2015 IEEE 56th Annual Symposium on Foundations of Computer Science

A Faster Cutting Plane Method and its Implications for Combinatorial and Convex Optimization

Yin Tat Lee
Department of Mathematics
MIT
Cambridge, USA
Email: yintat@mit.edu

Aaron Sidford
Department of EECS
MIT
Cambridge, USA
Email: sidford@mit.edu

Sam Chiu-wai Wong
Department of EECS
UC Berkeley
Berkeley, USA
Email: samcwong@berkeley.edu

Abstract

In this paper we improve upon the running time for finding a point in a convex set given a separation oracle. In particular, given a separation oracle for a convex set $K \subset \mathbb{R}^n$ that is contained in a box of radius R we show how to either compute a point in K or prove that K does not contain a ball of radius ϵ using an expected $O(n \log(nR/\epsilon))$ evaluations of the oracle and additional time $O(n^3 \log^{O(1)}(nR/\epsilon))$. This matches the oracle complexity and improves upon the $O(n^{\omega+1} \log(nR/\epsilon))$ additional time of the previous fastest algorithm achieved over 25 years ago by Vaidya [91] for the current value of the matrix multiplication constant $\omega < 2.373$ [98], [36] when $R/\epsilon = O(\mathsf{poly}(n))$.

Using a mix of standard reductions and new techniques we show how our algorithm can be used to improve the running time for solving classic problems in continuous and combinatorial optimization. In particular we provide the following running time improvements:

- *Submodular Function Minimization*: n is the size of the ground set, M is the maximum absolute value of function values and EO is the time for function evaluation.
  Our weakly and strongly polynomial time algorithms have a running time of $O(n^2 \log nM \cdot \text{EO} + n^3 \log^{O(1)} nM)$ and $O(n^3 \log^2 n \cdot \text{EO} + n^4 \log^{O(1)} n)$, improving upon the previous best of $O((n^4 \cdot \text{EO} + n^5) \log M)$ and $O(n^5 \cdot \text{EO} + n^6)$ respectively.
- *Submodular Flow*: $n = |V|$, $m = |E|$, C is the maximum edge cost in absolute value and U is maximum edge capacity in absolute value.
  We obtain a faster weakly polynomial running time of $O(n^2 \log nCU \cdot \text{EO} + n^3 \log^{O(1)} nCU)$, improving upon the previous best of $O(mn^5 \log nU \cdot \text{EO})$ and $O\left(n^4 h \min\{\log C, \log U\}\right)$ from 15 years ago by a factor of $\tilde{O}(n^4)$. We also achieve faster strongly polynomial time algorithms as a consequence of our result on submodular minimization.
- *Matroid Intersection*: n is the size of the ground set, r is the maximum size of independent sets, M is the maximum absolute value of element weight, $\mathcal{T}_{\text{rank}}$ and $\mathcal{T}_{\text{ind}}$ are the time for each rank and independence oracle query.
  We obtain a running time of $O((nr \log^2 n \mathcal{T}_{\text{rank}} + n^3 \log^{O(1)} n) \log nM)$ and $O((n^2 \log n \mathcal{T}_{\text{ind}} + n^3 \log^{O(1)} n) \log nM)$, achieving the first quadratic bound on the query complexity for the independence and rank oracles. In the unweighted case, this is the first improvement since 1986 for independence oracle.
- *Semidefinite Programming*: n is the number of constraints, m is the number of dimensions and S is the total number of non-zeros in the constraint matrices.
  We obtain a running time of $\tilde{O}(n(n^2 + m^\omega + S))$, improving upon the previous best of $\tilde{O}(n(n^\omega + m^\omega + S))$ for the regime S is small.

Index Terms

Cutting Plane Method; Ellipsoid Method; Submodular Function Minimization; Submodular Flow; Matroid Intersection; Semidefinite Programming

Introduction

The ellipsoid method and more generally, *cutting plane methods*, that is optimization algorithms which iteratively call a separation oracle, have long been central to theoretical computer science. In combinatorial optimization, since Khachiyan's seminal result in 1980 [60] proving that the ellipsoid method solves linear programs in polynomial time, the ellipsoid method has been crucial to solving discrete problems in polynomial time [44]. In continuous optimization, cutting plane methods have long played a critical role in convex optimization, where they are fundamental to the theory of non-smooth optimization [40].

Despite the key role that cutting plane methods have played historically in both combinatorial and convex optimization, over the past two decades progress on improving both the theoretical running time of cutting plane

0272-5428/15 $31.00 © 2015 IEEE
DOI 10.1109/FOCS.2015.68

methods as well as the complexity of using cutting plane methods for combinatorial optimization has stagnated. The theoretical running time of cutting plane methods for convex optimization has not been improved since the breakthrough result by Vaidya in 1989 [91], [93]. Moreover, for many of the key combinatorial applications of ellipsoid method, such as submodular minimization, matroid intersection and submodular flow, the running time improvements over the past two decades have been primarily combinatorial; that is they have been achieved by discrete algorithms that do not use numerical machinery such as cutting plane methods.

In this paper we make progress on these classic optimization problems on two fronts. First we show how to improve on the running time of cutting plane methods for a broad range of parameters that arise frequently in both combinatorial applications and convex programming (Section I). Second, we provide several frameworks for applying the cutting plane method and illustrate the efficiency of these frameworks by obtaining faster running times for semidefinite programming, matroid intersection, and submodular flow (Section II). Finally, we show how to couple our approach with the problem specific structure and obtain faster weakly and strongly polynomial running times for submodular function minimization, a problem of tremendous importance in combinatorial optimization (Section III). In both cases our algorithms are faster than previous best by a factor of roughly $\Omega(n^2)$.

We remark that many of our running time improvements come both from our faster cutting method and from new careful analysis of how to apply these cutting plane methods. In fact, simply using our reductions to cutting plane methods and a seminal result of Vaidya [91], [93] on cutting plane methods we provide improved running times for solving many of these problems. As such, we organized our presentation to hopefully make it easy to apply cutting plane methods to optimization problems and obtain provable guarantees in the future.

Our results demonstrate the power of cutting plane methods in theory and possibly pave the way for new cutting plane methods in practice. We show how cutting plane methods can continue to improve running times for classic optimization problems and we hope that these methods may find further use. As cutting plane methods such as analytic cutting plane method [38], [7], [39], [78], [99], [40] are frequently used in practice [43], [37], these techniques may have further implications.

This extended abstract summarizes our main contributions and approaches taken from the full version. Each section in this document is expanded into a part in the full version, demarcating the different types of problems considered and results achieved.

I. A Faster Cutting Plane Method

Throughout Section I we study the following *feasibility problem*:

Definition 1 (Feasibility Problem)**.** Given a separation oracle for a set $K \subseteq \mathbb{R}^n$ contained in a box of radius R either find a point $\vec{x} \in K$ or prove that K does not contain a ball of radius ϵ. Let SO be the cost of the oracle.

This feasibility problem is one of the most fundamental and classic problems in optimization. Since the celebrated result of Yudin and Nemirovski [100] in 1976 and Khachiyan [60] in 1979 essentially proving that this problem can be solved in time $O(\mathsf{poly}(n) \cdot \mathrm{SO} \cdot \log(R/\epsilon))$, this problem has served as one of the key primitives for solving numerous problems in both combinatorial and convex optimization.

Despite the prevalence of this feasibility problem the best known running time for solving this problem has not been improved in over 25 years. In a seminal result of Vaidya in 1989 [91], he showed how to solve the problem in time $\widetilde{O}(n \cdot \mathrm{SO} \cdot \log(nR/\epsilon) + n^{\omega+1} \log(nR/\epsilon))$. However, while there have been interesting generalizations and practical improvements [3], [83], [38], [7], [39], [78], [99], [40], [12], the best theoretical guarantees for solving this problem have not been improved since.

In Section I we show how to improve upon Vaidya's running time in certain regimes. We provide a cutting plane algorithm which achieves an expected running time of $O(n \cdot \mathrm{SO} \cdot \log(nR/\epsilon) + n^3 \log^{O(1)}(nR/\epsilon))$, improving upon the previous best known running time for the current known value of $\omega < 2.373$ [98], [36] when $R/\epsilon = O(\mathsf{poly}(n))$.

We achieve our results by the combination of multiple techniques. First we show how to use techniques from the work of Vaidya and Atkinson to modify Vaidya's scheme so that it is able to tolerate random noise in the computation in each iteration. We then show how to use known numerical machinery [92], [87], [68] in combination with new techniques to implement each of these relaxed iterations efficiently. We hope that both these numerical techniques as well as our scheme for approximating complicated methods, such as Vaidya's, may find further applications.

While our paper focuses on theoretical aspects of cutting plane methods, we achieve our results via the careful application of practical techniques such as dimension reduction and sampling. As such we hope that ideas in this

Year	Algorithm	Complexity
1979	Ellipsoid Method [86], [100], [60]	$O(n^2\mathrm{SO}\log\kappa + n^4\log\kappa)$
1988	Inscribed Ellipsoid [61], [79]	$O(n\mathrm{SO}\log\kappa + (n\log\kappa)^{4.5})$
1989	Volumetric Center [91]	$O(n\mathrm{SO}\log\kappa + n^{1+\omega}\log\kappa)$
1995	Analytic Center [7]	$O\left(\begin{array}{c} n\mathrm{SO}\log^2\kappa + n^{\omega+1}\log^2\kappa \\ +(n\log\kappa)^{2+\omega/2} \end{array}\right)$
2004	Random Walk [10]	$O(n\mathrm{SO}\log\kappa + n^7\log\kappa)$
2013	This paper	$O(n\mathrm{SO}\log\kappa + n^3\log^{O(1)}\kappa)$

Figure I.1. Algorithms for the Feasibility Problem. $\kappa = nR/\epsilon$. Note that [10] requires a much weaker oracle, membership oracle.

paper may lead to improved practical[1] algorithms for non-smooth optimization.

A. Previous Work

Throughout this paper, we restrict our attention to algorithms for the feasibility problem that have a polynomial dependence on SO, n, and $\log(R/\epsilon)$. Such "efficient" algorithms, typically follow the following iterative framework. First, they compute some trivial region Ω that contains K. Then, they call the separation oracle at some point $\vec{x} \in \Omega$. If $\vec{x} \in K$ the algorithm terminates having successfully solved the problem. If $\vec{x} \notin K$ then the separation oracle must return a half-space containing K. The algorithm then uses this half-space to shrink the region Ω while maintaining the invariant that $K \subseteq \Omega$. The algorithm then repeats this process until it finds a point $\vec{x} \in K$ or the region Ω becomes too small to contains a ball with radius ϵ.

