CS 435, 2018
Lecture 10, Date: 24 May 2018
Instructor: Nisheeth Vishnoi

Convex Programming using the Ellipsoid Method

In this lecture, we show how to adapt the Ellipsoid method to solve more general convex programs other than linear programming. This allows us to give a polynomial time algorithm for submodular minimization and apply it to the problem of computing maximum entropy distributions. We end with variants of the Ellipsoid method due to Vaidya and Lee-Sidford-Wong.

Contents

1 Convex Optimization: What have we done so far?

The main problem studied in this course is convex programming, i.e.

$$\begin{aligned} \min \quad & f(x) \\ \text{s.t.} \quad & x \in K \end{aligned} \tag{1}$$

where $f : \mathbb{R}^n \to \mathbb{R}$ is a convex function and $K \subseteq \mathbb{R}^n$ is a convex set. Let us now summarize and briefly review what we have already learned in previous lectures.

1.1 First order methods for unconstrained optimization

We started by studying the unconstrained optimization case (i.e. $K = \mathbb{R}^n$) and introduced a very general scheme, called gradient descent, to tackle convex programs in such a form. The common feature of all algorithms based on this scheme was that they used only the first order information about f to perform iterations. The running time bounds for first order methods (even after applying acceleration) were of the form

$$\text{poly}\left(R, L, \frac{1}{\varepsilon}\right),$$

where R is the distance from the initial point x_0 to the optimal solution, L is some regularity parameter of f (either a smoothness bound or a bound on the maximum magnitude of the gradient) and $\varepsilon > 0$ is the desired precision.

One regime where one could get an improvement over the polynomial dependence on these parameters was under strong regularity conditions on f. More precisely, under the guarantee that $mI \preceq \nabla^2 f(x) \preceq MI$ and $\kappa := \frac{M}{m}$ a gradient descent-based algorithm can be made work in time roughly

$$O\left(\kappa \cdot \log \frac{R}{\varepsilon}\right).$$

Note that κ being small is a rather strong requirement. Indeed for non-smooth (for instance piecewise linear) functions, M is very large or even infinite, and m is 0 for linear functions.

1.2 Second order methods: path following

There are two clear downsides of first order methods, as presented above. The first of them is that they cannot really be applied to constrained optimization. In fact, other than for very simple sets K such as the hypercube or the unit ball, the gradient descent method is not applicable, since it requires efficient algorithms to perform projection onto K (in every iteration of the algorithm). The second downside concerns the running time and specifically the fact that it depends polynomially on the error ε and the distance to the optimizer R.

A consequence of these two issues is the fact that one cannot solve linear programming by applying first order methods directly (at least when we intend to solve it exactly or up to an exponentially small error). To bypass this issue, the second order Newton's method for optimization was introduced and shown to converge quadratically fast (with dependency $\log\log\frac{1}{\varepsilon}$) for so called self-concordant functions (satisfying a certain smoothness condition for the Hessian). This result we used as the crucial building block to derive the path following method for convex programming. It implies that whenever an efficient self-concordant barrier function $\phi : K \to \mathbb{R}$ can be constructed for K, then we can optimize any linear function over K in roughly $\sqrt{\nu}\log\frac{1}{\varepsilon}$ iterations, where ν is the so called complexity parameter of the barrier function ϕ.

One of the implications of the path following algorithm, obtained by considering the logarithmic barrier for polytopes, is a polynomial time algorithm for linear programming, i.e., the task of minimizing $c^\top x$ over $x \in P := \{x \in \mathbb{R}^n : Ax \leq b\}$ where $A \in \mathbb{Q}^{m\times n}$ and $b \in \mathbb{Q}^m$ and $c \in \mathbb{Q}^n$. Moreover, when combined with a fast Laplacian solver, it can be used to derive the fastest known algorithm for minimum cost flow.

1.3 Linear programming using ellipsoid

While the path following interior point method leads to some extremely fast algorithms (both in theory and in practice) it crucially relies on existence of efficient self-concordant barrier functions for the fesible set K. For explicitly described linear programs or for unit balls in various norms one can construct such barrier functions, however no efficient barrier functions are known for combinatorial polytopes such as the matching polytope or the spanning tree polytope. Still, there are many interesting optimization problems over these sets, one would like to solve.

Towards constructing an algorithm for optimizing over a matching polytope we observed that this polytope has an efficient (polynomial time) separation oracle, and thus the most basic computational problem – separation – can be solved over such a polytope. Subsequently, we developed the ellipsoid algorithm that gives a method to optimize over such a polytope in polynomial time given just a separation oracle. This is a rather surprising result, since the matching polytope has exponentially many facets.

1.4 Convex optimization using ellipsoid?

The ellipsoid algorithm for linear programming that was developed in the previous lecture had several desirable features:

1. it requires only a separation oracle to work (black box access),

2. its running time depends polylogarithmically on $\frac{1}{\varepsilon}$ (with $\varepsilon > 0$ being the error),

3. its running time depends polylogarithmically on R, where $R > 0$ is the radius of the initial ellipsoid (i.e., roughly the distance to optimal solution).

Property (1) shows that ellipsoid is applicable more generally than the interior point method, which in contrast requires a stronger access – a self-concordant barrier function. Properties (2), (3) say that (at least asymptotically), ellipsoid outperforms first order methods. Note however, that so far the above have been only derived for the linear programming problem, i.e., when K is a polytope and f is a linear function. In this lecture we are going to extend it to **general convex sets** and **convex functions** assuming appropriate (oracle) access to f and separation oracle access to K. More precisely, in Section 4 we prove the following theorem

Theorem 1 (Ellipsoid algorithm for Convex Optimization)**.** *There is an algorithm such that given*

1. *a* 1*st order oracle for a convex function* $f : \mathbb{R}^n \to \mathbb{R}$*,*
2. *a separation oracle for a convex set* $K \subseteq \mathbb{R}^n$*,*
3. *numbers* $r, R > 0$ *such that* $K \subseteq B(R, 0)$ *and* K *contains a Euclidean ball of radius* r*,*
4. *bounds* l_0, u_0 *such that* $\forall x \in K \quad l_0 \leq f(x) \leq u_0$*,*
5. *an* $\varepsilon > 0$*,*

outputs a point $\hat{x} \in K$ *such that*

$$f(\hat{x}) \leq f(x^\star) + \varepsilon,$$

where $x^\star$ *is any minimizer of* f *over* K*. The running time of the algorithm is*

$$O\left(\left(n^2 + T_K + T_f\right) \cdot n^2 \cdot \log\left(\frac{R}{r} \cdot \frac{u_0 - l_0}{\varepsilon}\right)^2\right),$$

where T_K *and* T_f *are the running times of the separation oracle for* K *and a gradient oracle for* f *respectively.*

A detailed discussion of the assumptions of Theorem 1 is provided in Section 4. Here let us mention only that a 1st order oracle for a function f is understood as a primitive that given x outputs the value $f(x)$ and any subgradient $g(x) \in \partial f(x)$.

1.5 Is convex programming in P?

Given an algorithm as promised in the above section, one might want to reconsider the question of polynomial time solvability of general convex programs that has been mentioned (and answered negatively) in Lecture 2. Indeed, even though the result proved in this lecture seems to imply that convex programming is in **P**, it is not the case, as it relies on a few subtle, yet important assumptions. To construct a polynomial time algorithm for a specific convex program one has to first find a suitable upper bound R on the magnitude of the optimal solution – no algorithm can (provably) efficiently find the solution without such a bound. For the ellipsoid algorithm, such a bound has to be provided as input to even run the algorithm (see Section 4). Moreover, the algorithm requires a separation oracle for K and an oracle to compute gradients of f – both these computational task might turn out hard, even $\#P$-hard in certain cases!

