Advanced Topics in Algorithms

Kent Quanrud\textsuperscript{1}

Course book for CS593ATA, Fall 2021\textsuperscript{2}

\textsuperscript{1}krq@purdue.edu.
\textsuperscript{2}Last compiled December 13, 2021.
Class Schedule

1. **August 27.** Introductions, LP duality, max-flow min-cut, Ford-Fulkerson (chapter 1, section 2.1).

2. **September 3.** Augmenting path algorithms for max-flow (rest of chapter 2). HW1 released.

3. **September 10.** Push-relabel, directed minimum cut (chapter 3).

4. **September 17.** Introduction to multicommodity flows and cuts, multicut, and 2-approximate multiway cuts (section 4.1 – section 4.3.1). HW1 due, HW2 released.

5. **September 24.** 1.5-approximate multiway cut, sparsest cut (rest of chapter 4, chapter 5).

6. **October 1.** $L_1$-metric embeddings for sparsest cut (chapter 5). HW2 due on Monday. HW3 released.

7. **October 8.** Expander lower bounds for sparsest cut. Introduce edge-disjoint paths and all-or-nothing flow (chapter 6).

8. **October 15.** All-or-nothing flow (chapter 6). Introduce cut-matching (chapter 7).


10. **October 29.** Low-stretch spanning trees and randomized tree metrics (chapter 8).

11. **November 5.** LP duality via approximation (chapter 9).

12. **November 12.** Oblivious routing, introduction to spectral graph theory and low conductance cuts (chapters 10 and 11).


14. **November 26.** Thanksgiving holiday.

15. **December 3.** Computing electrical flows (chapter 12).

16. **December 10.** Max flow via electrical flow (chapter 13).

---

1All future dates should be regarded as tentative. The schedule is typically updated after the lecture to reflect what was covered.
# Contents

## Class Schedule

## Lecture Notes

1. **LP Duality and Max-Flow Min-Cut**
   1.1 Packing and covering paths ........................................... 8
   1.2 Linear programming and LP duality ................................. 12
   1.3 Max-flow min-cut via LP duality .................................... 15
   1.4 Exercises ............................................................... 17
   1.5 Additional notes and references ................................. 18

2. **Augmenting Path Algorithms for Max Flow**
   2.1 Ford-Fulkerson and Menger’s Theorem .............................. 19
   2.2 Shortest augmenting paths ........................................... 25
   2.3 Blocking Flows ......................................................... 28
   2.4 Fewer blocking flows suffice in simple graphs ................. 30
   2.5 Data structures for blocking flows ............................... 32
   2.6 Short blocking flows suffice for integer capacities (via scaling) 34
   2.7 Exercises ............................................................... 39
   2.8 Additional notes and references ................................. 40

3. **Push-Relabel and Directed Minimum Cut**
   3.1 Push-Relabel Max Flow ................................................. 42
   3.2 Top-down push-relabel ................................................. 47
   3.3 Accelerating push-relabel with data structures ............... 49
   3.4 Scaling push-relabel .................................................. 51
   3.5 Directed minimum cut via one push-relabel ...................... 52
   3.6 Rooted sparsification in simple graphs ............................ 54
   3.7 Additional notes and references ................................... 57
   3.8 Exercises ............................................................... 57
# Contents

4 Multicut and Multiway Cut

4.1 Beyond \((s,t)\)-flows and cuts .................................................. 58
4.2 Multicut ................................................................. 60
4.3 Multiway cut .............................................................. 67
4.4 Exercises ................................................................. 74
4.5 Further notes and references ............................................... 75

5 Sparsest cut

5.1 The LP ................................................................. 76
5.2 Rounding via region growing (for uniform demands) ............... 77
5.3 Rounding via \(L_1\)-metric embeddings .............................. 82
5.4 Expander lower bounds ............................................... 90
5.5 Application: Minimum bisection ................................... 93
5.6 Exercises ................................................................. 94
5.7 Additional notes and references ....................................... 95

6 Edge-Disjoint Paths and All-or-Nothing Multicommodity Flow

6.1 Edge-Disjoint Paths .................................................. 96
6.2 The All-or-Nothing Multicommodity Flow Problem ............. 97
6.3 Well-linked terminals ................................................ 98
6.4 Approximating All-or-Nothing Flow ................................ 103
6.5 Exercises ................................................................. 106

7 Cut-Matching

7.1 (Sparse) cuts and (expanding) matchings: some intuition .......... 107
7.2 Preliminaries ............................................................ 109
7.3 Setting up the cut-matching game .................................. 109
7.4 A randomized cut strategy ........................................... 110

8 Tree Metrics

8.1 Introduction ........................................................... 114
8.2 Low-Stretch Spanning Trees ......................................... 115
8.3 Hierarchical Tree Metrics .......................................... 117
8.4 Exercises ................................................................. 121

9 LP duality by approximation

9.1 Introduction ........................................................... 122
9.2 Langrangian Relaxations ............................................ 122
9.3 A continuous algorithm ............................................. 123
9.4 Exercises ................................................................. 129
III  Course Information  195

C  Homework assignments  196
   C.1  Homework 1  196
   C.2  Homework 2  197
   C.3  Homework 3  198
   C.4  Homework 4  199

D  Syllabus  200
   D.1  Dates and times  200
   D.2  Course description  200
   D.3  Learning outcomes  201
   D.4  Textbooks  202
   D.5  Correspondence  202
   D.6  Grading  203
   D.7  Homework  203
   D.8  Academic integrity  205
   D.9  Posting Class Material  206
   D.10  Purdue’s Honor Pledge  206
   D.11  Grief Absence Policy  206
   D.12  Conduct and Courtesy  206
   D.13  Students with Disabilities  207
   D.14  Emergencies  207
   D.15  Violent Behavior Policy  208
   D.16  Mental Health and Wellness  208
   D.17  Nondiscrimination  209
   D.18  Privacy  209
   D.19  Changes to the syllabus  210
Part I

Lecture Notes
Chapter 1

LP Duality and
Max-Flow Min-Cut

The reader may (should) have seen max flow, the max-flow min-cut theorem, and some algorithms for max flow in an introductory class on algorithms. Here we present these results from a different perspective that sets the stage for other topics in this course.

1.1 Packing and covering paths

Let \( G = (V,E) \) be a directed graph\(^1\), and let \( s, t \in V \) be two distinct vertices. We call \( s \) the **source** and \( t \) the **sink**. A **path packing** is a collection of edge disjoint paths. An \((s,t)\)-path packing is a path packing of \((s,t)\)-paths. The **maximum \((s,t)\)-path packing** problem\(^2\) is to

\[
\text{find a maximum cardinality packing of } (s,t)\text{-paths.}
\]

---

\(^1\)G is allowed to be a multi-graph, with multiple copies of the same edge.

\(^2\)Better known as the **uncapacitated maximum \((s,t)\)-flow problem** for reasons we discuss later.
An \((s,t)\)-cut is a set of edges whose removal disconnects \(s\) from \(t\). The **minimum \((s,t)\)-cut problem** is to

\[
\text{find the minimum cardinality \((s,t)\)-cut.}
\]

Both can be understood as a generalization of reachability. Reachability is concerned with whether there is a single connection from \(s\) to \(t\). Both the maximum flow and minimum cut problems measure the strength of the connection from \(s\) to \(t\).

We are interested in these questions both algorithmically and (graph-)structurally. Algorithmically the problems are highly non-trivial, as there are exponentially many possible paths from \(s\) to \(t\) to take into account. It is not obvious that there is a polynomial time algorithm for either problem.

### 1.1.1 Duality.

The \((s,t)\)-path packing and \((s,t)\)-cut problem are **dual** packing and covering problems, in the following sense. We are selecting paths that “pack in” to the edges of the graph – each path uses up all of its edges. Conversely, an \((s,t)\)-cut must contain at least one edge from every \((s,t)\)-path. That is, we are trying to “cover” the paths with edges, where we interpret each edge as a set that covers all the \((s,t)\)-paths that contain that edge. In short:

*paths pack into edges, and edges cover paths.*

As with any packing and covering problem, we have the following inequality.

\[
\text{(max \((s,t)\)-path packing)} \leq \text{(min \((s,t)\)-cut).}
\]  

1.1

Indeed, let \(P\) denote a path packing, and let \(C \subseteq E\) be an \((s,t)\)-cut. We have

\[
|P| = \sum_{p \in P} 1 \leq \sum_{p \in P} |p \cap C| \leq |C|.
\]

Above, we treat each path \(p \in P\) as a subset of edges. (a) is because, as an \((s,t)\)-cut, \(C\) contains at least one edge from every \((s,t)\)-path. (b) is because the paths \(p \in P\) are edge disjoint, so the sets \(p \cap C\) over \(p \in P\) are also disjoint. The following is a conceptual sketch of our argument.
The inequality (1.1) inspires some basic questions. Is the inequality ever equal? Is the inequality ever strict?

We now introduce some variations of the \((s,t)\)-paths and cut problems.

### 1.1.2 Capacities and costs.

A natural generalization of disjoint \((s,t)\)-paths allows edges to be reused, and extends the input to include **edge capacities** \(c : E \rightarrow \mathbb{R}_{>0}\) that set a numerical limit on how many times each edge can be used. For example, if an edge \(e\) has \(c(e) = 2\), then this implies we are allowed to use \(e\) twice (i.e., as if there are two copies of \(e\)). The same path is allowed to be selected multiple times. Formally, the problem becomes:

Find a collection of paths \(P\) of maximum cardinality \(|P|\) such that each edge \(e\) is contained in at most \(c(e)\) paths.

We denote the problem formalized above as \((\text{Max-Paths})\). Abusing notation, we will also let \((\text{Max-Paths})\) denote the optimum objective value of the \((\text{Max-Paths})\) problem.

Introducing edge capacities makes it even more challenging to find a polynomial time algorithm. Before, without edge capacities, it is clear that the optimum solution has a polynomial number of paths since each path must use up at least one edge. Once we introduce capacities – which may be large numbers expressed in a logarithmic number of bits – we can no longer assume that the maximum path packing has polynomial size!

Flip back to the dual \((s,t)\)-cut problem. A natural generalization introduces positive **edge costs** \(c : E \rightarrow \mathbb{R}_{>0}\); the problem becomes:

Find an \((s,t)\)-cut \(C \subseteq E\) of minimum cost \(\sum_{e \in C} c(e)\).

\((\text{Min-Cut})\)

The minimum cardinality cut problem is equivalent to the minimum cost cut problem with uniform costs \((c(e) = 1\) for all \(e)\). In contrast to edge capacities, edge costs do not invoke the risk of the optimum solution no longer being compact.

As with \((\text{Max-Flow})\), we write \((\text{Min-Cut})\) to denote both the problem described above and the optimum value of that problem.

When the capacities and costs are based on the same vector \(c\), then \((\text{Max-Paths})\) and \((\text{Min-Cut})\) are dual\(^3\) to one another. In particular one can show that \((\text{Max-Paths}) \leq (\text{Min-Cut})\) by a similar argument as before in the uncapacitated/uniform-cost setting. We leave the proof to the reader.

\(^3\)We will formalize the definition of dual soon.
Lemma 1.1. Let \( G = (V, E) \) be a directed graph and \( s, t \in V \). Let \( c : E \to \mathbb{R}_{>0} \) be a fixed set of capacities / costs. Then

\[
\text{(Max-Paths)} \leq \text{(Min-Cut)}.
\]

1.1.3 Fractionally packing and covering paths.

Another variation of the above problems allow for \textit{fractional solutions}. Consider first path packings. Let \( \mathcal{P}_{s,t} \) denote the family of all \((s,t)\)-paths. A \textbf{fractional collection of paths} is an assignment \( x : \mathcal{P}_{s,t} \to \mathbb{R}_{\geq 0} \) giving nonnegative weight to every path. A \textbf{fractional path packing} is a fractional collection of paths \( x \) that satisfies the capacity constraint \( c(e) \) for each edge \( e \) in the following quantitative sense:

\[
\sum_{p \ni e} x_p \leq c(e).
\]

(Here “\( p \ni e \)” is summing over all \( p \in \mathcal{P}_{s,t} \) such that \( e \in p \).) Subject to this constraint, the goal is to find the fractional path packing of maximum total quantity,

\[
\sum_{p \in \mathcal{P}_{s,t}} x_p.
\]

Putting it all together, the entire fractional path packing problem is given by:

\[
\begin{align*}
\text{maximize } & \sum_{p \in \mathcal{P}_{s,t}} x_p \text{ over } x : \mathcal{P}_{s,t} \to \mathbb{R}_{\geq 0} \\
\text{s.t. } & \sum_{p \ni e} x_p \leq c(e) \text{ for all } e \in E.
\end{align*}
\]

(Max-Flow)

Note that the objective \((\sum_p x_p)\) is a linear function of \( x \), and the constraints \((x_p \geq 0, \sum_{p \ni e} x_p \leq c(e)...)\) are linear inequalities. This feature is very important and we will return to it in a moment.

Due to the continuous nature of the fractional path packing problems – imagine \( x_p \) units of water flowing along the path \( p \) – it is commonly referred to as the \textbf{maximum flow} problem. For the rest of this section, we let \((\text{Max-Flow})\) refer to the maximum flow problem formulated above. Abusing notation, we will also let \((\text{Max-Flow})\) refer to the optimum value of \((\text{the problem})\) \((\text{Max-Flow})\). We also note that the (discrete) path packing is also called \textbf{integral maximum flow}.

Compare max flow with the (integer) path packing problem discussed above, where we can only select integer multiples of paths. Clearly, any integer solution is a feasible solution to the fractional version. For this reason, the fractional path
packing problem is called a relaxation of the integer path packing problem – every feasible solution to the latter is feasible in the former. As a relaxation of a maximization problem, we always have \((\text{Max-Flow}) \geq (\text{Max-Paths})\).

We can apply the same fractional perspective to the minimum cost \((s,t)\)-cut problem. Recall that an \((s,t)\)-cut contains at least one edge from every \((s,t)\)-path. A fractional \((s,t)\)-cut is a fractional combination of edges \(y : E \to \mathbb{R}_{\geq 0}\) that contains (in sum) one unit of edges from every \((s,t)\)-paths. An edge cost \(c(e)\) are now interpreted as the cost of one unit \(y(e)\). All put together, the fractional relaxation of minimum cost \((s,t)\)-cut is given by the following problem:

\[
\begin{align*}
\text{minimize} & \quad \sum_{e \in E} c(e)y_e \quad \text{over} \quad y : E \to \mathbb{R}_{\geq 0} \\
\text{s.t.} & \quad \sum_{e \in p} y_e \geq 1 \quad \text{for all} \quad p \in \mathcal{P}_{s,t}.
\end{align*}
\]

(1.2) has \(m\) variables but exponentially many constraints. This setup leads to the following situation. We encourage the reader to pause and consider the following question herself before reading on.

Suppose you were given a vector \(y \in \mathbb{R}^E_{\geq 0}\). How would you verify, in polynomial time, that \(y\) is a feasible solution? In particular, how does one verify that for every \((s,t)\)-path \(p\), the sum of \(y_e\)'s over \(e \in p\) is at least 1? (Is it even possible?)

The question is nontrivial because there is not enough time to enumerate every \((s,t)\)-path. But let us reformulate the question slightly: verifying every path \(p\) has \(\sum_{e \in p} y_e \geq 1\) is the same as verifying that the minimum \(\sum_{e \in p} y_e\), over all \(p \in \mathcal{P}_{s,t}\), is at least 1. Let us reinterpret the values \(y : E \to \mathbb{R}_{\geq 0}\) as edge lengths. Then the covering constraint is really saying that the length of the shortest \((s,t)\)-path w/r/t edge lengths \(y_e\), is \(\geq 1\); we can verify this constraint by computing the shortest \((s,t)\)-path w/r/t \(y\). An equivalent formulation of (1.2), then, is as follows.

Find the minimum cost set of edge lengths \(y : E \to \mathbb{R}_{\geq 0}\) subject to \(s\) and \(t\) having distance 1 in the shortest path metric induced by \(y\).

This problem, besides being a relaxation of \((s,t)\)-cut, is a very natural problem in its own right. For this reason, and to help distinguish the continuous nature of (1.2) for the discrete min-cut problem, we will also refer to the fractional min-cut problem as the \((s,t)\)-minimum cost metric problem. We will write \((\text{Min-Metric})\) to denote both the optimization problem and the value of the optimization problem formulated in (1.2) above.

### 1.2 Linear programming and LP duality

The fractional versions of the \((s,t)\)-path packing and cut problem described above are examples of linear programs, a special class of mathematical optimization problems which we now introduce.
Linear programs (LP’s) are constrained continuous optimization problems where the goal is to (a) select a vector \( x \in \mathbb{R}^n \) that (b) optimizes a linear objective subject to (c) linear equality and inequality constraints. That is, an optimization problem of the form

\[
\min/\max \langle b, x \rangle = \sum_{j=1}^{n} b_j x_j \text{ over } x \in \mathbb{R}^n
\]

\[
\text{s.t. } A_1 x \leq c_1, A_2 x = c_2, \text{ and } A_3 x \geq c_3.
\]

where \( A_1, A_2, A_3 \) are matrices and \( b, c_1, c_2, c_3 \) are vectors.

Clearly, linear programs are useful for modeling real problems where we seek continuous solutions. Throughout this class we will encounter many different uses for LP’s for understanding and solving discrete problems as well. A powerful feature of LP’s is that they are polynomial time solvable \(^4\), and conceptually it is easy to interact with these solvers as a black box. Moreover, real-world software for LP’s is well-developed and reliable in practice.

We now introduce two canonical classes of LP’s that capture most combinatorial problems.

**Packing LPs.** A packing LP is a linear program of the form

\[
\max \langle b, x \rangle \text{ over } x \in \mathbb{R}^n_{\geq 0} \text{ s.t. } Ax \leq c.
\]  

\((P)\)

where \( A \in \mathbb{R}^{m \times n} \), \( b \in \mathbb{R}^n_{>0} \), and \( c \in \mathbb{R}^m_{>0} \) all have nonnegative coefficients. We let \( \text{Opt}(P) \) denote the optimum value of the LP \((P)\).

The fractional path packing problem is our first example of an packing LP. For path packing, we have one variable for every path. Identifying edges and paths as coordinates, then, we have:

1. \( b = 1_{P_{s,t}} \), the all-ones vector in \( \mathbb{R}^{P_{s,t}} \)
2. \( c \in \mathbb{R}^E_{\geq 0} \) is the edge capacities.
3. \( A \in \{0,1\}^{E \times P_{s,t}} \) is the incidence matrix defined by

\[
A_{e,p} = \begin{cases} 
1 & \text{if } e \in p \\
0 & \text{if } e \notin p,
\end{cases}
\]

for each edge \( e \in E \) and path \( p \in P_{s,t} \).

Note that there are exponentially many variables in this LP so we could not even write it down in full in polynomial time, let alone apply a black box LP solver. Fortunately there are otherwise to solve the LP, as we will see.

\(^4\)More precisely, they are weakly polynomial time solvable, meaning the running times are polynomial in the bit complexity of the input.
Covering LPs. A covering LP is a linear program of the form
\[
\min \langle c, y \rangle \quad \text{over} \quad y \in \mathbb{R}^m_{\geq 0} \quad \text{s.t.} \quad A^T y \geq b,
\]  
where \( A \in \mathbb{R}^{m \times n}_{\geq 0} \), \( b \in \mathbb{R}^n_{> 0} \), and \( c \in \mathbb{R}^m_{> 0} \). We let \( \text{Opt}(C) \) denote the optimum value of the LP \( (C) \).

The minimum cost metric problem above is our first example of a covering LP. For minimum cost metric, we have one variable/column for each edge, and one row/constraint for each \((s,t)\)-path.

1. \( c \in \mathbb{R}^E_{\geq 0} \) is the edge costs.
2. \( b = 1_{\mathcal{P}_{s,t}} \) is the all-ones vector in \( \mathbb{R}^{\mathcal{P}_{s,t}} \).
3. \( A^T \in \{0,1\}^{\mathcal{P}_{s,t} \times E} \) is the \( \{0,1\} \)-incidence matrix defined by
   \[
   A^T_{p,e} = A_{e,p} = \begin{cases} 
   1 & \text{if } e \in p \\
   0 & \text{if } e \notin p,
   \end{cases}
   \]
   for each edge \( e \in E \) and path \( p \in \mathcal{P}_{s,t} \).

Note that \( A, b, c \) are the same between our two examples.

LP duality. LP duality is about the relationship between the linear programs \( (P) \) and \( (C) \), particular when the matrices and vectors \( A, b, c \) are the same for both problems. In this case \( (P) \) and \( (C) \) are said to be dual to one another.

Suppose we have dual pair of \( (P) \) and \( (C) \); i.e., \( A, b, c \) refer to the same objects in either problem. Let \( x \in \mathbb{R}^n_{\geq 0} \) be any feasible solution to \( (P) \) and let \( y \in \mathbb{R}^n_{\geq 0} \) be any feasible solution to \( (C) \). We have
\[
\langle b, x \rangle \overset{(c)}{\leq} \langle A^T y, x \rangle \overset{(d)}{=} \langle y, Ax \rangle \overset{(e)}{=} \langle y, c \rangle.
\]
Here \( (c) \) is because \( x \geq 0 \) and \( A^T y \geq b \). \( (d) \) is by definition of the transpose. \( (e) \) is because \( y \geq 0 \) and \( Ax \leq c \). Thus, for a packing problem \( (P) \) and a covering problem \( (C) \) linked by duality, we have
\[
\text{Opt}(P) \leq \text{Opt}(C).
\]

If this argument seems familiar, it is because we just saw it for packing and covering paths in section 1.1.1 above.

We ask the same question for packing and covering LP’s as we did for packing and covering paths. When, if ever, is \( \text{Opt}(P) = \text{Opt}(C) \)? The all-important LP duality theorem (here restricted to packing and covering problems) states that in fact they are always equal.

\(^5\)Of course, in \( (C) \), we could have written \( A \) instead of its transpose \( A^T \), and swapped \( b \) and \( c \), which would more closely resemble \( (P) \). It is convenient for the subsequent discussion on LP duality for \( A, b \) and \( c \) to have the same dimensions in \( (P) \) and \( (C) \).
Theorem 1.2 (LP Duality for packing and covering). \( \text{OPT}(P) = \text{OPT}(C) \).

We note that theorem 1.2 holds even if \( A, b, \) and \( c \) have negative coefficients.

We will see that LP duality has important consequences for many combinatorial problems of interest – starting with max flow in the present discussion. Recall that \((\text{Max-Flow})\) is a packing LP and \((\text{Min-Metric})\) is a covering LP. Moreover, they are dual to one another. The LP duality theorem then tells us that \((\text{Max-Flow}) = (\text{Min-Metric})\). As a relaxation, we also have that \((\text{Min-Metric}) \leq (\text{Min-Cut})\). That is:

\[
(\text{Max-Flow}) = (\text{Min-Metric}) \leq (\text{Min-Cut}).
\]

1.3 Max-flow min-cut via LP duality

We now prove the following well-known max-flow min-cut theorem.

