CPSC 536C - Algorithms for Convex Optimization

Lecture : Minimum Cut

23rd February 2026

Lecturer: Hasti Karimi *Scribe: Mitali Nanda*

In this lecture, we explore the intersection of convex optimization and combinatorial graph algorithms. We begin by examining how to solve optimization problems subject to strict equality constraints. Specifically, we look at how standard Gradient Descent (GD) can be modified using projection matrices to ensure that our iterative updates remain within a feasible solution space.

After establishing these optimization techniques, we transition to the classic combinatorial problem of finding the Minimum Cut in a graph. By formulating this problem as an Integer Program (IP) and subsequently relaxing it into a Linear Program (LP), we bridge the gap between discrete and continuous domains. Finally, we introduce fundamental graph matrices, such as the edge-vertex incidence matrix, to demonstrate how projected gradient methods can be directly applied to solve these relaxed graph problems [5]. The structure and derivations in this lecture closely follow the presentation found in Lap Chi Lau's course notes on this topic [4].

1 Linear Algebra Definitions

Before we can optimize over constraints, we must establish a few fundamental properties of matrices and vector spaces. These concepts define the geometric space in which our feasible solutions and gradient updates exist.

1.1 Null Space and Image Space

For any given matrix A, we define two critical subspaces:

- **Kernel or Null Space** (null(A)): The set of all vectors x that are mapped to zero by A. Formally,
$$\text{null}(A) = \{x \mid Ax = 0\}$$
In the context of optimization, moving along directions in the null space will not violate our equality constraints.

- **Span or Image Space** (Im(A)): The set of all possible outputs of the linear transformation A. Formally,
$$\text{Im}(A) = \{y \mid y = Ax \text{ for some } x\}$$
It is also known as the Column Space of A.

Equality Constrained Optimization via Null Space

We want to minimize a function $f(x)$ subject to a linear equality constraint:

$$\min_x f(x) \quad \text{s.t.} \quad Ax = b$$

Any solution x can be written as:

$$x = x_0 + Mz$$

Where:

- x_0 is one particular solution to $Ax = b$.
- M is a matrix whose columns are the basis vectors of $\ker(A)$ (the null space of A).
- z is an unconstrained variable vector.

Since M is in the null space of A, $AM = 0$:

$$Ax = A(x_0 + Mz) = Ax_0 + AMz = Ax_0 = b$$

Substituting our parameterized x into the objective function gives an unconstrained problem in terms of z:

$$\min_z f(x_0 + Mz) = g(z)$$

Applying the multivariate chain rule, the gradient of the new objective function $g(z)$ is:

$$\nabla g(z) = M^T \nabla f(x_0 + Mz)$$

Finding the basis of the null space matrix M takes:

$$\mathcal{O}(mn \cdot \min\{m, n\})$$

where M is a m x n matrix.

1.2 Projection Matrices

A projection matrix P is a square matrix that maps a vector onto a specific subspace. To be a valid orthogonal projection matrix, it must satisfy two key properties:

1. **Symmetric:** $P = P^T$.
2. **Idempotent:** $P^2 = P$. Intuitively, this means that once a vector has been projected onto a subspace, applying the projection again leaves it unchanged.

Thus, $\forall\ v : Pv \in S$ and $\forall\ b \in S : Pb = b$.

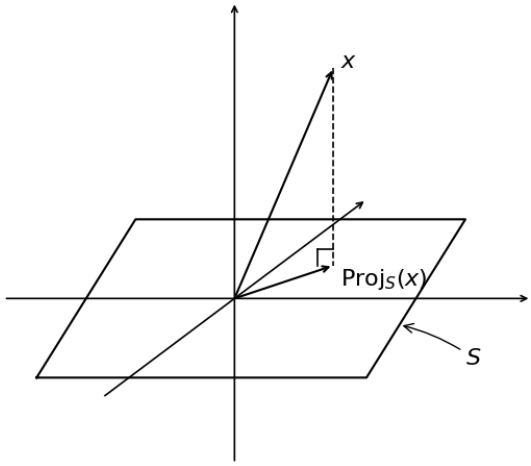

Figure 1: Projection of vector x onto subspace S.