In this lecture we provide two examples of problems: submodular function minimization and computing maximum entropy distributions. Both these problems can in the end can be reduced to optimizing convex functions over convex domains, however, the existence of a bound R for the entropy maximization and computability of gradients for both these problems, are far from trivial. This shows that even if a problem can be formulated as a convex program, still additional work is required to conclude polynomial time computability.

2 Submodular Function Minimization

In this section we study the submodular function minimization problem. In several lectures we have already encountered submodular functions and the problem of submodular minimization, although it has never been mentioned explicitly. Below we show how it arises rather naturally when trying to construct separation oracles for combinatorial polytopes.

2.1 Separation Oracles for $0-1$ polytopes

Recall that in the last lecture we proved that one can efficiently optimize linear functions over $0-1$ polytopes

$$P_{\mathcal{F}} = \mathrm{conv}\{1_S : S \in \mathcal{F}\} \subseteq [0,1]^n,$$

where $\mathcal{F} \subseteq 2^{[n]}$ (a family of subsets of $[n] := \{1,2,\ldots,n\}$ whenever one can provide an efficient separation oracle for $P_{\mathcal{F}}$. We saw also that for certain examples, such as the perfect matching polytope (see Problem Set 9), one can construct such polynomial time separation oracles by reducing it to an "easy to solve" discrete problem (in that case: minimum cut in a graph). Below we obtain a similar result – separation oracles for a large family of polytopes called *matroid polytopes*. We start by defining a rank function $r_{\mathcal{F}} : 2^{[n]} \to \mathbb{N}$ as

$$r_{\mathcal{F}}(S) = \max\{|T| : T \in \mathcal{F}, T \subseteq [n]\},$$

i.e. the maximum cardinality of a set in $T \in \mathcal{F}$ contained in S. Note that $r_{\mathcal{F}}(S)$ might not be well defined in general – however it is for downward-closed families (i.e. families $\mathcal{F}$ such that for every $S \subseteq T \subseteq [n]$, if $T \in \mathcal{F}$ then also $S \in \mathcal{F}$). Of special interest in this section are set families that are *matroids*

Definition 1. *A non-empty family of subsets $\mathcal{F}$ of a ground set $[n]$ is called a matroid if it satisfies the following conditions:*

1. *$\mathcal{F}$ is downward-closed,*
2. *if $A, B \in \mathcal{F}$ and $|B| > |A|$ then there exists $x \in B \setminus A$ such that $B \cup \{x\} \in \mathcal{F}$.*

The second condition is often called the exchange axiom. For a thorough treatment of matroid theory in combinatorial optimization we refer to [8]. Here, let us mention only that the set of all subsets of $[n]$ of cardinality at most k (for arbitrary k) is a matroid. Similarly, if G is an undirected graph, the set of all acyclic subsets of edges of G forms a matroid. The following important theorem provides a convenient description of the polytope $\mathcal{F}$ when $\mathcal{F}$ is a matroid, see [8] for a proof

Theorem 2 (Matroid Polytopes)**.** *Suppose that $\mathcal{F} \subseteq 2^{[n]}$ is a matroid, then*

$$P_{\mathcal{F}} = \{x \in [0,1]^n : \forall S \subseteq [n] \ \ \sum_{i \in S} x_i \leq r_{\mathcal{F}}(S)\}.$$

The inclusion from left to right ($\subseteq$) is true for all families $\mathcal{F}$, the opposite direction relies on the matroid assumption and is nontrivial. Theorem 2 suggests the following strategy for separation over matroid polytopes. Given $x \in [0,1]^n$ we denote by $F(S) := r_{\mathcal{F}}(S) - \sum_{i \in S} x_i$, we would like to find

$$S^\star := \operatorname*{argmin}_{S \subseteq [n]} f(S).$$

Indeed,

$$x \in P_{\mathcal{F}} \quad \Leftrightarrow \quad F(S^\star) \geq 0.$$

Furthermore, if $F(S^\star) < 0$ then $S^\star$ provides us with a separating hyperplane: $1_{S^\star}$. Thus, in order to solve the separation problem we need to solve the minimization problem for F. The crucial observation is now that F is not an arbitrary function – it is submodular, and thus has a lot of additional combinatorial structure.

Definition 2 (Submodular Function)**.** *A function $f : 2^{[n]} \to \mathbb{R}$ is called submodular if*

$$\forall S, T \subseteq [n] \quad F(S \cap T) + F(S \cup T) \leq F(S) + F(T).$$

As alluded before, the matroid rank function $r_{\mathcal{F}}$ is submodular, which is a classical fact in matroid theory.

Theorem 3 (Matroid Rank is Submodular)**.** *Suppose that $\mathcal{F} \subseteq 2^{[n]}$ is a matroid, then the rank function $r_{\mathcal{F}}$ is submodular.*

Given the above discussion, it is natural to consider the following general problem.

Submodular Function Minimization (SFM)

Input: A submodular function $F : 2^{[n]} \to \mathbb{R}$

Goal: Find a set $S^\star$ such that

$$S^\star = \operatorname*{argmin}_{S \subseteq [n]} F(S).$$

We have not specified how is the function F given on input – depending on the application it can be either succinctly described using a graph (or some other combinatorial structure), or given as an oracle. For the sake of generality, it is most convenient to think of F as given by an oracle that outputs $F(S)$ given $S \subseteq [n]$.

In Section A we present an application of SFM to the image denoising problem.

2.2 Theorem Statement

In this lecture we prove that we can minimize submodular functions in a very general scenario – all what we need is an oracle that given S outputs $f(S)$.

Theorem 4 (Polynomial Time Algorithm for SFM). *There is an algorithm that given an oracle access to a submodular function $F : 2^{[n]} \to [l, u]$ and an $\varepsilon > 0$ finds a set $S \subseteq [n]$ such that*

$$F(S) \leq F(S^\star) + \varepsilon,$$

where $S^\star$ is a minimizer of f. The algorithm runs in $\mathrm{poly}\left(n, \log \varepsilon^{-1}, \log(u-l)\right)$ *time and makes* $\mathrm{poly}\left(n, \log \varepsilon^{-1}, \log(u-l)\right)$ *queries to the oracle.*

As a consequence of Theorem 4 we obtain that the separation problem for matroid polytopes $P_{\mathcal{F}}$ can be solved given only an oracle that computes $r_{\mathcal{F}}$. This in turn is easy for any matroid given the weakest possible form of access to $\mathcal{F}$ – the membership oracle (given S, answer whether $S \in \mathcal{F}$ or not). It is one of the basic properties of matroids that one can find a maximum cardinality subset using a greedy procedure (see [8]). Thus, as a consequence we obtain the following general theorem.

Theorem 5 (Efficient Separation and Optimization over Matroid Polytopes). *There is an algorithm that given a membership oracle access to a matroid $\mathcal{F} \subseteq 2^{[n]}$ solves the separation problem over $P_{\mathcal{F}}$ in polynomial time.*

2.3 Towards an algorithm for SFM – Lovasz Extension

The first step in solving the SFM problem will be, as in the last lecture, to turn it into a continuous problem – ideally a convex optimization problem. We define the so called *Lovasz Extension* of $F : 2^{[n]} \to \mathbb{R}$.

Definition 3 (Lovasz Extension). *Let $F : 2^{[n]} \to \mathbb{R}$ be a function. We define the Lovasz Extension of F to be a function $f : [0,1]^n \to \mathbb{R}$ such that*

$$f(x) = \mathbb{E}\left[f(\{i : x_i < \lambda\})\right],$$

where the expectation is taken over a uniformly random choice of $\lambda \in [0,1]$.