Theorem 1.3 (Ford and Fulkerson [FF56] and Menger [Men27]). \((\text{Max-Flow}) = (\text{Min-Cut})\).

Typically this theorem is proven algorithmically the Ford-Fulkerson algorithm. But here we given an alternative proof based on LP duality\(^6\). Having already established that \((\text{Max-Flow}) = (\text{Min-Metric}) \leq (\text{Min-Cut})\), it suffices to prove that \((\text{Min-Cut}) \leq (\text{Min-Metric})\).

Let \( y \in \mathbb{R}^E_{\geq 0} \) be an optimum solution to the minimum cut LP, (1.2). We claim that

\[
\text{Given a fractional min-cut } y, \text{ we can find a discrete } (s,t)-\text{cut } C \subseteq E \text{ with total capacity, } \sum_{e \in C} c(e), \text{ less than or equal to the fractional capacity of } y, \langle c, y \rangle.
\]

This is our first example of rounding a fractional solution to a discrete one.

\(^6\)The author, in a fit of frenetic pandemic energy, once made a video of this proof which is available at https://youtu.be/J4yUdABv1tE.

\(^7\)With very little effort, thanks to LP duality.
Note. Before proceeding, we introduce some standard notation for cuts. For a set of vertices \( S \subset V \), the **directed out-cut of** \( S \), consisting of edges leaving \( S \), is denoted by
\[
\partial^+(S) \overset{\text{def}}{=} \{(u, v) \in E : u \in S, v \notin S\}
\]
The **(directed) in-cut of** \( S \), consisting of edges entering \( S \), is denoted by
\[
\partial^-(S) \overset{\text{def}}{=} \{(u, v) \in E : u \notin S, v \in S\}.
\]

### 1.3.1 Line embeddings and sweep cuts.

Recall that \( y \) also gives a set of edge lengths where the length of the shortest \((s, t)\)-path is 1. We leverage this insight to embed the vertices \( V \) on the real line – assigning values \( \alpha : V \to [0, +\infty) \) – as follows.

For each vertex \( v \), let \( \alpha_v \) be the length of the shortest \( s \to v \) path w.r.t the edge lengths \( y \in \mathbb{R}^n \). We have \( \alpha_s = 0 \) and \( \alpha_t \geq 1 \) because of the covering constraints in (Min-Metric).

Consider the following random cut. We pick a value \( \theta \in (0, 1) \) uniformly at random. Let \( S = \{v : x_v \leq \theta\} \), and let \( \tilde{S} = V \setminus S \). Since \( \alpha_s = 0 \) and \( \alpha_t \geq 1 \), the set of edges from \( S \) to \( \tilde{S} \) is always an \((s, t)\)-cut.

The value of this cut is randomized. Let us bound the cost of the directed cut from \( S \) to \( \tilde{S} \), in expectation. We have
\[
E \left[ \sum_{e \in \partial^+(S)} c(e) \right] \overset{(i)}{=} \sum_{e \in E} c(e) \Pr[e \in \partial^+(S)] \overset{(g)}{=} \sum_{e \in E} c(e) y_e = (\text{Min-Metric}) \quad (1.3)
\]
Here (f) is by linearity of expectation. The key inequality, (g), is based on the shortest path metric.

---

*See Appendix A for background on probability theory.*
For an edge \( e = \{u, v\} \), we have \( e \in \partial^+(S) \) iff \( \alpha_u \leq \theta \leq \alpha_v \), which happens with probability at most \( \alpha_v - \alpha_u \). Consider the shortest path from \( s \) to \( u \) w.r.t. \( y \), which has length \( \alpha_u \). Concatenating the shortest path from \( s \) to \( u \) with the edge \( e \),

\[
\begin{align*}
    s \xrightarrow{\alpha_u} u \xrightarrow{y_e} v,
\end{align*}
\]

gives a walk of length \( \alpha_u + y_e \), hence \( \alpha_v \leq \alpha_u + y_e \).

Consider now the inequality obtained in (1.3),

\[ E \left[ \sum_{e \in \partial^+(S)} c(e) \right] \leq (\text{Min-Metric}). \]

We have generated a randomized (discrete) cut that is on average no worse than the minimum fractional minimum cut. By the probabilistic method, there exists a value \( \theta \) where the \( (s, t) \)-cut has value at most this average. If not, then the average would have to be higher. This establishes the existence of a minimum cut with cost equal to the \( (s, t) \)-minimum cost metric, and establishes the max-flow min-cut theorem. To extract the cut, one can simply scan \( \theta \) over the interval \((0, 1)\) and check all \( n - 1 \) possible cuts. (In fact any \( \theta \in (0, 1) \) will work; see exercise 1.1.)

### 1.4 Exercises

**Exercise 1.1.** Consider the random cut, which was based on a threshold \( \theta \in (0, 1) \) chosen uniformly at random. Show that for all \( \theta \in (0, 1) \), the corresponding cut is a minimum \( \{s, t\}\)-cut.

**Exercise 1.2.** In most textbooks, max flow is presented as the following LP, which in particular has polynomial size in the input graph \( G \).

\[
\begin{align*}
    \text{maximize} \quad & \sum_{e \in \partial^+(s)} z_e - \sum_{e \in \partial^-(s)} z_e \quad \text{over} \quad z : E \rightarrow \mathbb{R}_{\geq 0} \\
    \text{s.t.} \quad & z_e \leq c(e) \quad \text{for all} \quad e \in E \quad \quad \tag{1.4} \\
    \quad & \sum_{e \in \partial^+(v)} z_e = \sum_{e \in \partial^-(v)} z_e \quad \text{for all} \quad v \in V \setminus \{s, t\}.
\end{align*}
\]

The second set of constraints are called flow conservation constraints. Show that the above LP is equivalent to the (fractional path packing version of) \( \text{(Max-Flow)} \) in the following sense.
1. Show that for every (feasible) fractional path packing $P$, there is a feasible solution $z$ to (1.4) with the same objective value.

2. Show that for every feasible solution $z$ to (1.4), there is a feasible path packing $P$ with the same objective value.\footnote{The second problem is trickier than the first. One should be able to prove it using only the ideas and results in this chapter (without retracing the flow algorithms of ensuing chapters).}

1.5 Additional notes and references

In the next few chapters we will develop many algorithms for maximum flow and minimum cut, and there we give many more additional pointers. We refer the reader to Schrijver [Sch03, Chapter 10] for many more references as well as historical notes on the maximum flow problem.
Chapter 2

Augmenting Path Algorithms for Max Flow

In the previous chapter we proved the max-flow min-cut theorem via LP duality and randomized rounding. While the rounding step itself was algorithmically simple, we do not have an algorithm for producing an LP solution to round. Yes, LP’s are polynomial time solvable, but recall that the \((s,t)\)-cut LP is not compact\(^1\).

In this chapter we turn to developing combinatorial algorithms for the max-flow/disjoint-paths problem, and the dual \((s,t)\)-cut. We first consider the uncapacitated, edge-disjoint paths problem for simplicity and discuss the Ford-Fulkerson algorithm, which the reader has likely seen in an introductory algorithms class before. We then discuss strongly polynomial algorithms for maximum flow, working our way through a sequence of successively maximum flow algorithms, all based on and improving the augmenting paths framework of Ford and Fulkerson.

2.1 Ford-Fulkerson and Menger’s Theorem

Consider for simplicity the (uncapacited) edge-disjoint \((s,t)\)-path packing problem, and the dual \((s,t)\)-cut problem.

2.1.1 Greedy heuristics

To start the discussion and build some intuition, we open with some simple greedy algorithms. For path packing, consider the obvious approach of repeatedly find \((s,t)\)-paths, removing (the edges of) each path from the graph and adding it to a growing collection \(P\).

\(^1\)That said, it could be solved by a particular type of LP solver called the ellipsoid method, which we will discuss later in the course.
2. Augmenting Path Algorithms for Max Flow

2.1. Ford-Fulkerson and Menger’s Theorem

Kent Quanrud
Fall 2021

Greedy-Path-Packing($G = (V,E), s,t$)

1. $P \leftarrow \emptyset$
2. While there is a path $p$ in $G$ from $s$ to $t$ // BFS or DFS
   A. Add $p$ to $P$
   B. Remove all the edges in $p$ from the graph
3. Return $P$

An equally simple and natural approach to the $(s,t)$-cut problem is as follows. Start with $C = E$; obviously $C$ is an $(s,t)$-cut. Then repeatedly try to remove edges from $C$ as long as $C$ remains an $(s,t)$-cut.

Greedy-Cut($G = (V,E), s,t$)

1. $C \leftarrow E$
2. While there is an edge $e \in C$ such that $C - e$ is an $(s,t)$-cut
   A. Remove $e$ from $C$
3. Return $C$

Ideally, we would like to show that these algorithms return the optimum solutions of their respective problems. For starters, it is clear from their terminating conditions that they produce locally optimal solutions. greedy-path-packing terminates when no more paths can be added to the current solution. greedy-cut terminates when no more edges can be removed from the current solution. That is:

**Lemma 2.1.** greedy-path-packing returns a maximal $(s,t)$-path packing $P$, and greedy-cut returns a minimal $(s,t)$-cut.

Tautologically, we have

(maximal path packing) ≤ (maximum path packing)

and

(minimal $(s,t)$-cut) ≥ (minimum $(s,t)$-cut).

But these relations are only useful if we can strengthen the inequalities to equalities. Unfortunately, simple counter examples show that they are not equal. Consider the following graph.
Let $s$ be the leftmost vertex and $t$ the rightmost vertex. Clearly the maximum packs two paths from left to right, by going along the top and along the bottom. But a single path using the middle edge also gives a maximal path packing.

Thus a maximal path packing is not a maximum path packing. It is not hard to find simple counter examples that exaggerate the difference in size between a maximal and maximum path packing. It is also easy to come up with counter examples that show that a minimal $(s,t)$-cut is not a minimum $(s,t)$-cut.

The point is that

\[
(\text{maximal path packing}) \neq (\text{maximum path packing})
\]

and

\[
(\text{minimal } (s,t)\text{-cut}) \neq (\text{minimum } (s,t)\text{-cut}).
\]

This is different from, say, spanning trees. There, one can show that any maximal forest is a maximum forest, and that any minimal spanning set of edges is a minimum spanning set. Moreover, the optimal solution in both problems have the same value, meeting at a spanning forest. This makes the spanning tree problems conducive to greedy algorithms where we simply keep adding edges to a forest, or keep removing edges from a spanning set, until we can no longer do so while maintaining feasibility. But such simple greedy algorithms apparently do not work for the $(s,t)$-flow and cut problems.

### 2.1.2 Residual graphs and augmenting paths

Recall our very simple counter example for the greedy path packing approach.

If we remove the selected path (in yellow), the two unused edges are separated and so there is no path remaining. But consider the following “path arithmetic”: 
Call two directed edges **opposite edges** if they have are the reverse of one another. The two paths on the LHS have a pair of opposite edges. We can “uncross” the two paths at the opposite edges, combining the initial part of one path with the remaining part of the other. The result is another path packing except now there are no opposite pairs of edges. This intuition leads to the following observation.

**Lemma 2.2.** Let \( p, q \) be two edge-disjoint \((s,t)\)-paths. Then there are two edge disjoint \((s,t)\)-paths \( p', q' \) with the following properties:

1. \( p' \) and \( q' \) use a subset of the edges in \( p \cup q \).
2. \( p' \) and \( q' \) have no opposite pairs of edges.

The proof is by induction on the number of opposite pairs of edges, and left to the reader.

Knowing now that we can uncross opposite pairs of edges and maintain a path packing, consider again our simple counter example. The problem before is that we used the middle edge going down, leaving a disconnect in the middle of the graph. But suppose we inserted a new edge in the opposite direction, going from bottom to top.

The new \((s,t)\)-path in this augmented, **residual graph** can be uncrossed with the previously selected path to obtain two paths in the original graph. This clever idea leads to the following algorithm, due to Ford and Fulkerson [FF56]. On first pass, the reader should ignore (3) and focus on the path packing portion of the algorithm.
augmenting-paths\((G = (V, E), s, t)\)

\[
\text{// [FF56]}
\]
1. \(P \leftarrow \emptyset\)
2. while there is an \((s, t)\)-path \(p : s \leadsto t\)
   A. remove all of \(p\)’s edges from the graph.
   B. add the reverse of all of \(p\)’s edges to the graph.
   C. uncross \(p\) with previously selected paths, and add \(p\) to \(P\).
3. \(S \leftarrow \{\text{vertices still reachable from } s\}\)
4. return the \((s, t)\)-path packing \(P\) and the \((s, t)\)-cut \(\partial^+(S)\)

A simulation of augmenting-paths on our simple counterexample is given in Figure 2.1. We remark that one can skip steps (1), (2.C), and (3) if the goal is only to compute the size of the maximum path packing.

Next we will argue that augmenting-paths returns an optimum solution. To analyze augmenting-paths we use the following definitions. First, we define the residual graph of a path packing as the graph obtained by removing the paths from the graph, and adding their reverse. The augmenting-paths algorithm maintains a residual graph of the current path-(or walk)-packing at any point in time. The key to the proof is the following observation.

**Observation 2.3.** Let \(S \subset V\) be any set of vertices with \(s \in S\) and \(t \notin T\). Let \(p : s \leadsto t\) be an \((s, t)\)-path. Then \(p\) has exactly one more edge for \(\partial^+(S)\) then from \(\partial^-(S)\). In particular, the residual graph w.r.t \(p\) decreases \(|\partial^+(S)|\) by 1 and increases \(|\partial^-(S)|\) by 1.

The augmenting-paths algorithm terminates only when the residual graph, here denoted \(G'\), has no \((s, t)\)-path. Let \(S\) denote the set of vertices reachable from \(S\) in the final residual graph, and let \(T = V \setminus S\) denote the remaining vertices. Note that \(s \in S\) and \(t \in T\). Below, \(S\) is in green and \(T\) is in red.

---

2Strictly speaking, after uncrossing, we may end up with a collection of walks rather than paths. But any feasible “walk-packing” can be converted into a path-packing by shortcutting any cycles that may arise.
Let \( k = \partial^+(S \mid G) \) number of edges of the out cut of \( S \) in the original input graph. Let us observe the changes to \( \partial^+(S \mid G) \) over the course of the augmenting-paths algorithm. Each iteration, we increase our path packing by one, and the size of \( \partial^+(S) \) decreases by one in the residual graph. Thus the algorithm terminates in exactly \( k \) iterations, with \( |P| = k \). But we already know, from duality, that any path packing has at most \( k \) paths. Thus the path packing \( P \) produced by the augmenting-path algorithm is certifiably optimal. Moreover, since the size of any \((s, t)\)-path packing is a lower bound for the size of any \((s, t)\)-cut, we simultaneously obtain the fact that \( \partial^+(S) \) is a minimum \((s, t)\)-cut.

We have now accomplished two things. First, we have proven a variant of max-flow min-cut for packing edge-disjoint paths. In some sense, since \((\text{Max-Paths}) \leq (\text{Max-Flow})\), this is stronger then our previous theorem, \((\text{Max-Flow}) = (\text{Min-Cut})\). However we only proved it for multi-graphs, which extends to integer capacities (by replacing capacities with parallel edges), but does not apply to general capacities. (More on general capacities in a moment,.) The edge-disjoint version of max-flow min-cut is called Menger’s theorem.

**Theorem 2.4** (Menger’s theorem). *In any directed graph with integer capacities, and...*
2. Augmenting Path Algorithms for Max Flow

2.2. Shortest augmenting paths

fixed source and sink, we have

\[(\text{Max-Paths}) = (\text{Min-Cut}).\]

In particular, there exists an integral maximum flow.

Second, we have obtained an algorithm with the following guarantee.

**Theorem 2.5** ([FF56]). Let \( G = (V,E) \) be a directed graph with \( m \) edges, and let \( s,t \in V \) be distinct vertices. Let \( \lambda \) be the size of the max-flow/min-cut from \( s \) to \( t \). Then augmenting-paths\((G,s,t)\) produces both a maximum packing and a minimum cut in \( O(m\lambda) \) time.

**Proof.** We have already shown the correctness of the algorithm. For the running time, each successive path can be obtained by BFS or DFS. By maintaining the paths as doubly linked lists, it is easy to “uncross” the selected paths in \( O(n) \) time per path. ■

For uncapacitated graphs, \( \lambda \) is at most \( m \), so augmenting-paths is polynomial time. However in the presence of (integer) capacities, \( \lambda \) may be exponentially large, and this running time is no longer satisfactory.

2.2 Shortest augmenting paths

In this section, we strengthen the Ford-Fulkerson augmenting path approach to obtain strongly polynomial time algorithms, even with general capacities. Continuing a theme from the last chapter, the key insight is to focus on shortest paths (here, in the residual graph at each iteration).

To this end, it is helpful to introduce the following terminology. Given a fixed residual graph, we arrange the vertices in layers by their (unweighted, shortest path) distance from \( s \). These layers can be obtained by BFS. An edge \((u,v)\) is called a forwards edge if \( u \) has strictly smaller distance than \( v \); i.e., \( u \)'s layer is one closer to \( s \) than \( v \). A backwards edge is an edge \((u,v)\) where \( v \) has distance less than \( u \). A neutral edge is an edge \((u,v)\) where \( u \) and \( v \) have the same distance from \( s \). For each vertex \( v \), we write \( d(v) \) to denote the distance from \( s \) to \( v \).

The reader should verify for herself the following observations.

**Observation 2.6.**

1. A path from \( s \) is a shortest path iff it only contains forwards edges.

2. Adding a backwards or neutral edge does not decrease any shortest path distances from \( s \).
2. Augmenting Path Algorithms for Max Flow

2.2. Shortest augmenting paths

Figure 2.2: Vertices arranged in BFS layers from $s$. Forward edges are solid; backwards and neutral edges are dashed.

3. Removing an edge does not decrease any shortest path distances from $s$.

4. If $d(t) \geq n$, then there is no $(s,t)$-path, and the maximum $(s,t)$-flow has value 0.

2.2.1 A strongly polynomial time algorithm

We now consider a subtle modification of the Ford-Fulkerson algorithm by Dinic [Din70] and Edmonds and Karp [EK01]. As with Ford-Fulkerson, each iteration selects a path and routes flow along it. The subtle change is that we always select the shortest augmenting path. This strategy has the following nice property.

**Lemma 2.7.** Suppose we route flow along a shortest $(s,t)$-path in the residual graph. Then the shortest path distances from $s$ in the residual graph do not decrease.

**Proof.** Let $p$ denote the shortest $(s,t)$-path along which we augment. All the edges in $p$ are forward edges. When updating the residual graph, suppose we do so via the following two steps, in order.

First, we add all the reverse edges of $p$ to the graph (if not already present). Since each edge in $p$ is a forwards edge, all newly introduced edges are backwards edges, so the distance from $s$ to $t$ does not decrease.

Second, we removing any edges in $p$ whose capacity is saturated by the augmentation. Removing edges from the residual graph does not decrease the distance from $s$ to $t$ either.

**Theorem 2.8** (Dinic [Din70] and Edmonds and Karp [EK01]). The $(s,t)$-maximum flow can be computed in $O(m^2n)$ time.
Proof. Consider the following algorithm, where we start with an empty flow and repeatedly execute the following:

1. Search for the shortest \((s, t)\)-path in the residual graph.
2. If there is such a path, then augment the maximum possible amount of flow along the path, update the residual graph, and repeat.
3. Otherwise return the flow computed so far as the maximum flow.

In short, the algorithm iteratively routes a maximum amount of flow along the shortest \((s, t)\)-path in the residual graph. Each iteration, by maximizing the amount of flow we route, the algorithm saturates at least one edge along the augmenting path. This edge was a forwards edge in the residual graph and it is removed from the residual graph. We charge the iteration to this edge. The removed edge can only return to the residual graph as a backwards edge (in the sense described above), which implies that at least one of its endpoints have increased in distance. Each edge has its endpoints range from 1 to \(n\), so we charge each edge at most \(n\) times. This gives a total of \(O(mn)\) iterations, each of which augments along one path found in \(O(m)\) time.

\[\blacksquare\]

2.2.2 Flow decompositions

The shortest augmenting path algorithm of theorem 2.8 establishes our first strongly polynomial running time. The fact that it only requires \(O(mn)\) iterations then implies that

Any flow decomposes to a fractional path packing of at most \(O(mn)\) paths.

One can do better. Let \(f\) be a fixed \((s, t)\)-flow, treated as a weighted and directed set of edges. Recall the greedy algorithm from section 2.1.1, where we simply remove \((s, t)\)-paths from the graph. We can modify the greedy algorithm so that each time we select an \((s, t)\)-path \(p\), we take as much as of \(p\) as possible (just as in section 2.2.1 above). Now imagine running this modified greedy algorithm on \(f\) itself. Observe that each time we subtract an \((s, t)\)-path from \(f\) (and decreasing the edge weights accordingly), \(f\) is still an \((s, t)\)-flow. By maximizing the quantity we take of \(p\), we also ensure that \(f\) has at least one less (nonzero) edge. Thus at termination \(f\) will have no edges left, and the path packing attained consists of \(m\) distinct paths.

---

\(\text{i.e., let } \gamma \text{ be the smallest residual capacity along the path. Augment } \gamma \text{ units of flow along this path.}\)

\(\text{A previous version of this section misstated that flows decompose to } O(mn) \text{ paths, rather than } O(m).\)
Theorem 2.9. Every \((s,t)\)-flow decomposes to a path packing of \(O(m)\) paths.

2.3 Blocking Flows

If we reflect on the success of the shortest augmenting path algorithm, we see that the key is to focus on the length of the shortest augmenting path. The length of the shortest augmenting \((s,t)\)-path is increasing over time, and once the length exceeds a certain length, we know that there are no augmenting paths and so we have a maximum flow. That is to say that the algorithm makes progress by increasing the length of the shortest augmenting path over time. This motivates the following notion of blocking flows introduced by Dinic [Din70].

Definition 2.10. A \((s,t)\)-blocking flow (or simply a blocking flow) is a maximal (fractional) packing of shortest \((s,t)\)-paths.

An alternative (and perhaps more common) definition of blocking flows is as follows. Given a residual graph, consider the layered subgraph obtained by retaining only the forwards edges (from \(s\)). A blocking flow is a maximum flow in this layered graph.

The key point of blocking flows is that they increase the length of the shortest augmenting path in the residual graph.

Lemma 2.11. Let \(f\) be a maximal packing of shortest paths. Augmenting along \(f\) increases the shortest \((s,t)\)-path distance in the residual graph.