1.3 The Complementary Projection

If P projects a vector onto a specific subspace, the matrix $(I - P)$ projects a vector onto the orthogonal complement of that subspace.

We can verify that $(I - P)$ is also a valid projection matrix by proving it is idempotent:

$$\begin{aligned}(I-P)^2 &= (I-P)(I-P) \\ &= I - 2P + P^2\end{aligned}$$

Since $P^2 = P$, this simplifies perfectly:

$$I - 2P + P = I - P$$

Furthermore, we can mathematically confirm that these two projection spaces are perfectly orthogonal to one another:

$$(I-P)P = P - P^2 = P - P = 0$$

$$\implies \ker(P) = \text{Im}(I-P)$$

$$\therefore \text{Solutions for } Px = b: \quad b + (I-P)y$$

2 Gradient Descent with Equality Constraints

Now we apply our geometric understanding of projection matrices to continuous optimization. Suppose we want to minimize a continuous function $f(y)$ subject to a set of linear equality constraints. We can define our optimization problem as:

$$\min f(y) \quad \text{subject to} \quad Ay = b \tag{1}$$

Let $S = \{y \mid Ay = b\}$ be the set of all feasible solutions.

If we were to use standard Gradient Descent, our update rule would be:

$$y_{t+1} = y_t - \frac{1}{\beta}\nabla f(y_t)$$

where $1/\beta$ is the step size (learning rate). The problem with this standard update is that the negative gradient $-\nabla f(y_t)$ might point outside of our feasible set S. Taking a step in this direction would violate our constraint $Ay = b$.

2.1 The Projected Update Rule

To fix this, we need our iterative steps to only move along directions that do not change the value of Ay. In other words, our step direction must exist entirely within the null space of A (null(A)).

We achieve this by applying a projection matrix $(I - P)$, which projects any vector onto null(A). The modified Projected Gradient Descent update rule becomes:

$$y_{t+1} = y_t - \frac{1}{\beta}(I - P)\nabla f(y_t) \tag{2}$$

This modified update rule is equivalent to taking a standard gradient step and then performing a Euclidean projection back onto the feasible constraint set $S = \{y \mid Ay = b\}$.

Assume our current point $y_t \in S$ and $Ay_t = b$. Let z be the point resulting from a standard, unconstrained gradient step:

$$z = y_t - \frac{1}{\beta}\nabla f(y_t)$$

We want to find the Euclidean projection of z onto the affine set S. Because y_t is already in S, the set can be rewritten as $S = y_t + \text{null}(A)$. Projecting z onto S is therefore equivalent to taking y_t and adding the projection of the step direction $(z - y_t)$ onto the null space of A:

$$\text{Proj}_S(z) = y_t + \text{Proj}_{\text{null}(A)}(z - y_t)$$

Substitute our unconstrained step $(z - y_t) = -\frac{1}{\beta}\nabla f(y_t)$ into the equation. Since the matrix $(I - P)$ is the projection operator onto null(A), we get:

$$\text{Proj}_S(z) = y_t + (I - P)\left(-\frac{1}{\beta}\nabla f(y_t)\right)$$

$$\text{Proj}_S(z) = y_t - \frac{1}{\beta}(I - P)\nabla f(y_t)$$

This perfectly matches our modified update rule in Equation (2), confirming that projecting the gradient before the step is identical to projecting the point after an unconstrained step.

2.2 Proof of Continuous Feasibility

The primary advantage of using projection matrices is that once we find a single valid starting point, every subsequent step is mathematically guaranteed to remain valid.

Suppose we initialize our algorithm with a feasible point $y_0 \in S$, meaning $Ay_0 = b$. Let us observe what happens when we calculate the next iteration, y_1, using Equation (2). We multiply both sides by A:

$$Ay_1 = A(y_0 - \frac{1}{\beta}P\nabla f(y_0))$$
$$Ay_1 = Ay_0 - \frac{1}{\beta}AP\nabla f(y_0)$$

Because P is explicitly defined as the projection onto the null space of A, any vector multiplied by P is in null(A). Therefore, by definition, $AP = 0$. Substituting this back into our equation gives:

$$Ay_1 = Ay_0 - \frac{1}{\beta}(0)\nabla f(y_0)$$
$$Ay_1 = Ay_0$$

Since $Ay_0 = b$, it directly follows that $Ay_1 = b$. Thus, $y_1 \in S$. By induction, if we start with $y_0 \in S$, all subsequent y_t's will also be in S. This allows us to smoothly optimize complex constrained functions without needing to constantly re-check or enforce constraints at every step.