We derive several important properties of the Lovasz extension in Section 5. Here let us only observe that the Lovasz extension of F is always a continuous function that agrees with F on integer vectors. Moreover, it is a simple exercise to show that

$$\min_{x \in [0,1]^n} f(x) = \min_{S \subseteq [n]} F(S).$$

Thus, to minimize F it is enough to design an algorithm to minimize f. A crucial theorem saying that under submodularity the resulting function over $[0,1]^n$ is convex was shown by [7].

Theorem 6 (Convexity of the Lovasz Extension [7])**.** *If a set function $F : 2^{[n]} \to \mathbb{R}$ is submodular then its Lovasz extension $f : [0,1]^n \to \mathbb{R}$ is convex.*

Thus our goal becomes now (seemingly) much more approachable than before, find:

$$\min_{x \in [0,1]^n} f(x), \tag{2}$$

where f is a convex function.

As a first attempt one can try to use first order methods, such as gradient descent or mirror descent, to solve this problem. In fact (given appropriate access to f) one can use[1] some of these methods to solve (2) up to an arbitrary precision ε in time polynomial in ε^{-1} and the maximum magnitude of a subgradient of f. As discussed in Lecture 2, such an algorithm is not regarded as polynomial time in the input parameters.

For the application to constructing separation oracles for matroid polytopes a polynomial running time with respect to $\log \varepsilon^{-1}$ (where $\varepsilon > 0$ is the precision) is necessary. In fact, one has to recover an exact optimum to provide a separating hyperplane when facing a point outside of the polytope. This can be achieved using similar techniques (perturbation and rounding) as used in Lecture 9 to optimize over $0-1$ polytopes. However, it crucially relies on the logarithmic dependency on the error, to yield a polynomial time algorithm.

2.4 Polynomial time algorithm for submodular function minimization

To obtain a polynomial time algorithm for optimizing convex functions over convex sets, in Section 4 we generalize the ellipsoid algorithm, introduced in the previous lecture. The following aspects of the ellipsoid algorithm need to be adjusted for it to be applicable in this new setting.

- Previously we worked with feasible sets being polytopes only – now we want to drop this assumption and work with general convex bodies.[2]

- We need to generalize the algorithm from linear objectives to convex objectives.

[1] The Lovasz extension is not smooth hence one would need to use mirror descent, as introduced in Lecture 4. This then requires a bound on the maximum magnitude G of the subgradient of f and yields an additive approximation of $\varepsilon > 0$ in $\approx \frac{G^2 n}{\varepsilon^2}$ iterations.

[2] Even though the domain of the convex problem resulting from submodular minimization is the hypercube $[0,1]^n$ (a polytope) we still need to consider more general domains, as the reduction from optimization to feasibility might yield such: level sets of the objective function.

Both these steps bring new challenges. However, the overall scheme of an algorithm that we will use to solve this general problem is similar to the previous lecture (ellipsoid algorithm combined with binary search). In the setting of optimizing a convex function f over a hypercube, the result we develop in Section 4 implies

Theorem 7 (Informal; see Theorem 1)**.** *Let $f : [0,1]^n \to \mathbb{R}$ be a convex function and suppose the following conditions are satisfied:*

1. *we are given access to a polynomial time oracle computing the value and the (sub)gradient of f,*
2. *the values of f over $[0,1]^n$ lie in the interval $[l,u]$ for some $l,u \in R$.*

Then, there is an algorithm that given $\varepsilon > 0$ outputs an ε-approximate solution to $\min_{x\in[0,1]^n} f(x)$ in time $\mathrm{poly}(n, \log(u-l), \log\varepsilon^{-1})$.

Note that given the above theorem the only remaining step is to establish efficient oracles for computing the values and subgradients of the Lovasz extension. (Note that the Lovasz extension is piecewise linear and thus is not differentiable, hence it does not have gradients everywhere.) This is a consequence of the below stated lemma (for a proof see Section 5)

Lemma 1 (Efficient computability of Lovasz extension)**.** *Let $F : 2^{[n]} \to \mathbb{R}$ be any function and $f : [0,1]^n \to \mathbb{R}$ be its Lovasz extension. There is an algorithm that given $x \in [0,1]^n$ computes $f(x)$ and a subgradient $g(x) \in \partial f(x)$ in time*

$$O(nT_F + n^2),$$

where T_F is the running time of the evaluation oracle for F.

We are now ready to establish a proof of Theorem 4.

Proof of Theorem 4. As concluded in the above discussion, minimizing a submodular function $F : 2^{[n]} \to \mathbb{R}$ is equivalent to minimizing its Lovasz extension $f : [0,1]^n \to \mathbb{R}$. Moreover, Theorem 6 asserts that f is a convex function. Therefore, one can compute an ε-approximate solution $\min_{x\in[0,1]^n} f(x)$ given just an oracle that provides values and subgradients of f (follows from Theorem 7). If the range of F is contained in $[l,u]$ then so is the range of f, hence the running time bound in Theorem 4 follows.

Finally, we show how to round an ε-approximate solution $\hat{x} \in [0,1]^n$ to a set $S \subseteq 2^{[n]}$. From the definition of the Lovasz extension it follows that

$$f(\hat{x}) = \sum_{i=0}^{n} \lambda_i F(S_i),$$

for some sets $S_0 \subseteq S_1 \subseteq \cdots \subseteq S_n$ and $\lambda \in \Delta_{n+1}$. Thus, for at least one i it holds that $f(S_i) \leq f(\hat{x})$ and we can output this S_i, as it satisfies $F(S_i) \leq f(\hat{x}) \leq F^\star + \varepsilon$. □

3 Maximum Entropy Distributions

3.1 Finding a distribution under constraints

Let Ω be a universe of cardinality N and consider the problem of picking a probability distribution $p \in \Delta_\Omega$ (recall that Δ_Ω denotes the probability simplex: the set of all probability distributions over Ω) over elements of Ω so as to preserve constraints of the form

$$\begin{cases} \mathbb{E}\left[h_1(X)\right] = \theta_1, \\ \mathbb{E}\left[h_2(X)\right] = \theta_2, \\ \dots \\ \mathbb{E}\left[h_n(X)\right] = \theta_n, \end{cases} \tag{3}$$

where above X is a random variable distributed according to p and the constraints are defined with respect to functions $h_1, h_2, \dots, h_n : \Omega \to \mathbb{R}$ and $\theta_1, \theta_2, \dots, \theta_n \in \mathbb{R}$ denote the desired values for their expectations. Constraints, as in (3) can be used to capture prior information about the distribution p for instance that $\Pr[X \in \Omega_1] = 0.5$ where $\Omega_1 \subseteq \Omega$ and many other statistical properties of p.

3.2 Example: Spanning trees in a graph

For a concrete, interesting example of the above consider the case when Ω is the set of all spanning trees T of an undirected graph $G = (V, E)$. For every edge $e \in E$ we can introduce a constraint

$$\Pr[e \in T] = \theta_e,$$

saying that the probability of the edge e being part of a random spanning tree (where T is distributed according to $p : \Omega \to [0,1]$) is θ_e. By combining all these $|E|$ constraints as above we can write them in the form

$$\sum_{T \in \Omega} p_T 1_T = \theta, \tag{4}$$

where $b \in [0,1]^E$ is the so called marginal vector (i.e., θ_e is the probability of $e \in E$ appearing in a random spanning tree), p_T is the probability of $T \in \Omega$ and $1_T \in \{0,1\}^E$ is the indicator vector of a tree T.