Proof. Lemma 2.7 already implies that the shortest \((s,t)\)-path distance does not decrease; here we are making a stronger claim that the shortest \((s,t)\)-path strictly decreases.

Fix the distances and layers from \(s\) at the point before augmenting along \(f\); for the rest of the proof, we refer to forward, backwards, and neutral edges w/r/t these distances. Let \(k\) denote the length of the shortest \((s,t)\)-path; all the paths in \(f\) have exactly \(k\) edges, and \(t\) is in the \((k+1)\)th layer from \(s\).

Since \(f\) only uses forward edges, augmenting along \(f\) only introduces backwards edges into the graph. If there is a path \(q\) in the residual graph of \(f\) using only forward edges, then some nonzero quantity of \(q\) could have been added to \(f\) – a contradiction to the maximality of \(f\). So every \((s,t)\)-path in the residual graph of \(f\) must use a neutral or backwards edge. But this implies that every such path has strictly more than \(k\) edges. (At least \(k\) forward edges to traverse \(k+1\) layers, plus at least one neutral or backwards edge.)

Lemma 2.12. A maximum flow can be computed via \(O(n)\) blocking flows.
2. Augmenting Path Algorithms for Max Flow

2.3. Blocking Flows

Proof. Each blocking flow increases the distance from \( s \) to \( t \) in the residual graph by 1; once the distance is greater than \( n - 1 \), there is no path from \( s \) to \( t \), and we have computed a maximum flow.

It remains to figure out how to compute blocking flows.

2.3.1 Computing blocking flows in unit capacity graphs

Lemma 2.13. In an uncapacitated flow network, a maximal packing of shortest paths can be found in \( O(m) \) time.

Proof. One first builds out the layered graph consisting of all the forwards edges with a single breadth-first search (BFS) from the source \( s \). Consider the directed acyclic graph (DAG) consisting of only these forwards edges; every \((s, t)\)-path in the layered graph is a shortest \((s, t)\)-path in the input graph.

A simple but inefficient way is to run the greedy algorithm from section 2.1.1 on this graph. Each iteration we search for an \((s, t)\)-path; if we find such a path, we add it to our path packing, and remove its edges from the graph. This produces a maximal path packing.

It is inefficient to traverse any edge \( e \) more than once. Indeed, if we traverse \( e \) and find it does not lead to the sink \( t \), we know that \( e \) will not lead to \( t \) in any future iteration. We should exclude \( e \) from any future search for \((s, t)\)-paths.

Putting this all together, one can compute a maximal packing of \((s, t)\)-paths by a variation of depth-first search in this layered graph. We first modify DFS by marking edges as we traverse them (rather than vertices), ensuring that we only traverse edge once\(^5\). We also maintain parent pointers through the search. Whenever we visit \( t \) in the search, we use the reverse pointers to extract the corresponding \((s, t)\)-path \( p \), adding \( p \) to a growing collection and removing all edges of \( p \) from the path. The running time is bounded by observing that we only traverse each edge once.

Thus, in unit capacity graphs, computing \( O(n) \) blocking flows gives the following running time.


This running time is not actually better than Ford-Fulkerson in simple graphs, since the max flow is at most \( n \) in a simple graph. We will return to this point shortly in section 2.4.

---

\(^5\)With some simple preprocessing, we can produce the next unmarked edge leaving a given vertex in constant time.
2.3.2 Computing blocking flows in capacitated graphs

We now consider blocking flows in capacitated graphs. The algorithm presented here is very similar to the one above simple graphs, but the running time is slower because we may only saturated and remove one edge per augmenting path.

**Lemma 2.15.** In a capacitated flow network, a maximal packing of shortest paths can be found in $O(mn)$ time.

**Proof.** We first build out the layered graph in $O(m)$ time, and henceforth work only in this subgraph graph. We build a maximal fractional path packing by repeating the following modified DFS in this layered graph.

Search for an $(s, t)$-path with DFS. If no path is found, then the paths found so far comprise a maximal path packing. Otherwise, let $p$ denote the $(s, t)$-path found, and let $A$ denote the set of all edges visited in this search. Note that the DFS takes $|A|$ time (which may be less than $m + n$). Let $A' = A \setminus p$ (treating $p$ as a set of edges); we know that there are no $(s, t)$-paths using edges in $A'$. We delete all edges in $A'$ from the graph. We also take the maximum quantity of $p$ as fits in the current capacities, and decrease the capacities along $p$ accordingly. Any saturated edge (and there is always at least one) is also dropped.

Each iteration takes at most $O(|A'| + n)$ time, where $|A'|$ is the set of edges removed in that iteration. Since each edge is removed at most once, the sum of $|A'|$ over all iterations is at most $m$. Meanwhile there are at most $O(m)$ iterations because each iteration removes at least one edge from the graph. This gives a total running time of $O(mn)$.

**Corollary 2.16.** In capacitated graphs, augmenting along blocking flows gives a $O(mn^2)$-time algorithm for max-flow.

This improves the $O(mn^2)$-time algorithm obtained by augmenting shortest paths in section 2.2.1.

2.4 Fewer blocking flows suffice in simple graphs

In section 2.3, we ran blocking flows for $O(n)$ iterations to compute a flow. In uncapacitated graphs, a blocking flow takes $O(m)$ time, which gives a $O(mn)$ running time. For simple graphs\(^6\), Even and Tarjan [ET75] gave a stronger argument that showed that much fewer blocking flows suffice. The key idea is that when the shortest $(s, t)$-path is large enough, so that there are many layers in the layered graph between $s$ and $t$, one of the cuts between layers must be a small cut – certifying that we nearly have a maximum flow. Even and Tarjan [ET75] give two flavors of this argument that give slightly different running times.

---

\(^6\)A **simple graph** is an unweighted/uncapacitated graph with no parallel edges.
Lemma 2.17. Consider a fixed flow $f$. Suppose the shortest augmenting path in the residual flow network has length $k$. Then the residual $(s,t)$-flow is at most $\frac{m}{k}$.

Proof. Consider the maximum flow in the residual graph. Every path has at least $k$ edges. Thus every unit of flow requires a unit of flow from at least $k$ edges. ■

It follows that $O(\sqrt{m})$ blocking flows suffice to obtain a maximum flow.

Theorem 2.18 (Even and Tarjan [ET75]). An integral maximum flow in a simple graph can be found in $O(\sqrt{m})$ blocking flows, hence in $O(m\sqrt{m})$ time.

Proof. After $O(\sqrt{m})$ blocking flows, the residual flow network has flow value at most $\sqrt{m}$. These can be routed by $O(\sqrt{m})$ additional blocking flows. (For the second stage, augmenting paths would work as well.) ■

The second running time of Even and Tarjan [ET75] begins with the following lemma, which replaces lemma 2.17.

Lemma 2.19. Suppose the shortest augmenting path in a flow network has length $k$. The

$$(\max \text{ (s,t)-flow}) \leq O\left(\frac{n^2}{k^2}\right).$$

Proof. Let $V_i$ be the set of vertices at distance $i$ from the source. There are at most $k/2$ values of $i$ where $|V_i| > 2n/k$. By the pigeonhole principle, there are two consecutive indices $i, i+1 < k$ such that both $|V_i|$ and $|V_{i+1}|$ have less than $2n/k$ vertices. There can be at most $O\left(\frac{n^2}{k^2}\right)$ between $V_i$ and $V_{i+1}$, and these edge given an $(s,t)$-cut. See also Figure 2.3. ■

A similar argument as theorem 2.18, except replacing lemma 2.17 with lemma 2.19, shows that that $O\left(n^{2/3}\right)$ blocking flows suffice to obtain a maximum flow. We leave the proof as an exercise to the reader.

Theorem 2.20 (Even and Tarjan [ET75]). In a unit capacity multi-graph, an integral maximum flow can be found in $O\left(n^{2/3}\right)$ blocking flows, hence $O\left(mn^{2/3}\right)$ time.

Remark 2.21. The $O(\sqrt{m})$-iteration bound extends to multigraphs; the $O\left(n^{2/3}\right)$ bound does not.
2.5 Data structures for blocking flows

Let us now return to capacitated graphs. Previously we showed that (essentially) DFS in the layered graph gives a $O(mn)$-time subroutine. There are better approach to computing blocking flows. In fact, one can compute blocking flows in nearly linear time by incorporating some sophisticated data structures into the search.

**Lemma 2.22** (Goldberg and Tarjan [GT90]). In a capacitated flow network, a blocking flow can be found in $O(m \log(n^2/m))$ time.

*Proof references.* Nearly linear running times with improving logarithmic dependencies obtained were obtained by [GN80; GT90; Shi78; Sle80; ST81; ST83]. Here we sketch the approach of Sleator and Tarjan [ST83] based on dynamic trees, to the effect of showing a $O(m \log(n))$ running time. With more care the $\log(n)$ can be improved to $\log(n^2/m)$; we refer the reader to [GT90] for these details.

[link-cut tree](https://en.wikipedia.org/wiki/Link-cut_tree) is based on a dynamic graph data structure called a link-cut tree. We briefly review the link-cut tree data structure, and refer the reader to the original paper [ST85] and lecture notes [Dem12; Qua21d] for further details.

The data structure maintains a collection of disjoint rooted trees, which we will treat as directed trees with edges always directed towards the root. In our case, these trees will always be subtrees of the residual graph and in particular the data structure also stores the capacities of each edge. The collection of trees is modified by the following two operations each in $O(\log(n))$ amortized time.

---

[32] Dem12 also links to a very good recorded lecture.
1. \text{link}(u, v): \text{Given two vertices } u \text{ and } v, \text{ where } u \text{ is the root of its tree, and } v \text{ is a vertex in a different tree from } u, \text{ add a directed edge from } u \text{ to } v \text{ (making } u \text{ a child of } v).\\

2. \text{cut}(e): \text{Delete an edge } e \text{ from its containing tree, splitting the tree into two trees.}\\

For each vertex \( u \), we have the following queries which are executed in the tree containing \( u \), again in \( O(\log n) \) amortized time.\\

3. \text{root}(u): \text{Get the vertex at the root of } u\text{'s tree.}\\

4. \text{min-capacity}(u): \text{Given a vertex } u, \text{ return the edge with minimum capacity among all edges on the } u\text{-to-root path in } u\text{'s tree.}\\

5. \text{decrease-capacity}(u, \delta): \text{Given a vertex } u, \text{ and a value } \delta, \text{ subtract } \delta \text{ from the capacity of every edge on the } u \text{ to } \delta \text{ path.}\\

In general, the link-cut trees allows for values along edges and vertices and efficient aggregate queries and updates along node-to-root paths in the rooted trees. Above, we have defined two such operations (\text{min-capacity} and \text{decrease-capacity}) specifically for computing blocking flows in residual graphs.

Recall the approach in section 2.6. We run DFS in the layered graph, repeatedly finding \((s, t)\)-paths, while discarding parts of the graph that we have finished searching. Suppose we find an \((s, t)\)-path \( p \). Then we decrease the capacities along the path as to saturate the minimum capacity edge \( e \) along the path, removing \( e \) from the search. The approach in section 2.6 starts the searching over from \( s \). To some extent, however, this is a waste of the two partial paths of \( p - e \) (from \( s \) and to \( t \), respectively.) The idea is to store these partial paths in link-cut trees.

In the link-cut trees, \( s \) is always a leaf, and \( t \) is always a root. The path from \( s \) to the root of \( s \)'s tree represents a partial path from \( s \) in the residual graph. The goal is to keep extending this path (at \( s \)'s root) until we reach \( t \). To this end we repeat the following step. We first query the root of \( s \)'s tree – call this vertex \( r \) – and unless \( r \) is equal to \( t \), scan the (unexamined) edges leaving \( r \).

If there is no edge at all, then there is no \((s, t)\)-path via \( r \). We backtrack by deleting \( r \) and calling \text{cut} on the appropriate edge. (Or, if \( r = s \), we finish the search altogether.)

Otherwise, we find a directed edge \((r, u)\), and call \text{link}(r, u). This makes \( s \)'s tree a subtree of \( u \), and \( s \)'s new root becomes the root of \( u \). If we are lucky, the new root is \( t \), but in general we repeat the process.

If \( r \) is equal to \( t \) – that is, \( t \) is the root of \( s \)'s tree – then the \( s \)-to-root path gives the desired \((s, t)\)-path. We query the minimum capacity along the path and decrease all the capacities along the path by the same amount. We cut any edge
whose capacity becomes 0 (identified via $\min$-capacity). We then repeat the process.

We continue the search until $r = s$ and there are no edges leaving $s$ left to explore.

To bound the running time, observe that the search does a constant number of tree operations per edge in the residual graph and per augmenting path obtained by the search. For each edge $e$, we link $e$ as it is searched, and cut $e$ as we either backtrack from $e$ or saturate $e$. (When $e$ cut by saturation, it is also preceded by a $\min$-capacity query that identifies it.) For each augmenting path, we execute a constant number of tree operations (querying $\min$-capacity and decrease-capacity) to update the residual graph. Each augmenting path saturates an edge that is then deleted from the residual graph, so there are at most $m$ augmenting paths.

### 2.6 Short blocking flows suffice for integer capacities (via scaling)

The blocking flows approach, which is based on the length of the shortest $(s,t)$-path in the residual graph, has lead to a faster strongly polynomial time algorithm in section 2.3 and very fast running times in section 2.4 for simple graphs. It is desirable to extend the running times for simple graphs – e.g. $O(mn^{2/3})$ – to capacitated graphs as well.

Unfortunately, the argument for an $O(mn^{2/3})$ running time in section 2.4 does not extend to capacitated graphs, or even to multigraphs. It is easy to pinpoint where the argument breaks. The $O(n^{2/3})$ iteration count comes from observing that, when the augmenting paths are sufficiently long, there are two consecutive layers in the distance-layered graph that both have a small number of vertices. Since there are at most one or two edges between any two vertices, the capacity of this cut is at most the product of the number of vertices in the layers. For capacitated graphs, each of these edges may have large capacities. While the number of edges across the cut is still limited, the capacity of the cut is unbounded.

On the other hand, if all the capacities were bounded by a constant, then the argument would go through with some loss proportional to the value of the constant. This motivates the following idea by Goldberg and Rao [GR98]. At any point in time, divide the edges into two classes. We have big edges, with capacity at least $2B$ for a fixed parameter $B > 0$, and small edges, with capacity less than $2B$. When computing shortest paths from $s$, we don’t count the big edges in the length; i.e., big edges are given edge-length 0. This forces all the big edges to be either neutral or backward edges and excludes them from any of the layered
2. Augmenting Path Algorithms for Max Flow

2.6. Short blocking flows suffice for integer capacities (via scaling)

(s,t)-cuts. Now, if the length of the shortest augmenting path with respect to these {0,1}-edge lengths is long, and we repeat the same argument, we will again find an (s,t)-cut between layers with a small number of edges. This time, however, all the edges in the cut have their capacity bounded above by B. Thus we arrive at the same bounds as in section 2.4 except with some dependency on B. This bound is much more useful than before.

Thus the high-level approach is to reduce to the simple / bounded-capacity setting by applying a threshold B. There are still many issues to iron out. A simple obstacle is that there may be length-0 paths consisting of only big edges. Fortunately, in this case, we can then route B units of flow along any such path (which is a lot). But more generally, the presence of these length 0 edges forces us to revisit some of the basic arguments underlying blocking flows. In particular, as presently constructed, it is not necessarily true that the distances are always increasing, because a small edge can become big and its length then decreases. This will be handled by making an exception for a class of “special edges”.

We first formalize the (initial) length function ℓ.

**Definition 2.23.** Let G = (V,E) be a directed graph with edge capacities c : E → R>0. Let s ∈ V be a fixed source vertex and let t ∈ V be a fixed sink vertex. For a fixed threshold B > 0, we define a length function ℓ : E → {0,1} indicating whether an edge has capacity < 2B, as follows.

\[ \ell(e) = \begin{cases} 1 & \text{if } c(e) < 2B \\ 0 & \text{otherwise.} \end{cases} \]

This length function allows us to extend the Even-Tarjan layered-cut argument as there are no length-0 edges in these cuts.

**Lemma 2.24.** Let k be the length of the shortest (s,t)-path w/r/t ℓ. Then

\[ (\max (s,t)\text{-flow}) \leq O\left(B \min \left\{ \frac{m}{k}, \frac{n^2}{k^2} \right\} \right) \]

**Proof.** As in the unit capacity case, we can argue that there is an (s,t)-cut between two layers of the layered graph that have less than \(O(m/k)\) and \(O(n^2/k^2)\) edges. Each of these edges have length 1, and therefore capacity ≤ 2B. See Figure 2.4.

We now design an algorithm that routes flow iteratively. The goal in each iteration is to either:

1. Augment B units of flow.
2. Augmenting Path Algorithms for Max Flow

2.6. Short blocking flows suffice for integer capacities (via scaling)

Figure 2.4: An \((s,t)\)-cut between two layers with edges of capacity less than \(2B\).

2. Augment a blocking flow.

To this end, one first contracts the strongly connected components of length 0 edges, and finds a blocking flow in the contracted graph. If the flow has value \(\geq B\), then we reduce it to \(B\). We then uncontract the graph which breaks the flow at the strongly connected components. Since each contracted edge has capacity \(\geq B\) (and in fact, \(\geq 2B\)), and the flow has size \(\leq B\), we can route flow through each strongly connected component and fix the flow.

If the flow has value \(\geq B\), then we are happy to have routed a lot of flow. Otherwise, if the flow has value \(< B\), then we want to argue that the length of the shortest \((s,t)\) has strictly increased. However, there is an issue where we may decrease the lengths of some edges to 0, when their capacity increases to \(\geq 2B\). To anticipate this exception, [GR98] introduced the notion of special edges.

**Definition 2.25.** An edge \(e = (u,v)\) is special if it is a neutral edge \((d(u) = d(v))\) with capacity \(\geq B\).

The reason that special edges excludes forwards and backwards edges is because (a) forward edges will never have their capacity increased, and (b) backward edges will not effect the shortest paths. Neutral edges are the only real liability.

Next we observe that decreasing the length of special edges is not corrupt the shortest paths.

**Lemma 2.26.** Suppose we decrease the length of a special edge to 0. Then the shortest path distances from \(s\) remain the same.

**Proof.** The endpoints already had the same distance. ■

\(^8\)Each strongly connected component of length-0 edges is contained in the same layer – why?
Let $\ell_0 : E \to \{0, 1\}$ denote the set of edge lengths obtained from $\ell$ by setting length of each special edge to 0. By the preceding lemma, $\ell$ and $\ell_0$ induce the same layered graph.

### 2.6.1 The blocking-flow-like subroutine

Overall, the algorithm is based on the following procedure to identify either a blocking flow or a flow of size $B$.

1. Compute the shortest paths from $s$ w/r/t the edge lengths $\ell : E \to \{0, 1\}$ defined above.
2. Reduce the length of all special edges to 0.
3. Contract all strongly connected components of length 0 edges.
4. Find a blocking flow in the contracted graph.
5. Extract either a flow of size $B$ or a blocking flow in the original graph, and route it.

**Lemma 2.27.** Any flow of size $\leq B$ in the contracted graph is a feasible flow in the original flow graph. Moreover, the corresponding flow in the original graph can be found in linear time.

**Proof.** Feasibly follows from the fact that all the contracted edges have capacity $\geq B$. To route though the strongly connected components in linear time, see [EH02; GR98; HT08].

Before routing, the shortest path lengths w/r/t $\ell$, and the shortest path lengths w/r/t $\ell_0$, are equal. In the following discussion, where we evaluate the shortest paths after routing, we specifically mean w/r/t $\ell$, with the adjustments for special edges discarded.

The following observations revise those of section 2.2.1. Here we fix the layers w/r/t $\ell$ before augmenting, and refer to backwards, neutral, and forwards edges relative to these layers.
Observation 2.28.

1. A shortest path from $s$ w/r/t $\ell$ consists of only forward edges of length 1 and neutral edges of length 0.

2. Routing along a shortest path from $s$ introduces only backwards edges and neutral edges (possibly of length 0).

3. Introducing a backwards or neutral edge (of any length) does not decrease the shortest path distance.

Lemma 2.29. Routing $\leq B$ units of flow along a collection of shortest $(s,t)$-paths (w/r/t $\ell_0$) does not decrease any shortest path distances from $s$.

Proof. Here the claim goes beyond the simpler setting of uniform edge lengths, because routing flow along an edge can also reduce the length of the reverse edge.

Fix the distances and edge lengths to the point before we route the flow. First, routing the flow does not introduce any forwards edges. Second, consider an edge $e = (u,v)$ with length 1 whose length would reduce to length 0 after routing the flow. This can only occur if we are routing so much flow on the reverse edge $(v,u)$ that the capacity of $(u,v)$ increases to over $2B$. But $(v,u)$ is either a neutral or forwards edge, so $d(u) = d(v)$. Thus $(u,v)$ is either a backwards or neutral edge, so the length of 0 does not decrease the shortest paths. ■

Lemma 2.30. Suppose $f$ is a maximal packing of shortest $(s,t)$-paths with total size $< B$. Then augmenting along $f$ increases the shortest path length (w/r/t $\ell$) by at least 1.

Proof. Fix the layers and edges lengths to the point before we route the flow, and let $k$ denote the length of the shortest paths. Since we are only interested in the effect of $f$ on the residual capacities, it suffices to assume that $f$ is un-crossed; i.e., $f$ is only using one direction of every opposite pair of edges.

Now, when we route the flow, updating the residual graph consists of increasing some capacities (and introducing edges) and decreasing other capacities (and deleting edges). It suffices to show that, in the “semi-residual-graph” obtained by only increasing capacities and introducing edges, $f$ is still a maximal packing of shortest $(s,t)$-paths.

Augmenting the path packing does not introduce any forwards edges as it only uses forwards and neutral edges. It also does not introduce any length 0 neutral edges that were not already special (and already length 0)\(^9\).

Now, if $f$ was not also maximal in the $(s,t)$-path, then there is an $(s,t)$-path $p$ of length $k$ of which some nonzero quantity can be added to $f$. Such a path

\(^9\)This is where we use the special edges
must be composed of neutral length-0 and \( k \) forward edges. We want to argue that some quantity \( p \) could also have been added to \( f \) in the original graph, contradicting the maximality of edges. To this end, for each edge \( e \in p \), we show that \( e \) has some leftover capacity. We have two cases.

1. If \( e \) is a forwards edge, then its capacity in the semi-residual-graph is no greater than in the original graph. Thus if \( e \) has extra capacity after \( f \) in the semi-residual graph, then it also.

2. If \( e \) is a length-0 neutral edge, then its capacity in the original graph is at least least \( B \). Since \( f \) has size less than \( B \), so there must be additional capacity in \( e \) after packing \( f \).