3 Minimum Cut Formulation

The Min-Cut problem involves partitioning nodes such that the minimum number of edges exist between the partitions.

To formalize this, we apply a labeling scheme where a node v is assigned a value $x_v = 1$ if it is in the source set S, and $x_v = 0$ if it is in the sink set t. The total number of edges crossing the cut is denoted by $|\delta(S)|$.

We can formulate the problem of finding the minimum cut as the following objective function, which minimizes the sum of differences across all edges:

$$\min \sum_{uv \in E} |x_u - x_v| \tag{3}$$

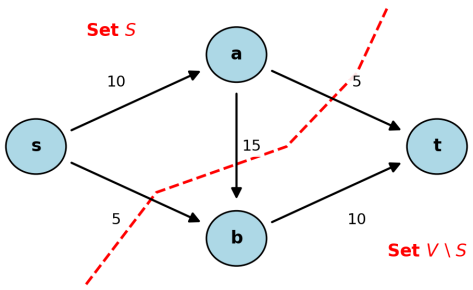

Figure 2: Example of a Min-Cut partition where s is the source node and t is the sink node

This is subject to constraints that anchor the source node s and sink node t into separate partitions:

$$\text{s.t.} \quad x_s = 1, \quad x_t = 0 \implies x_s - x_t = 1$$

Additionally, every node must be strictly assigned to one partition or the other:

$$x_v \in \{0, 1\}$$

The logic behind the objective function is that $|x_u - x_v| = 0$ for any two connected nodes that are placed in the same set. Therefore, $|x_u - x_v| = 1$ only when one node is in the source set S and the other is in the sink set T.

Because the variables x_v are restricted to discrete binary values, this formulation is an Integer Program.

Note that (3) is not a linear objective function in the given form; however, it can be reformulated as one using the following trick:

Let $y_{uv} = |x_u - x_v|$ for every edge $(u, v) \in E$. Then y_{uv} is the maximum of $(x_u - x_v)$ and $-(x_u - x_v)$. This can be written as two linear constraints:

1. $y_{uv} \geq x_u - x_v$
2. $y_{uv} \geq x_v - x_u$

Because our objective is to minimize the sum of all y_{uv}, the optimization solver will push y_{uv} down as far as it can go. The constraints force it to stop exactly at the larger of the two values (either $x_u - x_v$ or $x_v - x_u$), meaning y_{uv} will perfectly equal $|x_u - x_v|$ at the optimal solution.
Moreover, since $x_v \in \{0, 1\}$ for all nodes $v \in V$, $y_{uv} \in \{0, 1\}$ for all edges $uv \in E$.

Thus, we now have a linear objective function

$$\min \sum_{uv \in E} y_{uv}$$

subject to the linear constraints

- Edge cut constraints:

$$y_{uv} - x_u + x_v \geq 0 \quad \text{for all } uv \in E$$
$$y_{uv} - x_v + x_u \geq 0 \quad \text{for all } uv \in E$$

- Terminal constraints:

$$x_s = 1$$
$$x_t = 0$$

- Integrality constraints:

$$x_v \in \{0, 1\} \quad \text{for all } v \in V$$
$$y_{uv} \in \{0, 1\} \quad \text{for all } uv \in E$$

This reformulation allows us to relax IP to LP in the next part.

3.1 Relaxation to Linear Program

To simplify the Integer Program, we relax the strict binary constraint to a continuous interval:

$$\text{Relaxation to LP:} \quad 0 \leq x_v \leq 1$$

This relaxes the terminal constraints in the above IP formulation to $x_s - x_t = 1$, and the other integral constraint to $0 \leq y_{uv} \leq 1$. The edge cut constraints remain unchanged.

Let the optimal objective value of this continuous Linear Program be p. Because the LP relaxes the constraints of the IP, its optimal value serves as a lower bound for the true IP optimal value, although we will show below that they are actually equivalent for our case.

