receive more at s', without paying a larger per-stop cost or violating the per-step capacity bound. Projecting this back to G yields an s–s walk which generates more excess at s than the original cycle.

Flow-generating objects at t. For given nodes s and t, let $\mathcal{G}(s,t)$ denote the collection of simple flow-generating objects at t consisting of a flow-generating cycle at s, followed by an s–t path. Let $\mathcal{G}(t) := \bigcup_{s\in V} \mathcal{G}(s,t)$, be the collection of all simple flow-generating objects at t.

Lemma 3.19. *For any $s,t \in V$, there is an $O(m^2\bar{m}^2)$-sized collection $\mathcal{H}$ of $O(n)$-recurrent flow-generating objects, each consisting of a flow-generating s-s walk along with an s–t walk, such that $f_{\mathcal{H}}^+ \geq f_{\mathcal{G}(s,t)}^+$.*

Let $\mathcal{R}$ be the collection of flow-generating s–s walks guaranteed by Theorem 3.18, and let $\mathcal{W}$ be the collection of s–t walks guaranteed by Theorem 3.14. The collection $\mathcal{H}$ of flow-generating objects is obtained by taking all possible combinations of walks in $\mathcal{R}$ and $\mathcal{W}$, i.e., $\mathcal{H} := \{(R,W) : R \in \mathcal{R}, W \in \mathcal{W}\}$. Observe that for any flow-generating s–s walk R and any s–t walk W, the flow-generating object $U := (R,W)$ satisfies $f_U^+(\lambda) = \vec{f}_W(\lambda, f_R^+(\lambda))$. Therefore, $f_{\mathcal{H}}^+ \geq f_{\mathcal{G}(s,t)}^+$ by our choice of $\mathcal{R}$ and $\mathcal{W}$.

By taking the union of the dominating collections for $\mathcal{G}(s,t)$ for each $s \in V$, we immediately get a dominating set of size $O(nm^2\bar{m}^2)$ for $\mathcal{G}(t)$.

Theorem 3.20. *For any $t \in V$, there is an $O(nm^2\bar{m}^2)$-sized collection $\mathcal{H}$ of $O(n)$-recurrent flow-generating objects at t such that $f_{\mathcal{H}}^+ \geq f_{\mathcal{G}(t)}^+$.*

Absorbing versions. A completely symmetric version of the above concerns, instead of the maximum excess we can generate at t, the maximum deficit we can create at t. Let $\mathcal{A}(t)$ the collection of all simple flow-absorbing objects at t.

Theorem 3.21. *For any $t \in V$, there is an $O(nm^2\bar{m}^2)$-sized collection $\mathcal{B}$ of $O(n)$-recurrent flow-absorbing objects at t such that $f_{\mathcal{B}}^- \geq f_{\mathcal{A}(t)}^-$.*

3.4 Dominating Simple Conservative Objects

We are now ready to prove the main domination theorem for simple conservative objects.

Theorem 3.22. *Let $K \in \mathbb{Z}_+$ such that for every node s, there exist*

(i) a collection $\mathcal{H}_s$ of $O(n)$-recurrent flow-generating objects at s where $f_{\mathcal{H}_s}^+ \geq f_{\mathcal{G}(s)}^+$ and $|\mathcal{H}_s| \leq K$; and

(ii) a collection $\mathcal{B}_s$ of $O(n)$-recurrent flow-absorbing objects at s where $f_{\mathcal{B}_s}^- \geq f_{\mathcal{A}(s)}^-$ and $|\mathcal{B}_s| \leq K$.

Then for any arc e, there is an $O(m\bar{m}+K)$-sized collection $\mathcal{D}$ of $O(n)$-recurrent conservative objects such that $x_e^{\mathcal{D}} \geq x_e^{\mathcal{E}}/8$.

Combined with Theorem 3.20, Theorem 3.21 and Lemma 3.11, this gives a bound of $O(nm^2\bar{m}^2)$ on the SLC with neighborhood size $\eta = \Omega(1/(m^2n))$, proving Theorem 3.1.

Fix an arc e, with tail s and head t. In what follows, we only sketch the proof for dominating the collection $\mathcal{E}_p \subseteq \mathcal{E}$ of simple conservative objects $C = (C_g, C_p, C_a)$ in which $e \in E(C_p)$. The case where e lies on the flow-generating or flow-absorbing cycle is more involved. We refer the reader to the full version for the complete proof.