This shows that some quantity of \( p \) could have been added to \( f \) in the input graph as well, a contradiction. \( \square \)

### 2.6.2 Putting it all together

It remains to employ the thresholded blocking flow subroutine in an overall algorithm for maximum flow. One decision is where to set the value of \( B \). Here we introduce scaling (loosely defined): we choose \( B \) as a function of some upper bound \( F \) on the value of the maximum flow. Rather than solving the maximum flow problem all at once, we argue that a limited number of iterations are required to guarantee that the remaining maximum flow is at most a constant factor smaller than \( F \). We decrease \( F \) and \( B \) accordingly, and repeat.

**Lemma 2.31.** Let \( F \) be an upper bound on the maximum flow. Let \( k = O(1) \min \{ \sqrt{m}, n^{2/3} \} \). Let \( B = F / 4k \). After \( O(k) \) iterations of either a blocking flow or a flow of size \( B \), the remaining maximum flow is at most some constant factor less than \( F \).

Iterating the above lemma leads to the following overall running time. Here we start with an obvious upper bound \( F \), such as the sum of all capacities. It takes \( O(\min\{ \sqrt{m}, n^{2/3} \}) \) iterations to decrease \( F \) by a constant factor, and we repeat this until the upper bound is less than one. If the capacities were integral, then this implies that we have a maximum flow.

**Theorem 2.32.** For polynomially bounded integer capacities, the maximum flow can be computed in \( \tilde{O}(\sqrt{m}) \) or \( \tilde{O}(n^{2/3}) \) blocking flow computations, hence in \( \tilde{O}(m\sqrt{m}) \) or \( \tilde{O}(mn^{2/3}) \) time.

### 2.7 Exercises
Exercise 2.1.  
1. Prove the following: Given any \((s,t)\)-flow problem with max flow value \(\lambda > 0\), there exists an \((s,t)\)-path where the minimum capacity edge is at least \(\lambda/m\).

2. Describe a \(O(m \log n)\) algorithm to find the path described above.

3. Based on the two parts above, design and analyze an \((s,t)\)-max flow algorithm that runs in \(O(m^2 \log(\lambda))\) time for integer capacities, where \(\lambda\) denotes the value of the maximum flow. (This is a weakly polynomial running time.)

Exercise 2.2. Let \(G = (V,E)\) be a bipartite graph where the two sides have \(n_1\) and \(n_2\) vertices. Design and analyze a strongly polynomial time maximum flow algorithm that runs in \(O(m \min\{n_1, n_2\} \log(n))\) time on such a graph.

Exercise 2.3. Recall that the maximum vertex-disjoint \((s,t)\)-path problem asks for a maximum packing of \((s,t)\)-paths whose interior vertices are disjoint. (Of course this probably can be solved by a well-known reduction to maximum flow.)

Design and analyze an algorithm, based on the ideas in section 2.4, that solves the maximum vertex-disjoint \((s,t)\)-path problem in \(O(m \sqrt{n})\) time.

2.8 Additional notes and references

Some of these algorithms (especially Ford-Fulkerson and shortest augmenting paths) are covered in most introductory textbooks on algorithms. The author has also written lecture notes [Qua21a; Qua21b] on Ford-Fulkerson and Edmonds-Karp in previous classes (which have been adapted here). A recent book by Williamson [Wil19] (freely available online) covers all of the algorithms presented in this chapter. Many of the algorithms are in the book by Ahuja, Magnanti, and Orlin [AMO93]. See also [Scho3, Chapter 9, 10]. For blocking flows, we also recommend the lecture notes by Karger [Kar06].

Goldberg [Golo8] has some helpful slides on the Goldberg-Rao algorithm, in which he comments that despite the good worst-case running time, the Goldberg-Rao algorithm is “not as robust as push-relabel in practice”. “Push-relabel” is  

\textit{Update, September 11:} An earlier version of this problem instead asked for \(\tilde{O}(mn^2 \log(\lambda))\) running time; here we strengthen the claim to \(\tilde{O}(m^2 \log(\lambda))\). We point out that the section on flow decomposition, section 2.2.2, has also been revised.

\textit{Even if this subroutine eludes you, you can black box it for the next part.}

\textit{A possibly helpful bit of math: for (small) \(\epsilon > 0\), \(\log_{1+\epsilon}(x) \leq O(\log(x)/\epsilon)\) is a good approximation (which you may want to verify for yourself).}
a very different type of max-flow algorithm that we discuss next chapter; the relevant point here is that the worst case running time of push-relabel (about $\tilde{O}(mn)$) is not as good as Goldberg-Rao (for polynomial capacities).
Chapter 3

Push-Relabel and Directed Minimum Cut

We begin the chapter by discussing and analyzing multiple versions of the push-relabel max flow algorithm, a different and (in a sense) dual approach to the augmenting path algorithms from last chapter. We then transition to the directed minimum cut problem, starting with the algorithm of Hao and Orlin [HO94] which adapts the push-relabel algorithms to directed minimum cut.

Throughout this section, let $G = (V, E)$ be a directed graph and let $c : E \to \mathbb{R}_{>0}$ be a fixed set of capacities.

3.1 Push-Relabel Max Flow

The push-relabel framework for maximum flow was first introduced by Goldberg [Gol85] and further developed by Goldberg and Tarjan [GT88]. This approach departs from the Ford-Fulkerson framework in the sense that it does not augment along paths and may not obtain a feasible flow until the very end. The two main components of the algorithm are a preflow and a set of vertex labels, which are relaxations of flows and distance layers, respectively. These relaxations (defined below) are still similar enough to flows and distance labels to allow us to define residual graphs and forwards/backwards/neutral edges, and similar to blocking flows, one can explore many different interesting ideas within this framework.

3.1.1 Preflows

An $(s,t)$-preflow is a relaxation of an $(s,t)$-flow where non-terminals are allowed to have excess flow. Formally, it is a vector $f : E \to \mathbb{R}_{\geq 0}$ subject to the following invariants.
1. **Capacity constraints:** For all edges $e \in E$, the flow on $e$ is at most the capacity:

$$0 \leq f(e) \leq c(e).$$

2. **One-sided conservation constraints.** For all non-terminals $v \in V - \{s,t\}$, the total flow entering a vertex is at least the total flow leaving it:

$$\sum_{e \in \partial^{-}(v)} f(u,v) \geq \sum_{e \in \partial^{+}(v)} f(v,u).$$

The (nonnegative) difference between the amount of flow entering a (non-terminal) vertex $v$ and the amount of flow leaving it is called the **excess** or the **net flow** at $v$. For ease of notation, we denote it as

$$\hat{f}(v) \overset{\text{def}}{=} \sum_{e \in \partial^{-}(v)} f(u,v) - \sum_{e \in \partial^{+}(v)} f(v,u).$$

We define the residual graph $G_f = (V, E_f)$ of a preflow $f$ in the same way as for flows: decreasing capacities by the amount of flow on the edge; increasing capacities by the amount of flow on the opposite edge.

**Lemma 3.1.** Let $f$ be a preflow. Then $f$ decomposes to a path packing that contains $\hat{f}(v)$ units of fractional $(s,v)$-paths for every $v \in V - \{s\}$. In particular, $f$ contains an $(s,t)$-flow of size $\hat{f}(t)$.

**Proof.** Create a new auxiliary vertex $t'$, and connect every vertex with positive excess (include the sink $t$) to $t'$ with an edge with infinite capacity. Consider the $(s,t')$-flow $f'$ where we send all the positive excess to $t'$ via the auxiliary edges. This flow decomposes to a fractional $(s,t')$-path packing that includes $\hat{f}(v)$ paths that go through the auxiliary $(v,t')$ edge. Removing this last edge gives the desired path decomposition. ■

### 3.1.2 Labels

The **push-relabel framework**\(^{1}\) maintains a preflow $f$ and a set of **vertex levels** $\ell : V \rightarrow \mathbb{Z}_{\geq 0}$ such that:

(I) **The sink has level 0 and the source has level $n$:**

$$\ell(t) = 0 \text{ and } \ell(s) = n.$$

---

\(^{1}\)Frank [Fra11] presents a slight variation of this, where $s$ may have a level $\leq n$, which is particularly convenient for the Hao and Orlin [HO94] algorithm discussed later. The most important features are that the sink is at level 0, and each edge in the residual graph goes down at most one level.
(II) Every residual edge goes down (towards \( t \)) by at most one level:

\[
\ell(v) \geq \ell(u) - 1 \text{ for all } (u, v) \in E_f.
\]

Initially the algorithm will have \( s \) at level \( n \) (as required) and all other vertices at level 0:

\[
\ell(v) = \begin{cases} 
  n & \text{if } v = s \\
  0 & \text{otherwise}.
\end{cases}
\]

The initial preflow \( f \) starts with an empty flow and then saturates all the edges leaving the source \( s \):

\[
f(u, v) = \begin{cases} 
  c(u, v) & \text{if } u = s \\
  0 & \text{otherwise}.
\end{cases}
\]

In particular, this removes all edges leaving \( s \) from the residual graph so that we satisfy (II).

Invariant (II) can be interpreted as a relaxation of distances in the following senses.

**Lemma 3.2.** For all \( x, y \in V \),

\[
\ell(x) - \ell(y) \leq d(x, y),
\]

where \( d(x, y) \) is the (unweighted) distance from \( x \) to \( y \) in the residual graph.

**Proof.** Suppose there is an \((x, y)\)-path of \( k \) edges. Each of these edge in the path can go down at most one level. \( \square \)

By lemma 3.2 and the fact that \( \ell(t) = 0, \ell(v) \) is a lower bound on the distance from \( v \) to the sink \( t \). In particular, we have \( d(s, t) \geq \ell(s) = n: \text{there is no } (s,t)\text{-path in the residual graph of } f \). In this sense push-relabel is dual to augmenting path algorithms – the preflow \( f \) always induces an \((s, t)\)-cut (via its residual graph). But because \( f \) is not a feasible flow, we cannot certify this cut as a minimum \((s, t)\)-cut.

By the same token, if \( f \) is also a flow (with no non-terminals with positive excess) then it is immediately a maximum flow. Additionally, the levels \( \ell \) will induce a minimum \((s, t)\)-cut, as follows.

**Lemma 3.3.** Let \( f \) be an \((s, t)\)-preflow and \( \ell \) a set of levels satisfying the invariants (I) and (II). Suppose \( f \) is also an \((s, t)\)-flow. Then \( f \) is a maximum \((s, t)\)-flow. Moreover, there is an index \( i \in \{1, \ldots, n-1\} \) such that for \( U = \{v : \ell(e) > i\} \), \( \partial^+(U) \) induces a minimum \((s, t)\)-cut.
Proof. As discussed, \( f \) is maximum because there is no \((s,t)\)-path in the residual graph. To extract the minimum \((s,t)\)-cut from \( \ell \), observe that there exists an index \( i \in \{1, \ldots, n - 1\} \) for which there are no vertices of level \( i \). Let \( U = \{v : \ell(e) > i\} \); since the \( i \)th level is empty, we have \( \bar{U} = \{v : \ell(e) < i\} \). \( U \) contains \( s \) and \( \bar{U} \) contains \( t \), so \( \partial^+(U) \) represents an \((s,t)\)-cut in the input graph. By invariant (II), there are no edges from \( U \) to \( \bar{U} \) in the residual graph of \( f \). This implies that in the input graph, \( \partial^+(U) \) has capacity equal to the size of \( f \). 

\[ \square \]

**Forward, backwards, and neutral edges.** While \( \ell : V \rightarrow \mathbb{Z}_{\geq 0} \) are not actually distance labels, invariants (I) and (II) install just enough structure \( \ell \) to serve the same role (as we have already seen).

We say that a residual edge \((u, v) \in E_f\) is **forward** if the above inequality is tight; that is,

\[ \ell(v) = \ell(u) - 1. \]

Likewise we have a **neutral** edge when \( \ell(u) = \ell(v) \) and a backwards edge when \( \ell(v) > \ell(u) \).

We say that a vertex \( v \) “has a forward edge” if in particular there is a forward edge starting from \( v \).

3.1.3 The generic push-relabel algorithm and analysis

There are many different push-relabel algorithms but they generally follow the same framework consisting of two types of operations, called **push** and **relabel**. Within this framework that are different strategies for employing these operations, some of which we discuss in greater detail below.

Recall that the only difference between a flow and a preflow is that a preflow allows intermediate vertices to carry positive excess. In fact, by lemma 3.3, this is the only difference between a preflow and a **max** flow. Call a non-terminal vertex **active** if \( \hat{f}(v) > 0 \). Loosely speaking, push-relabel algorithms try to route the excess from active vertices – one vertex at a time – towards the sink \( t \) or the source \( s \). (The latter implies there there was too much flow out for all of it to be routed to \( t \).) Given an active vertex \( v \), the algorithm executes one of the following two local operations at \( v \).

1. We **push** flow along a forward edge \( e \) leaving \( v \). Usually\(^2\) we push as much flow as possible subject to the preflow constraints. This quantity is the minimum of the residual capacity of \( e \) and the excess at \( v \). It is helpful in the analysis to classify each push as follows.

\(^2\)Some scaling algorithms do not necessarily push the maximum amount.
A saturating push is a push that uses up all of the capacity of the edge.

(b) A non-saturating push is a push that does not use up all of the capacity of the edge, and (necessarily) uses all of the excess at the vertex.

2. We relabel the level of \( v \) to the next level. To preserve invariant (II), this is only permitted when \( v \) has no forward edges.

The algorithm repeats the operations above until there are no more active vertices. At that point, by lemma 3.3, \( f \) is a maximum flow.

It is not at all obvious that this algorithm should terminate. At a high level, we want to extinguish all of the active vertices. But pushing flow out of one active vertex \( v \), and as to deactivate \( v \), may activate many more vertices one level below! It’s not clear that we’re making much progress.

We will analyze the push-relabel framework by bounding each of the above operations – saturating pushes, non-saturating pushes, and relabelings – separately. Besides these “basic operations”, one needs to account for the running time overhead for selecting a tight vertex, and identifying a forward edge from a given vertex, and so forth. But in most cases, it is easy to see how to use some simple data structures to facilitate the operations (e.g., tracking the active vertices in a list), and the bottleneck will come the total number of basic operations.

Relabelings.

**Lemma 3.4.** For each vertex \( v \), \( \ell(v) \) is nonnegative, nondecreasing, and bounded above \( 2n - 1 \). Consequently, each vertex is relabeled at most \( 2n - 1 \) times and there are at most \( O(n^2) \) relabelings in all.

*Proof.* Indeed, if \( v \) is active, then there must be a path in the residual graph from \( v \) to \( s \). But the level of \( s \) is fixed at \( n \). So every active vertex has level \( \leq \ell(s) + n - 1 = 2n - 1 \). Moreover, only active vertices get relabeled. ■

Saturating pushes.

**Lemma 3.5.** There are at most \( n - 1 \) saturating pushes to any fixed edge. Therefore there are at most \( O(mn) \) saturating pushes total.

*Proof.* When an edge \( e = (u,v) \) is removed from the residual graph, we have \( \ell(v) < \ell(u) \). The edge \( e \) can reappear after pushing along the opposite edge \( (v,u) \), which requires \( \ell(u) < \ell(v) \). Since levels are monotonically increasing, this requires \( \ell(u) \) to have increased by at least two. But \( \ell(u) \) increases at most \( 2n - 1 \) times in all. ■
Non-saturating pushes.

The trickiest to analyze is non-saturating pushes.

**Lemma 3.6.** There are at most $O(mn^2)$ non-saturating pushes.

**Proof.** Define a potential function by

$$
\Phi = \sum_{v \text{ active}} \ell(v).
$$

Each non-saturating push decreases $\Phi$ by one. A saturating push increases $\Phi$ by at most $O(n)$. A relabel increases $\Phi$ by one.

Since there are at most $O(mn)$ saturating pushes, and $O(n^2)$ relabels, the total increase to $\Phi$ is $O(mn^2)$. Moreover, $\Phi$ is initially zero, and always nonnegative. Charging each non-saturating push to a decrease in $\Phi$, we conclude that there are at most $O(mn^2)$ non-saturating pushes. ■

Thus push-relabel requires at most $O(mn^2)$ of the push/relabel operations. With some minor bookkeeping with simple data structures, it is easy to implement the algorithm in $O(mn^2)$ time.

Let us highlight the point that the bottleneck is non-saturating pushes. The relabelings and saturating pushes – summing to $O(mn)$ total operations – are actually very fast. We now explore different variations of push-relabel that obtain better running times; each of these strategies are designed to limit the number of non-saturating pushes.

### 3.2 Top-down push-relabel

Section 3.1.3 gave a very general description and analysis of the push-relabel algorithm. We introduced the two push and relabel operations – with pushes classified saturating or non-saturating – and showed that any algorithm following these operations makes $O(mn^2)$ total operations. More specifically, any push-relabel algorithm makes at most $O(mn^2)$ non-saturating pushes, which is the bottleneck. We also showed that there are at most $O(n^2)$ relabels and $O(mn)$ saturating pushes.

There are many possible strategies within this framework, and perhaps different approaches can lead to different performance in practice, or better upper bounds (especially for non-saturating pushes). The main decision is which active vertex to select at a given moment in time. Here are a few possible examples.

1. Select the active vertex $v$ with highest label $\ell(v)$.

2. Select the active vertex $v$ with lowest label $\ell(v)$. 
3. Select active vertices \( v \) in FIFO order (e.g., add vertices to a queue as they become active).

4. Select active vertices \( v \) in LIFO order (e.g., add vertices to a stack as they become active).

5. Select the active vertex \( v \) with greatest excess \( \hat{f}(v) \).

6. Select the active vertex \( v \) with lowest excess \( \hat{f}(v) \).

7. Select an active vertex \( v \) uniformly at random.

In this section we consider the first strategy, always addressing the active vertex of highest level.

**Lemma 3.7.** In top-down push-relabel, there are at most \( O(n^2) \) non-saturating pushes from any fixed vertex \( v \). Therefore there are at most \( O(n^3) \) non-saturating pushes in all.

**Proof.** Suppose we make a non-saturating push from a vertex \( v \), making \( v \) inactive. Since \( v \) was the highest level active vertex, there is no way for \( v \) to receive flow and become active without some vertex of \( v \) being promoted to a higher label. Each vertex gets promoted at most \( O(n) \) times. ■

The lemma above reduces the number of We leave it to the reader to show that the algorithm can also be implemented in \( O(n^3) \) time.

**Theorem 3.8.** There is a highest-level push-relabel algorithm that runs in \( O(n^3) \) time.

\( O(n^3) \) is already better than our best strongly polynomial algorithm in chapter 2 among those that did not leverage sophisticated data structures. We note that the FIFO strategy also obtains a \( O(n^3) \)-running time; we leave the proof to the reader as exercise 3.1.

### 3.2.1 \( O(n^2\sqrt{m}) \) non-saturating pushes

Cheriyan and Maheshwari [CM89] and Tunçel [Tun94] showed that for top-down push-relabel, one can obtain a better upper bound of \( O(n^2\sqrt{m}) \) non-saturating pushes.

**Theorem 3.9** (Cheriyan and Maheshwari [CM89] and Tunçel [Tun94]). The highest-level push-relabel algorithm (appropriately implemented) runs in \( O(n^2\sqrt{m}) \) time.

**Proof.** The following argument is due to Cheriyan and Mehlhorn [CM99]. We restrict our attention to showing that the highest-level rule makes at most \( O(n^2\sqrt{m}) \) non-saturating pushes; we leave it to the reader to devise a \( O(n^2\sqrt{m}) \) implementation.
Define a potential $\Phi$ by

$$\Phi = \sum_{v \text{ active}} |w : \ell(w) \leq \ell(v)|.$$  

This potential counts, for every active vertex $v$, the total number of vertices “below” or at the same level as $v$.

$\Phi$ is initially 0. A relabeling or a saturated push can increase $\Phi$ by at most $n$. So the total increase to $\Phi$ is at most $O(mn^2)$. On the other hand, a saturating push at a vertex $v$ de-activates $v$ and decreases $\Phi$, by at least the number of vertices $v$ at $v$’s level, $|w : \ell(w) = \ell(v)|$. (In the worst case, the non-saturating push from $v$ activates a new vertex $w$. Since $\ell(w) = \ell(v) - 1$, the difference in contribution between $v$ and $w$ is precisely the number of vertices at $v$’s level.)

Call a level “small” if it has $< k$ vertices, and “big” if it has $\geq k$ vertices. Call a push “small” if it is from a vertex on a small level, and “big” if it is from a vertex at a big level.

Now, we always treat active vertices at the highest level. Let a “phase” be defined as a consecutive sequence of basic operations made by the algorithm on active vertices from the same level. Phases are terminated by increasing the highest active level (by relabeling an active vertex), or by decreasing the highest active level (by deactivating the last active vertex with a push). Now there are only $O(n^2)$ relabels so we only increase the highest level $O(n^2)$ times. Each decrease can be attributed to a previous increase$^3$. So there are at most $O(n^2)$ phases.

Let a $k \in \mathbb{N}$ be a parameter to be determined. Call a non-saturating push “small” if the highest active level has $\leq k$ active vertices, and “big” if the highest active level has $> k$ active vertices. There are most $k$ small non-saturating pushes per phase, hence $O(n^2k)$ small non-saturating pushes overall. Meanwhile each big push decreases $\Phi$ by at least $O(k)$, so there are at least $O(mn^2/k)$ big pushes. Balancing terms at $k = \sqrt{m}$, we conclude that there are at most $O(n^2\sqrt{m})$ non-saturating pushes.

$^3$Like an elevator.

### 3.3 Accelerating push-relabel with data structures

Goldberg and Tarjan [GT88] showed how to use dynamic data structures to accelerate the push-relabel running time algorithm. Goldberg and Tarjan specifically consider the FIFO strategy for selecting active vertices. However, it appears most strategies would work; the key idea is to use data structures drastically cut the computational work generated by non-saturating pushes. In fact, their argument does not actually decrease the total number of non-saturating pushes; rather,
most of the non-saturating pushes are simulated efficiently by the data structures (similar to blocking flows).

**Theorem 3.10.** With link-cut trees, the push-relabel algorithm can be made to run in $O(mn \log(n^2/m))$ time.

**Proof.** For ease of exposition we will only show how to obtain an $O(mn \log(n^2/m))$ running time. The $O(mn \log(n^2/m))$ comes from a more careful application of these techniques. We refer the reader to Goldberg and Tarjan [GT88] or Goldberg’s thesis [Gol87, Chapter 1] for these details.

We will use link-cut trees to make non-saturating pushes more efficient. Before discussing push-relabel, it is helpful to briefly recall the use of link-cut trees in accelerating blocking flows. The inspiration there was that whenever we find and augment along an $(s, t)$, and remove a bottleneck edge from the residual graph, the remaining parts of the paths can still be useful. Rather than discard this progress, we maintain the explored edges in a dynamic tree. We use the root operation to quickly identify where the search left off, link to add an edge to the search, aggregate queries along the $s$-to-root path to identify bottleneck edges and route flow, and cut to delete edges as they are saturated and removed from the residual graph. See section 2.5 for more details and explanation of the dynamic tree operations.