3.3 Entropy Maximization

One can ask: given information on p in the form of (4), what is the most natural choice of p satisfying all these constraints? Note that typically there are uncountably many choices for such a p and priori there is no canonical way of picking it. In such a case, the *Maximum Entropy Princeple* introduced by Jaynes [2, 3] states that we should pick a distribution p that maximizes the entropy

$$\sum_{T \in \Omega} p_T \log \frac{1}{p_T}$$

subject to $p \in \Delta_\Omega$ and p satisfying (4). The rationale behind chosing the entropy-maximizing distribution is that in a certain sense it is the most "general" choice and does not include any other information about p other than that given by constraints – see [4] for a thorough discussion on this topic. This leads to a computational problem

Entropy Maximization Problem

Input: A domain Ω, a collection of vectors $\{v_\omega\}_{\omega\in\Omega} \subseteq \mathbb{R}^n$ and $\theta \in \mathbb{R}^n$.

Goal: Find a distribution $p^\star$ over Ω, the optimal solution to the problem:

$$\begin{aligned} \max \quad & \sum_{\omega\in\Omega} p_\omega \log \frac{1}{p_\omega} \\ \text{s.t.} \quad & \sum_{\omega\in\Omega} p_\omega v_\omega = \theta \\ & p \in \Delta_\Omega \end{aligned} \tag{5}$$

3.4 Solving entropy maximization over small domains

Since the entropy function is concave, the above entropy maximization problem is in fact a convex program (or rather a concave program) and thus can be treated using convex optimization tools. In particular, if the domain Ω is of size N, then one can find an ε-approximate solution to (5) in time polynomial in N, $\log \varepsilon^{-1}$ and the bit complexity of the input vectors. This can be done for instance using the interior point method.[3]

3.5 Issue with large domains

However, when trying to apply this direct approach to the spanning tree example, one immediately realizes that N is exponential in the size of the graph and thus a polynomial in N algorithm is in fact exponential in the number of vertices (or edges) in G. In fact, even the input size (when given explicitly) to this entropy maximization problem is of exponential size – one would need to provide exponentially many vectors: indicator vectors of all spanning trees of G.

Finally, the issue which seems hardest to bypass is that the output size of the problem is of exponential size (a vector p of dimension $N = |\Omega|$), hence even if we used G as a compact representation of the input and had a fast algorithm to compute the entropy-maximizing distribution then still it would take exponential time to output it. In the next section we show that, surprisingly, all these problems can be dealt with.

[3] For that, one needs to construct a self-concordant barrier function for the sublevel set of the entropy function – see Exercise 3 in Problem Set 6 for an example of such.

3.6 Duality

To counter the problems related to solving the entropy maximization problem for exponentially large domains Ω we use duality to transform this problem into one that has only n variables. More precisely, consider the following dual problem to (5).

Problem Dual to Entropy Maximization

Input: A domain Ω, a collection of vectors $\{v_\omega\}_{\omega\in\Omega} \subseteq \mathbb{R}^n$ and $\theta \in \mathbb{R}^n$.

Goal: Find a vector $y^\star \in \mathbb{R}^n$, the optimal solution to the problem:

$$\begin{aligned} \min \quad & \log\left(\sum_{\omega\in\Omega} e^{\langle y, v_\omega - \theta\rangle}\right) \\ \text{s.t.} \quad & y \in \mathbb{R}^n \end{aligned} \tag{6}$$

It is an important exercise to prove that problems (5) and (6) are dual to each other and that strong duality holds, hence their optimal values coincide. The dual problem (6) looks arguably simpler to solve since it is an unconstrained optimization problem. However, as expected, the difficulty does not disappear, since its objective involves an exponential-size sum and hence cannot be evaluated directly in polynomial time.

Let us for a moment forget about the computablity problem for (6) and assume that $y^\star$ is the optimal solution. Then, $y^\star$ can be seen as a compact representation of the optimal distribution $p^\star$ as the following lemma demonstrates

Lemma 2 (Succinct representation of max-entropy distribution)**.** *Suppose that $y^\star \in \mathbb{R}^n$ is the optimal solution to the dual to entropy maximization problem* (6)*. Then the optimal solution $p^\star$ to the entropy maximization problem* (5) *can be recovered as*

$$\forall \omega \in \Omega \qquad p^\star_\omega = \frac{e^{\langle y^\star, v_\omega\rangle}}{\sum_{\omega'\in\Omega} e^{\langle y^\star, v_{\omega'}\rangle}}.$$

Thus it is enough to compute $y^\star$ in order to obtain a succinct representation of the maximum entropy distribution $p^\star$. Therefore, from now on we can focus on the task of solving the dual problem (6).

3.7 Solving the dual problem

Let us denote the objective of the dual program by

$$f(y) := \log\left(\sum_{\omega\in\Omega} e^{\langle y, v_\omega - \theta\rangle}\right).$$

We would like to find the minimum of $f(y)$. For this we apply the general algorithm for convex optimization based on the ellipsoid method that we derive in Section 4. When translated to this setting it has the following consequence.

Theorem 8 (Informal; see Theorem 1). *Suppose the following conditions are satisfied:*

1. *we are given access to a polynomial time oracle computing the value and the gradient of f,*
2. *$y^\star$ is guaranteed to lie in a ball $B(0, R)$ for some $R > 0$,*
3. *the values of f over $B(0, R)$ stay in the interval $[-M, M]$ for some $M > 0$.*

Then, there is an algorithm that given $\varepsilon > 0$ outputs an ε-approximate solution to (6) *in time* $\mathrm{poly}(n, \log R, \log M, \log \varepsilon^{-1})$ *(when an oracle query is treated as a unit operation).*

We use the above to obtain a polynomial time algorithm in the case of spanning trees.

3.8 Polynomial time algorithm for the spanning tree case

Theorem 9 (Solving the Dual to Max Entropy Problem for Spanning Trees). *Given a graph $G = (V, E)$ and a vector $\theta \in \mathbb{R}^E$ such that $B(\theta, \eta) \subseteq P_{ST}(G)$ there is an algorithm to find an ε-approximate solution to*

$$\min_{y \in \mathbb{R}^E} \log \left(\sum_{T \in \mathcal{T}_G} e^{\langle y, 1_T - \theta \rangle} \right) \tag{7}$$

in time $\mathrm{poly}(|V|, \eta^{-1}, \log \varepsilon^{-1})$*, where $\mathcal{T}_G$ is the set of all spanning trees in G.*

Note importantly that the above gives a polynomial time algorithm only if θ is "sufficiently deep" in the polytope P_{ST} (or in other words, when it is far away from the boundary). If the point θ comes close to the boundary (i.e. $\eta \approx 0$) then the bound deteriorates to infinity. This assumption turns out to not be necessary (see [10]), however the interiority assumption makes the proof simpler and easier to understand – for this, we follow the treatment by [9].

Proving a bound on R

We start by verifying that the second condition in Theorem 8 is satisfied, by proving an appropriate upped bound on the magnitude of the optimal solution.

Lemma 3 (Bound on the norm of the optimal solution). *Suppose that $G = (V, E)$ is an undirected graph and $\theta \in \mathbb{R}^E$ satisfies $B(\theta, \eta) \subseteq P_{ST}(G)$, for some $\eta > 0$. Then $y^\star$ – the optimal solution to the dual to the max entropy problem* (7) *satisfies*

$$\|y^\star\|_2 \leq \frac{|E|}{\eta}.$$

Note that the bound detoriates as $\eta \to 0$ and thus becomes rather useless when θ is close to the bundary. By a slightly different reasoning one can obtain a variant of Lemma 3 avoiding the dependency on η [10]. A proof of Lemma 3 appears in Section 6.

Efficient evaluation of f

We now proceed to verifying the first condition in Theorem 8. To this end, we need to show that f can be evaluated efficiently (proved as an exercise in Problem Set 10).