To round the continuous variables back into discrete sets, we define a cut based on a threshold l. A node v is placed into the set S_l if its continuous value x_v falls at or below this threshold:

$$S_l = \{v \mid x_v \leq l\}$$

The total number of edges crossing the boundary between S_l and the remaining vertices $V \setminus S_l$ is denoted as $|\delta(S_l)|$.

The core of the proof relies on expressing the LP objective value as an average or integral over all possible thresholds from 0 to 1. An edge (u, v) crosses the cut S_l if and only if the threshold l falls between their respective values, meaning $x_u \leq l \leq x_v$. The size of this interval is $|x_u - x_v|$. Summing this over all edges yields the central identity:

$$p = \sum_{uv \in E} |x_u - x_v| = \int_0^1 |\delta(S_l)| dl \tag{4}$$

This identity can also be understood via randomized rounding. Suppose we pick a random threshold l uniformly from the interval $[0, 1]$ and output the corresponding cut S_l. The probability that any given edge (u, v) is cut is exactly the distance between their assigned values, $|x_u - x_v|$.

By linearity of expectation, the expected size of this randomly generated cut is exactly equal to the optimal LP value, p.

Since the integral (or average/expected value) of the cut sizes over the interval is p, there must exist at least one threshold l that produces a cut whose size is at most p. Because the number of edges must be an integer, this cut size is bounded by $\lfloor p \rfloor$.

Therefore, we can bound the optimal IP value from both sides. It is bounded below by the LP minimum, and bounded above by the specific integer cut we proved must exist:

$$OPT_{LP} \leq OPT_{IP} \leq \lfloor p \rfloor$$

Since $OPT_{LP} = p$, this implies that the LP relaxation finds the exact integer solution for the Min-Cut problem.

3.2 Smoothing the LP Objective for Gradient Descent

The objective function of our Linear Program involves an absolute value, which is non-differentiable. To apply Gradient Descent, we introduce a smooth approximation using a small parameter μ. The original objective is bounded by this new smooth function:

$$\text{LP:} \quad \min \sum_{uv \in E} |x_u - x_v| \leq \sum_{uv \in E} \sqrt{(x_u - x_v)^2 + \mu^2} \tag{5}$$

This substitution works because the smooth function tightly bounds the absolute value. The error introduced by this approximation is at most μ per edge:

$$|x_u - x_v| \leq \sqrt{(x_u - x_v)^2 + \mu^2} \leq |x_u - x_v| + \mu$$

Because of this per-edge bound, the optimal value of the newly smoothed Linear Program is bounded by the original LP's optimal value (denoted as F) plus the accumulated error across all edges in the graph, $\mu|E|$:

$$OPT_{\text{new LP}} \leq OPT_{LP} + \mu|E| = F + \mu|E|$$

Running Gradient Descent on this smoothed function yields an approximate solution $\overline{F}$. This solution includes both the smoothing error $\mu|E|$ and the optimization error from Gradient Descent, denoted as δ. We want this final solution to be a $(1+\epsilon)$-approximation of the true optimal value F:

$$\overline{F} \leq F + \mu|E| + \delta = (1+\epsilon)F \tag{6}$$

To satisfy this condition, the total allowed error ϵF is divided equally between the optimization error and the smoothing error:

$$\text{i.e. set} \quad \delta = \frac{\epsilon F}{2}, \quad \mu = \frac{\epsilon F}{2|E|}$$

NOTE: In practice, the true optimal cut value F is unknown at the start of the algorithm. To implement the parameter setting without knowing F, we use a standard algorithm design technique known as **logarithmic guessing**.

1. We know that the optimal cut size must be an integer bounded by $1 \leq F \leq |E|$.
2. We can guess the value of F in powers of 2. Let our guess be $F' \in \{1, 2, 4, 8, \ldots, |E|\}$.
3. For each guess F', we set our parameters $\mu = \frac{\epsilon F'}{2|E|}$ and $\delta = \frac{\epsilon F'}{2}$ and run the projected gradient descent algorithm for the required number of iterations.
4. Each run will produce a continuous solution that we round to a discrete cut (as described via the thresholding in section 3.1). We simply keep track of the cuts produced and return the minimum cut found across all runs.