Let us return to push-relabel and first give some high-level motivation. We generally prefer to make saturating pushes or relabels as these already have good upper bounds. Of course only a non-saturating push may be available. When we execute a non-saturating push from (say) $u$ to $v$, $v$ becomes active, and we can keep searching at $v$ for a saturating push or a relabeling. If indeed we can make then relabel $v$, or make a saturating push at $v$, then we can charge the preceding non-saturating push to this operation. But perhaps there is only a non-saturating push at $v$, and so we continue the search until we find something other than a non-saturating push. In this manner we find a path of non-saturating pushes terminated by either a saturating push or a relabeling. Of course the path may be long with many non-saturating pushes, so this approach is not yet efficient.

The idea is to store these non-saturating pushes (represented by their edges) in dynamic trees. As we search out non-saturating pushes (along forward edges), we keep these edges in a dynamic tree. We use the root operation to fast-forward to the leading non-saturating push-point, and then look for the next non-saturating push from there.

We can charge each newly discovered non-saturating push to inserting the edge in the dynamic forest. Meanwhile a forward edge $e = (u, v)$ is deleted from the dynamic forest data structure iff either

1. $e$ is saturated.
3. Push-Relabel and Directed Minimum Cut

3.4 Scaling push-relabel

2. \(v\) is relabeled. In particular, a fixed edge is deleted at most \(O(n)\) times. Thus an edge is also inserted at most \(O(n)\) times, and we only have a “new” non-saturating push \(O(n)\) times per edge.

This leads to an overall running time bounded by \(O(mn)\) tree operations. We leave it to the reader to sort out the remaining implementation details, or otherwise refer the reader to the aforementioned references.

3.4 Scaling push-relabel

The final version of push-relabel we present is a scaling algorithm due to Ahuja and Orlin [AO89]. Here we assume that the capacities \(c : E \rightarrow [U]\) are integers between 1 and \(U\) (for some \(U \in \mathbb{N}\)); the running time will depend on \(\log(U)\).

The high level idea as follows. For a fixed \(\Delta > 0\), we maintain the invariant that each vertex has excess at most \(2\Delta\). We only make non-saturating pushes of size exactly \(\Delta\). (That is, even if we could push more, we truncate the push to \(\Delta\) units of flow.) We will always push an active vertex at the lowest possible level, which preserves the invariant that the maximum excess at most \(2\Delta\).

Suppose all active vertices have excess less than \(\Delta\). Then we can divide \(\Delta\) in half, and continue with this smaller value of \(\Delta\).

Since we are repeatedly halving, it is convenient to maintain \(\Delta\) as a power of 2. Initially we set \(\Delta = 2^\lfloor \log_2 U \rfloor\). The algorithm maintains integrality throughout, so when \(\Delta < 1/2\), this implies that there are no active vertices. In particular, there are \(O(\log(U))\) halvings before terminations.

Now, consider the following potential functions which allows us to amortize non-saturating pushes.

\[
\Phi = \frac{1}{\Delta} \sum_{v \in V} \hat{f}(v)\ell(v).
\]

Lemma 3.11. Relabeling and saturating pushes take \(O(1)\) amortized time (w/r/t \(\Phi\)). Non-saturating pushes take no amortized time.

Proof. The first claims are easy; for non-saturating pushes, we observe that each non-saturating push moves \(\Delta\) units of flow to a lower-level.

The remaining operation to analyze – unique to scaling – is when we decrease \(\Delta\). Here we have the following.

Lemma 3.12. Dividing \(\Delta\) in half takes \(O(n^2)\) amortized time.

51
3. Push-Relabel and Directed Minimum Cut

3.5. Directed minimum cut via one push-relabel

Kent Quanrud
Fall 2021

Proof. Observe the potential is at most $O(n^2)$ because $\hat{f}(v) \leq 2\Delta$ for all $v$. Halving $\Delta$ doubles the potential $\Phi$, which gives the amortized running time.

This leads to the following running time. The broad strokes of obtaining this running time are captured by the above lemmas; we leave it to the reader to flesh out the remaining implementation details (to identify the active vertex at lowest level, to identify a forwards edge at the active vertex, etc.).

**Theorem 3.13** (Ahuja and Orlin [AO89]). *In a flow network with $m$ edges, $n$ vertices, and integer capacities between 1 and $U$, push-relabel with scaling (appropriately implemented) computes the maximum flow in $O(mn + n^2 \log(U))$ time.*

3.5 Directed minimum cut via one push-relabel

We now shift gears to a different but related problem: minimum directed cut. A directed cut is a set of edges $C \subset E$ whose removal makes the graph not strongly connected. That is, there exists some pair $s, t \in V$ for which $C$ is an $(s, t)$-cut. The goal is to find the minimum weight directed cut.

Of course, we can reduce the problem to $(s, t)$-cut by enumerating all pairs. This requires $O(n^2)$ $(s, t)$-cut computations. We can do better by fixing $s$, and then enumerating all $t \in V - s$. For each $t$, we compute the minimum $(s, t)$-cut and the minimum $(t, s)$-cut. The question is whether one can do better.

**Rooted connectivity.** A slight change in perspective that is very helpful is to instead focus on the rooted connectivity problem. Fix a vertex $r \in V$, called the root. A *rooted cut*, or $r$-cut, is a set of edges forming an $(r, t)$-cut for some $t \in V - r$. The minimum rooted cut asks for the minimum weighted rooted cut. Of course an algorithm for minimum $r$-cut solves the minimum directed cut problem, by picking any $r \in V$, and computing the minimum $r$-cut in both $G$ and the graph obtained by reversing all the edges in $r$.

Now, to find the minimum $r$-cut, we can of course enumerate all $t \in V - r$ and compute the minimum $(r, t)$-cut for each. This approach seems excessive for a couple reasons.

1. After computing the minimum $(r, t)$-cut for some $t$, if this cut is not the minimum $r$-cut, then we know that $t$ must be on the $r$-side of the minimum $r$-cut. So we can reduce the size of the graph slightly by contracting $t$ into $r$ before continuing to the next choice of $t$.

2. After computing the minimum $(r, t)$-cut, we (typically) also have a maximum $(r, t)$-flow to certify it. It seems somewhat wasteful to throw away all this
work when computing the \((r, t')\)-cut for the next sink \(t'\). Perhaps there is some way to extend

These informal observations motivate Hao and Orlin’s algorithm [HO94], which computes the minimum \(r\)-cut with a single push-relabel algorithm. The algorithm still enumerates all choices of \(t \in V - r\), but is careful to continue the progress from one \((r, t)\)-cut problem to the next. At a high level, the algorithm is as follows.

1. Pick some vertex \(t_1 \in V - r\), and compute the minimum \((r, t_1)\)-cut with push-relabel. (The cut is certified by a preflow, which we explain in greater detail below.) Take note of the value of this cut.

2. Now we want to pick a new vertex \(t_2\) as the next sink, and convert \(t_1\) into a source (effectively contracting it with \(r\)). We pick a vertex \(t_2\) with \(\ell(t_2) = 1\) (the existence of which is assured by some modifications to the push-relabel algorithm) as the next sink. We make \(t_2\) a source by pushing as much flow as possible out of \(t_2\) as to saturate all its edges, and set \(\ell(t_1) = n\). (So \(t\) will acts as a second source in addition to \(r\).) Taking the existing flow and labels as a starting point, we continue the push-relabel algorithm as to compute the minimum \((\{r, t_1\}, t_2)\)-cut.

3. We then select \(t_2\) with \(\ell(t_3) = 2\), convert \(t_2\) into a source (just like for \(t_1\)), and continue. In this fashion we iterate through all the other sinks.

Above we only give a high-level description and many details have yet to be filled in. An important difference from before is that we only run push-relabel until we find a minimum \((r, t)\)-cut certified by a preflow – and not necessarily to the point where \(f\) is a flow. To this we make the following observations about preflows.

**Max-preflow min-cut.** Given a preflow \(f\), we define the size of \(f\) to be the net flow at the sink vertex. lemma 3.1 can be rephrased as saying that every preflow \(f\) contains a flow \(f'\) of the same size. This implies the following.

**Lemma 3.14.** The maximum size of any \((s, t)\)-preflow equals the minimum capacity of any \((s, t)\)-cut.

**Lemma 3.15.** Let \(f\) be a preflow and \(\ell\) be a set of levels satisfying the (I) and (II). Suppose there exists an index \(i > 0\) such that:

1. There are no vertices \(v\) with level \(\ell(v) = i\).

2. There are no active vertices \(v\) with level \(\ell(v) < i\).

Then for \(T = \{v : \ell(v) < i\}\), \(\partial^-(T)\) is a minimum \((s, t)\)-cut.
Proof. In general, the capacity of $\partial^-(T)$ in the residual graph is reduced by exactly the total excess of all vertices in $T$. Since $T$ has no active vertices, this is exactly the size of $f$. Meanwhile the capacity of $\partial^-(T)$ is 0 in the residual graph. ■

Now we are prepared to discuss the modifications to the push-relabel algorithm. Our goal is to compute a preflow $f$ and level set $\ell$ as to satisfy the conditions of lemma 3.15. To this end, we always treat the active vertex of the lowest level. (This strategy is compatible with both the link-cut trees approach of ?? and the scaling approach of ??). If, in the course of treating the lowest level first, a level becomes empty, we immediately trigger lemma 3.15 and have obtained a minimum $(r, t)$-cut. Now, we want to ensure if the $i$th sink $t_i$ is at level $\ell(t_i) = i - 1$, then we will be able to find the $(i + 1)$th sink $t_{i+1}$ at level $i$. This could be an issue, for example, if the empty layer induces the minimum $(r, t)$-cut is at level $i$.

To address this, after creating the empty layer via relabel and extracting the cut, we undo that relabel. (Alternatively, we cancel the relabel and output the cut anyway, knowing that it would have been certified as the minimum $r$-cut). Thus we finish the iteration without actually creating an empty layer, and guaranteeing that the next sink can be found in the next level.

Note that in this algorithm, we never have to relabel a vertex beyond level $n - 1$ (except when a sink $t$ is converted to a source, in which case its level is set to $n$).

If we use the link-cut tree implementation, then we obtain the following running time.

**Theorem 3.16** (Hao and Orlin [HO94]). The minimum $r$-cut can be found in $O(mn \log \frac{n^2}{m})$ deterministic time.

Similarly obtains the analogous result via scaling. See also [Fra11] for the description of a $O(n^3)$ running time without data structures or scaling.

### 3.6 Rooted sparsification in simple graphs

We now consider rooted min cut for the special case of simple graphs.

Let $T \subset V$ be the sink-cut of the minimum $r$-cut; that is, $\emptyset \subset T \subset V - r$, and the minimum $r$-cut is $\partial^-(T)$. Let $\lambda$ denote the size of the cut; i.e., $\lambda = |\partial^-(T)|$. Suppose for the sake of discussion that we knew the value of $k$. (In truth, a constant factor estimate is sufficient, and eventually will be guessed with only logarithmic overhead.)

Besides the Hao and Orlin’s algorithm [HO94], here are two different approaches for computing the minimum $r$-cut, parametrized by $\lambda$ and $k$.

1. When the connectivity $\lambda$ is very small, then there is a deterministic algorithm by Gabow [Gab95] that runs in $\tilde{O}(m \lambda)$ time.
2. When the number of vertices \( k \) in the sink component \( T \) is very large, then we can try to randomly sample \( t \in T \) and find the minimum \( r \)-cut as the minimum \( (r, t) \)-cut.

Let \( MF(m, n) \) time denote the running time of \( (s, t) \)-max flow in a simple graph with \( m \) edges and \( n \) vertices. The second approach, by sampling \( n \log(n)/k \) vertices, takes \( O((n \log(n)/k) \ MF(m, n)) \) time and succeeds with high probability. Now, the best theoretical running time for \( MF \) is \( \tilde{O}(m + n^{1.5}) \) (!), recently obtained by Brand, Lee, Liu, Saranurak, Sidford, Song, and Wang [BLL+21]. This gives a running time of \( \tilde{O}(mn/k + n^{2.5}/k) \) for the second approach.

To review, we have running times of \( \tilde{O}(mk) \) and \( \tilde{O}((mn)/k + n^{2.5}/k) \). Their combination makes for a good algorithm if \( \lambda \) is small or \( k \) is large; the big weakness is when \( \lambda \) is large and \( k \) is small.

### 3.6.1 Small sink, small cut

**Lemma 3.17.** Let \( k \) be the minimum number of vertices in a sink component of a minimum \( r \)-cut. Then either \( k = 1 \) or \( \lambda < k \).

**Proof.** Let \( T \) be the set of vertices on the sink-side of a cut with \( \lambda \) edges. Suppose \( k = |T| > 1 \). Every vertex in \( T \) has in-degree \( > \lambda \). Consider all edges with head in \( T \). Of all the edges with head in \( T \), at most \( k(k - 1) \) edges have their tail in \( T \) as well. Thus \( \lambda > k\lambda - k(k - 1) \). Rearranging, we have \( k(k - 1) > (k - 1)\lambda \), hence either \( k = 1 \) or \( k > \lambda \).

This has immediate algorithmic implications. First of all, the case \( k = 1 \) is easy, since then \( T \) is a single vertex and we need only inspect the in-degree of all \( v \in V - r \). For \( k > 1 \), lemma 3.17 says that the \( \lambda < k \). Plugging this into the running time of Gabow’s algorithm [Gab95], we now have running times of the form \( \tilde{O}(mk) \) and \( \tilde{O}((mn + n^{2.5})/k) \).

1. If \( m \geq n^{1.5} \), then the second running time becomes \( \tilde{O}(mn/k) \). We run Gabow’s algorithm [Gab95] for \( k \leq \sqrt{n} \) and the sampling approach for \( k \geq \sqrt{n} \). This gives an overall running time of \( \tilde{O}(m\sqrt{n}) \).

2. If \( m \leq n^{1.5} \), then the second running time becomes \( \tilde{O}(n^{2.5}/k) \). We run Gabow’s algorithm [Gab95] for \( k \leq \sqrt{n^{2.5}/m} \) and the sampling approach for \( k \geq \sqrt{n^{2.5}/m} \). This gives an overall running time of \( \tilde{O}(n^{1.25}\sqrt{m}) \).

This is already a sizeable improve on Hao and Orlin’s algorithm [HO94]. But let’s keep going.
3.6.2 Rooted sparsification

Lemma 3.18 ("Rooted sparsification"). Let \( k \in \mathbb{N} \). Consider the graph \( \bar{G} \) obtained by contracting all vertices with in-degree \( \geq 2k \) into \( r \). Let \( \bar{r} \) denote the contracted vertex in \( \bar{G} \). Then we have the following.

1. \( \bar{G} \) is a multigraph with less than \( 2nk \) edges (with multiplicity).
2. If the minimum number of vertices in a sink component of a minimum \( r \)-cut has greater than 1 and less than or equal to \( k \) vertices, then the minimum \( r \)-cut and the minimum \( \bar{r} \)-cut are the same.

Proof. It is easy to see that contracting all vertices with in-degree \( \geq 2k \) into \( r \) results in a multigraph \( \bar{G} \) in which every vertex has in-degree \( < 2k \), hence there are at most \( 2nk \) edges total.

Let \( T \) be the sink component of a minimum \( r \)-cut. Observe that contracting into \( r \) cannot decrease the edge connectivity. If one can show that no vertices in \( T \) are contracted into \( \bar{r} \), then \( T \) is the sink component of a minimum \( \bar{r} \)-cut as well.

By lemma 3.17, the minimum \( r \)-cut has size \( \lambda < k \). Because \( G \) is simple and \( T \) has \( \leq k \) vertices, every vertex in \( T \) has in-degree less than \( \lambda + k < 2k \). Thus any contracted vertex is outside of \( T \). This completes the proof. ■

Thus, given \( k \), we can apply the rooted sparsification technique to effectively reduce – that is to say, sparsify – the graph to have at most \( nk \) edges. Now, if we use Gabow’s algorithm on the sparsified graph – where \( m \leq nk \), and \( \lambda \leq k \) – we obtain a running time of \( \tilde{O}(nk^2) \). The running time for \((s,t)\)-flow also improves to \( \text{MF}(nk, n) = \tilde{O}(nk + n^{2.5}) \). If we run Gabow’s algorithm for \( k \leq \sqrt{n} \) and the sampling approach for \( k \geq \sqrt{n} \), we obtain a total running time of \( \tilde{O}(n^2) \).

Of course, we may not know the true value of \( k \). Fortunately we can guess it efficiently. Observe that for all the arguments above, it suffices to have a constant factor estimate of \( k \) – e.g., a value \( k \) such that \( k/2 \leq |T| \leq k \) – while only increasing the running time by a constant factor. We can guarantee a factor two estimate by exhausting each power of 2: run the algorithm above for \( k = 1, 2, 4, \ldots, \lfloor \log_2 n \rfloor \). One of these values of \( k \) will be correct and return the desired cut with high probability. In conclusion, we have obtained the following.

Theorem 3.19 ([CQ21]). One can compute minimum \( r \)-cut in a simple graph with high probability in \( \tilde{O}(n^2) \)-time.

Remark 3.20. The argument easily extends to integer edge weights between 1 and \( U \), with a factor \( U \) increasing in the running time. See [CQ21].
3.7 Additional notes and references

Preflows were introduced by Karzanov [Kar74], which were employed as part of a blocking flow subroutine. The push-relabel algorithm was pioneered by [Gol85], which was then improved by Goldberg and Tarjan [GT88]. Goldberg’s thesis [Gol87, Chapter 1] is a good exposition of these developments. This lead to many follow-up works within the push-relabel framework, some of which we cover. There has also been experimental work on the algorithm [CG97; CGM98], by which push-relabel has acquired a reputation for being very good in practice.

The push-relabel algorithm is discussed in the algorithms textbook by Kleinberg and Tardos [KT06], and in books by Ahuja, Magnanti, and Orlin [AMO93], Frank [Fra11], Schrijver [Sch03], and Williamson [Wil19]. [Fra11; Wil19] also include the Hao-Orlin minimum cut algorithm from section 3.5. We also recommend the lecture notes by Blum and Gupta [BG13]. For the improved top-down push-relabel bound, we recommend a nice note by Tarjan [Tar13].

We did not discuss the minimum cut algorithm by Gabow [Gab95] which remains the best algorithm for small connectivities in multigraphs. Gabow’s algorithm is based on Edmonds’s [Edm75]’s directionless tree packing theorem. It is also a very nice example of matroid intersection, another topic unto itself. We may discuss Edmonds’s theorem [Edm75] and matroid intersection later in the course, at which point we will also discuss Gabow’s algorithm [Gab95].

The rooted min-cut algorithm for simple graphs (section 3.6) is very recent [CQ21].

3.8 Exercises

Exercise 3.1. In section 3.2, we described but did not analyze the “first-in first-out” strategy for selecting active vertices. (Vertices are placed in a queue as they are made active, from which we find the next active vertex.) Show that this push-relabel algorithm (appropriately implemented) takes $O(n^3)$ to find the maximum flow.
Chapter 4

Multicut and Multiway Cut

4.1 Beyond \((s,t)\)-flows and cuts

To this point, most of our discussion have been about the \((s,t)\) max-flow and min-cut problems. (We also discussed directed minimum cut which reduces to \((s,t)\)-flow.) A natural and very useful generalization is to allow for \textit{multiple} \((s,t)\)-pairs. In these problems, the input (like before) consists of a graph \(G = (V,E)\) and edge capacities \(c : E \to \mathbb{R}_{>0}\). The input also specifies a collection of \textbf{terminal pairs} \((s_1,t_1), \ldots, (s_k,t_k) \in \binom{V}{2}\), also called \textbf{commodities}. Given these commodities, a \textbf{multicommodity flow} consists of an \((s_i,t_i)\)-flow \(f_i : E \to \mathbb{R}_{\geq 0}\) for each \(i\). The different \(f_i\)'s \textit{do not} get to cancel out flow with one another, and instead they must share the capacities of the edges. Given a multicommodity flow \(\{f_i : i \in [k]\}\), the \textbf{load} on an edge \(e\) is the sum of flows on that edge, \(\sum_{i=1}^{k} f_i(e)\). The congestion on an edge \(e\) is the load divided by the capacity, \(\frac{\sum_{i=1}^{k} f_i(e)}{c(e)}\).

In the multicommodity setting — and in contrast to single commodity flow before — we say “\((s,t)\)-flow” only to mean that flow is conserved at every vertex besides \(s\) and \(t\), and not necessarily that it satisfies the capacity constraints. In many congestion minimization problems, it is natural to allow flows to exceed edge capacities, and develop algorithms that try to minimize the congestion. (Up to scaling, the distinction is not so important anyway, as will be made clear.)

In typical multicommodity flow problems, at a very high level, the more flow we can route within the capacities, the better. (Equivalently the less congestion the better.) But how we define “more flow” leads to different problems. For example, on can try to maximize the sum of sizes of the flows \(f_1, \ldots, f_k\). Or, one can try to maximize the minimum of the sum of sizes of the flows \(f_1, \ldots, f_k\). Sometimes the input also includes \textbf{demands} \(b_1, \ldots, b_k > 0\), which says that each \(f_i\) should be of size \(b_i\). One can then try to minimize the congestion among all multicommodity flows that meet the demands.
As we will see, (fractional) multicommodity flow can often solve problems in polynomial time because they can be expressed as an LP. However, there are discrete formulations that are typically NP-Hard. For example, one can require each $f_i$ to consist of a single $(s_i, t_i)$-path. The goal may be either to minimize the congestion of these paths (treated as flows of size 1); or, to try to route as many of the commodities as possible within the capacity constraints.

Dual to multicommodity flow problems are multicommodity cut problems. Here the goal is to find cost-efficient cuts that separate the $(s_i, t_i)$-pairs. These problems are often NP-Hard. Besides cut problems trying to separate all pairs simultaneously, we will study problems where one tries to optimize the ratio of cost-per-separated-pair. By considering, for example, the special case of all \( \binom{n}{2} \) possible terminal pairs, these problems can also be motivated as graph partitioning problems, where the general goal is to divide up the graph into substantially smaller parts while throwing out as little information (typically edges) as possible.

Below, we describe a number of related problems in this framework. We keep the descriptions brief as we will discuss several of them in greater detail later. Each of following problems can be considered in directed or undirected graphs. The directed versions are generally harder than the undirected versions, both in the sense that the algorithms and analyses are (usually) more complicated for directed graphs, and in the sense that the approximation factors are worse for directed problems (with lower bounds to support these gaps). Our study will be almost exclusively take place in undirected graphs.