Previous work on efficient algorithms for the feasibility problem all follow this iterative framework. They vary in terms of what set Ω they maintain, how they compute the center to query the separation oracle, and how they update the set. In Table I.1, we list the previous running times for solving the feasibility problem. As usual SO indicates the cost of the separation oracle. To simplify the running times we let $\kappa \stackrel{\text{def}}{=} nR/\epsilon$. The running times of some algorithms in the table depend on R/ϵ instead of nR/ϵ. However, for many situations, we have $\log(R/\epsilon) = \Theta(\log(nR/\epsilon))$ and hence we believe this is still a fair comparison.

The first efficient algorithm for the feasibility problem is the *ellipsoid* method, due to Shor [86], Nemirovksii and Yudin [100], and Khachiyan [60]. The ellipsoid method maintains an ellipsoid as Ω and uses the center of the ellipsoid as the next point. It takes $\Theta(n^2\log\kappa)$ calls which is far from the lower bound $\Omega(n\log\kappa)$ calls [77].

To alleviate the problem, the algorithm could maintain all the information from the oracle, i.e., the polytope created from the intersection of all half-spaces obtained. The center of gravity method [69] achieves the optimal oracle complexity using this polytope and the central of gravity of this polytope as the next point. However, computing central of gravity is computationally expensive and hence we do not list the running time of it in Table I.1. The Inscribed Ellipsoid Method [61] also achieved an optimal oracle complexity using this polytope as Ω but instead using the center of the maximal inscribed ellipsoid in the polytope to query the separation oracle. We listed it as occurring in year 1988 in Table I.1 because it was [79] that yielded the first polynomial time algorithm to actually compute this maximal inscribed ellipsoid for polytope.

Vaidya [91] obtained a faster algorithm by maintaining an approximate of this polytope and using a different center, namely the volumetric center. Although the oracle complexity of this volumetric center method is very good, the algorithm is not extremely efficient as each iteration involves matrix inversion. Atkinson and Vaidya [7] showed how to avoid this computation in certain settings however they were unable to achieve the desired convergence rate.

Bertsimas and Vempala [10] also provide an algorithm that avoids these expensive linear algebra operations while maintaining the optimal convergence rate by using techniques in sampling convex sets. Even better, this result works for a much weaker oracle, membership oracle. However, the additional cost of this algorithm is relatively high in theory. We remark that while there are considerable improvement on the sampling techniques [71], [58], [68], the additional cost is still quite high compared to standard linear algebra.

[1] Although cutting plane methods are often criticized for their empirical performance, recently, Bubeck, Lee and Singh [12] provided a variant of the ellipsoid method that achieves the same convergence rate as Nesterov's accelerated gradient descent with a better empirical performance. It suggests that cutting plane methods can be as aggressive as first order methods if designed properly.

B. Challenges in Improving Previous Work

Our algorithm builds upon the previous fastest algorithm of Vaidya [93]. Ignoring implementation details and analysis, Vaidya's algorithm is quite simple. This algorithm simply maintains a polytope $P^{(k)} = \{x \in \mathbb{R}^n : \mathbf{A}\vec{x} - \vec{b} \geq \vec{0}\}$ as the current Ω and uses the *volumetric center*, the minimizer of the *volumetric barrier function*

$$\arg\min_{\vec{x}} \frac{1}{2} \log\det\left(\mathbf{A}^T \mathbf{S}_{\vec{x}}^{-2} \mathbf{A}\right) \quad \text{where} \quad \mathbf{S}_{\vec{x}} \stackrel{\text{def}}{=} \mathbf{diag}(\mathbf{A}\vec{x} - \vec{b}) \tag{I.1}$$

as the point at which to query the separation oracle. The polytope is then updated by adding shifts of the half-spaces returned by the separation oracle and dropping unimportant constraints. By choosing the appropriate shift, picking the right rule for dropping constraints, and using Newton's method to compute the volumetric center he achieved a running time of $O(n \cdot SO \cdot \log\kappa + n^{1+\omega} \log\kappa)$.

While Vaidya's algorithms' dependence on SO is essentially optimal, the additional per-iteration costs of his algorithm could possibly be improved. The computational bottleneck in each iteration of Vaidya's algorithm is computing the gradient of $\log\det$ which involves computing the leverage scores $\vec{\sigma}(\vec{x}) \stackrel{\text{def}}{=} \text{diag}(\mathbf{S}_x^{-1}\mathbf{A}\left(\mathbf{A}^T\mathbf{S}_x^{-2}\mathbf{A}\right)^{-1}\mathbf{A}^T\mathbf{S}_x^{-1})$, a commonly occurring quantity in numerical analysis and convex optimization [87], [16], [70], [68], [67]. As the best known algorithms for computing leverage scores exactly in this setting take time $O(n^\omega)$, directly improving the running time of Vaidya's algorithm seems challenging.

However, since an intriguing result of Spielman and Srivastava in 2008 [87], it has been well known that using Johnson Lindenstrauss these leverage scores can be computed up to a multiplicative $(1 \pm \epsilon)$ error by solving $O(\epsilon^{-2}\log n)$ linear systems involving $\mathbf{A}^T\mathbf{S}_x^{-2}\mathbf{A}$. While in general this still takes time $O(\epsilon^{-2}n^\omega)$, there are known techniques for efficiently maintaining an inverse of a matrix so that solving linear systems take amortized $O(n^2)$ [92], [67], [68]. Consequently if it could be shown that computing approximate leverage scores sufficed, this would potentially decrease the amortized cost per iteration of Vaidya's method.

Unfortunately, Vaidya's method does not seem to tolerate this type of multiplicative error. If leverage scores were computed this crudely then in using them to compute approximate gradients for (I.1), it seems that any point computed would be far from the true center. Moreover, without being fairly close to the true volumetric center, it is difficult to argue that such a cutting plane method would make progress.

To overcome this issue, it is tempting to directly use recent work on improving the running time of linear programming [67]. In this work, the authors faced a similar issue where a volumetric, i.e. $\log\det$, potential function had the right analytic and geometric properties, however was computational expensive to minimize. To overcome this issue the authors instead computed a weighted analytic center:

$$\arg\min_{\vec{x}} \sum_{i\in[m]} w_i \log s_i(\vec{x}) \quad \text{where} \quad \vec{s}(\vec{x}) \stackrel{\text{def}}{=} \mathbf{A}\vec{x} - \vec{b} \quad .$$

For carefully chosen weights this center provides the same convergence guarantees as the volumetric potential function, while each step can be computed by solving few linear systems (rather than forming the matrix inverse).

Unfortunately, it is unclear how to directly extend the work in [67] on solving an explicit linear program to the feasibility problem specified by a separation oracle. While, it is possible to approximate the volumetric barrier by a weighted analytic center in many respects, proving that this approximation suffices for efficient convergence remains open. In fact, the volumetric barrier function as used in Vaidya's algorithm is well approximated simply by the standard analytic center

$$\arg\min_{\vec{x}} \sum_{i\in[m]} \log s_i(\vec{x}) \quad \text{where} \quad \vec{s}(\vec{x}) \stackrel{\text{def}}{=} \mathbf{A}\vec{x} - \vec{b} \quad .$$

as all the unimportant constraints are dropped during the algorithm. However, despite decades of research, the best running times known for solving the feasibility problem using the analytic center are Vaidya and Atkinson algorithm from 1995 [7]. While the running time of this algorithm could possibly be improved using approximate leverage score computations and amortized efficient linear system solvers, unfortunately at best, without further insight this would yield an algorithm which requires a suboptimal $O(n\log^{O(1)}\kappa)$ queries to the separation oracle.

As pointed out in [7], the primary difficulty in using any sort of analytic center is quantifying the amount of progress made in each step. We still believe providing direct near-optimal analysis of weighted analytic center is a tantalizing open question warranting further investigation. However, rather than directly address the question of the performance of weighted analytic centers for the feasibility problem, we take a slightly different approach that side-steps this issue. We provide a partial answer that still sheds some light on the performance of the weighted analytic center while still providing our desired running time improvements.

C. Our Approach

To overcome the shortcoming of the volumetric and analytic centers we instead consider a hybrid barrier function

$$\arg\min_{\vec{x}} \sum_{i\in[m]} w_i \log s_i(\vec{x}) + \log\det(\mathbf{A}^T\mathbf{S}_x^{-1}\mathbf{A}) \quad \text{where} \quad \vec{s}(\vec{x}) \stackrel{\text{def}}{=} \mathbf{A}\vec{x} - \vec{b} \quad .$$

for careful chosen weights. Our key observation is that for correct choice of weights, we can compute the gradient of this potential function. In particular if we let $\vec{w} = \vec{\tau} - \vec{\sigma}(\vec{x})$ then the gradient of this potential function is the same as the gradients of $\sum_{i\in[m]} \tau_i \log s_i(\vec{x})$, which we can compute efficiently. Moreover, since we are using $\log\det$, we can use analysis similar to Vaidya's algorithm [91] to analyze the convergence rate of this algorithm.

Unfortunately, this simple observation and does not immediately change the problem substantially. It simply pushes the problem of computing gradients of $\log\det$ to computing $\vec{w}$. Therefore, for this scheme to work, we would need to ensure that the weights do not change too much and that when they change, they do not significantly hurt the progress of our algorithm. In other words, for this scheme to work, we would still need very precise estimates of leverage scores.

However, we note that the leverage scores $\vec{\sigma}(\vec{x})$ do not change too much between iterations. Moreover, we provide what we believe is an interesting technical result that an unbiased estimates to the changes in leverage scores can be computed using linear system solvers such that the *total error* of the estimate is bounded by the total change of the leverage scores. Using this result our scheme simply follows Vaidya's basic scheme in [91], however instead of minimizing the hybrid barrier function directly we alternate between taking newton steps we can compute, changing the weights so that we can still compute Newton steps, and computing accurate unbiased estimates of the changes in the leverage scores so that the weights do not change adversarially by too much.

To make this scheme work, there are two additional details that need to be dealt with. First, we cannot let the weights vary too much as this might ultimately hurt the rate of progress of our algorithm. Therefore, in every iteration we compute a single leverage score to high precision to control the value of w_i and we show that by careful choice of the index we can ensure that no weight gets too large.

Second, we need to show that changing weights does not affect our progress by much more than the progress we make with respect to $\log\det$. To do this, we need to show the slacks are bounded above and below. We enforce this by adding regularization terms and instead consider the potential function

$$p_{\vec{c}}(\vec{x}) = -\sum_{i\in[m]} w_i \log s_i(\vec{x}) + \frac{1}{2}\log\det\left(\mathbf{A}^T\mathbf{S}_x^{-2}\mathbf{A} + \lambda\mathbf{I}\right) + \frac{\lambda}{2}\|x\|_2^2$$

This allows us to ensure that the entries of $\vec{s}(\vec{x})$ do not get too large or to small and therefore changing the weighting of the analytic center cannot affect the function value too much.