Lemma 4 (Polynomial time evaluation oracle for f and its gradient)**.** *Suppose that $G = (V, E)$ is an undirected graph and f is defined as*

$$f(y) := \log\left(\sum_{T\in\mathcal{T}_G} e^{\langle y, 1_T - \theta\rangle}\right),$$

then there is an algorithm that given y outputs the value $f(y)$ and the gradient $\nabla f(y)$ in time $\operatorname{poly}(\|y\|, |E|)$*.*

Note that the running time of the oracle is polynomial time in $\|y\|$ and not in $\log\|y\|$ as one would perhaps desire (for a polynomial time oracle). This is a consequence of the fact that even a single term in the sum $e^{\langle y, 1_T - \theta\rangle}$ might be as large as $\approx e^{\|y\|}$ and hence it requires up to $\|y\|$ bits to represent, and it is hard to avoid calculations on such numbers.

Summary

We are now ready to conclude a proof of Theorem 9 from Theorem 8 using the above stated lemmas.

Proof of Theorem 9. We need to verify all the conditions: 1., 2. and 3. in Theorem 8.

Condition 2. is satisfied for $R = O(\frac{m}{\eta})$ by Lemma 3. Given a bound $\|y\| \le R$ we can now provide a lower and upper bound on $f(y)$ (to verify condition 3.). We have

$$\begin{aligned}
f(y) &= \log\left(\sum_{T\in\mathcal{T}_G} e^{\langle y, 1_T - \theta\rangle}\right) \\
&\le \log\left(\sum_{T\in\mathcal{T}_G} e^{\|y\|_2\cdot\|1_T - \theta\|}\right) \\
&\le \log|\mathcal{T}_G| + \|y\|_2 \cdot O(\sqrt{m}) \\
&\le m + \frac{m^{3/2}}{\eta} \\
&= \operatorname{poly}\left(m, \frac{1}{\eta}\right)
\end{aligned}$$

Similarly, we can obtain a lower bound and thus whenever $\|y\| \le R$, $-M \le f(y) \le M$ with $M = \operatorname{poly}\left(m, \frac{1}{\eta}\right)$.

Condition 3. is also satisfied by Lemma 4, however it adds a $\text{poly}(R)$ factor to the running time (since the oracle is polynomial in $\|y\|$) hence the final bound on the running time (as given by Lemma 8) becomes

$$\text{poly}(|E|, \log R, \log M, \log \varepsilon^{-1}) \cdot \text{poly}(R) = \text{poly}(|E|, \eta^{-1}, \log \varepsilon^{-1}).$$

□

4 Convex Optimization using Ellipsoid

This section is devoted to deriving an algorithm for solving convex optimization programs $\min_{x \in K} f(x)$ when only oracle access to K and f is given. To this end we start by generalizing the Ellipsoid algorithm from the last lecture to solve the feasibility problem for any convex set K (not only for polytopes) and further we show that this, combined with the standard binary search procedure yields an efficient algorithm under mild conditions on f and K.

4.1 From Polytopes to Convex sets

Note that the Ellipsoid algorithm for the feasibility problem for polytopes that we developed in the previous lecture did not use the fact that P is a polytope, only that P is convex and a separation oracle for P is available. Indeed Theorem 8 in Lecture 9 can be rewritten in this more general form to yield.

Theorem 10 (Solving the feasibility problem for convex sets)**.** *There is an algorithm such that given a separation oracle for a convex set $K \subseteq \mathbb{R}^n$, a radius $R > 0$ such that $K \subseteq B(0, R)$ and a parameter $r > 0$ gives one of the following outputs*

1. *YES (along with a point $\hat{x} \in K$) – proving that K is non-empty,*
2. *NO – in which case K is guaranteed to not contain a Euclidean ball of radius r.*

The running time of the algorithm is

$$O\left((n^2 + T_K) \cdot n^2 \cdot \log \frac{R}{r}\right),$$

where T_K is the running time of the separation oracle for K.

Note that the above is written in a slightly different form then Theorem 8 in the last lecture – indeed, there we assumed that K contains a ball of large radius and wanted to compute a point inside of K. Here the algorithm will proceed until the volume of the current ellipsoid becomes smaller than the volume of a unit balll of radius r and then outputs "NO" if no point in K is found. This variant can be seen as an approximate non-emptiness check:

1. If $K \neq \emptyset$ and $\text{vol}(K)$ is "large enough" then the algorithm outputs "YES",

2. If $K = \emptyset$ then the algorithm outputs "NO",
3. If $K \neq \emptyset$ but $\text{vol}(K)$ is small then the algorithm can answer either "YES" or "NO".

The uncertainty introduced when K has tiny volume is typically not a problem, as we can often mathematically force K to be either empty of have large volume when running the ellipsoid algorithm on it.

4.2 Algorithm for Convex Optimization

We now build upon the ellipsoid algorithm for checking non-emptiness of a convex set K to derive a very general method for convex optimization. Let $f : \mathbb{R}^n \to \mathbb{R}$ be a convex function, to find the minimum of f over K the idea is to use the binary search method to find the optimal value. The subproblem solved at every step is then simply checking non-emptiness of

$$K^g := \{x \in K : f(x) \leq g\},$$

for some value g. This can be done as long as a separation oracle for K^g is available. One also needs to be careful about the uncertainty introduced in the non-emptiness check, as it might potentially cause errors in the binary search procedure. For this, it is assumed that K contains a ball of radius r and at every step of the binary search the ellipsoid algorithm is called with an appropriate choice of the inner ball parameter. Details are provided in the pseudocode below, where it is assume that all values of f over K lie in the interval $[l_0, u_0]$.

```
Ellipsoid Method for Convex Optimization

  1. Set $l := l_0$ and $u := u_0$
  2. while $u - l > \varepsilon/2$
      (a) set $g := \frac{l+u}{2}$
      (b) Apply the Ellipsoid algorithm to the set
              $K^g := \{x \in K : f(x) \leq g\}$
          with parameter $r' := \frac{r \cdot \varepsilon}{2(u_0 - l_0)}$
            • If Ellipsoid answers YES: set $u := g$ and let $\hat{x} \in K^g$ be the point
              returned by the Ellipsoid algorithm
            • Else: set $l := g$
  3. Output $\hat{x}$
```

At this point it is not clear that the above scheme indeed gives a correct algorithm. Indeed – we need to verify that the choice of r' guarantees that the binary search algorithm gives correct answers most of the times. Further, it is not even clear how one would implement such an algorithm, as so far we have not yet talked about how f is given. It turns out that access to values and gradients of f is enough in this setting – we call this 1st order oracle access to f.

Definition 4 (1st order oracle)**.** *A first order oracle for a function $f : \mathbb{R}^n \to \mathbb{R}$ is a primitive that given $x \in \mathbb{Q}^n$ outputs the value $f(x) \in \mathbb{Q}$ and a vector $g(x) \in \mathbb{Q}^n$ such that*

$$\forall z \in \mathbb{R}^n \;\; f(z) \geq f(x) + \langle z - x, g(x) \rangle .$$

In particular, if f is differentiable then $g(x) = \nabla f(x)$, but more generally $g(x)$ can be just any subgradient of f at x. We note that typically one cannot hope to get an exact 1st order oracle, but rather an approximate one – for simplicity we work with exact oracles. An extension to approximate oracles is possible yet requires introducing so called weak separation oracles which in turn creates new technical difficulties – we refer to [1] for a thorough discussion on this topic.