1. **Multicut**: $G$ is undirected and the input specifies a set of terminal pairs $(s_1, t_1), (s_2, t_2), \ldots, (s_k, t_k)$. The goal is to compute a minimum cost cut separating every terminal pair $(s_i, t_i)$.

2. **Multiway cut**: The input contains a set $t_1, \ldots, t_k \in V$ of terminals. The goal is to compute a minimum cost cut separating every pair of terminals \( \{t_i, t_j\} \).

3. **Multicommodity flow**: The input specifies a set of terminals pairs $(s_1, t_1), \ldots, (s_k, t_k)$, and for each pair $(s_i, t_i)$, a demand $d_i > 0$. The goal is to compute, for each $i$, an $(s_i, t_i)$ flow $f_i : E \rightarrow \mathbb{R}_{\geq 0}$ such that the sum of flows fit in the capacities.

4. **Balanced cut**: Given a parameter $b \in (0, 1/2]$, find the set $S$ inducing a cut $\partial(S)$ of minimum total cost subject to $bn \leq |S| \leq (1 - b)n$. An important special case ($b = 1/2$) asks to split the graph in half, and is called the **minimum bisection** problem.
5. **Sparsest cut**: The input specifies a set of terminal pairs \((s_1, t_1), \ldots, (s_k, t_k)\). The goal is to compute a cut \(C = \partial(S) \subseteq E\) minimizing the ratio of cost-to-separated-pairs, also called the sparsity of the cut. An important special case is where we have all \(\binom{n}{2}\) terminal pairs, which is called uniform sparsest cut. A well-studied generalization is whether the terminal pairs are weighted; the goal then is to minimize the ratio of edge-costs to the weight-of-separated-pairs.

6. **Unsplittable flow**: Unsplittable flow is a discretization of multicommodity flow where each flow \(f_i\) is restricted to being a single path.

7. **All-or-nothing multicommodity flow**: We are given demands \(d_i\) for each commodity \((s_i, t_i)\), and the goal is to find a multicommodity flow that (completely) satisfies the demands of as many commodities \((s_i, t_i)\) as possible while satisfying the capacity constraints.

There are many more problems besides those above, including combinations of the problems above. (For example, one can consider the all-or-nothing unsplittable flow). Special cases of these problems (e.g., in special cases of graphs such as trees, interval graphs, or geometric settings) are also of interest.

In this chapter we will discuss some of the cut problems listed above; in the next chapter, we will discuss some of the flow problems.

### 4.2 Multicut

Let \(G\) be undirected, and let \((s_1, t_1), \ldots, (s_k, t_k) \in \binom{V}{2}\) be a set of \(k\) terminal pairs. Recall that a **multicut** is a set of edges that is simultaneously a cut for all \((s_i, t_i)\) pairs. Given costs (i.e., capacities) on the edges, the goal is to compute the minimum cost multicut. The problem is NP-Hard for \(k \geq 3\) [DJP+94].

In an implicit sense (discussed previously in chapter 1), the minimum multicut is a covering problem: the edges in the cut must cover all \((s_i, t_i)\)-paths for all \(i\). The dual packing problem, then, is to

\[
\text{Pack as many paths between the } (s_i, t_i)\text{-pairs as possible subject to the capacity constraints.}
\]

This is the same formulation as \((s, t)\)-flow except our path packing allows for \((s_i, t_i)\)-paths for all \(i\). The fractional version of this problem is sometimes called **sum multicommodity flow**.
4. Multicut and Multiway Cut

4.2. Multicut

As mentioned, the multicut problem is a covering problem requiring at least one edge from every \((s_i, t_i)\)-path, for all \(i\). The fractional relaxation asks for a fractional edge set with at least one total edge from the same family of the paths. Thus we have the following LP relaxation for the minimum cost multicut.

\[
\begin{align*}
\text{minimize} & \quad \sum_e c(e)y(e) \\
\text{subject to} & \quad \sum_{e \in p} y(e) \geq 1 \quad \text{for all } p \in P_{s_i, t_i}, \ i = 1, \ldots, k
\end{align*}
\]

(4.1)

Recall that for \((s, t)\)-flow, the LP relaxation could be understand as computing a shortest path metric at which \(s\) and \(t\) are at distance one. We can also recast (4.1) as a minimum cost metric problem, this time trying to meet distance requirements for all \((s_i, t_i)\)-pairs, as follows.

*Find the minimum cost edge lengths such that each \((s_i, t_i)\)-pair has shortest path distance \(\geq 1\).*

Our next task is to convert a fractional solution \(y\) for (4.1) to a feasible multicut with comparable cost. We will give two different approaches illustrating two useful techniques: randomized rounding (extending ideas from chapter 1), and deterministic “region growing”.

**Preliminaries.** Throughout the rest of this section, fix an optimum solution \(y\) to (4.1). We let \(\text{OPT}_L = \sum_e c(e)y(e)\) denote the optimum value of the LP. \(\text{OPT}_L\) is at most the cost of the (discrete) minimum multicut. By LP duality, \(\text{OPT}_L\) equals the size of the maximum sum multicommodity flow.

Our constructions will focus on the metric induced by \(y\). For two vertices \(u, v\), let \(d(u, v)\) denote length of the shortest \((u, v)\)-path w/r/t edge lengths given by \(y\). For a vertex \(s\) and radius \(r \geq 0\), we let \(B(s, r)\) denote the closed ball of all vertices at distance at most \(r\) from \(s\):

\[B(s, r) = \{v \in V : d(s, v) \leq r\}.
\]

4.2.2 Randomized rounding

Recall the LP-rounding proof of the \((s, t)\) max-flow min-cut theorem (chapter 1). We first obtain edges lengths \(y : E \to \mathbb{R}_{\geq 0}\) such that \(s\) and \(t\) are distance 1. We then draw \(\theta \in (0, 1)\) uniformly at random, and output the cut \(\partial^+(S)\) where \(S\) was the set of vertices with distance \(\theta\) or less from \(s\) (which always includes \(s\), and always excludes \(t\)). This cut was a minimum cut in expectation.
Perhaps the same idea extends here. Let $\theta \in (0,1)$ be drawn uniformly at random. For each $i$, let $A_i = B(s_i, \theta)$ be the ball of size $\theta$ centered at $i$. Consider the edge set $C = \bigcup_i \partial(A_i)$; $C$ is a multicut. Now, for each edge $e$, and each index $i$, we have

$$P[e \in \partial(A_i)] \leq y(e)$$

by the exact same argument as for $(s,t)$-cuts in chapter 1. By the union bound,

$$P[e \in C] \leq \sum_i P[e \in \partial(A_i)] \leq ky(e).$$

Thus the expected cost of $C$ is

$$E\left[\sum_{e \in C} c(e)\right] = \sum_e c(e) P[e \in C] \leq k \sum_e y(e)c(e) = k \OPT.$$

Thus we obtain a $k$-approximate multicut in expectation – the expected cost is at most $k$ times the cost of the minimum multicut. Actually our bounds are a little stronger: the randomized cut $C$ has cost at most $k$ times the value of the LP, $\OPT_L$. This value may be less than the value of the min multicut. But taking a step back, a $k$-approximation is hardly impressive. We could instead have computed the minimum $(s_i, t_i)$-cut for each $i$ – each of which has cost at most the value of the optimum multicut – and the union of the minimum $(s_i, t_i)$-cuts gives a $k$-approximate multicut (see also exercise 4.1).

**$O(\log k)$ approximation.** Next we show how to modify the $\theta$-rounding algorithm above and improve the bound from $k$ to $O(\log k)$.

To build up some intuition, let us point out some redundancy that produces the factor of $k$ in our first approach. Consider a fixed edge $e$. We upper-bounded the probability of $e \in C$ by taking a union bound...

Consider the following “scooping” algorithm that creates each ball $B(s_i, \theta)$ one at a time, removing the vertices $B(A_i, \theta)$ from the graph before proceeding to the next. Besides the aspect of removing parts out of the graph, we highlight two technical details. First, we restrict $\theta$ to $(0,1/2)$, which ensures we produce a multicut (lemma 4.1). Second, and more subtle, we scoop out the balls $B(s_i, \theta)$ in uniformly random order of $s_i$. The full algorithm is as follows.

1. Draw $\theta \in [0,1/2]$ uniformly at random, and sample a random permutation $\pi$ of $[k]$. Let $U = V$.

2. For each index $i$ in the randomized order:
   (a) Set $A_i = B(s_i, \theta)$, and $B_i = A_i \cap U$
Lemma 4.1. \( C \) is a multicut.

Proof. For each \((s_i, t_i)\) pair, \( s_i \in A_i \) implies that \( s_i \in B_j \) for some index \( j \) (either \( i \) itself, or another index randomly ordered before \( i \)). \( B_j \) has diameter \( < 1 \), and \( d(s_i, t_i) > 1 \), so \( t_i \notin B_j \). That is, \( \partial(B_j) \) is an \((s_i, t_i)\)-cut.

In the following lemma, \( H_k = 1 + 1/2 + \cdots + 1/k \) is the \( k \) harmonic number and is at most \( \ln(k + 1) \).

Lemma 4.2. For each edge \( e \in E \), \( \Pr[e \in C] \leq 2H_k y(e) \).

Proof. Fix an edge \( e = \{u, v\} \in E \). Let us reindex the indices in nondecreasing order of \( \min\{d(s_i, u), d(s_i, v)\} \). For each index \( i = 1, \ldots, k \), consider the event that the set \( B_i \) contributes \( e \) to \( C \); that is, \( B_i \) is the first set (in the randomized order) to cut \( e \). We claim that for all \( i \),

\[
\Pr[B_i \text{ contributes } e \text{ to } C] \leq \frac{2y(e)}{i} \tag{4.2}
\]

If the bound above holds true, then

\[
\Pr[e \in C] = \sum_i \Pr[B_i \text{ contributes } e \text{ to } C] \leq \sum_{i=1}^k \frac{2y(e)}{i} = 2H_k y(e),
\]

as desired.

Fix \( i \); we want to prove (4.2). Now, \( B_i \) occurs iff \( e \in \partial(A_i) \) and \( e \) was not already “scooped” out by another set \( A_j \); the latter occurs iff \( e \cap A_j \neq \emptyset \), and index \( j \) was randomly ordered before \( i \).

Consider first the event that \( e \in \partial(A_j) \). By essentially the same analysis as we have done multiple times before, we have

\[
\Pr[e \in \partial(A_j)] = \Pr[\min\{d(u, s_i), d(v, s_j)\} \leq \theta \leq \max\{d(u, s_i), d(v, s_j)\}] \leq 2|d(u, s_i) - d(v, s_j)| \leq 2y(e).
\]

Now, consider the event that \( e \) was not “scooped” out by another set \( A_j \), conditional on \( e \in \partial(A_i) \). Since \( e \in \partial(A_i) \), we know that \( e \cap A_j \neq \emptyset \) for all \( j < i \). So the event requires \( i \) to be ordered before \( j \) for all indices \( A_j \). The random ordering is independent of \( e \in \partial(A_i) \), and by symmetry, \( i \) is ordered first among indices \( \{1, \ldots, i\} \) with probability \( 1/i \). In conclusion,

\[
\Pr[B_i \text{ contributes } e \text{ to } C] = \Pr[e \in \partial(A_i)] \Pr[B_i \text{ contributes } e \text{ to } C | e \in \partial(A_i)] \leq \Pr[e \in \partial(A_i)] \Pr[i \text{ ordered before all } j < i] \leq \frac{2y(e)}{i},
\]

which establishes (4.2) and completes the proof.
4. Multicut and Multiway Cut

4.2. Multicut

Putting everything together, we have the following result.

**Theorem 4.3.** The randomized rounding algorithm described above returns a multicut with value at most $2H_k \text{OPT}_{\text{LP}}$, where $H_k$ is the $k$th harmonic number, and $\text{OPT}_{\text{LP}}$ is the optimum value of the LP.

**Proof.** By lemma 4.1, the set $C$ is a multicut. The expected cost is

$$E \left[ \sum_{e \in C} c(e) \right] = \sum_{e \in C} P[e \in C]c(e) = \sum_{e \in C} 2H_k c(e)y(e) = 2H_k \text{OPT}_{\text{LP}},$$

as desired. Here (a) is by linearity of expectation and (b) is by lemma 4.2. ■

4.2.3 Deterministic rounding via region growing

We now describe and analyze a deterministic alternative that also obtains a $O(\log k)$-approximation, which highlights an important technique called **region growing**. This algorithm was the first $O(\log k)$-approximation for multicut, by Garg, Vazirani, and Yannakakis [GVY96]. (The region growing technique itself is from Leighton and Rao [LR99] for the sparsest cut problem, which we will discuss in ??.)

The algorithm is simple and in some ways similar the randomized algorithm above. In the randomized algorithm, we scooped out balls centered at the sources $s_i$. The balls had a randomly sampled radius $\theta$, and they were scooped out in random order. The algorithm here also scoops out balls. It processes the sources in any order, and for each $s_i$, identifies a “good” radius $r_i$ and scoops out $B(s_i, r_i)$ from the graph. Here a “good” $r_i$ is one where the cost of the cut $\partial(B(s_i, r_i))$ is bounded relative to the total cost of all the edges scooped out by the ball. Good radii us to charge all the cut edges to the overall cost of the LP solution. The key is showing that a good radius exists, which is where the “region growing” argument comes into play.

Formally, we analyze the following algorithm.

1. For $i = 1, 2, \ldots, k$
   (a) Identify a radius $r_i \in [0, 1/2)$ such that
   $$\partial(B(s_i, r_i)) \leq 2\ln(k + 1) \sum_{e \in \varepsilon: x \cap B(s_i, r_i) \neq \emptyset} c(e)y(e). \quad (4.3)$$
   (b) Let $B_i = B(S_i, r_i)$ and set $V \leftarrow V \setminus B_i$.
2. return $C = \bigcup_i \partial(B_i)$.

Step 1a requires justification for the existence of such an $r_i$, and this is addressed by the following lemma.
4. Multicut and Multiway Cut

4.2. Multicut

Lemma 4.4. For each $i$, there exists a radius $r_i$ satisfying (4.3).

Proof. Fix $i$, and for ease of notation, let $s = s_i$. Define a function $W : \mathbb{R}_{\geq 0} \to \mathbb{R}_{\geq 0}$ by setting

$$W(0) = \text{OPT}/k,$$
$$W'(r) = c(\partial(B(s, r))).$$

Note that $W'(r)$ is piecewise linear, hence $W(r)$ is well-defined. To help interpret $W'(r)$, let

$$E[B(s, r)] = \{e \in E : e \cap B(s, r) \neq \emptyset\}$$

denote the set of edges incident to a vertex in $B(s, r)$. We have

$$W(r) - W(0) = \int_0^r c(\partial(B(s, \theta))) \, d\theta$$

\begin{align*}
\quad &\stackrel{(a)}{=} \sum_{e \in E} \int_0^r c(e) 1[e \in \partial(B(s, \theta))] \, d\theta \\
\quad &\stackrel{(b)}{=} \sum_{e \in E[B(s, r)]} c(e)y_e.
\end{align*}

Above, in (a), $1[e \in \partial(B(s, \theta))]$ denotes the \{0,1\}-indicator of whether $e \in \partial(B(s, \theta))$. (b) is because for each edge $e = \{u, v\} \in E$, $e \in \partial(B(s, \theta))$ iff $\min\{d(u, s), d(v, s)\} \leq \theta \leq \max\{d(u, s), d(v, s)\}$, and this interval has length at most $y(e)$.

Informally speaking, we can interpret $W'(r)$ as a sort of weight of the region $B(s, r)$, growing in $r$. The increase $W(r) - W(0)$ is a lower bound on the cost of all edges incident to a vertex in $B(s, r)$.

The derivation above also implies that $W'(r)$ is bounded above by

$$W(r) \leq W(0) + \sum_{e \in E} c(e)y_e = (1 + 1/k) \text{OPT}.$$ 

Now, suppose by contradiction that there was no $r \in [0, 1/2]$ satisfying (4.3); in terms of $W$, this implies that

$$W'(r) > 2\ln(k)W(r)$$

for $r \in [0, 1/2]$. The inequality above inequality $W'(r) > 2\ln(k)W(r)$ implies that

$$W(r) > e^{2\ln(k+1)r}W(0) = (k + 1)^{2r} \text{OPT}/k$$

for $r \in [0, 1/2]$. (Recall that the differential equation $f'(t) = \alpha f(t)$ is solved by $f(t) = e^{\alpha t} f(0)$.) For $r = 1/2$ we have $W(r) > (1+1/k) \text{OPT}$, a contradiction. ■
Now we prove the overall approximation.

**Theorem 4.5** (Garg, Vazirani, and Yannakakis [GVY96]). The deterministic algorithm described above returns a \((2 \ln (k + 1))\)-approximate minimum multicut.

**Proof.** The algorithm forces \(B_i\) to have radius at most \(1/2\). Consequently \(C\) is always a multicut by the same argument as lemma 4.1.

We now turn to the cost of \(C\), which is the union of \(\partial(B_i)\) over \(i = 1, \ldots, k\). For each \(i\), the contribution of \(\partial(B_i)\) is at most \(2 \ln (k + 1)c(E_i)\) where \(E_i\) is the set of edge remaining and incident to some vertex in \(B(s, r_i)\). In particular the \(E_i\)'s are disjoint. Thus \(\sum_i c(E_i) \leq \text{OPT}_L\) and we conclude that \(c(C) \leq 2 \ln (k + 1)c\).
4.3 Multiway cut

Let $G$ be undirected, and let $T = \{t_1, \ldots, t_k\} \subset V$ be a set of $k$ terminals. A multiway cut is a set of edges separating every pair of terminals in $T$ simultaneously; i.e., a special case of multicut for all pairs in $T$. The goal is to compute the minimum cost multicut. The problem is NP-Hard for $k \geq 3$ [DJP+94]. We already have a $O(\log(k))$-approximation via multicut; however, due to the symmetry of terminals, one might hope for better.

As with multicut, the minimum multicut is a covering problem. The edges in the cut must cover all paths with endpoints in $T$. The dual packing problem packs as many paths between $T$ as possible subject to the capacity constraints. This is the sum multicommodity flow problem with all pairs of terminals induced by $T$.

4.3.1 2-approximation via isolating cuts

Let $C \subset E$ denote the optimum multiway cost. In $G - C$, every $t_i$ is in a connected component of vertices $S_i \subset V$ that excludes all other terminals. Fix $i$ and consider the cut $\partial(S_i)$. The cut $\partial(S_i)$ isolates $t_i$ from all other terminals. Now, for fixed $i$, the minimum isolating cut for $t_i$ can be computed by $(s, t)$-flow, by treating $t_i$ as a source and grouping the remaining vertices $T - t_i$ as a sink. Let $D_i$ denote the minimum isolating cut for $t_i$, and observe that the union denoted $D \defeq D_1 \cup \cdots \cup D_k$ gives a multiway-cut.

It remains to relate the value of $D$ to the optimum solution $C$. We first observe that for each $i$, we have

$$c(D_i) \leq c(\partial(S_i))$$

by choice of $D_i$ as the minimum cost $t_i$-isolating cut. We also have

$$\sum_i c(\partial(S_i)) \leq 2c(C)$$

because each edge $e \in C$ appears in at most two of the isolating cuts $\partial(S_i)$. (Namely, for the sets $S_i$ containing the endpoints of $e$.) Altogether, we have

$$c(D) \leq \sum_i c(D_i) \leq \sum_i c(\partial(S_i)) \leq 2c(C).$$

We say that $D$ is a 2-approximate minimum cost multicut – $D$ is a multicut, and its objective value is within a multiplicative factor of 2 of the optimum value.

One can improve the approximation factor slightly to $(2 - 2/k)$. Observe that for all $i$, the set

$$D' \defeq D_1 \cup \cdots \cup D_{i-1} \cup D_{i+1} \cup \cdots \cup D_k,$$
which omits \( D_i \), is also a multicut. Let \( D_i \) be the isolating cut of maximum cost \( c(D_i) \); then

\[
c(D') \leq \sum_{j \neq i} c(D_j) \leq \frac{k-1}{k} \sum_{j=1}^{k} c(D_j) \leq 2 \left( 1 - \frac{1}{k} \right) c(C),
\]

as desired. In summary:

**Theorem 4.6** (Dahlhaus, Johnson, Papadimitriou, Seymour, and Yannakakis [DJP+94]). There is a \((2 - 2/k)\)-approximation for minimum cost multicut via isolating cuts.

### 4.3.2 Randomized rounding

We now take an LP-based approach to the multiway cut problem. The following relaxation of multiway cut is essentially the same as for multicut:

Com*pute the minimum cost edge lengths such that every pair of terminals has distance at least 1.*

To express this as an LP, let \( \mathcal{P}_T \) denote the set of all paths with (distinct) endpoints in \( T \). We have the LP,

\[
\text{minimize } \sum_e c(e) y(e) \text{ s.t. } \sum_{e \in \rho} y(e) \geq 1 \text{ for all } \rho \in \mathcal{P}_T.
\] (4.4)

Given a feasible solution \( y \) to (4.4), we want to round \( y \) to a multiway cut. The following algorithm is essentially the same algorithm as for multicut.

1. Draw \( \theta \in [0, 1/2] \) uniformly at random, and order the terminals \( T = \{t_1, \ldots, t_k\} \) uniformly at random.
2. For \( i = 1, \ldots, k-1 \)
   (a) \( C_i \leftarrow B(t_i, \theta) \) and \( V \leftarrow V \setminus C_i \)
3. Return \( \bigcup_{i \in T} C_i \).

One can show that, for every edge \( e = \{u, v\} \), the probability that \( e \) is cut is at most \( 2y(e) \) (see exercise 4.2). It follows from linearity of expectation that the expected cost of the multiway cut is \( 2 \sum_e c(e) y(e) = 2 \text{OPT}_{(4.4)} \). With care the bound can be improved to \((2 - 2/k)\) (exercise 4.3). In conclusion:

**Theorem 4.7.** Given a feasible solution \( y \) to LP (4.4), the randomized rounding algorithm described above returns a cut with expected value \((2 - 2/k) \sum_e c(e) y(e)\).
Can one do better? (Always a good question.) Consider multiway cut on a star graph. The vertices consist of a set of $k$ terminals, $T$, and an additional vertex $u$. We have an edge $\{u, t\}$ for every terminal $t \in T$. This creates a star with $u$ in the center and the terminals at the leaves. Clearly any multiway cut must include all but one edge – for a total of $k - 1$ edges. But there is a feasible LP solution that places $1/2$ on every edge, for a fractional total of $k/2$:

This gives an example where $\text{OPT} = k - 1$ and $\text{OPT}(4.4) \leq k/2$. The ratio of the two is bounded below by

$$\frac{\text{OPT}}{\text{OPT}(4.4)} \geq \frac{k - 1}{k/2} = 2 - \frac{2}{k}.$$

Recall that the rounding algorithm described above (and analyzed in exercises 4.2 and 4.3) obtains a multiway cut of expected value $(2 - 2/k) \text{OPT}(4.4)$. This also proves that $\text{OPT} \leq (2 - 2/k) \text{OPT}(4.4)$. The star example above also shows that, for this LP, one cannot obtain better worst-case bounds – any rounding algorithm on the star graph example must lose a factor of $2 - 2/k$.