Third, we need to make sure our potential function is convex. If we simply take $\vec{w} = \vec{\tau} - \vec{\sigma}(\vec{x})$ with $\vec{\tau}$ is an estimator of $\vec{\sigma}(\vec{x})$, $\vec{w}$ can be negative and the potential function could be non-convex. To circumvent this issue, we use $\vec{w} = c_e + \vec{\tau} - \vec{\sigma}(\vec{x})$ and make sure $\left\|\vec{\tau} - \vec{\sigma}(\vec{x})\right\|_\infty < c_e$.

Combining these insights, using efficient algorithms for solving a sequence of slowly changing linear systems [92], [67], [68], and providing careful analysis ultimately allows us to achieve a running time of $O(n\mathrm{SO}\log\kappa + n^3\log^{O(1)}\kappa)$ for the feasibility problem. Furthermore, in the case that K does not contain a ball of radius ϵ, our algorithm provides a proof that the polytope does not contain a ball of radius ϵ. This proof ultimately allows us to achieve running time improvements for strongly polynomial submodular minimization in Part III.

Theorem 2 (Our Cutting Plane Method)**.** *Let $K \subseteq \mathbb{R}^n$ be a non-empty set contained in a box of radius R, i.e. $K \subseteq \{\vec{x} : \left\|\vec{x}\right\|_\infty \leq R\}$. For any $\epsilon \in (0, R)$ in expected time $O(n\mathrm{SO}\log(nR/\epsilon) + n^3\log^{O(1)}(nR/\epsilon))$ our cutting plane method either outputs $\vec{x} \in K$ or finds a polytope $P = \{\vec{x} : \mathbf{A}\vec{x} \geq \vec{b}\} \supseteq K$ such that*

1) *P has $O(n)$ many constraints (i.e. $\mathbf{A} \in \mathbb{R}^{O(n)\times d}$ and $\vec{b} \in \mathbb{R}^{O(d)}$).*
2) *Each constraint of P is either an initial constraint from $B_\infty(R)$ or of the form $\langle\vec{a},\vec{x}\rangle \geq b + \delta$ where $\langle\vec{a},\vec{x}\rangle \geq b$ is a hyperplane returned by the separation oracle and $\delta = \Omega\left(\frac{\epsilon}{\sqrt{n}}\right)$.*
3) *The polytope P has small width with respect to some direction $\vec{a}_1$ given by one of the constraints, i.e.*

$$\max_{\vec{y}\in P\cap B_\infty(R)} \langle\vec{a}_1,\vec{y}\rangle - \min_{\vec{y}\in P\cap B_\infty(R)} \langle\vec{a}_1,\vec{y}\rangle \leq O\left(n\epsilon\ln(R/\epsilon)\right)$$

4) *Furthermore, the algorithm produces a proof of the fact above involving convex combination of the constraints, namely, non-negatives $t_2, ..., t_{O(n)}$ and $\vec{x} \in P$ such that*

a) $\left\|\vec{x}\right\|_2 \leq 3\sqrt{n}R$,
b) $\left\|\vec{a}_1 + \sum_{i=2}^{O(n)} t_i \vec{a}_i\right\|_2 = O\left(\frac{\epsilon}{R}\sqrt{n}\log(R/\epsilon)\right)$,
c) $\vec{a}_1^T \vec{x} - \vec{b}_1 \leq \epsilon$,
d) $\left(\sum_{i=2}^{O(n)} t_i a_i\right)^T \vec{x} - \sum_{i=2}^{O(n)} t_i b_i \leq O(n\epsilon \log(R/\epsilon))$.

II. A User's Guide to Cutting Plane Methods

Cutting plane methods have long been employed to obtain polynomial time algorithms for solving optimization problems. However, for many problems cutting plane methods are often regarded as inefficient both in theory and in practice.In the full version, we provide several techniques for applying cutting plane methods efficiently. Moreover, we illustrate the efficacy and versatility of these techniques by showing how they allows us to achieve improved running times for solving multiple problems including semidefinite programming, matroid intersection, and submodular flow.

We hope these results revive interest in ellipsoid and cutting plane methods. We believe the results demonstrate that these techniques can be useful not just for showing that a problem is solvable in polynomial time, but in many settings they can be used to yield substantial running time improvements. Some of these techniques are new and some are extensions of previously known techniques.

In the remainder of this section we give an overview of the main techniques we employ to apply our cutting plane method (Section II-A, II-B, and II-C), demonstrate the versatility of our approach (Section II-D), and provide an overview of many of the improved running times that can be achieved by our algorithms (Section II-E).

A. Technique 0: From Feasibility to Optimization

Although cutting plane methods are typically introduced as algorithms for finding a point in a convex set (as we did with the feasibility problem in Section I), this is often not the easiest way to apply the methods. Moreover, improperly applying results on the feasibility problem to solve convex optimization problems can sometimes lead to vastly sub-optimal running times.

Here, we explain how to use our cutting plane method to efficiently solve convex optimization problems using an approximate subgradient oracle. Our result is based on a result of Nemirovski [76] in which he showed how to use a cutting plane method to solve convex optimization problems without smoothness assumptions on the function and with minimal assumptions on the size of the function's domain. We generalize his proof to accommodate for an approximate separation oracle, an extension which is essential for our later applications. We use this result as the starting point for two new techniques we discuss below. We define an approximate separation oracle and provide our main theorem regarding convex optimization below.

Definition 3 (Separation Oracle for a Function)**.** For any convex function f, $\eta \geq 0$ and $\delta \geq 0$, a (η, δ)-separation oracle on a convex set Γ for f is a function on $\mathbb{R}^n$ such that for any input $\vec{x} \in \Gamma$, it either asserts $f(\vec{x}) \leq \min_{\vec{y}\in\Gamma} f(\vec{y}) + \eta$ or outputs a half space H such that

$$\{\vec{z} \in \Gamma : f(\vec{z}) \leq f(\vec{x})\} \subset H \stackrel{\text{def}}{=} \{\vec{z} : \vec{c}^T\vec{z} \leq \vec{c}^T\vec{x} + b\} \tag{II.1}$$

with $b \leq \delta\left\|\vec{c}\right\|$ and $\vec{c} \neq \vec{0}$. We let $\mathrm{SO}_{\eta,\delta}(f)$ be the time complexity of this oracle.

If the function f is differentiable, the gradient of f is a $(0, 0)$-separation oracle and it can usually be computed by explicit formula.

Theorem 4. *Let f be a convex function on $\mathbb{R}^n$ and Ω be a convex set that contains a minimizer of f. Suppose we have a (η, δ)-separation oracle for f and Ω is contained inside $B_\infty(R) \stackrel{\text{def}}{=} \{\vec{x} : \left\|\vec{x}\right\|_\infty \leq R\}$. For any $0 < \alpha < 1$, we can compute $\vec{x} \in \mathbb{R}^n$ such that*

$$f(\vec{x}) - \min_{\vec{y}\in\Omega} f(\vec{y}) \leq \eta + \alpha\left(\max_{\vec{y}\in\Omega} f(\vec{y}) - \min_{\vec{y}\in\Omega} f(\vec{y})\right)$$

with an expected running time of

$$O\left(n\mathrm{SO}_{\eta,\delta}(f)\log\left(\frac{n\kappa}{\alpha}\right) + n^3\log^{O(1)}\left(\frac{n\kappa}{\alpha}\right)\right),$$

where $\delta = \Theta\left(\frac{\alpha\mathrm{MinWidth}(\Omega)}{n^{3/2}\ln(\kappa)}\right)$, $\kappa = \frac{R}{\mathrm{MinWidth}(\Omega)}$ *and* $\mathrm{MinWidth}(\Omega) = \min_{\|\vec{a}\|_2=1} \max_{\vec{x},\vec{y}\in\Omega} \langle\vec{a}, \vec{x} - \vec{y}\rangle$.

Observe that this algorithm requires no information about Ω (other than that $\Omega \subseteq B_\infty(R)$) and does not guarantee $\vec{x}_j \in \Omega$. Hence, even though Ω can be complicated to describe, the algorithm still gives a guarantee related to the gap $\max_{\vec{x}\in\Omega} f(\vec{x}) - \min_{\vec{x}\in\Omega} f(\vec{x})$. For specific applications, it is therefore advantageous to pick a Ω as large as possible while the bound on function value is as small as possible. Specifically, using John's ellipsoid, it can be shown that any convex set Ω can be transformed linearly such that (1) $B_\infty(1)$ contains Ω and, (2) $\mathrm{MinWidth}(\Omega) = \Omega(n^{-3/2})$. In other words, κ can be effectively chosen as $O(n^{3/2})$. Therefore if we are able to find such a linear transformation, the running time is simply $O\left(n\mathrm{SO}(f)\log(n/\alpha) + n^3 \log^{O(1)}(n/\alpha)\right)$. In general, this can be done easily using the structure of the particular problem and the running time does not depend on the size of domain at all.

B. Technique 1: Dimension Reduction through Duality

Now, we discuss how cutting plane methods can be applied to obtain both primal and dual solutions to convex optimization problems. Moreover, we show how this can be achieved while only applying the cutting plane method in the space, primal or dual, which has a fewer number of variables. Thus we show how to use duality to improve the convergence of cutting plane methods while still solving the original problem.

To illustrate this idea consider the following very simple linear program (LP)

$$\min_{x_i \geq 0, \sum x_i = 1} \sum_{i=1}^{n} w_i x_i$$

where $\vec{x} \in \mathbb{R}^n$ and $\vec{w} \in \mathbb{R}^n$. Although this LP has n variables, it should to be easy to solve purely on the grounds that it only has one equality constraint and thus dual linear program is simply

$$\max_{y \leq w_i \forall i} y\,,$$

i.e. a LP with only one variable. Consequently, we can apply e.g. the cutting plane method to solve it efficiently. Thus, while we can use duality to decrease dimensions, it is not always obvious how to recover the optimal primal solution x variable given the optimal dual solution y. Indeed, for many problems their dual is significantly simpler than itself (primal).

One such recent example is the linear programming result of [67]. In this result, they show how to take advantage of this observation and get a faster LP solver and maximum flow algorithm. It is interesting to study how far this technique can extend, that is, in what settings can one recover the solution to a more difficult dual problem from the solution to its easier primal problem?

There is in fact another precedent for such an approach. Grötschel, Lovász and Schrijver[45] showed how to obtain the primal solution for linear program by using a cutting plane method to solve the linear program exactly. This is based on the observation that cutting plane methods are able to find the active constraints of the optimal solution and hence one can take dual of the linear program to get the dual solution. This idea was further extended in [63] which also observed that cutting plane methods are incrementally building up a LP relaxation of the optimization problem. Hence, one can find a dual solution by taking the dual of that relaxation.