Any $C \in \mathcal{E}_p$ can be viewed as the composition of a flow-generating object $U_g = (C_g, P_1)$ at s, the arc e, and a flow-absorbing object $U_a = (P_2, C_a)$ at t (so $C_p = P_1 \oplus e \oplus P_2$). But this suggests a straightforward choice of a dominating collection: take the collection $\mathcal{H}_s$ of flow-generating objects at s which dominates all simple flow-generating objects at s, and the collection $\mathcal{B}_t$ of flow-absorbing objects at t which dominates (in the sense of Theorem 3.21) all simple flow-absorbing objects at t, to form a collection $\mathcal{Q}$ of conservative objects given by

$$\mathcal{Q} := \{U_g' \oplus e \oplus U_a' : U_g' \in \mathcal{H}_s, U_a' \in \mathcal{B}_t\}.$$

This does the job; for any cost bound λ, there is some $U_g' \in \mathcal{H}_s$ that can create at least as much excess at s as U_g; similarly some $U_a' \in \mathcal{B}_t$ can get rid of at least as much flow at t as U_a; and so $U_g' \oplus e \oplus U_a' \in \mathcal{Q}$ can send at least as much flow through e as C.

ACKNOWLEDGMENTS

This project has received funding from the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme (grant agreement nos. 757481–ScaleOpt and 805241–QIP), and from the Dutch Science Foundation (NWO) under the NWO Talent Programme (grant 016.Vidi.189.087).

Part of this work was done during the ICERM programme "Combinatorics and Optimization" and the Simons Institute programme "Data Structures and Optimization for Fast Algorithms".

REFERENCES

[1] Ilan Adler and Steven Cosares. 1991. A strongly polynomial algorithm for a special class of linear programs. *Oper. Res.* 39, 6 (1991), 955–960. https://doi.org/10.1287/OPRE.39.6.955

[2] Ravindra K Ahuja, Thomas L Magnanti, and James B Orlin. 1993. *Network Flows: Theory, Algorithms, and Applications.* Prentice Hall.

[3] Xavier Allamigeon, Pascal Benchimol, Stéphane Gaubert, and Michael Joswig. 2018. Log-barrier interior point methods are not strongly polynomial. *SIAM J. Appl. Algebra Geom.* 2, 1 (2018), 140–178. https://doi.org/10.1137/17M1142132

[4] Xavier Allamigeon, Daniel Dadush, Georg Loho, Bento Natura, and László A. Végh. 2022. Interior point methods are not worse than Simplex. https://doi.org/10.1109/FOCS54457.2022.00032

[5] Xavier Allamigeon, Daniel Dadush, Georg Loho, Bento Natura, and László A Végh. 2024. Interior point methods are not worse than Simplex. arXiv:2206.08810v3 [math.OC]

[6] Xavier Allamigeon, Stéphane Gaubert, and Nicolas Vandame. 2022. No self-concordant barrier interior point method is strongly polynomial. In *Proc. ACM STOC*. 515–528. https://doi.org/10.1145/3519935.3519997

[7] Lenore Blum, Mike Shub, and Stephen Smale. 1989. On a theory of computation and complexity over the real numbers: NP-completeness, recursive functions and universal machines. *Bull. Amer. Math. Soc.* 21, 1 (1989), 1–46. https://doi.org/10.1142/9789812792839_0013

[8] Jan Van Den Brand, Li Chen, Richard Peng, Rasmus Kyng, Yang P. Liu, Maximilian Probst Gutenberg, Sushant Sachdeva, and Aaron Sidford. 2023. A deterministic almost-linear time algorithm for minimum-cost flow. In *Proc. IEEE FOCS*. https://doi.org/10.1109/focs57990.2023.00037

[9] Li Chen, Rasmus Kyng, Yang P Liu, Richard Peng, Maximilian Probst Gutenberg, and Sushant Sachdeva. 2022. Maximum flow and minimum-cost flow in almost-linear time. In *Proc. IEEE FOCS*. 612–623. https://doi.org/10.1109/FOCS54457.2022.00064

[10] Edith Cohen and Nimrod Megiddo. 1994. Improved algorithms for linear inequalities with two variables per inequality. *SIAM J. Comput.* 23, 6 (1994), 1313–1347. https://doi.org/10.1137/S0097539791256325

[11] Edith Cohen and Nimrod Megiddo. 1994. New algorithms for generalized network flows. *Math. Program.* 64, 1–3 (1994), 325–336. https://doi.org/10.1007/bf01582579

[12] Michael B Cohen, Yin Tat Lee, and Zhao Song. 2019. Solving linear programs in the current matrix multiplication time. In *Proc. ACM STOC*. 938–942. https://doi.org/10.1145/3313276.3316303

[13] Daniel Dadush, Sophie Huiberts, Bento Natura, and László A. Végh. 2023. A scaling-invariant algorithm for linear programming whose running time depends only on the constraint matrix. *Math. Program.* (2023). https://doi.org/10.1007/s10107-023-01956-2 (in press).