Because we only guess in powers of 2, there are at most $\mathcal{O}(\log |E|)$ guesses, meaning this adds at most a logarithmic factor to the overall runtime.Furthermore, because this continuous algorithm actually runs faster for larger values of F (since the number of iterations scales with $\frac{1}{\sqrt{F}}$ in the edge space formulation), we can optimize the search by guessing top-down (starting from $F' = |E|$ and halving it). We can stop as soon as we find a valid cut of size $\leq (1 + \epsilon)F'$. The total running time will be bounded by a geometric series that is heavily dominated by the runtime of the final, smallest guess. As a result, the asymptotic runtime bound of the algorithm is perfectly preserved.

The smoothness parameter of this new function, denoted as β, is defined by the inverse of the smoothing parameter.

$$\text{Smoothness of new func}^n : \quad \beta = \frac{1}{\mu}$$

Proof: Let's write $y_e = |x_u - x_v|$ where $e = (u, v) \in E$ and

$$g(y) = \sum_{e \in E} \sqrt{y_e^2 + \mu^2}$$

.

Then, the gradient $\nabla g(y)$ is easy to compute:

$$\frac{\partial}{\partial y_e} g(y) = \frac{\partial}{\partial y_e} \sum_{e \in E} \sqrt{y_e^2 + \mu^2} = \frac{\partial}{\partial y_e} \sqrt{y_e^2 + \mu^2} = \frac{y_e}{\sqrt{y_e^2 + \mu^2}}$$

Note that because this partial derivative depends only on y_e, all mixed partial derivatives evaluate to zero (i.e., $\frac{\partial^2}{\partial y_e \partial y_{e'}} g(y) = 0$ for $e \neq e'$), i.e., the off-diagonal entries of the Hessian are zero. Now,

$$\frac{\partial^2}{\partial y_e^2} g(y) = \frac{\partial}{\partial y_e} \left(\frac{y_e}{\sqrt{y_e^2 + \mu^2}} \right) = \frac{-y_e^2}{(y_e^2 + \mu^2)^{\frac{3}{2}}} + \frac{1}{(y_e^2 + \mu^2)^{\frac{1}{2}}} = \frac{\mu^2}{(y_e^2 + \mu^2)^{\frac{3}{2}}}$$

Hence the Hessian matrix is just a diagonal matrix with entries

$$\frac{\mu^2}{(y_e^2 + \mu^2)^{\frac{3}{2}}} \leq \frac{\mu^2}{(\mu^2)^{\frac{3}{2}}} = \frac{1}{\mu}$$

The maximum eigenvalue of the Hessian is at most $\frac{1}{\mu}$ and so the smoothness parameter is $\frac{1}{\mu}$. □

Now, let $S = \{x \mid Ax = b\}$ be our feasible set and f be a β-smooth convex function. Given a starting point x_0 and a gradient oracle $(I - P)\nabla f(x)$, we want to find a point x_T such that the error is bounded by ϵ:

$$\implies x_T : f(x_T) - f(x^*) \leq \epsilon$$

To understand the cost of the gradient oracle, we must evaluate the projection step. In this specific context, our only constraint is the terminal condition $x_s - x_t = 1$. We can express this as a linear constraint $a^T x = 1$, where $a = \chi_s - \chi_t$ is a vector with 1 at index s, -1 at index t, and 0 elsewhere (in general, χ_i represents the indicator vector for a node i, containing a 1 at the index corresponding to i and 0s everywhere else). The projection matrix P onto the one-dimensional subspace spanned by this constraint is simply:

$$P = \frac{aa^T}{\|a\|^2} = \frac{(\chi_s - \chi_t)(\chi_s - \chi_t)^T}{2}$$

When we apply the projection operator $(I - P)$ to our gradient vector $\nabla f(x)$, we are effectively computing:

$$(I - P)\nabla f(x) = \nabla f(x) - \frac{1}{2}(\chi_s - \chi_t)(\nabla f(x)_s - \nabla f(x)_t)$$

Notice that this projection operation only reads and modifies the s and t coordinates of the gradient. Because it requires just $\mathcal{O}(1)$ arithmetic operations, it is extremely easy and fast to compute. This guarantees that projecting the gradient back into the feasible set adds no asymptotic overhead to the $\mathcal{O}(|E|)$ time required to compute the gradient $\nabla f(x)$ itself.