In the full paper, we show that this idea allows us to recover a dual optimal solution from an approximately optimal primal solution. Unfortunately, the performance of this approach seems quite problem-dependent. We therefore simply use semidefinite programming (SDP) to illustrate it. As a by-product, we obtain a faster SDP solver in both the primal and dual formulations of the problem.

C. Technique 2: Using Optimization Oracles Directly

In the seminal works of Grötschel, Lovász, Schrijver and independently Karp and Papadimitriou [44], [59], they showed the equivalence between optimization oracles and separation oracles, and gave a general method to construct a separation oracle for a convex set given an optimization oracle for that set, that is an oracle for minimizing linear functionals over the set. This seminal result led to the first weakly polynomial time algorithm for many algorithms such as submodular function minimization. Since then, this idea has been used extensively in various settings [57], [13], [14], [20].

Unfortunately, while this equivalence of separation and optimization is a beautiful and powerful tool for polynomial time solvability of problems, in many case it may lead to inefficient algorithms. In order to use this reduction to get a separation oracle, the optimization oracle may need to be called multiple times – essentially the number of times needed to run a cutting plane method and hence may be detrimental to obtaining small asymptotic running times. Therefore, it is an interesting question of whether there is a way of using an optimization oracle more directly.

We consider a special class of problems, that we call the *intersection problem*. For these problems we demonstrate how to achieve running time improvements by using optimization oracles directly. The problem we consider is as follows. We wish to solve the problem for some cost vector $\vec{c}$ and convex set K. We assume that the convex set K can be decomposed as $K = K_1 \cap K_2$ such that $\max_{\vec{x} \in K_1} \langle \vec{c}, \vec{x} \rangle$ and $\max_{\vec{x} \in K_2} \langle \vec{c}, \vec{x} \rangle$ can each be solved efficiently.

We show that by considering a carefully regularized variant, we obtain a problem such that optimization oracles for K_1 and K_2 immediately yield a separation oracle for this regularized problem. By analyzing the regularizers and bounding the domains of the problem we are able to show that this allows us to efficiently compute highly accurate solutions to the intersection problem without using a complicated iterative scheme, such as cutting plane methods which rely on the equivalence between separation and optimization. To state our result, we define the optimization oracle as follows:

Definition 5. Given a convex set K and $\delta > 0$. A δ-optimization oracle for K is a function on $\mathbb{R}^n$ such that for any input $\vec{c} \in \mathbb{R}^n$, it outputs $\vec{y}$ such that

$$\max_{\vec{x} \in K} \langle \vec{c}, \vec{x} \rangle \leq \langle \vec{c}, \vec{y} \rangle + \delta.$$

We denote by $\mathrm{OO}_\delta(K)$ the time complexity of this oracle.

In the full version, we show the following result

Theorem 6. *Consider the optimization problem $\max_{\vec{x} \in K_1 \cap K_2} \langle \vec{c}, \vec{x} \rangle$ where $\max_{\vec{x} \in K_1} \|\vec{x}\|_2 < M$, $\max_{\vec{x} \in K_2} \|\vec{x}\|_2 < M$ and $\|\vec{c}\|_2 \leq M$. Assume that $K_1 \cap K_2 \neq \emptyset$ and that we have ϵ-optimization oracle for every $\epsilon > 0$. For $0 < \delta < 1$, we can find $\vec{z} \in \mathbb{R}^n$ such that*

$$\max_{\vec{x} \in K_1 \cap K_2} \langle \vec{c}, \vec{x} \rangle \leq \delta + \langle \vec{c}, \vec{z} \rangle$$

and $\left\|\vec{z} - \vec{x}\right\|_2 + \left\|\vec{z} - \vec{y}\right\|_2 \leq \delta$ for some $\vec{x} \in K_1$ and $\vec{y} \in K_2$. Also, it takes time only

$$O\left(n \left(\mathrm{OO}_\eta(K_1) + \mathrm{OO}_\eta(K_2) \right) \log\left(\frac{nM}{\delta} \right) + n^3 \log^{O(1)}\left(\frac{nM}{\delta} \right) \right)$$

where $\eta = \Omega\left(\left(\frac{\delta}{nM} \right)^{O(1)} \right)$.

We note that this intersection problem can be viewed as a generalization of the matroid intersection problem and we show our reduction gives a faster algorithm in certain parameter regimes. As another example, we show our reduction gives a substantial polynomial improvement in the running time for the submodular flow problem.

D. Affine Subspace of Convex Set

We stress that while some results in this section are problem-specific, the techniques introduced here are quite general and are applicable to a wider range of problems. To illustrate this, here we demonstrate how one can take mixture of the two techniques above to get an generalization but less efficient version of the linear programming result of [66]. Consider the following optimization problem

$$\max_{\vec{x} \in K \text{ and } \mathbf{A}\vec{x} = \vec{b}} \langle \vec{c}, \vec{x} \rangle \tag{II.2}$$

where $\vec{x}, \vec{c} \in \mathbb{R}^n$, K is a convex subset of $\mathbb{R}^n$, $\mathbf{A} \in \mathbb{R}^{r \times n}$ and $\vec{b} \in \mathbb{R}^m$. Here, we assume a slightly stronger optimization oracle for K:

Definition 7. Given a convex set K and $\delta > 0$. A δ-2nd-order-optimization oracle for K is a function on $\mathbb{R}^n$ such that for any input $\vec{c} \in \mathbb{R}^n$ and $\lambda > 0$, it outputs $\vec{y}$ such that

$$\max_{\vec{x} \in K} \left(\langle \vec{c}, \vec{x} \rangle - \lambda \left\|\vec{x}\right\|^2 \right) \leq \delta + \langle \vec{c}, \vec{y} \rangle - \lambda \left\|\vec{y}\right\|^2.$$

We denote by $\mathrm{OO}^{(2)}_{\delta,\lambda}(K)$ the complexity of this oracle.

In the full paper, we show how to obtain the following result:

Theorem 8. *Assume that $\max_{\vec{x} \in K} \|\vec{x}\|_2 < M$, $\left\|\vec{b}\right\|_2 < M$, $\left\|\vec{c}\right\|_2 < M$, $\left\|\mathbf{A}\right\|_2 < M$ and $\lambda_{\min}(\mathbf{A}) > 1/M$. Assume that $K \cap \{\mathbf{A}\vec{x} = \vec{b}\} \neq \emptyset$. Suppose that we have ϵ-2nd-order-optimization oracle for every $\epsilon > 0$. For $0 < \delta < 1$, we can find $\vec{z} \in K$ such that*

$$\max_{\vec{x} \in K \text{ and } \mathbf{A}\vec{x} = \vec{b}} \langle \vec{c}, \vec{x} \rangle \leq \delta + \langle \vec{c}, \vec{z} \rangle$$

Authors	Years	Running times
Nesterov, Nemirovsky[80]	1992	$\tilde{O}(\sqrt{n}(nm^{\omega}+n^{\omega-1}m^2))$
Anstreicher [4]	2000	$\tilde{O}((mn)^{1/4}(nm^{\omega}+n^{\omega-1}m^2))$
Krishnan, Mitchell [64]	2003	$\tilde{O}(m(n^{\omega}+m^{\omega}+S))$ (dual SDP)
This paper	2015	$\tilde{O}(m(n^2+m^{\omega}+S))$

Figure II.1. Previous algorithms for solving a $n \times n$ SDP with m constraints and S non-zeros entries

and $\left\|\mathbf{A}\vec{z}-\vec{b}\right\|_2 \leq \delta$*; and the algorithm takes*

$$O\left(r\mathrm{OO}^{(2)}_{\eta,\lambda}(K)\log\left(\frac{nM}{\delta}\right)+r^3\log^{O(1)}\left(\frac{nM}{\delta}\right)\right)$$

where r *is the number of rows in* $\mathbf{A}$*,* $\eta=\left(\frac{\delta}{nM}\right)^{\Theta(1)}$*,* $\lambda=\left(\frac{\delta}{nM}\right)^{\Theta(1)}$*.*

Remark 9. In [66], they considered the special case $K=\{\vec{x}:0\leq x_i\leq 1\}$ and showed that it can be solved in $\tilde{O}(\sqrt{r})$ iterations using interior point methods. This gives the current fastest algorithm for the maximum flow problem on directed weighted graphs. Our result generalizes their result to any convex set K but with $\tilde{O}(r)$ iterations.

E. Applications

Here we demonstrate the efficacy of our approach by providing algorithms that improve upon the previous best known running times for solve several classic problems in combinatorial and continuous optimization. We provide a an overview of these applications, previous work on these problems, and our results.

In order to avoid deviating from our main discussion, our coverage of previous methods and techniques is brief. Given the large body of prior works on SDP, matroid intersection and submodular flow, it would be impossible to have an in-depth discussion on all of them. In order to demonstrate the power of our techniques, we do however provide a running time comparison of our algorithms with the relevant previous methods.

Semidefinite Programming

Consider the semidefinite programming (SDP) problem

$$\max_{\mathbf{X}\succeq\mathbf{0}}\mathbf{C}\bullet\mathbf{X}\text{ s.t. }\mathbf{A}_i\bullet\mathbf{X}=b_i\text{ (primal)}\qquad\min_{\vec{y}}\vec{b}^T\vec{y}\text{ s.t. }\sum_{i=1}^{n}y_i\mathbf{A}_i\succeq\mathbf{C}\text{ (dual)}$$

where $\mathbf{X}$, $\mathbf{C}$, $\mathbf{A}_i$ are $m\times m$ symmetric matrices, $\vec{b},\vec{y}\in\mathbb{R}^n$, and $\mathbf{A}\bullet\mathbf{B}\stackrel{\text{def}}{=}\mathrm{Tr}(\mathbf{A}^T\mathbf{B})$. For many problems, $n\ll m^2$ and hence the dual problem has fewer variables than the primal. There are many results and applications of SDP; see [94], [89], [74] for a survey on this topic. Since our focus is on polynomial time algorithms, we do not discuss pseudo-polynomial algorithms such as the spectral bundle methods [46], multiplicative weight update methods [5], [6], [56], [2], etc.

Currently, there are two competing approaches for solving SDP problems, namely interior point methods (IPM) and cutting plane methods. Typically, IPMs require fewer iterations than the cutting plane methods, however each iteration of these methods is more complicated. For SDP problems, interior point methods require the computations of the Hessian of the function $-\log\det\left(\mathbf{C}-\sum_{i=1}^{n}y_i\mathbf{A}_i\right)$ whereas cutting plane methods usually only need to compute minimum eigenvectors of the slack matrix $\mathbf{C}-\sum_{i=1}^{n}y_i\mathbf{A}_i$.