We are now ready to go back to Theorem 1 that formally describes the efficiency of the above provided scheme. We note that the algorithm requires access to a lower and upper bound on f over K as well as an enclosing ball $B(0, R)$ for K and a radius $r > 0$ such that $B(\widetilde{x}, r) \subseteq K$ for some $\widetilde{x} \in K$ (note however that $\widetilde{x}$ is not provided on input). Moreover, in order to obtain a polynomial time algorithm using the above theorem one also has to construct an efficient separation oracle for K and an efficient 1st order oracle for f. Consequently, the above **does not imply that all convex programs can be solved in polynomial time** even though the dependency on the error $\varepsilon > 0$ is $\log \varepsilon^{-1}$ – as desired for polynomial running time. Depending on the application, coming up with appropriate bounds r, R or l_0, u_0 might be quite non-trivial and even lead to quantities with exponential bit complexity. Similarly, constructing separation oracles or 1st order oracles for certain sets might lead to hard computational problems. For this reason, before claiming that a given convex program is polynomial time solvable, a thorough discussion of these issues must be performed.

4.3 Analysis

The goal of this section is to provide a proof of Theorem 1. The are two main components to it: one is to show that given a separation oracle for K and a 1st order oracle for f we can obtain a separation oracle for

$$K^g := \{x \in \mathbb{R}^n : f(x) \leq g\}$$

and the second component is to show that by using the Ellipsoid algorithm to test the non-emptiness of K^g with the parameter r' specified in the algorithm, we are guaranteed to obtain a correct answer up to a good precision. We discuss these two components in separate steps and conclude the result afterwards.

Constructing a separation oracle for K^g

In the lemma below we show that a separation oracle for the sublevel set $S^g := \{x \in \mathbb{R}^n : f(x) \leq g$ can be constructed using a 1st order oracle for f.

Lemma 5 (Separation over sublevel sets using a 1st order oracle)**.** *Suppose there is a* 1*st order oracle to a convex function $f : \mathbb{R}^n \to \mathbb{R}$ with running time T_f. Then, for any $g \in \mathbb{Q}$ there is a separation oracle for $S^g := \{x \in \mathbb{R}^n : f(x) \leq g\}$ with running time T_f.*

Proof. The construction of the oracle is rather simple: given $x \in \mathbb{R}^n$ we first compute $f(x)$. If $f(x) \leq g$ then the oracle outputs YES. Otherwise let u be the subgradient of f at x (obtained using the 1st order oracle of f), then using the subgradient property of convex functions:

$$\forall z \in \mathbb{R}^n \qquad f(z) \geq f(x) + \langle z - x, u \rangle .$$

Since $f(x) > g$, for every $z \in S^g$ we have

$$g + \langle z - x, u \rangle < f(x) + \langle z - x, u \rangle \leq f(z) \leq g,$$

or in other words

$$\langle z, u \rangle < \langle x, u \rangle ,$$

hence u provides us with a separating hyperplane. The running time of such an oracle is clearly $O(T_f)$. □

By noting that $K^g = K \cap S^g$ we simply obtain that a separation oracle for K^g can be constracted using the oracles for K and f.

Bounds on the volume of sublevel sets

In this step we would like to give a lower bound on the smallest ball contained in K^g depending on how close g is to the minimum $f^\star$ of f on K. This is required to claim that the Ellipsoid algorithm will correctly determine whether $K^g \neq \emptyset$ in various steps of the binary search. More precisely we would like to see that if $g \approx f^\star + \varepsilon$ then still K^g contains a large enough ball so that Ellipsoid returns "YES".

Lemma 6 (Lower bound on volume of sublevel sets)**.** *Let $K \subseteq \mathbb{R}^n$ be a convex set containing a Euclidean ball of radius $r > 0$ and $f : K \to [f^\star, f_{\max}]$ be a convex function on K with $f^\star = \min_{x \in K} f(x)$. For any $g := f^\star + \delta$ (for $\delta > 0$), define $K^g := \{x \in K : f(x) \leq g\}$. Then K^g contains a Euclidean ball of radius*

$$r \cdot \frac{\delta}{f_{\max} - f^\star}.$$

Proof. Let $x^\star \in K$ be any minimizer of f over K. In this argument, we can assume without loss of generality that $x^\star = 0$. Define $\eta := \frac{\delta}{f^\star - f_{\max}}$, we claim that

$$\eta K \subseteq K^g, \tag{8}$$

where $\eta K := \{\eta x : x \in K\}$. Since the ball of radius r contained in K becomes a ball of radius ηr in ηK, the claim implies the lemma. Thus we now focus on proving 8. Let $x \in \eta K$, we would like to show that $f(x) \leq g$. We have $\frac{x}{\eta} \in K$ and hence from convexity of f we have

$$f(x) \leq (1-\eta) f(0) + \eta f\left(\frac{x}{\eta}\right) \leq (1-\eta) f^\star + \eta f_{\max} = f^\star + \eta(f_{\max} - f^\star) = f^\star + \delta = g,$$

hence $x \in K^g$ and the claim follows. □

Summary

Given lemmas 5 and 6 we are well equipped to prove Theorem 1.

Proof of Theorem 1. Let $f^\star = f(x^\star)$, we observe first that whenever $g \geq f^\star + \varepsilon/2$ then K^g contains a Euclidean ball of radius

$$r' = \frac{r \cdot \varepsilon}{2(u_0 - l_0)}$$

and hence for such a g, the Ellipsoid algorithm terminates with verdict YES and outputs a point $\hat{x} \in K^g$. This is a direct consequence of Lemma 6.

Let l, u be the values of these variables at the moment of termination of the algorithm. From the above reasoning it follows that

$$u \leq f^\star + \varepsilon,$$

indeed $u \leq l + \varepsilon/2$ and the Ellipsoid can answer "NO" only for $g \leq f^\star + \varepsilon/2$, hence $l \leq f^\star + \varepsilon/2$. Therefore the $\hat{x}$ output by the algorithm belongs to K^u and hence

$$f(\hat{x}) \leq f^\star + \varepsilon.$$

This proves correctness of the algorithm. It remains to analyze the running time. The Ellipsoid algorithm is run $\log \frac{u_0 - l_0}{\varepsilon/2}$ times, and every such execution takes

$$O\left((n^2 + T_{K^g}) \cdot n^2 \cdot \log \frac{R}{r'}\right) = O\left((n^2 + T_K + T_f) \cdot n^2 \cdot \log\left(\frac{R}{r} \cdot \frac{u_0 - l_0}{\varepsilon}\right)\right)$$

time, where we used Lemma 5 to conclude that $T_{K^g} \leq T_K + T_f$. □

Remark 7 (Avoiding binary search)**.** *By taking a closer look at the algorithm and the proof of Theorem 1 one can observe that one does not need to restart the Ellipsoid algorithm at every iteration of the binary search. Indeed, one can just reuse the ellipsoid obtained in the previous call. This leads to an algorithm with a slightly reduced running time*

$$\left((n^2 + T_K + T_f) \cdot n^2 \cdot \log\left(\frac{R}{r} \cdot \frac{u_0 - l_0}{\varepsilon}\right)\right),$$

(Note that there is no square in the logarithmic factor.)