[14] Daniel Dadush, Zhuan Khye Koh, Bento Natura, and László A Végh. 2022. An accelerated Newton–Dinkelbach method and its application to two variables per inequality systems. *Math. Oper. Res.* (2022). https://doi.org/10.1287/moor.2022.1326

[15] Daniel Dadush, Bento Natura, and László A. Végh. 2020. Revisiting Tardos's framework for linear programming: faster exact solutions using approximate solvers. In *Proc. IEEE FOCS*. https://doi.org/10.1109/focs46700.2020.00091

[16] Samuel I Daitch and Daniel A Spielman. 2008. Faster approximate lossy generalized flow via interior point algorithms. In *Proc. ACM STOC*. 451–460. https://doi.org/10.1145/1374376.1374441

[17] Jack Edmonds. 1967. Systems of distinct representatives and linear algebra. *J. Res. Nat. Bur. Standards Sect. B* 71, 4 (1967), 241–245. https://doi.org/10.6028/jres.071B.033

[18] DR Fulkerson. 1968. Networks, frames, blocking systems. *Mathematics of the Decision Sciences, Part I, Lectures in Applied Mathematics* 2 (1968), 303–334.

[19] Mehrdad Ghadiri, Richard Peng, and Santosh S Vempala. 2023. The bit complexity of efficient continuous optimization. In *Proc. IEEE FOCS*. https://doi.org/10.1109/FOCS57990.2023.00125

[20] Andrew V. Goldberg, Serge A. Plotkin, and Éva Tardos. 1991. Combinatorial algorithms for the generalized circulation problem. *Math. Oper. Res.* 16, 2 (1991), 351–381. https://doi.org/10.1287/moor.16.2.351

[21] Donald Goldfarb, Zhiying Jin, and James B. Orlin. 1997. Polynomial-time highest-gain augmenting path algorithms for the generalized circulation problem. *Math. Oper. Res.* 22, 4 (1997), 793–802. https://doi.org/10.1287/moor.22.4.793

[22] Martin Grötschel, László Lovász, and Alexander Schrijver. 1988. *Geometric algorithms and combinatorial optimization*. Springer. https://doi.org/10.1007/978-3-642-97881-4

[23] Dorit S. Hochbaum. 2004. Monotonizing linear programs with up to two nonzeroes per column. *Oper. Res. Lett.* 32, 1 (2004), 49–58. https://doi.org/10.1016/s0167-6377(03)00074-9

[24] Dorit S. Hochbaum and Joseph Naor. 1994. Simple and fast algorithms for linear and integer programs with two variables per inequality. *SIAM J. Comput.* 23, 6 (1994), 1179–1192. https://doi.org/10.1137/S0097539793251876

[25] Shunhua Jiang, Zhao Song, Omri Weinstein, and Hengjie Zhang. 2021. A faster algorithm for solving general LPs. In *Proc. ACM STOC*. https://doi.org/10.1145/3406325.3451058

[26] L. V. Kantorovich. 1939. Mathematical methods of organizing and planning production. *Publication House of the Leningrad State University* (1939), 68. https://doi.org/10.1287/mnsc.6.4.366 English translation in *Management Science* 6(4):366-422, 1960..

[27] Adam Karczmarz. 2022. Improved strongly polynomial algorithms for deterministic MDPs, 2VPI feasibility, and discounted all-pairs shortest paths. In *Proc. ACM-SIAM SODA*. 154–172. https://doi.org/10.1137/1.9781611977073.8

[28] Narendra Karmarkar. 1984. A new polynomial-time algorithm for linear programming. In *Proc. ACM STOC*. 302–311. https://doi.org/10.1007/BF02579150

[29] Leonid G Khachiyan. 1979. A polynomial algorithm in linear programming. In *Doklady Academii Nauk SSSR*, Vol. 244. 1093–1096. https://doi.org/10.1016/0041-5553(80)90061-0

[30] Guanghui Lan, Renato DC Monteiro, and Takashi Tsuchiya. 2009. A polynomial predictor-corrector trust-region algorithm for linear programming. *SIAM J. Optim.* 19, 4 (2009), 1918–1946. https://doi.org/10.1137/070693461

[31] Yin Tat Lee and Aaron Sidford. 2014. Path finding methods for linear programming: Solving linear programs in $\tilde{O}(\sqrt{\text{rank}})$ iterations and faster algorithms for maximum flow, Part II. In *Proc. IEEE FOCS*. 424–433. https://doi.org/10.1109/focs.2014.52

[32] Nimrod Megiddo. 1983. Towards a genuinely polynomial algorithm for linear programming. *SIAM J. Comput.* 12, 2 (1983), 347–353. https://doi.org/10.1137/0212022