In [4], Anstreicher provided the current fastest IPM for solving the dual SDP problem using a method based on the volumetric barrier function. This method takes $O((mn)^{1/4})$ iterations and each iteration is as cheap as usual IPMs. For general matrices $\mathbf{C},\mathbf{X},\mathbf{A}_i$, each iteration takes $O(nm^{\omega}+n^{\omega-1}m^2)$ time where ω is the fast matrix multiplication exponent. If the matrices are sparse, then [35], [75] show how to use matrix completion inside the IPM. However, the running time depends on the extended sparsity patterns which can be much larger than the total number of non-zeros.

In [64], Krishnan and Mitchell observed that the separation oracle for dual SDP takes only $O(m^{\omega}+S)$ time, where $S=\sum_{i=1}^{n}\mathrm{nnz}(\mathbf{A}_i)$ be the total number of non-zeros in the constant matrix Hence, the cutting plane method by [93] gives a faster algorithm for SDP for many regimes. For $\omega=2.38$, the cutting plane method is faster when the problem is not too dense, i.e. $\sum_{i=1}^{n}\mathrm{nnz}(\mathbf{A}_i)<n^{0.63}m^{2.25}$. It is known how to make use of cutting plane methods to get the primal solution [63]; however, to the best knowledge of the authors, we are not aware any worst case running time analysis on this.

Authors	Years	Running times
Edmonds [22]	1968	not stated
Aigner, Dowling [1]	1971	$O(nr^2\mathcal{T}_{\text{ind}})$
Tomizawa, Iri [90]	1974	not stated
Lawler [65]	1975	$O(nr^2\mathcal{T}_{\text{ind}})$
Edmonds [24]	1979	not stated
Cunningham [18]	1986	$O(nr^{1.5}\mathcal{T}_{\text{ind}})$
This paper	2015	$O(n^2 \log n\mathcal{T}_{\text{ind}} + n^3 \log^{O(1)} n)$ $O(nr \log^2 n\mathcal{T}_{\text{rank}} + n^3 \log^{O(1)} n)$

Figure II.2. Previous algorithms for (unweighted) matroid intersection. Here n is the size of the ground set, $r = \max\{r_1, r_2\}$ is the maximum rank of the two matroids, $\mathcal{T}_{\text{ind}}$ is the time needed to check if a set is independent (independence oracle), and $\mathcal{T}_{\text{rank}}$ is the time needed to compute the rank of a given set (rank oracle).

Authors	Years	Running times
Edmonds [22]	1968	not stated
Tomizawa, Iri [90]	1974	not stated
Lawler [65]	1975	$O(nr^2\mathcal{T}_{\text{ind}} + nr^3)$
Edmonds [24]	1979	not stated
Frank [28]	1981	$O(n^2r(\mathcal{T}_{\text{circuit}} + n))$
Orlin, Ahuja [82]	1983	not stated
Brezovec, Cornuéjols, Glover[11]	1986	$O(nr(\mathcal{T}_{\text{circuit}} + r + \log n))$
Fujishige, Zhang [34]	1995	$O(n^2r^{0.5} \log rM \cdot \mathcal{T}_{\text{ind}})$
Shigeno, Iwata [85]	1995	$O((n + \mathcal{T}_{\text{circuit}})nr^{0.5} \log rM)$
This paper	2015	$O((n^2 \log n\mathcal{T}_{\text{ind}} + n^3 \log^{O(1)} n) \log nM)$ $O((nr \log^2 n\mathcal{T}_{\text{rank}} + n^3 \log^{O(1)} n) \log nM)$

Figure II.3. Previous algorithms for weighted matroid intersection. In additions to the notations used in the unweighted table, $\mathcal{T}_{\text{circuit}}$ is the time needed to find a fundamental circuit and M is the bit complexity of the weights.

In the full version, we present improved algorithms for finding the dual solution and prove carefully how to obtain the primal solution.

Theorem 10. *Consider the semidefinite programming (SDP) problem:*

$$\max_{\mathbf{X} \succeq \mathbf{0}} \mathbf{C} \bullet \mathbf{X} \text{ s.t. } \mathbf{A}_i \bullet \mathbf{X} = b_i \text{ and } \min_{\vec{y}} \vec{b}^T \vec{y} \text{ s.t. } \sum_{i=1}^{n} y_i \mathbf{A}_i \succeq \mathbf{C}$$

Suppose that for some $M \geq 1$ we have

1) $\|b\|_2 \leq M$, $\|\mathbf{C}\|_F \leq M$ *and* $\|\mathbf{A}_i\|_F \leq M$ *for all* i.
2) *The primal feasible set lies inside the region* $\text{Tr}\mathbf{X} \leq M$.
3) *The dual feasible set lies inside the region* $\|\vec{y}\|_\infty \leq M$.

Let OPT *be the optimum solution. Then, with high probability, we can find* $\mathbf{X}$ *and* $\vec{y}$ *such that*

1) $\mathbf{X} \succeq \mathbf{0}$, $\text{Tr}\mathbf{X} = O(M)$, $\sum_i |b_i - \langle \mathbf{X}, \mathbf{A}_i \rangle| \leq \epsilon$ *for all* i *and* $\mathbf{C} \bullet \mathbf{X} \geq \text{OPT} - \epsilon$.
2) $\|\vec{y}\|_\infty = O(M)$, $\sum_{i=1}^n y_i \mathbf{A}_i \succeq \mathbf{C} - \epsilon \mathbf{I}$ *and* $\vec{b}^T \vec{y} \leq \text{OPT} + \epsilon$.

in expected time $O\left(\left(nS + n^3 + nm^{\omega+o(1)}\right) \log^{O(1)}\left(\frac{nM}{\epsilon}\right)\right)$ *where* S *is the sparsity of the problem defined as* $\text{nnz}(\mathbf{C}) + \sum_{i=1}^n \text{nnz}(\mathbf{A}_i)$ *and* ω *is the fast matrix multiplication constant.*

Our result on the faster cutting plane method suggests that matrix multiplication can sometimes be avoided by carefully analyzing the changes from one iteration to the next. We suspect that this may also be possible for our SDP algorithm.

Conjecture 11. *Is there a* $O\left(\left(nS + n^3 + nm^{2+o(1)}\right) \log^{O(1)}\left(\frac{nM}{\epsilon}\right)\right)$ *time algorithm for SDP?*

Authors	Years	Running times
Fujishige [30]	1978	not stated
Grötschel, Lovász, Schrijver[44]	1981	weakly polynomial
Zimmermann [101]	1982	not stated
Barahona, Cunningham [9]	1984	not stated
Cunningham, Frank [19]	1985	$\rightarrow O(n^4 h \log C)$
Fujishige [31]	1987	not stated
Frank, Tardos [29]	1987	strongly polynomial
Cui, Fujishige [96]	1988	not stated
Fujishige, Röck, Zimmermann[33]	1989	$\rightarrow O(n^6 h \log n)$
Chung, Tcha [15]	1991	not stated
Zimmermann [102]	1992	not stated
McCormick, Ervolina [73]	1993	$O(n^7 h^* \log nCU)$
Wallacher, Zimmermann [97]	1994	$O(n^8 h \log nCU)$
Iwata [47]	1997	$O(n^7 h \log U)$
Iwata, McCormick, Shigeno [52]	1998	$O\left(n^4 h \min\left\{\log nC, n^2 \log n\right\}\right)$
Iwata, McCormick, Shigeno [53]	1999	$O\left(n^6 h \min\left\{\log nU, n^2 \log n\right\}\right)$
Fleischer, Iwata, McCormick[27]	1999	$O\left(n^4 h \min\left\{\log U, n^2 \log n\right\}\right)$
Iwata, McCormick, Shigeno [54]	1999	$O\left(n^4 h \min\left\{\log C, n^2 \log n\right\}\right)$
Fleischer, Iwata [25]	2000	$O(mn^5 \log nU \cdot \mathrm{EO})$
This paper	2015	$O(n^2 \log nCU \cdot \mathrm{EO} + n^3 \log^{O(1)} nCU)$

Figure II.4. Previous algorithms for Submodular Flow with n vertices, maximum cost C and maximum capacity U. The factor h is the time for an exchange capacity oracle, h^* is the time for a "more complicated exchange capacity oracle" and EO is the time for evaluation oracle of the submodular function. The arrow,$\rightarrow$, indicates that it used currently best maximum submodular flow algorithm as subroutine which was non-existent at the time of the publication.

Matroid Intersection

Matroid intersection is one of the most fundamental problems in combinatorial optimization. The first algorithm for matroid intersection is due to the seminal paper by Edmonds [22]. In Figures 9.2 and 9.3 we provide a summary of the previous algorithms for unweighted and weighted matroid intersection as well as the new running times we obtain in this paper. While there is no total ordering on the running times of these algorithms due to the different dependence on various parameters, we would like to point out that our algorithms outperform the previous ones in regimes where r is close to n and/or the oracle query costs are relatively expensive. In particular, in terms of oracle query complexity our algorithms are the first to achieve the quadratic bounds of $\tilde{O}(n^2)$ and $\tilde{O}(nr)$ for independence and rank oracles. We hope our work will revive the interest in the problem of which progress has been mostly stagnated for the past 20-30 years.

Theorem 12. *Suppose that the weight w is integer with $\|w\|_\infty \leq M$. Then, we can find*

$$S \in \arg\min_{S \in \mathcal{I}_1 \cap \mathcal{I}_2} w(S)$$

in time $O\left(n\mathrm{GO}\log(nM) + n^3 \log^{O(1)}(nM)\right)$ where GO *is the cost of greedy method for $\mathcal{I}_1$ and $\mathcal{I}_2$.*

Conjecture 13. *What is the optimal rank and independence oracle complexity for matroid intersection? Is our quadratic bound tight?*

One nice aspect of this open problem is that oracle complexity can be established without any computational hardness assumption as it is a purely information theoretic measure. If our bound is indeed tight, one possible approach would be to examine the tight instances for our algorithm which might in turn give a lower bound.

Minimum-Cost Submodular Flow

(Minimum-cost) Submodular Flow is a very general problem in combinatorial optimization which generalizes many problems such as minimum cost flow, the graph orientation, polymatroid intersection, directed cut covering [32]. In Figure 9.4 we provide an overview of the previous algorithms for submodular flow.

Many of the running times carry a parameter h, which is the time required for computing an "exchange capacity". To our knowledge, the best way of computing an exchange capacity is to solve an instance of submodular minimization which previously took time $\tilde{O}(n^4\mathrm{EO}+n^5)$ but now $\tilde{O}(n^2\mathrm{EO}+n^3)$ thanks to our result in Section III. Readers may wish to substitute $h = \tilde{O}(n^2\mathrm{EO} + n^3)$ when reading the table.