From Nesterov's accelerated gradient descent [6], the running time (number of iterations) required to achieve this accuracy is bounded by:

$$T = \mathcal{O}\left(\sqrt{\frac{\beta}{\delta}}\|x_0 - x^*\|_2\right)$$

This provides a quadratic improvement over standard Gradient Descent, which would require $\mathcal{O}(\frac{\beta}{\delta}\|x_0 - x^*\|_2^2)$ iterations. This acceleration is a crucial component in achieving the fast overall running time for our minimum cut algorithm.

We substitute our specific parameters into this bound. We know $\beta = \frac{1}{\mu} = \frac{2|E|}{\epsilon F}$, and the distance to the optimal solution is bounded by $\|x_0 - x^*\|_2 \leq \sqrt{|V|}$. This gives the total number of iterations:

$$T = \mathcal{O}\left(\frac{\sqrt{|E|}}{\epsilon F} \cdot \|x_0 - x^*\|_2\right) = \mathcal{O}\left(\frac{\sqrt{|E||V|}}{\epsilon F}\right) \quad \text{\# iterations}$$

The time required to compute the gradient at each step scales with the number of edges:

$$\text{Time for gradient: } \mathcal{O}(|E|)$$

Multiplying the number of iterations by the per-iteration time gives the total time complexity for the algorithm:

$$\therefore \text{Total time} = \mathcal{O}\left(\frac{|E|^{3/2}\sqrt{|V|}}{\epsilon F}\right) \tag{7}$$

At first glance, this bound might not seem uniformly competitive. There already exist classical combinatorial algorithms that can solve the minimum cut problem exactly in $\mathcal{O}(|E|^{1.5})$ time. However, for a large optimal cut value of $F = \sqrt{|V|}$, the runtime of our Accelerated Gradient Descent approach simplifies to $\mathcal{O}(|E|^{1.5}/\epsilon)$, which effectively matches the best known combinatorial methods at the time. This analysis reveals a fascinating property: the continuous algorithm runs faster as the optimal cut value F becomes larger, which is the exact opposite behavior of traditional augmenting path algorithms. Despite this advantage for large cuts, it is important to note that the Gradient Descent approach is not uniformly faster in general, and furthermore, it only computes an ϵ-approximate flow or cut rather than an exact solution.

3.3 Incidence Matrix and Subspace Projection

To further improve the running time of our algorithm, we will now change the dimensionality of our Linear Program. The crucial idea introduced by Lee, Rao, and Srivastava (LRS)[5] is to perform a change of variables so that we optimize over the edge space $\mathbb{R}^{|E|}$ instead of the vertex space $\mathbb{R}^{|V|}$. To map our problem into this new space, we first introduce a fundamental graph matrix.

The Incidence Matrix $B_{m\times n}$ maps vertices to edges. For an edge uv, its row contains a 1 at column u and -1 at column v.

$$B_{m\times n} = \begin{array}{c} \\ uv \end{array} \begin{array}{c} \begin{array}{cccccccccc} & & & u & & & & v & & \end{array} \\ \begin{bmatrix} & & & & & & & & & \\ & & & & & & & & & \\ 0 & \dots & 0 & 1 & 0 & \dots & 0 & -1 & 0 & \dots 0 \\ & & & & & & & & & \\ & & & & & & & & & \end{bmatrix} \end{array}$$

Multiplying B by a node vector x yields the edge differences:

$$Bx \implies x_u - x_v \longrightarrow \chi_u - \chi_v \begin{bmatrix} x_u \\ x_v \end{bmatrix}$$

We perform a change of variables by setting $y = Bx$. This reformulates the problem from the vertex space to the edge space:

$$\min \sum |x_u - x_v| \text{ s.t. } x_s - x_t = 1 \implies \min f(y) \text{ s.t. } Py = b$$
$$\text{where } \sum |x_u - x_v| = \|Bx\|_1 = \sum_{e \in |E|} |y_e|$$

(Here, $Py = b$ encapsulates the transformed constraints on our new variable y; we will derive the exact form of P and b below.)