5 Proofs of Lemmas Regarding Submodular Function Minimization

5.1 Computability of Lovasz extension

Proof of Lemma 1. Without loss of generality assume that the point $x \in [0,1]^n$ on which we have to evaluate f satisfies

$$x_1 \leq x_2 \leq \cdots \leq x_n.$$

Then, it is easy to see that

$$f(x) = x_1 \cdot F(\emptyset) + (x_2 - x_1)F([1]) + (x_3 - x_2)F([2]) + \ldots + (x_n - x_{n-1})F([n-1]) + (1 - x_n)F([n]).$$

For this reason, $f(x)$ is efficiently computable (requires just $O(n)$ evaluations of F). To compute the subgradient of f at x one can assume without loss of generality that $x_1 < x_2 \ldots < x_n$, since otherwise we can perform a small perturbation to reduce to this case. Now, on the set $S = \{x \in [0,1]^n : x_1 < x_2 < \cdots < x_n\}$ the function $f(x)$ (as demonstrated above) is just a linear function, hence its gradient can be evaluated efficiently. □

6 Proofs of Lemmas Regarding Maximum Entropy Distributions

6.1 Bound on the norm of the optimal dual solution

Proof of Lemma 3. Denote $m = |E|$ and let $\theta \in \mathbb{R}^m$ be such that $B(\theta, \eta) \subseteq P_{ST}(G)$. Suppose that $y^\star$ is the optimal solution to the dual program. If f denotes the dual objective we know from strong duality that any upper bound on the optimal primal solution is an upper bound on $f(y^\star)$. For this reason, since the maximum achievable entropy of a distribution over $\mathcal{T}_\mathcal{G}$ is $\log|\mathcal{T}_\mathcal{G}|$, we have

$$f(y^\star) \leq \log|\mathcal{T}_\mathcal{G}| \leq \log 2^m = m. \tag{9}$$

Recall now that

$$f(y^\star) = \log\left(\sum_{T \in \mathcal{T}_G} e^{\langle y^\star, 1_T - \theta\rangle}\right),$$

thus from (9) we obtain, for every $T \in \mathcal{T}_G$

$$\langle y^\star, 1_T - \theta\rangle \leq m.$$

By convexity it follows that for every point $x \in P_{ST}(G)$ we have

$$\langle y^\star, x - \theta\rangle \leq m.$$

Hence, from the assumption that $B(\theta, \eta) \subseteq P_{ST}(G)$, we conclude that

$$\forall v \in B(0, \eta) \quad \langle y^\star, v\rangle \leq m.$$

In particular, by taking $v := \eta \frac{y^\star}{\|y^\star\|_2}$ we obtain that

$$\|y^\star\|_2 \leq \frac{m}{\eta}.$$

□

7 Other variants of the Cutting Plane method

In this section we describe certain variants of the cutting plane method that can be used in place of the ellipsoid method to derive the result of this and the previous lecture. Recall that the cutting plane method is typically used to solve the following problem: given a convex set $K \subseteq \mathbb{R}^n$ (along with a ball $B(0, R)$ containing it) by a separation oracle the goal is to either find a point $x \in K$ or correctly determine that K does not contain a ball of radius $r > 0$, where r is provided as input.

To solve this problem, the general cutting plane method maintains a set $E_k \supseteq K$ and shrinks it in every step, so that

$$E_0 \supseteq E_1 \supseteq E_2 \supseteq \cdots \supseteq K.$$

As a measure of progress one typically uses the volume of E_k, thus ideally one wants to decrease the volume of E_k at every step

$$\mathrm{vol}(E_{k+1}) < \alpha \mathrm{vol}(E_k),$$

where $0 < \alpha < 1$ is the volume drop parameter. The ellipsoid algorithm achieves $\alpha \approx 1 - \frac{1}{2n}$ and hence it requires roughly $O(n \log \alpha^{-1} \log \frac{R}{r}) = O(n^2 \log \frac{R}{r})$ iterations to terminate. It is known that this volume drop rate is tight when using ellipsoids, however one can ask whether this can be improved to say a constant $\alpha < 1$ when using different kind of sets E_k to approximate the convex body K.

7.1 Maintaining a polytope instead of an ellipsoid

Recall that in the cutting plane method the set E_{k+1} is picked so as to satisfy

$$E_k \cap H \subseteq E_{k+1},$$

where $H := \{x : \langle x, h_k \rangle \leq \langle x_k, h_k \rangle\}$ is the halfspace passing through the point $x_k \in E_k$ determined by the separating hyperplane output by the separation oracle for K (see Section 2 in Lecture 8). In the ellipsoid method E_{k+1} is chosen as the minimum volume ellipsoid containinf $E_k \cap H$. One could perhaps argue that this choice of E_{k+1} is not at all efficient, as we already know that $K \subseteq E_k \cap H$, hence $E_{k+1} := K \cap E_k$ would be much more reasonable. By following this strategy, starting with E_0 being a box $[-R, R]^n$ we produce a sequence of polytopes (instead of ellipsoids) containing K. The crucial question that is arises is how to pick a point $x_k \in E_k$, so that no matter what halfspace H through

x_k is output by the separation oracle, we can still guarantee that $E_{k+1} := E_k \cap H$ has a significcantly smaller volume than E_k. Note that when we pick a point x_k close to the boundary of E_k, H might "cut out" only a small piece of E_k and hence $\mathrm{vol}(E_k) \approx \mathrm{vol}(E_{k+1})$ and hence no progress is made in such a step. For this reason it seems reasonable to pick a point x_k that is in the "center" of the polytope.

It follows by a theorem of Grunbaum that if x_k is chosen to be the centroid[4] of the polytope then no matter what halfspace H through x_k is chosen, it holds that

$$\mathrm{vol}(E_k \cap H) \leq \left(1 - \frac{1}{e}\right) \mathrm{vol}(E_k),$$

where $e \approx 2.71$ is the Euler number. In other words, we obtain a method where $\alpha \approx 0.67$ is a constant. While this sounds promising, this method has a significant drawback: the centroid is not at all easy to compute and even for polytopes there are no known fast algorithms to find the centroid. Given such a difficulty there has been many attempts to define alternatives notions of a center of the polytope so as both: make it easy to compute and get a constant $\alpha < 1$. We now give brief overviews of two such approaches.

7.2 The Volumetric Center method of Vaidya

The idea of Vaidya [11] is to use the so-called volumetric center of E_k as x_k. To define it, suppose that E_k is defined by the system of inequalities $\langle a_i, x \rangle \leq b_i$ for $i = 1, 2, \ldots, m$. Let $F : E_k \to \mathbb{R}$ be the logarithmic barrier

$$F(x) = -\sum_{i=1}^{m} \log\left(b_i - a_i^\top x\right),$$

and similarly let $V : E_k \to \mathbb{R}$ be the volumetric barrier

$$V(x) = \log\det \nabla^2 F(x),$$

then, the volumetric center x_k of E_k is defined as

$$x_k := \operatorname*{argmin}_{x \in E_k} V(x).$$

The intuition for using the volumetric center x_k as the queried point at step k is that x_k is the point around which the Dikin ellipsoid has the largest volume. Since the Dikin Ellipsoid is a decent approximation of the polytope (see Lecture 7), then one should expect that a hyperplane through the center of this ellipsoid should divide the polytope into two roughly equal pieces and hence we should expect α to be small. What Vaidya proves is that indeed, on average (over a large number of iterations)

$$\frac{\mathrm{vol}(E_{k+1})}{\mathrm{vol}(E_k)} \leq 1 - 10^{-6},$$

[4] The centroid $c \in \mathbb{R}^n$ of a measurable, bounded set $K \subseteq \mathbb{R}^n$ is defined to be $c := \int_K x dx$, i.e., the mean of the uniform distribution over K.

and hence α is indeed a constant (slightly) smaller than 1. Further the volumetric center x_k of E_k does not have to be recomputed from scratch every single time – in fact since E_{k+1} has only one constraint more than E_k, Vaidya uses the Newton's method to compute x_{k+1} with x_k as the starting point. This recentering step can be implemented in $O(n^\omega)$ time. In fact, to achieve this running time, the number of facets in E_k needs to be kept small throughout the iterations, hence occasionally the algorithm has to drop certain (low impact) constraints.

To summarize: Vaidya's method attains the optimal, constant rate of volume drop, hence requires roughly $O(n \log \frac{R}{r} \log \frac{1}{\varepsilon})$ iterations to terminate. However, the update time per iteration is $n^\omega \approx n^{2.38}$ is slower as compared to the update time for the ellipsoid method, which is $O(n^2)$ (as a rank-1 update to an $n \times n$ matrix).