The previous fastest weakly polynomial algorithms for submodular flow are by [54], [25], [27], which take time $\tilde{O}(n^6\mathrm{EO} + n^7)$ and $O(mn^5 \log nU \cdot \mathrm{EO})$, assuming $h = \tilde{O}(n^2\mathrm{EO} + n^3)$. Our algorithm for submodular flow has a running time of $\tilde{O}(n^2\mathrm{EO} + n^3)$, which is significantly faster by roughly a factor of $\tilde{O}(n^4)$.

For strongly polynomial algorithms, our results in this section do not yield a speedup but we remark that our faster strongly polynomial algorithm for submodular minimization in Section III improves the previous algorithms by a factor of $\tilde{O}(n^2)$ as a corollary (because h requires solving an instance of submodular minimization).

Theorem 14. *Given a directed graph $G = (V, E)$ with $|E| = m$ and a submodular function f on $\mathbb{R}^V$ with $|V| = n$, $f(\emptyset) = 0$ and $f(V) = 0$. Let A be the incidence matrix of G. Consider the submodular flow problem*

$$\begin{aligned} &\textit{Minimize} && \langle c, \varphi \rangle \\ &\textit{subject to} && l(e) \le \varphi(e) \le u(e) \quad \forall e \in E \\ &&& x(v) = (A\varphi)(v) \quad \forall v \in V \\ &&& \sum_{v \in S} x(v) \le f(S) \quad \forall S \subseteq V \end{aligned} \tag{II.3}$$

with $c \in \mathbb{Z}^E$, $l \in \mathbb{Z}^E$, $u \in \mathbb{Z}^E$ where $C = \left\|\vec{c}\right\|_\infty$ and $U = \max\left(\left\|u\right\|_\infty, \left\|l\right\|_\infty, \max_{S \subset V} |f(S)|\right)$. We can solve the submodular flow problem (II.3) in time $O\left(n^2\mathrm{EO}\log(mCU) + n^3 \log^{O(1)}(mCU)\right)$ where EO is the cost of function evaluation.

In Section III, we provide a faster weakly polynomial algorithm for submodular minimization and then show how to make it strongly polynomial. We believe that this can also be done for submodular flow because of the similarity between the two problems.

Conjecture 15. *Is there an analogous faster strongly polynomial time algorithm for submodular flow?*

III. Submodular Function Minimization

Submodular functions and submodular function minimization (SFM) are fundamental to the field of combinatorial optimization. Examples of submodular functions include graph cut functions, set coverage function, and utility functions from economics. Since the seminal work by Edmonds in 1970 [23], submodular functions and the problem of minimizing such functions (i.e. submodular function minimization) have served as a popular modeling and optimization tool in various fields such as theoretical computer science, operations research, game theory, and most recently, machine learning. Given its prevalence, fast algorithms for SFM are of immense interest both in theory and in practice.

Throughout Section III, we consider the standard formulation of SFM: we are given a submodular function f defined over the subsets of a n-element ground set. The values of f are integers, have absolute value at most M, and are evaluated by querying an oracle that takes time EO. Our goal is to produce an algorithm that solves this SFM problem, i.e. finds a minimizer of f, while minimizing both the number of oracle calls made and the total running time.

We provide new $O(n^2 \log nM \cdot \mathrm{EO} + n^3 \log^{O(1)} nM)$ and $O(n^3 \log^2 n \cdot \mathrm{EO} + n^4 \log^{O(1)} n)$ time algorithms for SFM. These algorithms improve upon the previous fastest weakly and strongly polynomial time algorithms for SFM which had a a running time of $O((n^4 \cdot \mathrm{EO} + n^5) \log M)$ [49] and $O(n^5 \cdot \mathrm{EO} + n^6)$ [81] respectively. Consequently, we improve the running times in both regimes by roughly a factor of $O(n^2)$.

Both of our algorithms bear resemblance to the classic approach of Grötschel, Lovász and Schrijver [44], [45] using the Lovász extension. In fact our weakly polynomial time algorithm directly uses the Lovász extension as well as the results of Section II to achieve these results. Our strongly polynomial time algorithm also uses the Lovász extension, along with more modern tools from the past 15 years.

At a high level, our strongly polynomial algorithms apply our cutting plane method in conjunction with techniques originally developed by Iwata, Fleischer, and Fujishige (IFF) [51]. Our cutting plane method is performed for

enough iterations to sandwich the feasible region in a narrow strip from which useful structural information about the minimizers can be deduced. Our ability to derive the new information hinges on a significant extension of IFF techniques.

Over the past few decades, SFM has drawn considerable attention from various research communities, most recently in machine learning [8], [62]. Given this abundant interest in SFM, we hope that our ideas will be of value in various practical applications. Indeed, one of the critiques against existing theoretical algorithms is that their running time is too slow to be practical. Our contribution, on the contrary, shows that this school of algorithms can actually be made fast theoretically and we hope it may potentially be competitive against heuristics which are more commonly used.

A. Previous Work

Here we provide a brief survey of the history of algorithms for SFM. For a more comprehensive account of the rich history of SFM, we refer the readers to recent surveys [72], [50].

The first weakly and strongly polynomial time algorithms for SFM were based on the ellipsoid method [60] and were established in the foundational work of Grötschel, Lovász and Schrijver in 1980's [44], [45]. Their work was complemented by a landmark paper by Cunningham in 1985 which provided a pseudopolynomial algorithm that followed a flow-style algorithmic framework [17]. His tools foreshadowed much of the development in SFM that would take place 15 years later. Indeed, modern algorithms synthesize his framework with inspirations from various max flow algorithms.

The first such "flow style" strongly polynomial algorithms for SFM were discovered independently in the breakthrough papers by Schrijver [84] and Iwata, Fleischer, and Fujishige (IFF) [51]. Schrijver's algorithm has a running of $O(n^8 \cdot \text{EO} + n^9)$ and borrows ideas from the push-relabel algorithms [41], [21] for the maximum flow problem. On the other hand, IFF's algorithm runs in time $O(n^7 \log n \cdot \text{EO})$ and $O(n^5 \cdot \text{EO} \log M)$, and applies a flow-scaling scheme with the aid of certain proximity-type lemmas as in the work of Tardos [88]. Their method has roots in flow algorithms such as [47], [42].

Subsequent work on SFM provided algorithms with considerably faster running time by extending the ideas in these two "genesis" papers [84], [51] in various novel directions [95], [26], [49], [81], [55]. Currently, the fastest weakly and strongly polynomial time algorithms for SFM have a running time of $O((n^4 \cdot \text{EO} + n^5) \log M)$ [49] and $O(n^5 \cdot \text{EO} + n^6)$ [81] respectively. Despite this impressive track record, the running time has not been improved in the last eight years.

We remark that all of the previous algorithms for SFM proceed by maintaining a convex combination of $O(n)$ BFS's of the base polyhedron, and incrementally improving it in a relatively local manner. As we shall discuss in Section III-B, our algorithms do not explicitly maintain a convex combination. This may be one of the fundamental reasons why our algorithms achieve a faster running time.

Finally, beyond the distinction between weakly and strongly polynomial time algorithms for SFM, there has been interest in another type of SFM algorithm, known as fully combinatorial algorithms in which only additions and subtractions are permitted. Previous such algorithms include [55], [49], [48]. We do not consider such algorithms in the remainder of the paper and leave it as an open question if it is possible to turn our algorithms into fully combinatorial ones.

B. Our Results and Techniques

In Section III we show how to improve upon the previous best known running times for SFM by a factor of $O(n^2)$ in both the strongly and weakly polynomial regimes. In Figure III.1 summarizes the running time of the previous algorithms as well as the running times of the fastest algorithms presented in this paper.

Both our weakly and strongly polynomial algorithms for SFM utilize a convex relaxation of the submodular function, called the *Lovász extension*. Our algorithms apply our cutting plane method from Section I using a separation oracle given by the subgradient of the Lovász extension. To the best of the author's knowledge, Grötschel, Lovász and Schrijver were the first to formulate this convex optimization framework for SFM [44], [45].

For weakly polynomial algorithms, our contribution is two-fold. First, we show that cutting plane methods such as Vaidya's [93] can be applied to SFM to yield faster algorithms. Second, as our cutting plane method, Theorem 4, improves upon previous cutting plane algorithms and consequently the running time for SFM as well. This gives a running time of $O(n^2 \log nM \cdot \text{EO} + n^3 \log^{O(1)} nM)$, an improvement over the previous best algorithm by Iwata [49] by a factor of almost $O(n^2)$.

Authors	Years	Running times	Remarks
Grötschel, Lovász, Schrijver [44], [45]	1981,1988	$\widetilde{O}(n^5 \cdot \text{EO} + n^7)$[72]	first weakly and strongly
Cunningham [17]	1985	$O(Mn^6 \log nM \cdot \text{EO})$	first pseudopoly
Schrijver [84]	2000	$O(n^8 \cdot \text{EO} + n^9)$	first combin. strongly
Iwata, Fleischer, Fujishige[51]	2000	$O(n^5 \cdot \text{EO} \log M)$ $O(n^7 \log n \cdot \text{EO})$	first combin. strongly
Iwata, Fleischer [26]	2000	$O(n^7 \cdot \text{EO} + n^8)$	
Iwata [49]	2003	$O((n^4 \cdot \text{EO} + n^5) \log M)$ $O((n^6 \cdot \text{EO} + n^7) \log n)$	current best weakly
Vygen [95]	2003	$O(n^7 \cdot \text{EO} + n^8)$	
Orlin [81]	2007	$O(n^5 \cdot \text{EO} + n^6)$	current best strongly
Iwata, Orlin [55]	2009	$O((n^4 \cdot \text{EO} + n^5) \log nM)$ $O((n^5 \cdot \text{EO} + n^6) \log n)$	
Our algorithms	2015	$O(n^2 \log nM \cdot \text{EO} + n^3 \log^{O(1)} nM)$ $O(n^3 \log^2 n \cdot \text{EO} + n^4 \log^{O(1)} n)$	

Figure III.1. Algorithms for submodular function minimization. Note that some of these algorithms were published in both conferences and journals, in which case the year we provided is the earlier one.

Theorem 16. *We have an $O(n^2 \log nM \cdot EO + n^3 \log^{O(1)} n \cdot \log nM)$ time algorithm for submodular function minimization.*

Our strongly polynomial algorithms, on the other hand, require substantially more innovation. We first begin with a very simple geometric argument that SFM can be solved in $O(n^3 \log n \cdot \text{EO})$ oracle calls (but in exponential time). This proof only uses Grunbaum's Theorem from convex geometry and is completely independent from the rest of the paper. It was the starting point of our method and suggests that a running time of $\widetilde{O}(n^3 \cdot \text{EO} + n^{O(1)})$ for submodular minimization is in principle achievable.