This introduces two constraints on y. First, y must lie in the column space of B. Let $\Pi = B^T(B^TB)^{-1}B$ be the projection matrix onto the span of B. Since $y = Bx$, it follows that $y = \Pi y$, giving our first constraint:

$$(I - \Pi)y = 0 \tag{8}$$

Second, we transform the boundary constraint $x_s - x_t = 1$. Let χ_s, χ_t be indicator vectors for the source and sink. We can rewrite the constraint as an inner product, insert the graph Laplacian $L = B^TB$ and its inverse L^{-1}, and use the adjoint property to shift B to the left side:

$$\langle x, \chi_s - \chi_t \rangle = 1$$
$$\langle x, B^TBL^{-1}(\chi_s - \chi_t) \rangle = 1$$
$$\langle Bx, BL^{-1}(\chi_s - \chi_t) \rangle = 1$$

By defining $y_{st} = BL^{-1}(\chi_s - \chi_t)$, this constraint becomes $\langle y, y_{st} \rangle = 1$.

We combine these conditions into a single equality constraint system $Py = b$, where:

$$P = (I - \Pi) + \frac{y_{st}y_{st}^T}{\|y_{st}\|^2}, \quad b = \frac{y_{st}}{\|y_{st}\|^2}$$

During Gradient Descent, we must project the gradient $\nabla f(y)$ using the operator $(I - P)$:

$$(I - P)\nabla f(y) = \left(\Pi - \frac{y_{st}y_{st}^T}{\|y_{st}\|^2}\right) \nabla f(y) \tag{9}$$

To compute the dominant term $\Pi \nabla f(y) = B^T\ L^{-1}B\nabla f(y)$ efficiently, we evaluate it step-by-step from right to left:

$$\underbrace{B^T\ \underbrace{L^{-1}\ \underbrace{B\nabla f(y)}_{\mathcal{O}(|E|)}}_{\mathcal{O}(|E|)}}_{\mathcal{O}(|E|)} \tag{10}$$

The operations $B\nabla f(y)$ and multiplication by B^T both take linear $\mathcal{O}(|E|)$ time because the incidence matrix is sparse(B has only $2|E|$ non-zero entries). Evaluating the middle term requires solving the linear system $Lx = z$ (where $z = B\nabla f(y)$). Because L is a graph Laplacian matrix,

we do not need to rely on slow, traditional matrix inversion techniques (which typically require $\mathcal{O}(|V|^3)$ time). Instead, we can utilize a specialized algorithm known as a Laplacian solver [3]. These solvers achieve dramatic speedups by leveraging the specific Symmetric Diagonally Dominant (SDD) structure of the Laplacian to approximate the solution in near-linear time, $\tilde{\mathcal{O}}(|E|)$. Thus, the overall time to compute the projected gradient is linear with respect to the number of edges.

Before moving to the runtime analysis, let's explicitly state the full smoothed formulation that we will solve using Projected Gradient Descent in the edge space. Drawing from our smoothing technique in Section 3.2 and our change of variables, the complete optimization problem is:

$$\min_y g(y) = \sum_{e \in E} \sqrt{y_e^2 + \mu^2}$$

$$\text{subject to } Py = b$$

where P and b encapsulate the span and terminal constraints as derived above. This is the precise continuous formulation to which we apply Nesterov's accelerated method.

3.4 Runtime Analysis

The primary motivation for reformulating the problem from the vertex space (x) to the edge space (y) is to obtain a significantly tighter bound on the initial distance to the optimal solution. In the original vertex space, we could only crudely bound the distance from our starting point to the optimal solution as $||x_0 - x^*||_2 \leq \sqrt{|V|}$. However, in the transformed edge space, our initial feasible point $y_0 = \frac{y_{st}}{||y_{st}||^2}$ is the minimum norm solution to the constraint system $Py = b$. Because it is the projection of the origin onto the solution space, the distance to the optimal edge-space solution y^* is strictly bounded by the norm of y^* itself. Since an optimal cut containing F edges corresponds to a vector y^* with exactly F non-zero entries of magnitude 1, we get a much tighter distance bound:

$$||y_0 - y^*||_2 \leq ||0 - y^*||_2 = \sqrt{F}$$

To determine the improved runtime, we recall the iteration complexity bound from Nesterov's accelerated gradient descent:

$$T = \mathcal{O}\left(\sqrt{\frac{\beta}{\delta}}||y_0 - y^*||_2\right)$$

As established in our previous smoothing analysis (Section 3.2), to achieve a $(1+\epsilon)$-approximation, we set our smoothness parameter to $\beta = \frac{2|E|}{\epsilon F}$ and our target accuracy to $\delta = \frac{\epsilon F}{2}$. By plugging these parameters and our new diameter bound into the iteration complexity formula, the number of required iterations drops significantly:

$$T = \mathcal{O}\left(\sqrt{\frac{2|E|/\epsilon F}{\epsilon F/2}} \cdot \sqrt{F}\right) = \mathcal{O}\left(\frac{\sqrt{|E|}}{\epsilon F} \cdot \sqrt{F}\right) = \mathcal{O}\left(\frac{1}{\epsilon}\sqrt{\frac{|E|}{F}}\right)$$

Because the Laplacian solver allows us to compute the projected gradient at each step in near-linear $\tilde{\mathcal{O}}(|E|)$ time, the total running time for this continuous approach becomes:

$$\text{Total Time} = \text{Iterations} \times \text{Time per Iteration} = \mathcal{O}\left(\frac{1}{\epsilon}\sqrt{\frac{|E|}{F}}\right) \cdot \tilde{\mathcal{O}}(|E|) = \tilde{\mathcal{O}}\left(\frac{|E|^{1.5}}{\epsilon\sqrt{F}}\right)$$

3.5 Comparison to Combinatorial Methods

This continuous gradient descent approach exhibits a surprising property: it solves the Min-Cut problem faster when the optimal cut size F is large. This directly contrasts with traditional augmenting-path and combinatorial algorithms, whose runtimes typically degrade as the cut size increases.

For instance, the combinatorial algorithm by Karger and Levine [1] operates in $\tilde{\mathcal{O}}(|E| + |V|F)$ time. In 2013, Lee, Rao, and Srivastava (LRS) [5] demonstrated that by balancing these two approaches, running the combinatorial algorithm when F is small ($F \leq \frac{|E|}{(|V|\epsilon)^{2/3}}$) and the continuous gradient descent algorithm when F is large, one could achieve a combined best-case runtime of $\mathcal{O}(|E||V|^{1/3}\epsilon^{-2/3})$.

Since then, further developments in continuous optimization methods, combined with combinatorial subroutines, have continued to push these bounds, eventually leading to algorithms [2][7] that can solve the Min-Cut problem in near-linear $\tilde{\mathcal{O}}(|E|)$ time using projected gradient descent.

There are some current shortcomings to this continuous approach, for example, it is not yet applicable to directed graphs and remains inefficient for weighted graphs. While there has been recent progress, many open questions still remain. Ultimately, the main takeaway is that relatively simple convex optimization methods can yield surprising improvements for classic combinatorial problems.

References

[1] D. R. Karger and M. S. Levine. Fast augmenting paths by random sampling from residual graphs. *SIAM Journal on Computing*, 44(2):310–327, 2015.

[2] J. A. Kelner, Y. T. Lee, L. Orecchia, and A. Sidford. An almost-linear-time algorithm for approximate max flow in undirected graphs, and its multicommodity generalizations. In *Proceedings of the Twenty-Fifth Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 217–226. SIAM, 2014.

[3] I. Koutis, G. L. Miller, and R. Peng. A nearly-m log n time solver for sdd linear systems. In *2011 IEEE 52nd Annual Symposium on Foundations of Computer Science (FOCS)*, pages 590–598. IEEE, 2011.

[4] L. C. Lau. Lecture 8: Minimum cut. CS 798: Convexity and Optimization, University of Waterloo, 2017. https://cs.uwaterloo.ca/ lapchi/cs798/L08.pdf.

[5] Y. T. Lee, S. Rao, and N. Srivastava. A new approach to computing maximum flow using electrical flows. In *Proceedings of the 45th Annual ACM Symposium on Theory of Computing (STOC)*, 2013.

[6] Y. E. Nesterov. A method of solving a convex programming problem with convergence rate $O(1/k^2)$. *Dokl. Akad. Nauk SSSR*, 269(3):543–547, 1983.

[7] R. Peng. Approximate undirected maximum flows in $O(m \text{ polylog}(n))$ time. In *Proceedings of the Twenty-Seventh Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 1862–1880. SIAM, 2016.