7.3 An improvement by Lee, Sidford and Wong

Given the result of Vaidya one might wonder whether it would be possible to achieve the optimal, constant volume drop rate and at the same time keep the running time per iteration as low as for the ellipsoid method $\approx n^2$. In their paper [6] Lee, Sidford and Wong showed that such an improvement is indeed possible. To obtain this improvement, the following barrier function over the polytope E_k is considered

$$G(x) = -\sum_{i=1}^{m} w_i \log s_i(x) + \frac{1}{2} \log\det(A^\top S_x^{-2} A + \lambda I) + \frac{\lambda}{2} \|x\|_2^2, \tag{10}$$

where $w_1, w_2, \ldots, w_m$ are certain positive weights and $\lambda > 0$ is a parameter. Similarly as in Vaidya's approach, the query point is determined as the minimum of the barrier $G(x)$, the "hybrid center"

$$x_k := \operatorname*{argmin}_{x \in E_k} G(x).$$

The use of regularized weighted logarithmic barriers (10) is inspired by similar barrier functions used to obtain an improved for path following interior point methods in [5]. It is proved in [6] that maintaining x_k and the set of weights w_k such that x_k is the minimum of $G(x)$ over $x \in E_k$ is possible in amortized time $O(n^2)$ per iteration. Moreover, such a choice of x_k guarantees a constant volume drop, on average with $\alpha \approx 1 - 10^{-27}$.

Using this variant of the cutting plane method, Lee, Sidford and Wong derive a host of implications to combinatorial optimization. In particular, they give asymptotically fastest known algorithms for submodular minimization, matroid intersection and semidefinite programming.

A Application of SFM: Image Denoising

Consider an image denoising problem as demonstrated in Figure 1. We are given a $H \times W$ noisy picture whose every pixel is either white or black and would like to remove the noise from it: i.e. intuitively make the edges smooth and turn it into a picture of a "solid object".

To propose a solution to this problem it is convenient to think of the picture as a graph $G = (V, E)$ where vertices correspond to pixels, i.e., $|V| = WH$, and edges corresponding to neighboring pixels. Let $|V| = n$ and $|E| = m$. The original picture is then nothing but a vector $p \in \{0,1\}^n$ and similarly the solution we are looking for is $q \in \{0,1\}^n$. To find q – a denoised version of p we would like to capture two constraints on q:

1. **(Smoothness)** For two neighboring pixels (connected by an edge in G) the value of q should be the same: $q_i = q_j$ when $ij \in E$
2. **(Approximation)** The denoised version should still largely resemble p, i.e., $q \approx p$

Clearly, the two above goals are contradictory and cannot be satisfied at the same time – the first condition is only satisfied when q is all zeros or all ones. For this reason, the way we state the problem is to say that for "most" edges, the value of q does not change. More precisely, we can set up the following optimization problem.

$$\min_{q\in\{0,1\}^n} \sum_{ij\in E} F(q) := \psi(q_i, q_j) + \sum_{i\in V} \phi(q_i, p_i), \tag{11}$$

where ψ and ϕ are penalty functions for being "non-smooth" and for differing from p respectively. The simples choice of ψ and ϕ is perhaps

$$\psi(b_1, b_2) = \phi(b_1, b_2) = \begin{cases} 0 & \text{if } b_1 = b_2, \\ 1 & \text{if } b_1 \neq b_2 \end{cases} \tag{12}$$

This means that there is a cost of 1 for transitioning from 0 to 1 (or the other way around) on an edge, and for modifying the value of any of the bits. Clearly, for better results one would probably need to put appropriate weights on ψ and ϕ, i.e. decide on a trade-off on how close to p do we want to be vs how smooth q do we want to make. Moreover we could as well have a different weight or a completely different penalty function ψ_{ij} for every edge or ϕ_i for every vertex. Such a need might arise when only part of the picture has noise and we would like to keep the remaining part intact.

We note however that the choice of the penalty function needs to be done with some care. If one uses ψ to be the "negation" of the one used above, i.e., $\psi(b_1, b_2)$ is 1 if $b_1 \neq b_2$ and 0 otherwise then suddenly (11) becomes NP-complete – it is able to capture the maximum independent set problem. Note however that the ψ function mentioned above does not seem to be well suited for a "non-smoothness" penalty function as it achieves the completely opposite goal. In fact the penalty function defined in (12) has the nice property

$$\psi(0,0) + \psi(1,1) \leq \psi(0,1) + \psi(1,0),$$

which in turn causes $F : 2^{[n]} \to \mathbb{R}$ (as defined in (11)) to satisfy

$$\forall S, T \subseteq [n] \quad F(S \cap T) + F(S \cup T) \leq F(S) + F(T), \tag{13}$$

where the binary input vector to F is treated as (an indicator vector of) a set; in other words F is submodular.

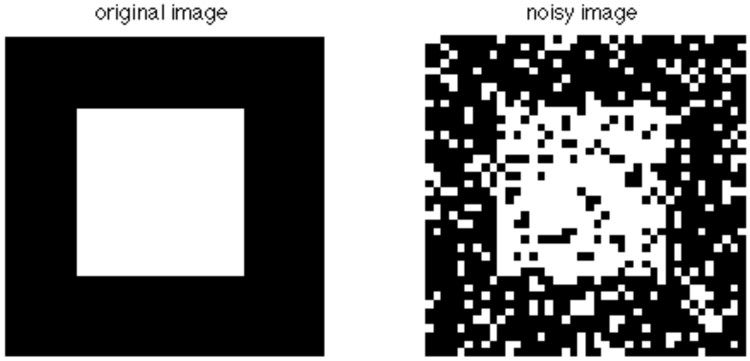

Figure 1: Example of image denoising: the original image and a version with noise.

References

[1] Martin Grötschel, Lászlo Lovász, and Alexander Schrijver. *Geometric Algorithms and Combinatorial Optimization*, volume 2 of *Algorithms and Combinatorics*. Springer, 1988.

[2] Edwin T. Jaynes. Information Theory and Statistical Mechanics. *Physical Review*, 106:620–630, May 1957.

[3] Edwin T. Jaynes. Information Theory and Statistical Mechanics. II. *Physical Review*, 108:171–190, October 1957.

[4] Edwin T. Jaynes. On the rationale of maximum-entropy methods. *Proceedings of the IEEE*, 70(9):939–952, 1982.

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

[6] Yin Tat Lee, Aaron Sidford, and Sam Chiu-wai Wong. A faster cutting plane method and its implications for combinatorial and convex optimization. In *IEEE 56th Annual Symposium on Foundations of Computer Science, FOCS 2015, Berkeley, CA, USA, 17-20 October, 2015*, pages 1049–1065, 2015.

[7] László Lovász. Submodular functions and convexity. In *Mathematical Programming The State of the Art*, pages 235–257. Springer, 1983.

[8] Alexander Schrijver. *Combinatorial optimization: polyhedra and efficiency*, volume 24. Springer Science & Business Media, 2002.

[9] Mohit Singh and Nisheeth K. Vishnoi. Entropy, optimization and counting. In *Proceedings of the 46th Annual ACM Symposium on Theory of Computing*, pages 50–59. ACM, 2014.

[10] Damian Straszak and Nisheeth K. Vishnoi. Computing maximum entropy distributions everywhere. *CoRR*, abs/1711.02036, 2017.

[11] P. M. Vaidya. A new algorithm for minimizing convex functions over convex sets. In *30th Annual Symposium on Foundations of Computer Science*, pages 338–343, Oct 1989.