Lemma 17. *Given a function $f : 2^V \longrightarrow \mathbb{R}^n$ (not necessarily submodular) admits a relaxation $\hat{f}$ on $[0,1]^n$ such that $\hat{f}$ is convex and the set of minimizers of $\hat{f}$ is the convex hull of the set of minimizers of f. Then, one can compute a minimum point of f with only $O(n^2 \log n)$ calls of a subgradient oracle of $\hat{f}$ and $O(n^2)$ calls of a function value oracle of $\hat{f}$.*

To turn this observation into an efficient algorithm, we first run cutting plane, Theorem 2, for enough iterations such that we compute either a minimizer or a set P containing the minimizers that fits within in a narrow strip. This narrow strip consists of the intersection of two approximately parallel hyperplanes. If our narrow strip separates P from one of the faces $x_i = 0$, $x_i = 1$, we can effectively eliminate the element i from our consideration and reduce the dimension of our problem by 1. Otherwise a pair of elements p, q can be identified for which q is guaranteed to be in any minimizer containing p (but p may not be contained in a minimizer). Our first algorithm deduces only one such pair at a time. This technique immediately suffices to achieve a $\widetilde{O}(n^4 \cdot \text{EO} + n^5)$ time algorithm for SFM. We then improve the running time to $\widetilde{O}(n^3 \cdot \text{EO} + n^4)$ by showing how to deduce many such pairs simultaneously in time proportional to the number of pairs recovered. Similar to past algorithms, this structural information is deduced from a point in the so-called base polyhedron.

Theorem 18. *We have an $O(n^3 \log^2 n \cdot EO + n^4 \log^{O(1)} n)$. time algorithm for submodular function minimization.*

Readers well-versed in SFM literature may recognize that our strongly polynomial algorithms are reminiscent of the scaling-based approach first used by IFF [51] and later in [49], [55]. While both approaches share the same skeleton, there are differences as to how structural information about minimizers is deduced.

Finally, there is one more crucial difference between these algorithms which we believe is responsible for much of our speedup. One common feature shared by all the previous algorithms is that they maintain a convex combination of $O(n)$ BFS's of the base polyhedron, and incrementally improve on it by introducing new BFS's by making local changes to existing ones. Our algorithms, on the other hand, choose new BFS's by the cutting plane method. Because of this, our algorithm considers the geometry of the existing BFS's where each of them has influences over the choice of the next BFS. In some sense, our next BFS is chosen in a more "global" manner.

One natural question that arises from our work is whether our algorithm achieves the best possible time complexity. Earlier works on SFM (e.g. Schrijver's algorithm [84]) suggest that the logarithmic factors in our running time should be avoidable. More interestingly, we wonder if the oracle complexity of $\widetilde{O}(n^3)$ can be improved to $\widetilde{O}(n^{3-\epsilon})$. A positive solution would require a breakthrough in our understanding of the structure of submodular functions.

Conjecture 19. *Can one remove the logarithmic factor dependence in the running time of our algorithms? Is a polynomial improvement, i.e. an oracle complexity of $\widetilde{O}(n^{3-\epsilon})$, possible?*

ACKNOWLEDGMENTS

We thank Matt Weinberg for insightful comments about submodular minimization and minimizing the intersection of convex sets that were deeply influential to our work. We thank Yan Kit Chim, Jonathan A. Kelner, Robert Kleinberg, Pak-Hin Lee, Christos Papadimitriou, and Chit Yu Ng and for many helpful conversations. This work was partially supported by NSF awards 0843915 and 1111109, NSF grants CCF0964033 and CCF1408635, Templeton Foundation grant 3966, NSF Graduate Research Fellowship (grant no. 1122374). Part of this work was done while the first two authors were visiting the Simons Institute for the Theory of Computing, UC Berkeley. Lastly, we would like to thank Vaidya for his beautiful work on his cutting plane method.

REFERENCES

[1] Martin Aigner and Thomas A Dowling. Matching theory for combinatorial geometries. *Transactions of the American Mathematical Society*, 158(1):231–245, 1971.

[2] Zeyuan Allen-Zhu, Yin Tat Lee, and Lorenzo Orecchia. Using optimization to obtain a width-independent, parallel, simpler, and faster positive sdp solver. *arXiv preprint arXiv:1507.02259*, 2015.

[3] Kurt M Anstreicher. On vaidya's volumetric cutting plane method for convex programming. *Mathematics of Operations Research*, 22(1):63–89, 1997.

[4] Kurt M Anstreicher. The volumetric barrier for semidefinite programming. *Mathematics of Operations Research*, 25(3):365–380, 2000.

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

[6] Sanjeev Arora and Satyen Kale. A combinatorial, primal-dual approach to semidefinite programs. In *Proceedings of the thirty-ninth annual ACM symposium on Theory of computing*, pages 227–236. ACM, 2007.

[7] David S Atkinson and Pravin M Vaidya. A cutting plane algorithm for convex programming that uses analytic centers. *Mathematical Programming*, 69(1-3):1–43, 1995.

[8] Francis Bach. Learning with submodular functions: A convex optimization perspective. *Foundations and Trends in Machine Learning*, 2013.

[9] Francisco Barahona and William H Cunningham. A submodular network simplex method. In *Mathematical Programming at Oberwolfach II*, pages 9–31. Springer, 1984.

[10] Dimitris Bertsimas and Santosh Vempala. Solving convex programs by random walks. *Journal of the ACM (JACM)*, 51(4):540–556, 2004.

[11] Carl Brezovec, Gerard Cornuéjols, and Fred Glover. Two algorithms for weighted matroid intersection. *Mathematical Programming*, 36(1):39–53, 1986.

[12] Sébastien Bubeck, Yin Tat Lee, and Mohit Singh. A geometric alternative to nesterov's accelerated gradient descent. *arXiv preprint arXiv:1506.08187*, 2015.

[13] Yang Cai, Constantinos Daskalakis, and S Matthew Weinberg. Optimal multi-dimensional mechanism design: Reducing revenue to welfare maximization. In *Foundations of Computer Science (FOCS), 2012 IEEE 53rd Annual Symposium on*, pages 130–139. IEEE, 2012.

[14] Yang Cai, Constantinos Daskalakis, and S Matthew Weinberg. Reducing revenue to welfare maximization: Approximation algorithms and other generalizations. In *Proceedings of the Twenty-Fourth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 578–595. SIAM, 2013.

[15] Nam-Kee Chung and Dong-Wan Tcha. A dual algorithm for submodular flow problems. *Operations research letters*, 10(8):489–495, 1991.

[16] Michael B. Cohen, Yin Tat Lee, Cameron Musco, Christopher Musco, Richard Peng, and Aaron Sidford. Uniform sampling for matrix approximation. *CoRR*, abs/1408.5099, 2014.

[17] William H Cunningham. On submodular function minimization. *Combinatorica*, 5(3):185–192, 1985.

[18] William H Cunningham. Improved bounds for matroid partition and intersection algorithms. *SIAM Journal on Computing*, 15(4):948–957, 1986.

[19] William H Cunningham and András Frank. A primal-dual algorithm for submodular flows. *Mathematics of Operations Research*, 10(2):251–262, 1985.

[20] Constantinos Daskalakis and S Matthew Weinberg. Bayesian truthful mechanisms for job scheduling from bi-criterion approximation algorithms. In *Proceedings of the Twenty-Sixth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 1934–1952. SIAM, 2015.

[21] EA Dinic. An algorithm for the solution of the max-flow problem with the polynomial estimation. *Doklady Akademii Nauk SSSR*, 194(4):1277–1280, 1970.

[22] Jack Edmonds. Matroid partition. *Mathematics of the Decision Sciences*, 11:335–345, 1968.

[23] Jack Edmonds. Submodular functions, matroids, and certain polyhedra. *Edited by G. Goos, J. Hartmanis, and J. van Leeuwen*, page 11, 1970.

[24] Jack Edmonds. Matroid intersection. *Annals of discrete Mathematics*, 4:39–49, 1979.

[25] Lisa Fleischer and Satoru Iwata. Improved algorithms for submodular function minimization and submodular flow. In *Proceedings of the thirty-second annual ACM symposium on Theory of computing*, pages 107–116. ACM, 2000.

[26] Lisa Fleischer and Satoru Iwata. A push-relabel framework for submodular function minimization and applications to parametric optimization. *Discrete Applied Mathematics*, 131(2):311–322, 2003.

[27] Lisa Fleischer, Satoru Iwata, and S Thomas McCormick. A faster capacity scaling algorithm for minimum cost submodular flow. *Mathematical Programming*, 92(1):119–139, 2002.

[28] András Frank. A weighted matroid intersection algorithm. *Journal of Algorithms*, 2(4):328–336, 1981.

[29] András Frank and Éva Tardos. An application of simultaneous diophantine approximation in combinatorial optimization. *Combinatorica*, 7(1):49–65, 1987.

[30] S. Fujishige. Algorithms for solving the independent-flow problems. *Journal of the Operations Research Society of Japan*, 1978.

[31] Satoru Fujishige. An out-of-kilter method for submodular flows. *Discrete applied mathematics*, 17(1):3–16, 1987.

[32] Satoru Fujishige and Satoru Iwata. Algorithms for submodular flows. *IEICE TRANSACTIONS on Information and Systems*, 83(3):322–329, 2000.

[33] Satoru Fujishige, Hans Röck, and Uwe Zimmermann. A strongly polynomial algorithm for minimum cost submodular flow problems. *Mathematics of Operations Research*, 14(1):60–69, 1989.

[34] Satoru Fujishige and Zhang Xiaodong. An efficient cost scaling algorithm for the independent assignment problem. *Journal of the Operations Research Society of Japan*, 38(1):124–136, 1995.

[35] Mituhiro Fukuda, Masakazu Kojima, Kazuo Murota, and Kazuhide Nakata. Exploiting sparsity in semidefinite programming via matrix completion i: General framework. *SIAM Journal on Optimization*, 11(3):647–674, 2001.

[36] François Le Gall. Powers of tensors and fast matrix multiplication. *arXiv preprint arXiv:1401.7714*, 2014.

[37] J-L Goffin, Jacek Gondzio, Robert Sarkissian, and J-P Vial. Solving nonlinear multicommodity flow problems by the analytic center cutting plane method. *Mathematical Programming*, 76(1):131–154, 1997.

[38] Jean-Louis Goffin, Zhi-Quan Luo, and Yinyu Ye. Complexity analysis of an interior cutting plane method for convex feasibility problems. *SIAM Journal on Optimization*, 6(3):638–652, 1996.