The worst-case (multiplicative) difference between the LP and the discrete problem is called the **integrality gap**. Here we have shown an integrality gap of $2 - 2/k$. We always have $\text{OPT} \leq (2 - 2/k) \text{OPT}(4.4)$ and we have an algorithm that obtains this. We also have examples where $\text{OPT} \geq (2 - 2/k) \text{OPT}(4.4)$. So the integrality gap for (4.4) is exactly $2 - 2/k$.

Now, by LP-duality, $\text{OPT}(4.4)$ is also equal to the max sum multicommodity flow between $T$ (where $c$ becomes capacities). Our integrality gap becomes a **flow-cut gap** between the maximum sum multicommodity flow and the minimum multiway cut, for the symmetric setting of $k$ terminals.

**Theorem 4.8.** For a set of $k$ terminals in an undirected graph,

$$\max \text{multicommodity flow} \leq \min \text{multiway cut} \leq (2 - 2/k) \max \text{multicommodity flow}.$$

Moreover, there are examples showing that the inequalities are tight.

For this particular bound we have matching upper and lower bounds on the integrality gap. For other problems, we may have upper and lower bounds on the integrality gap that are not the same. This means it is still an open problem to improve either the upper or lower bounds and close the integrality gap.
4.3.3 A stronger LP

We now describe a better approximation algorithm of Călinescu, Karloff, and Rabani [CKR00] that obtains a 1.5-approximation for multiway cut. The starting point of Călinescu, Karloff, and Rabani’s algorithm [CKR00] is a stronger LP relaxation of multiway-cut.

Recall that a cut can be interpreted as a \( \{0,1\} \)-metric \( d : V \times V \rightarrow \{0,1\} \) such that \( d(u,v) = 1 \) iff \( u \) and \( v \) are separated by the cut. Previously we had considered the following LP.\(^1\)

\[
\begin{align*}
\text{minimize} & \quad \sum_{e = \{u,v\} \in E} c(e)d(u,v) \quad \text{over } d : V \times V \rightarrow \mathbb{R}_{\geq 0} \nonumber \\
\text{s.t.} & \quad d \text{ is a metric}, \\
& \quad d(t_1,t_2) \geq 1 \quad \text{for all } \{t_1,t_2\} \subset T
\end{align*}
\]

The interpretation is straightforward: we want to the minimum cost metric such that every pair of terminals is separated in the fractional sense; i.e., has distance 1.

We also saw the limits of the above LP – no rounding algorithm will give better than a 2-approximation. In this situation, one approach is to try to strengthen the LP – adding additional inequalities that still describe the optimum discrete solution, and might also decrease the integrality gap. (In particular, we want to rule out the star graph example above.) What else do know about multiway cuts? For one, we know that the multiway cut induces a partition of \( V \), where each vertex \( v \) is connected to at most one terminal. That is, each \( v \in V \) is separated from (at least) \( k - 1 \) of the terminals. Thus the following inequality holds for all metrics induced by multiway cuts:

\[
\sum_{t \in T} d(v,t) \geq k - 1 \quad \text{for all } v \in V.
\]

The following are also valid inequalities: For all pairs \( \{u,v\} \subset V \),

\[
\sum_{t \in T} |d(t,u) - d(t,v)| \leq 2d(u,v).
\]

---

\(^1\)We point out that the constraint that “\( d \) is a metric” is easily expressed by linear inequalities, as follows:

\[
\begin{align*}
d(v,v) &= 0 \quad \text{for all } v \in V \\
d(u,v) &= d(v,u) \quad \text{for all } u,v \in V \\
d(u,v) &\leq d(u,w) + d(w,v) \quad \text{for all } u,v,w \in V.
\end{align*}
\]
Indeed, fix a multiway cut and let \( d : V \times V \to \{0,1\} \) be the corresponding \( \{0,1\}\)-distance metric. If \( u \) and \( v \) are in the same component of the multiway cut, then \( d(u,t) = d(v,t) \) for all \( t \) and we have 0 on both sides. If \( u \) and \( v \) are in different components, then \( d(u,v) = 1 \), and \( d(u,t) = d(v,t) = 1 \) for all but 2 terminals corresponding to \( u \) and \( v \)'s separated components, where the absolute difference is 1. This gives us 2 on both sides of the inequality.

If we add these two sets of valid inequalities to the basic LP, we obtain Călinescu, Karloff, and Rabani’s stronger LP [CKR00], as follows.

\[
\begin{align*}
\text{minimize} & \quad \sum_{e=(u,v) \in E} c(e) d(u,v) \quad \text{over } d : V \times V \to \mathbb{R}_\geq 0 \\
\text{subject to} & \quad d(t_1, t_2) \geq 1 \quad \text{for all } \{t_1, t_2\} \subset T \\
& \quad \sum_{t \in T} d(v,t) \geq k-1 \quad \text{for all } v \in V \\
& \quad \sum_{t \in T} |d(t,u) - d(t,v)| \leq 2d(u,v) \quad \text{for all } u,v \in V^2
\end{align*}
\]

To build some intuition for (CKR), recall the example of a star from section 4.3.2. We had a set of \( k \) terminals \( T \) each attached to a center vertex \( u \). The multiway cut has size \( k - 1 \). For the basic LP (4.4), there is a feasible solution assigning 1/2 to every edge for a total size of \( k/2 \). This fractional solution is invalid for (CKR), however. In particular it violates the constraint that \( \sum_{t \in T} d(v,t) \geq k-1 \) (for each \( v \)). In fact this constraint directly force any feasible solution of (CKR) to have at least \( k - 1 \) fractional edges – as desired, for the star graph.

Călinescu, Karloff, and Rabani [CKR00] analyzed essentially the same randomized scooping algorithm as described in the previous section, except now applied to an optimum solution to (CKR). The algorithm selects \( \theta \in (0,1) \) uniformly at random and processes the terminals \( t_i \) in random order. For each, it scoops the ball \( S_i = B(t_i, \theta) \); this results in \( k \) components \( S_1, \ldots, S_k \) that induce a multiway cut. One can show that this gives an \((3/2)\)-approximation – we refer the reader to [CKR00] or the textbooks by Vazirani [Vaz01] and Williamson and Shmoys [WS11] for this analysis.

One can express the last set inequalities, involving the absolute values, with a family of linear inequalities. First, we introduce nonnegative variables \( A(t,u,v) \geq 0 \) for all \( t \in T \) and all ordered pairs \( u,v \in V \). We then include the constraints

\[
\begin{align*}
A(t,u,v) & \geq d(t,u) - d(t,v) \quad \text{for all } t \in T \text{ and } u,v \in V \\
\sum_{t \in T} A(t,u,v) + A(t,v,u) & \leq 2d(u,v) \quad \text{for all } u,v \in V.
\end{align*}
\]

The interpretation is that \( A(t,u,v) + A(T,v,u) \geq |d(t,u) - d(t,v)| \), and the optimum solution will set \( A(t,u,v) + A(T,v,u) = |d(t,u) - d(t,v)| \).
Instead we will describe an alternative rounding scheme that showcases the power of submodularity.

### 4.3.4 Rounding by uncrossing

The following rounding algorithm is due to Chekuri and Ene [CE11]; it is motivated by a more general problem called “symmetric submodular multiway partition” to which this algorithm can be extended to solve.

1. Pick $\theta \in (0, 1)$ uniformly at random.
2. For each $t \in T$ let $A_t = \{v : d(v, t) \leq \theta\}$.
3. Uncross the $A_t$’s to make them disjoint:
   
   (a) Let $B_t = A_t$ for each $t \in T$.
   
   (b) While there is a pair $B_s, B_t$ with $B_s \cap B_t \neq \emptyset$:
       
       i. Replace $B_s$ and $B_t$ with the better\footnote{Here “better” means the pair with the small sum of cut costs.} of $B_s$ and $B_t \setminus B_s$, or $B_t$ and $B_s \setminus B_t$.

4. Return the multiway cut $\bigcup_{t \in T} \partial(B_t)$.

Why do we insist on uncrossing the $A_t$’s to obtain disjoint $B_t$’s? We point out that $\bigcup_{t \in T} \partial(A_t)$ is already a multiway cut, with cost bounded above by $\sum_{t \in T} c(\partial(A_t))$ The reason we uncross is because then the value of the output multiway cut is

$$\frac{1}{2}\left(c(\partial(U)) + \sum_{t \in T} c(\partial(B_t))\right)$$

where $U = \bigcup_{t \in T} A_t$

– note the $1/2$ – because each edge in the multiway cut appears twice in the individual cuts $\partial(U)$ and $\{\partial(B_t) : t \in T\}$.

Of course uncrossing the $A_t$’s might create problems too. Perhaps $\sum_t c(\partial(B_t))$ is much $\sum_t c(\partial(A_t))$. The key claim is that uncrossing (step 3) does not increase the sum of cut values, $\sum_t c(\partial(A_t))$. More precisely, we will show that for any overlapping pair $B_s$ and $B_t$, either

1. $c(\partial(B_s)) + c(\partial(B_t \setminus B_s)) \leq c(\partial(B_t)) + c(\partial(B_s))$, or
2. $c(\partial(B_t)) + c(\partial(B_s \setminus B_t)) \leq c(\partial(B_s)) + c(\partial(B_t))$.

A set function $f : 2^V \to \mathbb{R}$ is symmetric if $f(S) = f(\overline{S})$ for all sets $S \subset V$. A set function $f : 2^V \to \mathbb{R}$ is submodular if

$$f(S) + f(T) \geq f(S \cup T) + f(S \cap T)$$
for all sets $S, T \subseteq V$. The cut function $f(S) = c(\partial(S))$ is easily seen to be symmetric and submodular.

Uncrossing cuts (and symmetric submodular functions) is an invaluable technique used in many places in combinatorial optimization and approximation algorithms.

**Lemma 4.9.** Let $f : 2^V \to \mathbb{R}$ be a symmetric submodular set function. Then for any two sets $A, B \subseteq V$,

$$f(A) + f(B) \geq \min\{f(A) + f(B - A), f(B) + f(A - B)\}.$$  

**Proof.** We have

$$f(A) + f(B) = f(A) + f(B) = \begin{cases} f(A) + f(B) & (a) \\ f(A) + f(B) & (b) \\ f(A) + f(B - A) & (c) \end{cases} = \begin{cases} f(A) + f(B) & (a) \\ f(A) + f(B) & (b) \\ f(B - A) + f(A - B) & (c) \end{cases}.$$  

by (a) symmetry and (b) submodularity, and (c) is by symmetry and De Morgan’s law. Thus

$$2f(A) + 2f(B) \geq f(A) + f(B - A) + f(B) + f(A - B),$$

which implies the claim claim.

The inequality obtained in (4.5), $f(A) + f(B) = f(B - A) + f(A - B)$ for symmetric submodular $f$, is called **posi-modularity**. We now return to the multiway cut problem and bounding the overall LP.

**Theorem 4.10** (Chekuri and Ene [CE11]). The uncrossing-based rounding algorithm described above yields a randomized $1.5$-approximation for minimum cost multiway cut.

**Proof.** Let $U = \bigcup_t A_t$. We have

$$c(C) = \frac{1}{2} \left( c(\partial(U)) + \sum_t c(\partial(B_t)) \right) \leq \frac{1}{2} \left( c(\partial(U)) + \sum_t c(\partial(A_t)) \right)$$

by lemma 4.9. It remains to bound the RHS.

Fix an edge $e = \{u, v\}$. For each terminal $t \in T$, we have

$$e \in \partial(A_t) \iff \min\{d(u, t), d(v, t)\} \leq \theta \leq \max\{d(u, t), d(v, t)\};$$

the RHS describes an interval of size $|d(u, t) - d(v, t)|$. Since $\theta$ is drawn uniformly from $[0, 1]$, we have

$$\mathbb{P}[e \in \partial(A_t)] = |d(u, t) - d(v, t)|.$$
Summed over all $t$, we have
\[ \sum_{t \in T} P[e \in \partial(A_t)] = \sum_{t \in T} |d(u, t) - d(v, t)| \leq 2d(u, v). \]

By linearity of expectation, then,
\[ \sum_t E[c(\partial(A_t))] \leq 2 \sum_{u, v} c(u, v)d(u, v) = 2 \text{OPT}_{(CKR)}. \]

Next we analyze $P[e \in \partial(U)]$. Without loss of generality suppose that $\min_t d(u, t) \leq \min_t d(v, t)$; then
\[ e \in \partial(U) \iff \min_t d(u, t) \leq \theta \leq \min_t d(v, t). \]

The RHS describes an interval of size $\min_t d(v, t) - \min_t d(u, t)$; we have
\[ \min_t d(v, t) - \min_t d(u, t) \leq \max_t d(v, t) - d(u, t) \leq d(u, v) \]
where (a) is by the triangle inequality. Thus $P[e \in \partial(U)] \leq d(u, v)$, and $E[c(\partial(U))] = \text{OPT}_{(CKR)}$.

Altogether, we have
\[ E[c(C)] \leq \frac{1}{2} E[c(\partial(U))] + \frac{1}{2} \sum_t E[c(\partial(A_t))] \leq \frac{3}{2} \text{OPT}_{(CKR)}, \]
as desired.

Remark 4.11. With a small tweak the approximation bound can be improved to slightly better than $3/2$ – this is left as an exercise to the reader.

### 4.4 Exercises

**Exercise 4.1.** In the warmup to the randomized rounding approximation algorithm for multicut (section 4.2.2), we pointed out that simply computing the minimum $(s_i, t_i)$-cut for all $i$ and outputting the union gives a $k$-approximation to the minimum multicut. In fact the union is within a factor $k$ of the multicut LP, (4.1): why?

**Exercise 4.2.** Prove that the rounding algorithm in section 4.3.2 obtains a factor 2 approximation, by showing in particular that every edge $e = \{u, v\}$ is cut with probability at most $2y(e)$. To do so, fix $e = \{u, v\}$. Consider the following cases and for each show that the probability of $e$ being cut is at most $2y(e)$.
4. Multicut and Multiway Cut

4.5. Further notes and references

Kent Quanrud  
Fall 2021

1. Neither $u$ nor $v$ is in $B(t_i, 1/2)$ for any $t_i$. (This is easy.)

2. $u, v \in B(t_i, 1/2)$ for some terminal $t_i$.

3. $u \in B(t_i, 1/2)$ for some terminal $t_i$, and $v \notin B(t_j, 1/2)$ for all $j$.

4. $u \in B(t_i, 1/2)$ and $v \in B(t_j, 1/2)$ for distinct terminals $t_i$ and $t_j$.

**Exercise 4.3.** Improve the analysis of the algorithm from section 4.3.2 to show that it obtains a $2(1 - 1/k)$-approximation.

**Exercise 4.4.** Adjust the algorithm and improve the analysis of theorem 4.10 to obtain a $(1.5 - 1/k)$-approximation.

## 4.5 Further notes and references

Approximations algorithms are a well-established and constructive approach to tackling many useful but NP-Hard optimization problems. We restrict ourselves to approximation algorithms for graph problems to keep the course compact, but these represent only a small sliver of the broad scope of approximation algorithms. Two well-known textbooks on approximation algorithms are by Vazirani [Vaz01] and Williamson and Shmoys [WS11]. This chapter overlaps with Chapters 4, 19, 20, and 21 in [Vaz01] and Chapter 8 in [WS11].

We learned the topic of approximation algorithms primarily from Chandra Chekuri, who has compiled numerous slides, lecture notes, and references over several iterations of his approximation algorithms course [Che06; Che09; Che11; Che13; Che16; Che18b]. We highlight his notes [Che18a] on Călinescu, Karloff, and Rabani's improved multiway-cut rounding algorithm [CKR00] which we follow closely. The presentation was also greatly influenced and improved by discussions with Chandra.

---

*Both textbooks are available online to Purdue students – [Vaz01] via the school library, and [WS11] at [https://www.designofapproxalgs.com](https://www.designofapproxalgs.com).*

75
Chapter 5
Sparsest cut

Let $G$ be undirected, and let $b : (V \rightarrow R_{\geq 0}$ be a set of nonnegative demands. Given a set $S$, the sparsity of $S$ is defined as the ratio

$$\frac{c(\partial(S))}{\sum_{u \in S, v \notin S} b(u, v)}.$$

(For $S = \emptyset$ or $S = V$, we treat the sparsity of $S$ as $+\infty$). The sparsest cut problem is to compute the set $S$ of minimum sparsity. An important special case is **uniform sparsest cut** where the demands are uniformly $b(u, v) = 1$. Then the sparsity has the simpler form

$$\frac{c(\partial(S))}{|S||\bar{S}|}.$$

To minimize the uniform sparsity, we (of course) want to minimize the numerator and maximize the denominator. Minimizing the numerator is to find small cuts (as usual). The denominator (by AM-GM) is maximized by choosing $|S| \approx n/2$. So the uniform sparsest cut is looking for a tradeoff between the capacity of the cut and how “balanced” the cut is. (In fact, sparsest cut is used as a subroutine for the *balanced cut* problem, as we will discuss.)

To drive this point further, observe that

$$\frac{1}{n} \cdot \frac{c(\partial(S))}{\min\{|S|, |\bar{S}|\}} \leq \frac{c(\partial(S))}{|S||\bar{S}|} \leq \frac{2}{n} \cdot \frac{c(\partial(S))}{\min\{|S|, |\bar{S}|\}}$$

because $n/2 < \max\{|S|, |\bar{S}|\} \leq n$. That is, up to a constant factor, we are trying to minimize the ratio

$$\frac{c(\partial(S))}{\min\{|S|, |\bar{S}|\}}.$$
Here we clearly see the expense of choosing a very small set $S$.

In this chapter, we describe a very influential result of Leighton and Rao [LR99] that obtains a $O(\log(n))$ approximation ratio for the uniform sparsest cut problem. Their algorithm is based on applying region growing to the metric induced by the dual LP. We then discuss a randomized $O(\log(n))$ approximation algorithm for general demands based on $\ell_1$-metric embeddings. We will also discuss lower bounds and some applications of sparsest cut.

5.1 The LP

Leighton and Rao’s algorithm [LR99] is based on rounding an LP relaxation of the sparsest cut problem with region growing. However, since the sparsest cut optimizes a ratio, obtaining the linear relaxation is not as obvious. As a step in this general direction, consider the following (nonlinear) relaxation of the sparsest cut problem.

**Compute a metric $d : V \times V \rightarrow \mathbb{R}_{\geq 0}$ minimizing the ratio**

$$\frac{\sum_{e = \{u,v\}} c(e) d(u,v)}{\sum_{\{u,v\}} b(u,v) d(u,v)}.$$

Now, we can scale the distances up or down with no effect on the ratio. In particular, we can fix the denominator to be 1, which gives the following optimization problem which is a linear program.

*Find the minimum $c$-cost metric such that the $b$-weighted sum of distances is at least 1.*

That is:

$$\min \sum_{e = \{u,v\} \in E} c(e) d(u,v)$$

over all metrics $d : V \times V \rightarrow \mathbb{R}_{\geq 0}$

s.t. $\sum_{\{s,t\}} b(s,t) d(s,t) \geq 1$. 

(5.1)

5.1.1 The dual

To obtain the dual, it is helpful to rewrite (5.1) as a pure covering problem. Recall the correspondence between metrics and edge lengths, via shortest path distances. Then (5.1) is the same as:
Find the minimum cost edge lengths such that the $b$-weighted sum of shortest path distances is at least 1.

To make this more explicit, for $s,t \in V$, let $\mathcal{P}_{s,t}$ denote the family of all $(s,t)$-paths. Let us define a path bundle as a collection of paths $P$ consisting of an $(s,t)$-path $P_{s,t} \in \mathcal{P}_{s,t}$ for every pair $(s,t)$. We let $\mathcal{P}_s \equiv \prod_{(s,t)} \mathcal{P}_{s,t}$ denote the family of all path bundles. Then we can express the problem above as follows.

$$\min \sum_{e \in E} c(e) y(e) \quad \text{over } y : E \to \mathbb{R}_{\geq 0}$$

s.t. $\sum_{(s,t)} b(s,t) \sum_{e \in P_{s,t}} y(e) \geq 1$ for all $P \in \mathcal{P}_s$. (5.2)

We can separate this LP by computing the shortest $(s,t)$-path for every pair $(s,t)$, and verifying the covering constraint for this bundle of shortest paths.

(5.2) covers path bundles with edges; thus, the dual packing LP packs path bundles into edges.

$$\max \sum_{P \in \mathcal{P}_s} x(P) \quad \text{over } x : \mathcal{P}_s \to \mathbb{R}_{\geq 0}$$

s.t. $\sum_{P \in \mathcal{P}_s} x(P) \sum_{(s,t) \in P_{s,t}} b(s,t) \leq c(e)$ for all $e \in E$. (5.3)

In (5.3), each path bundle $P$ represents a choice of paths for every $(s,t)$-pair to concurrently route the demands $b$. So (5.3) is trying to concurrently route $b$ as much as possible subject to the capacity constraints. This problem is called concurrent flow or demand multicommodity flow.

5.2 Rounding via region growing (for uniform demands)

Leighton and Rao [LR99] addressed the setting of uniform demands $b = 1$, as well as “product demands” where the demands are of the form $b(u,v) = \pi(u)\pi(v)$ for a fixed vector $\pi \in \mathbb{R}^V_{\geq 0}$. In this section we assume uniform demands, $b = 1$.

Then the sparsity of a set $S$ has the cleaner form of

$$\frac{c(\partial(S))}{|S||\bar{S}|}.$$ 

Let $d$ be an optimum metric to the metric relaxation (5.1). For uniform demands we may assume that $d(u,v) \leq 1$ for all $u,v$. The goal is to compute a set of vertices $S$ with

$$\frac{c(\partial(S))}{|S||\bar{S}|} \leq O(\log(n)) \cdot \text{OPT}_{(5.1)}.$$
Our analysis is divided into two cases. The first, dubbed the “concentrated” case, is when there is a high concentration of points within a small ball of radius $r_0 \approx 1/n^2$. The second “non-concentrated” case is when this does not occur. In the concentrated case we show that the line embedding from the center of the concentrated ball induces a sparse cut. In the non-concentrated case we partition the graph into pieces with a region-growing technique; the non-concentrated setting implies that these pieces are individually small enough to be reassembled to give a sparse cut.

For the remainder of this section, let $r_0 = 1/2n^2$.

### 5.2.1 Concentrated case.

First we consider the concentrated case.