[33] Nimrod Megiddo, Shinji Mizuno, and Takashi Tsuchiya. 1998. A modified layered-step interior-point algorithm for linear programming. *Math. Program.* 82, 3 (1998), 339–355. https://doi.org/10.1007/BF01580074

[34] Renato D. C. Monteiro and Takashi Tsuchiya. 2003. A variant of the Vavasis-Ye layered-step interior-point algorithm for linear programming. *SIAM J. Optim.* 13, 4 (2003), 1054–1079. https://doi.org/10.1137/S1052623401388926

[35] Renato D. C. Monteiro and Takashi Tsuchiya. 2005. A new iteration-complexity bound for the MTY predictor-corrector Algorithm. *SIAM J. Optim.* 15, 2 (2005), 319–347. https://doi.org/10.1137/S1052623402416803

[36] Carolyn Haibt Norton, Serge A Plotkin, and Éva Tardos. 1992. Using separation algorithms in fixed dimension. *J. Algorithms* 13, 1 (1992), 79–98. https://doi.org/10.1016/0196-6774(92)90006-X

[37] Neil Olver and László A Végh. 2020. A simpler and faster strongly polynomial algorithm for generalized flow maximization. *J. ACM* 67, 2 (2020), 1–26. https://doi.org/10.1145/3055399.3055439

[38] Tomasz Radzik. 2004. Improving time bounds on maximum generalised flow computations by contracting the network. *Theor. Comput. Sci.* 312, 1 (2004), 75–97. https://doi.org/10.1016/s0304-3975(03)00403-1

[39] Mateo Restrepo and David P. Williamson. 2007. A simple GAP-canceling algorithm for the generalized maximum flow problem. *Math. Program.* 118, 1 (2007), 47–74. https://doi.org/10.1007/s10107-007-0183-8

[40] R. Tyrrell Rockafellar. 1969. The elementary vectors of a subspace of R^N. In *Combinatorial Mathematics and Its Applications: Proceedings North Carolina Conference, Chapel Hill, 1967*. The University of North Carolina Press, 104–127.

[41] Stephen Smale. 1998. Mathematical problems for the next century. *The Mathematical Intelligencer* 20, 2 (1998), 7–15. https://doi.org/10.1007/BF03025291

[42] Éva Tardos. 1985. A strongly polynomial minimum cost circulation algorithm. *Combinatorica* 5, 3 (1985), 247–255. https://doi.org/10.1007/BF02579369

[43] Éva Tardos. 1986. A strongly polynomial algorithm to solve combinatorial linear programs. *Oper. Res.* (1986), 250–256. https://doi.org/10.1287/opre.34.2.250

[44] Éva Tardos and Kevin D. Wayne. 1998. Simple Generalized Maximum Flow Algorithms. In *Proc. IPCO*. 310–324. https://doi.org/10.1007/3-540-69346-7_24

[45] Pravin M Vaidya. 1989. Speeding-up linear programming using fast matrix multiplication. In *Proc. IEEE FOCS*. 332–337. https://doi.org/10.1109/SFCS.1989.63499

[46] Jan van den Brand. 2020. A deterministic linear program solver in current matrix multiplication time. In *Proc. ACM-SIAM SODA*. 259–278.

[47] Jan van den Brand, Yin Tat Lee, Aaron Sidford, and Zhao Song. 2020. Solving tall dense linear programs in nearly linear time. arXiv:2002.02304 [cs.DS]

[48] Jan van den Brand and Daniel Zhang. 2023. Faster high accuracy multi-commodity flow from single-commodity techniques. arXiv:2304.12992 [cs.DS]

[49] Stephen A. Vavasis and Yinyu Ye. 1996. A primal-dual interior point method whose running time depends only on the constraint matrix. *Math. Program.* 74, 1 (1996), 79–120. https://doi.org/10.1007/BF02592148

[50] László A. Végh. 2017. A strongly polynomial algorithm for generalized flow maximization. *Math. Oper. Res.* 42, 2 (2017), 179–211. https://doi.org/10.1287/moor.2016.0800

[51] Kevin D. Wayne. 2002. A polynomial combinatorial algorithm for generalized minimum cost flow. *Math. Oper. Res.* 27, 3 (2002), 445–459. https://doi.org/10.1287/moor.27.3.445.313

[52] Yinyu Ye, Michael J. Todd, and Shinji Mizuno. 1994. An $O(\sqrt{nL})$-iteration homogeneous and self-dual linear programming algorithm. *Math. Oper. Res.* 19, 1 (1994), 53–67. https://doi.org/10.1287/moor.19.1.53

[53] Manru Zong, Yin Tat Lee, and Man-Chung Yue. 2023. Short-step methods are not strongly polynomial-time. *Math. Program.* (2023). https://doi.org/10.1007/s10107-023-02002-x

Received 13-NOV-2023; accepted 2024-02-11