[39] Jean-Louis Goffin and Jean-Philippe Vial. Shallow, deep and very deep cuts in the analytic center cutting plane method. *Mathematical Programming*, 84(1):89–103, 1999.

[40] Jean-Louis Goffin and Jean-Philippe Vial. Convex nondifferentiable optimization: A survey focused on the analytic center cutting plane method. *Optimization Methods and Software*, 17(5):805–867, 2002.

[41] Andrew V Goldberg and Robert E Tarjan. A new approach to the maximum-flow problem. *Journal of the ACM (JACM)*, 35(4):921–940, 1988.

[42] Andrew V Goldberg and Robert E Tarjan. Finding minimum-cost circulations by successive approximation. *Mathematics of Operations Research*, 15(3):430–466, 1990.

[43] Jacek Gondzio, O Du Merle, Robert Sarkissian, and J-P Vial. Accpmï¿œxa library for convex optimization based on an analytic center cutting plane method. *European Journal of Operational Research*, 94(1):206–211, 1996.

[44] Martin Grötschel, László Lovász, and Alexander Schrijver. The ellipsoid method and its consequences in combinatorial optimization. *Combinatorica*, 1(2):169–197, 1981.

[45] Martin Grötschel, László Lovász, and Alexander Schrijver. Geometric algorithms and combinatorial optimization. Springer, 1988.

[46] Christoph Helmberg and Franz Rendl. A spectral bundle method for semidefinite programming. *SIAM Journal on Optimization*, 10(3):673–696, 2000.

[47] Satoru Iwata. A capacity scaling algorithm for convex cost submodular flows. *Mathematical programming*, 76(2):299–308, 1997.

[48] Satoru Iwata. A fully combinatorial algorithm for submodular function minimization. *Journal of Combinatorial Theory, Series B*, 84(2):203–212, 2002.

[49] Satoru Iwata. A faster scaling algorithm for minimizing submodular functions. *SIAM Journal on Computing*, 32(4):833–840, 2003.

[50] Satoru Iwata. Submodular function minimization. *Mathematical Programming*, 112(1):45–64, 2008.

[51] Satoru Iwata, Lisa Fleischer, and Satoru Fujishige. A combinatorial strongly polynomial algorithm for minimizing submodular functions. *Journal of the ACM (JACM)*, 48(4):761–777, 2001.

[52] Satoru Iwata, S Thomas McCormick, and Maiko Shigeno. A faster algorithm for minimum cost submodular flows. In *SODA*, pages 167–174, 1998.

[53] Satoru Iwata, S Thomas McCormick, and Maiko Shigeno. A strongly polynomial cut canceling algorithm for the submodular flow problem. In *Integer Programming and Combinatorial Optimization*, pages 259–272. Springer, 1999.

[54] Satoru Iwata, S Thomas McCormick, and Maiko Shigeno. A fast cost scaling algorithm for submodular flow. *Information Processing Letters*, 74(3):123–128, 2000.

[55] Satoru Iwata and James B Orlin. A simple combinatorial algorithm for submodular function minimization. In *Proceedings of the twentieth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 1230–1237. Society for Industrial and Applied Mathematics, 2009.

[56] Rahul Jain and Penghui Yao. A parallel approximation algorithm for positive semidefinite programming. In *Foundations of Computer Science (FOCS), 2011 IEEE 52nd Annual Symposium on*, pages 463–471. IEEE, 2011.

[57] Klaus Jansen. Approximate strong separation with application in fractional graph coloring and preemptive scheduling. *Theoretical Computer Science*, 302(1):239–256, 2003.

[58] Ravindran Kannan and Hariharan Narayanan. Random walks on polytopes and an affine interior point method for linear programming. *Mathematics of Operations Research*, 37(1):1–20, 2012.

[59] Richard M Karp and Christos H Papadimitriou. On linear characterizations of combinatorial optimization problems. *SIAM Journal on Computing*, 11(4):620–632, 1982.

[60] Leonid G Khachiyan. Polynomial algorithms in linear programming. *USSR Computational Mathematics and Mathematical Physics*, 20(1):53–72, 1980.

[61] LG Khachiyan, SP Tarasov, and II Erlikh. The method of inscribed ellipsoids. In *Soviet Math. Dokl*, volume 37, pages 226–230, 1988.

[62] Andreas Krause. http://submodularity.org/.

[63] Kartik Krishnan and John E Mitchell. A unifying framework for several cutting plane methods for semidefinite programming. *Optimization methods and software*, 21(1):57–74, 2006.

[64] Kartik Krishnan and John E Mitchell. Properties of a cutting plane method for semidefinite programming. *Pacific Journal of Optimization*, 8(4):779–802, 2012.

[65] Eugene L Lawler. Matroid intersection algorithms. *Mathematical programming*, 9(1):31–56, 1975.

[66] Yin Tat Lee and Aaron Sidford. Path finding ii: An\˜ o (m sqrt (n)) algorithm for the minimum cost flow problem. *arXiv preprint arXiv:1312.6713*, 2013.

[67] Yin Tat Lee and Aaron Sidford. Path-finding methods for linear programming : Solving linear programs in õ(sqrt(rank)) iterations and faster algorithms for maximum flow. In *55th Annual IEEE Symposium on Foundations of Computer Science, FOCS 2014, 18-21 October, 2014, Philadelphia, PA, USA*, pages 424–433, 2014.

[68] Yin Tat Lee and Aaron Sidford. Efficient inverse maintenance and faster algorithms for linear programming. *arXiv preprint arXiv:1503.01752*, 2015.

[69] A. Yu Levin. On an algorithm for the minimization of convex functions. *Soviet Math. Doklady*, 1965.

[70] Mu Li, Gary L Miller, and Richard Peng. Iterative row sampling. 2012.

[71] László Lovász and Santosh Vempala. Simulated annealing in convex bodies and an $o^*(n^4)$ volume algorithm. *J. Comput. Syst. Sci.*, 72(2):392–417, 2006.

[72] S McCormick. *Submodular Function Minimization*. 2013.

[73] S Thomas and McCormick. Canceling most helpful total submodular cuts for submodular flow. In *IPCO*, pages 343–353, 1993.

[74] Renato DC Monteiro. First-and second-order methods for semidefinite programming. *Mathematical Programming*, 97(1-2):209–244, 2003.

[75] Kazuhide Nakata, Katsuki Fujisawa, Mituhiro Fukuda, Masakazu Kojima, and Kazuo Murota. Exploiting sparsity in semidefinite programming via matrix completion ii: Implementation and numerical results. *Mathematical Programming*, 95(2):303–327, 2003.

[76] Arkadi Nemirovski. Efficient methods in convex programming. 1994.

[77] D. B. Nemirovsky, A. S., & Yudin. Problem complexity and method efficiency in optimization. 1983.

[78] Yu Nesterov. Complexity estimates of some cutting plane methods based on the analytic barrier. *Mathematical Programming*, 69(1-3):149–176, 1995.

[79] Yu Nesterov and Arkadi Nemirovskiy. *Self-concordant functions and polynomial-time methods in convex programming*. USSR Academy of Sciences, Central Economic & Mathematic Institute, 1989.

[80] Yu Nesterov and A Nemirovsky. Conic formulation of a convex programming problem and duality. *Optimization Methods and Software*, 1(2):95–115, 1992.

[81] James B Orlin. A faster strongly polynomial time algorithm for submodular function minimization. *Mathematical Programming*, 118(2):237–251, 2009.

[82] James B Orlin, John VandeVate, et al. On a" primal" matroid intersection algorithm. 1983.

[83] Srinivasan Ramaswamy and John E Mitchell. A long step cutting plane algorithm that uses the volumetric barrier. *Department of Mathematical Science, RPI, Troy, NY*, 1995.

[84] Alexander Schrijver. A combinatorial algorithm minimizing submodular functions in strongly polynomial time. *Journal of Combinatorial Theory, Series B*, 80(2):346–355, 2000.

[85] Maiko Shigeno and Satoru Iwata. A dual approximation approach to weighted matroid intersection. *Operations research letters*, 18(3):153–156, 1995.

[86] Naum Z Shor. Cut-off method with space extension in convex programming problems. *Cybernetics and systems analysis*, 13(1):94–96, 1977.

[87] Daniel A Spielman and Nikhil Srivastava. Graph sparsification by effective resistances. *SIAM Journal on Computing*, 40(6):1913–1926, 2011.

[88] Éva Tardos. A strongly polynomial minimum cost circulation algorithm. *Combinatorica*, 5(3):247–255, 1985.

[89] Michael J Todd. Semidefinite optimization. *Acta Numerica 2001*, 10:515–560, 2001.

[90] Nobuaki Tomizawa and Masao Iri. Algorithm for determining rank of a triple matrix product axb with application to problem of discerning existence of unique solution in a network. *ELECTRONICS & COMMUNICATIONS IN JAPAN*, 57(11):50–57, 1974.

[91] Pravin M. Vaidya. A new algorithm for minimizing convex functions over convex sets (extended abstract). In *FOCS*, pages 338–343, 1989.

[92] Pravin M Vaidya. Speeding-up linear programming using fast matrix multiplication. In *Foundations of Computer Science, 1989., 30th Annual Symposium on*, pages 332–337. IEEE, 1989.

[93] Pravin M Vaidya. A new algorithm for minimizing convex functions over convex sets. *Mathematical Programming*, 73(3):291–341, 1996.

[94] Lieven Vandenberghe and Stephen Boyd. Semidefinite programming. *SIAM review*, 38(1):49–95, 1996.

[95] Jens Vygen. A note on schrijver's submodular function minimization algorithm. *Journal of Combinatorial Theory, Series B*, 88(2):399–402, 2003.

[96] S. Fujishige W. Cui. A primal algorithm for the submodular flow problem with minimum mean cycle selection. *Journal of the Operations Research Society of Japan*, 1988.

[97] C Wallacher and Uwe T Zimmermann. A polynomial cycle canceling algorithm for submodular flows. *Mathematical programming*, 86(1):1–15, 1999.

[98] Virginia Vassilevska Williams. Multiplying matrices faster than coppersmith-winograd. In *Proceedings of the forty-fourth annual ACM symposium on Theory of computing*, pages 887–898. ACM, 2012.

[99] Yinyu Ye. Complexity analysis of the analytic center cutting plane method that uses multiple cuts. *Mathematical Programming*, 78(1):85–104, 1996.

[100] David B Yudin and Arkadii S Nemirovski. Evaluation of the information complexity of mathematical programming problems. *Ekonomika i Matematicheskie Metody*, 12:128–142, 1976.

[101] U Zimmermann. Minimization on submodular flows. *Discrete Applied Mathematics*, 4(4):303–323, 1982.

[102] Uwe Zimmermann. Negative circuits for flows and submodular flows. *Discrete applied mathematics*, 36(2):179–189, 1992.