**Lemma 5.1.** Suppose there exists a vertex $s$ such that $|B(s, r_0)| \geq 2n/3$. Then there exists $r \geq r_0$ such that $B(s, r)$ has sparsity $O(1) \text{OPT}_L$.

**Proof.** Suppose not. Then for all $q > 0$, we have

\[
\begin{align*}
&\ c(\partial(B(s, r_0 + q))) \geq c \text{OPT}_L |B(s, r_0 + q)\| V \setminus B(s, r_0 + q) | \\
&\ \geq \frac{cn \text{OPT}_L}{3} |V \setminus B(s, r_0 + q) | \\
\end{align*}
\]

for a fixed universal constant $c > 0$ (to be determined). Here (a) observes that $|B(s, r_0 + q)| \geq |B(s, r_0)| \geq 2n/3$.

Define a region-growing weight function $W(r)$ differentially by

\[
\begin{align*}
W(0) &= 0 \\
W'(0) &= c(\partial(B(s, r)))
\end{align*}
\]

$W(r)$ is nondecreasing, bounded below by 0, and bounded above $\text{OPT}_L$ (by a similar calculation as in multicut). Now, we have

\[
1 \geq \frac{W(1) - W(r_0)}{\text{OPT}_L} = \int_{0}^{1-r_0} \frac{c(\partial(B(s, r_0 + q)))}{\text{OPT}_L} dq \\
\geq \frac{cn}{3} \int_{0}^{1-r_0} |V \setminus B(s, r_0 + q)| dq = \frac{cn}{3} \left( \sum_{v \in V \setminus B(s, r_0)} d(v, s) - r_0 \right) \\
\geq \frac{cn}{3} \sum_{v \in V} (d(v, s) - r_0) = \frac{cn}{3} \sum_{v \in V} d(v, s) - \frac{cnr_0}{3}.
\]

(b) implicitly interchanges changes. In (c), the inequality holds because the sum is extended only by nonpositive terms.
We also have
\[ 1 \leq \sum_{\{u,v\}} d(u,v) \leq \sum_{\{u,v\}} d(u,s) + d(s,v) < n \sum_v d(v,s), \]
where (d) is by the distance constraint in the LP and (e) is by the triangle inequality.

Plugging back into (5.4), we have \( 1 > c/6 \). For \( c \geq 6 \), then, we have a contradiction. ■

### 5.2.2 Non-concentrated case.

Now we address the remaining non-concentrated case.

**Lemma 5.2.** Suppose \( |B(s, r_0)| \leq 2n/3 \) for all \( s \in V \). Then one can compute, a cut with sparsity \( O(\log n) \) \( \text{OPT}_L \).

**Proof.** We will apply a region growing technique (similar to multicut) to partition \( V \) into sets \( S_1, \ldots, S_k \subset V \) such that

1. \( c(\partial(S_i)) \leq O(n^2 \log n) \) \( \text{OPT}_L \).
2. \( |S_i| \leq 2n/3 \) for all \( i \).

Assuming this holds, then, we obtain the desired sparse cut as follows. Since \( |S_i| \leq 2n/3 \) for all \( i \), we can group the \( S_i \)'s into two sets \( T \subset V \) and \( \bar{T} = V \setminus T \) such that \( n/3 \leq |T|, |\bar{T}| \leq 2n/3 \). Then \( T \) has sparsity

\[ \frac{c(\partial(T))}{|T||\bar{T}|} \leq \frac{c(\bigcup_i \partial(S_i))}{(2n/3)(n/3)} \leq O(\log n) \text{OPT}_L. \]

It remains to obtain the sets \( S_1, \ldots, S_k \). As alluded to above, we will use region growing to iteratively remove sets of the form \( B(s, r) \); the key is to show that there are always “good” radii \( r \).

Fix a center vertex \( s \in V \). Consider the region-growing weight function \( W(r) \) defined differentially by

\[ W(0) = \text{OPT}_L/n \]
\[ W'(r) = c(\partial(B(s, r))). \]

\( W(r) \) is nondecreasing in \( r \) and bounded by the range \([\text{OPT}_L/n, (1+1/n) \text{OPT}_L]\). Consequently, for a sufficiently large constant \( c_0 \), there always exists a radius \( r < r_0 \) such that

\[ W'(0) \leq c_0 n^2 \log(n) W(r), \]
since otherwise the differential inequality
\[ W'(0) \geq c_0 \log(n)n^2W(r) \]
implies that
\[ W(r_0) \geq e^{c_0n^2\log(n)r_0}(\text{OPT}_L/n) > (1 + 1/n)\text{OPT}_L, \]
a contradiction. Thus there exists a radius \( r \leq r_0 \) that, in particular, satisfies
\[
c(\partial(B(s, r))) = W'(r) \leq O\left(n^2 \log(n)\right)W(r)
\leq O\left(n^2 \log(n)\right) \left(\text{OPT}_L/n + \sum_{v \in B(s,r)} \sum_{e \in \partial(v)} c(e)y(e)\right).
\]
Moreover, by assumption, we have \(|B(s, r)| \leq 2n/3\).

We repeat the following steps until \( V \) is empty. Here we let \( i \) be the index of the iteration.

1. Pick \( s_i \in V \) arbitrarily.
2. Find \( r_i \in [0, r_0] \) such that
\[
c(\partial(B(s, r))) \leq O\left(n^2 \log n\right) \left(\frac{\text{OPT}_L}{n} + \sum_{v \in B(s,r)} \sum_{e \in \partial(v)} c(e)y(e)\right).
\]
3. Set \( S_i = B(s, r) \) and \( V = V \setminus S_i \).

We claim that these steps produce sets \( S_1, \ldots, S_k \) satisfying the two requirements above.

Clearly we have \(|S_i| \leq 2n/3\). We also have
\[
c\left(\bigcup_{i} \partial(S_i)\right) \leq \sum_{i} c(\partial(S_i))
\leq O\left(n^2 \log n\right) \left(\text{OPT}_L + \sum_{v \in S_i} \sum_{e \in \partial(v)} c(e)y(e)\right)
= O\left(n^2 \log n\right) \left(\text{OPT}_L + \sum_{v \in V} \sum_{e \in \partial(v)} c(e)y(e)\right)
= O\left(n^2 \log n\right) \text{OPT}_L,
\]
which completes the proof. ■
5.2.3 Putting it all together.

**Theorem 5.3** (Leighton and Rao [LR99]). The region-growing algorithm described above gives a $O(\log n)$-approximation to the uniform sparsest cut.

**Proof.** Let $y$ be an optimum solution to (5.2). It is easy to identify if $y$ falls in the concentrated or non-concentrated case. For the former, we obtain a $O(1)$-approximation; for the latter, a $O(\log n)$-approximation. ■

5.3 Rounding via $L_1$-metric embeddings

We now analyze a different approach to rounding the sparsest-cut metric, based on randomized embeddings. This version readily generalizes to general demands $b : \binom{V}{2} \rightarrow \mathbb{R}_{\geq 0}$.

Let $d$ be any metric that is a feasible solution to LP (5.1). Our goal is to convert $d$ to a cut with sparsity comparable to the cost of $d$.

5.3.1 Rounding line embeddings

We first observe that some special cases of metrics are very easy to round – namely, those related to line embeddings. Suppose there is a function $f : V \rightarrow \mathbb{R}$ such that $d(u,v) = |f(u) - f(v)|$ for all $u,v \in V$.

(Such a function $f$, placing $V$ on the real line, is called a **line embedding**.)

Rescaling and translating (which does not effect the sparsity), we may assume that $\min_u f(u) = 0$, and $\max_v f(v) = 1$.

Consider the following random cut $S$ (which we have seen before in proving the max-flow min-cut theorem). Pick $\theta \in (0,1)$ uniformly at random, and let $S = \{u : f(u) \leq \theta\}$. Observe that for each edge $e = \{u,v\}$, we have

$$
\Pr[e \in \partial(S)] = |f(u) - f(v)| = d(u,v).
$$

Thus we can rewrite the sparsity of $d$ as

$$
\text{(sparsity of $d$)} = \frac{\sum_{(u,v) \in E} c(u,v)d(u,v)}{\sum_{(s,t)} d(s,t)b(s,t)} = \frac{\mathbb{E}\left[\sum_{e \in \partial(S)} c(e)\right]}{\mathbb{E}\left[\sum_{s \in S, t \notin S} b(s,t)\right]}.
$$

(5.5)

Note that the RHS is **not** the expected sparsity of $S$. The expected sparsity of $S$ is the quantity

$$
\mathbb{E}[\text{sparsity of $S$}] = \mathbb{E}\left[\frac{\sum_{e \in \partial(S)} c(e)}{\sum_{s \in S, t \notin S} b(s,t)}\right],
$$

82
is not the same as the quantities in (5.5). This is in contrast to our proof of max-flow min-cut, where $S$ is a minimum $(s, t)$-cut on average, and the existence of a minimum $(s, t)$-cut follows immediately from the probabilistic method.

Still the probabilistic approach can be salvaged with a little more work. Observe that $S$ can only be one of $n - 1$ different sets $S_1, \ldots, S_{n-1}$ where $\emptyset \subsetneq S_1 \subset S_2 \subset S_3 \cdots \subset S_{n-1} \subsetneq V$. For each $i$, let $p_i = \mathbb{P}[S = S_i]$. Then

$$(5.5) = \frac{\sum_{i=1}^{n-1} p_i \sum_{e \in \partial(S_i)} c(e)}{\sum_{i=1}^{n-1} p_i \sum_{s \in S_i, t \in \bar{S}_i} b(s, t)}.$$ 

Now we apply the following elementary fact. (The proof is left as exercise 5.1.)

**Lemma 5.4.** Let $a_1, \ldots, a_h, b_1, \ldots, b_h > 0$. Then

$$\min_i \frac{a_i}{b_i} \leq \frac{\sum_i a_i}{\sum_i b_i} \leq \max_i \frac{a_i}{b_i}.$$ 

It follows that for some $S_i$, the sparsity of $S_i$ is at most the sparsity of $d$. In conclusion, we have shown the following.

**Lemma 5.5.** Let $d : (V^2) \to \mathbb{R}$ be a metric induced by a line embedding. Then one can partition the vertices into two sets $(S, \bar{S})$ such that

$$\frac{\sum_{e \in \partial(S)} c(e)}{\sum_{s \in S, t \in \bar{S}} b(s, t)} \leq \frac{\sum_{(u, v) \in E} c(u, v) d(u, v)}{\sum_{(s, t)} d(u, v) b(s, t)}.$$ 

**Rounding $L_1$-metrics** So much for metrics given by line embeddings. How about a metric obtained as a sum of line metrics? Recall that the $L_1$-metric on $\mathbb{R}^h$ is defined by

$$\|x - y\|_1 = \sum_{i=1}^h |x_i - y_i|.$$ 

Suppose $d$ was the $L_1$-metric of an embedding $f : V \to \mathbb{R}^h$. That is,

$$d(u, v) = \|f(u) - f(v)\|_1 = \sum_{i=1}^h |f_i(u) - f_i(v)|$$ 

for a function $f : V \to \mathbb{R}^h$. We can think of this as the sum of $h$ line metrics $f_1, \ldots, f_h$. The sparsity of $d$ expands out to

$$(\text{sparsity of } d) = \frac{\sum_{i=1}^h \sum_{e = (u, v)} c(e) |f_i(u) - f_i(v)|}{\sum_{i=1}^h \sum_{(u, v)} b(u, v) |f_i(u) - f_i(v)|}.$$
Applying lemma 5.4 again, we see that one of these line embeddings, say $f_j$, has sparsity no worse than $d$. From the line embedding $f_j : V \rightarrow \mathbb{R}$, we can extract a cut with sparsity at most that of $f_j$. This establishes the following.

**Lemma 5.6.** Let $d : (V^2) \rightarrow \mathbb{R}$ be the $L_1$-metric over an explicit embedding of $V$. Then one can partition the vertices into two sets $(S, \bar{S})$ such that

$$\frac{\sum_{e \in \partial(S)} c(e)}{\sum_{s \in S, t \in \bar{S}} b(s,t)} \leq \frac{\sum_{(u,v) \in E} c(u,v) d(u,v)}{\sum_{(s,t)} d(u,v) b(s,t)}.$$ 

To sum up: $L_1$ metrics can be rounded without loss. We can find an $L_1$-metric with sparsity within a factor $\alpha$ of the sparsest metric $d$, then we can convert that into an $\alpha$-approximate sparsest cut.

### 5.3.2 Randomized $L_1$-metric embeddings

We now know that $L_1$-metrics can be rounded to sparse cuts without any loss. But the LP for sparsest cut produces a generic metric $d$, that is not an $L_1$-metric.

Our new strategy, given a generic metric $d$, is to try to find an $L_1$-metric $d_1$ with sparsity comparable to $d$. We then invoke lemma 5.6 to round obtain a sparse cut from $d_1$. Our $L_1$-metric $d_1$ will be defined by a mapping $f : V \rightarrow \mathbb{R}^h$ (for some $h \in \mathbb{N}$), so that

$$d_1(u,v) = \|f(u) - f(v)\|_1.$$ 

We will prove the following theorem.

**Theorem 5.7.** Let $d : (V^2) \rightarrow \mathbb{R}_{\geq 0}$ be a metric and $\delta \in (0,1)$. For $h = O(\log(n) \log(1/\delta))$, one can construct a randomized embedding $f : V \rightarrow \mathbb{R}^h$ such that for all $u,v \in V$, we have

$$\|f(u) - f(v)\|_1 = O(\log(1/\delta)) d(u,v)$$ 

deterministically, and

$$\Pr[\|f(u) - f(v)\|_1 \leq d(u,v)] \leq \delta \quad (5.6)$$

For $\delta = 1/\text{poly}(n)$ and $h = O\left(\log^2 n\right)$, we can apply the union bound to (5.6) over all pairs $u,v$. This gives the following theorem.

**Corollary 5.8.** Let $d : (V^2) \rightarrow \mathbb{R}_{\geq 0}$ be a metric and $\delta \in (0,1)$. For $h = O\left(\log^2(n)\right)$, one can construct a randomized embedding $f : V \rightarrow \mathbb{R}^h$ such that with high probability, for all $u,v \in V$,

$$d(u,v) \leq \|f(u) - f(v)\|_1 \leq O(\log n) d(u,v).$$
The embedding \( f : V \rightarrow \mathbb{R}^h \) described in Corollary 5.8 is said to be an \( O(\log n) \)-distortion metric embedding as it maps points in one metric space into another while preserving all distances up to a \( O(\log n) \)-multiplicative factor.

The \( O(\log n) \)-distortion embedding into \( L_1 \) is the last ingredient for the following algorithm for sparsest cut.

1. Solve the LP (5.1) to obtain a sparsest metric \( d \).
2. Invoke Corollary 5.8 to obtain a \( O(\log n) \)-distortion embedding \( f : (V, d) \rightarrow \mathbb{R}^h \) from \( d \) into the \( L_1 \)-metric. The \( L_1 \)-metric via \( f \) has sparsity at most a \( O(\log n) \) factor greater than \( d \).
3. Invoke lemma 5.6 to round the \( L_1 \)-metric to a cut with sparsity at most the metric, which is a factor \( O(\log n) \) greater than the sparsity of \( d \) (and the optimum of (5.1)).

This algorithm is due to Linial, London, and Rabinovich [LLR95], and establishes the following.

**Theorem 5.9.** There is a \( O(\log n) \) randomized approximation to (non-uniform) sparsest cut (on undirected graphs).

Actually, Linial, London, and Rabinovich [LLR95] observed that one can do slightly better when there are demands for only \( k \) commodity pairs. (i.e., \( k \) pairs \((u, v)\) with \( d(u, v) > 0 \).)

**Theorem 5.10.** There is a \( O(\log k) \) randomized approximation to sparsest cut, where \( k \) is the number of commodity pairs with nonzero demand.

This approximation factor is obtained by building on the ideas in ??, and left as exercise 5.2.

Now we describe the randomized algorithm of Linial, London, and Rabinovich [LLR95] that computes the embedding in theorem 5.14. We note that previously Bourgain [Bou85] had obtained a deterministic embedding but the output dimension \( h \) was exponential. Linial, London, and Rabinovich’s algorithm [LLR95] can be interpreted as an efficient randomized implementation of Bourgain’s embedding [Bou85].

The algorithm, which we call random-Fréchet, is extremely simple. We generate \( \lfloor \log n \rfloor \) coordinates. For \( i = 1, \ldots, n \), we sample a set \( S_i \) where each point is sampled independently with probability \( 1/2^i \). For each vertex \( v \), we find the distance between \( v \) and (the closest point in) \( S_i \). This gives the \( i \)th coordinate of \( v \). Pseudocode is described in Figure 5.1.

Each coordinate of the randomized embedding is given by distances from a set. There is a name for this class of embeddings: Fréchet embeddings. Figure 5.2 attempts to visualize a single coordinate generated in this manner.
5. Sparsest cut

5.3. Rounding via $L_1$-metric embeddings

Let $S_i$ sample each $v \in V$ independently with probability $2^{-i}$

for each $v \in V$

1. $f_i(v) \leftarrow \min_{s \in S_i} d(s,v)$

2. return $f : V \to \mathbb{R}_{\geq 0}^{\lceil \log n \rceil}$

Figure 5.1: A $O(\log n)$ dimension, randomized Frechét embedding with $O(\log n)$ distortion in expectation

Figure 5.2: Level sets by distance from a set of points $S_i$, encoding one coordinate of a Frechét embedding.
5.3.3 Low-distortion in expectation

We now turn to proving theorem 5.14. Consider an instance of the random-Frechet algorithm, which computes a randomized embedding \( f : V \rightarrow \mathbb{R}^{O(\log(n))} \).

For ease of notation, for a vertex \( v \) and coordinate \( i \), we let \( v_i \) denote the \( i \)th coordinate of the embedding of \( v \).

**Lemma 5.11.** For \( u, v \in V \) and \( i \in \mathbb{N} \), \( |u_i - v_i| \leq d(u, v) \).

**Proof.** By the triangle inequality, we have both
\[
d(s, u) - d(s, v) \leq d(u, v) \quad \text{and} \quad d(s, v) - d(s, u) \leq d(u, v).
\]
for all \( s \in S_i \).

lemma 5.11 immediate implies the \( \|u - v\|_1 \leq O(\log n) d(u, v) \), since there are \( O(\log n) \) dimensions and each can contribute at most \( d(u, v) \). The harder part is showing the lower bound: informally, we want to show that \( \|u - v\|_1 \geq d(u, v) \), up to constant factors. This lower bound is too strong; instead, we settle for the same inequality but only in expectation.

**Lemma 5.12.** For \( u, v \in V \), we have \( \mathbb{E}[\|u - v\|_1] \geq c d(u, v) \) for some constant \( c > 0 \).

**Proof.** For ease of notation, let \( \delta = d(u, v) \). For each \( i \), let \( r_i \) be the minimum length \( r \) such that there are at least \( 2^i \) points at distance \( \leq r \) from \( u \), and \( 2^i \) points at distance \( \leq r \) from \( v \); i.e.,
\[
r_i = \arg \min_{r > 0} \{|\{x : d(u, x) \leq r\}| \geq 2^i, |\{x : d(v, x) \leq r\}| \geq 2^i\}
\]
We claim that

For each index \( i \), we have
\[
|u_{i+1} - v_{i+1}| \geq (\min\{r_i, \delta/2\} - \min\{r_{i-1}, \delta/2\})
\]
with constant probability \( c > 0 \).
Before proving the claim, suppose it holds true. Let $k$ be the largest index such that $r_{k-1} \leq \delta/2$ (for which the claim applies). We have

$$\mathbb{E}[\|u - v\|_1] \geq \sum_{i=0}^{k} \mathbb{E}[|u_{i+1} - v_{i+1}|]$$

\[(a) \geq c \sum_{i=0}^{k} (\min\{r_i, \delta/2\} - \min\{r_{i-1}, \delta/2\})\]

\[(b) \geq \frac{c\delta}{4},\]

as desired. Here (a) applies the claim and (b) is by telescoping sums and recalling that $r_{k+1} > \delta/2$.

It remains to prove the claim. We have two cases: (a) $r_i \leq \delta/2$, and (b) $r_{i-1} < \delta/2 \leq r_i$. We assume without loss of generality that $r_i$ is defined by $u$; i.e., $|\{x: d(u, x) < r_i\}| < 2^i$.

**Case 1: $r_i \leq \delta/2$.** Let $U = \{x: d(u, x) < r_i\}$, and let $V = \{x: d(v, x) \leq r_{i-1}\}$. We have $|U| < 2^i$ and $|V| \geq 2^{i-1}$. Since $r_{i-1} < r_i \leq \delta/2$, $U$ and $V$ are disjoint.

$S_{i+1}$ samples each point with probability $2^{-i-1}$. By direct calculation, $S_{i+1}$ samples no points from $U$ with constant probability, and at least one point from $V$ with constant probability. Since $U$ and $V$ are disjoint, whether any point from $U$ is sampled and whether any point from $V$ is sampled is independent. Thus $S_{i+1}$ samples a point from $V$ and no points from $U$ simultaneously with some constant probability $c > 0$. In this event, we have $u_{i+1} \geq r_i$ and $v_{i+1} \leq r_{i-1}$, so $u_{i+1} - v_{i+1} \geq r_i - r_{i-1}$. In expectation, we have

$$\mathbb{E}[|u_{i+1} - v_{i+1}|] \geq c(r_i - r_{i-1}),$$

as desired.
Case 2: \( r_i > \frac{\delta}{2} > r_{i-1} \). Let \( U = \{ x : d(u,x) \leq \frac{\delta}{2} \} \) (with \( \frac{\delta}{2} \) in place of \( r_i \)) and let \( V = \{ x : d(u,v) \leq r_{i-1}/2 \} \).

By the same argument as above, we have that \( S_{i+1} \) samples a point from \( V \) and no points in \( U \) with some constant probability \( c > 0 \). In this event, \( u_{i+1} - v_{i+1} \geq \frac{\delta}{2} - r_{i-1} \).

**Theorem 5.13.** randomized-Fréchet embeds \( V \) into \( \mathbb{R}^{\lceil \log n \rceil} \) such that

\[
\|u - v\|_1 \leq \lceil \log n \rceil d(u,v) \text{ and } E[\|u - v\|_1] \geq c d(u,v) \text{ for all } u, v \in V,
\]

for some absolute constant \( c > 0 \).

**5.3.4 Amplification**

Theorem 5.13 shows that a single instance of randomized-Fréchet obtains \( O(\log n) \) distortion “in expectation”, so to speak, for each pair of vertices. In particular the embedded distance is bounded above deterministically but below only in expectation. We want to strengthen this so that the lower bound holds with high probability.

**Theorem 5.14.** With probability of error \( 1/poly(n) \), the average of \( O(\log n) \) embeddings produced by randomized-Fréchet is an embedding \( V \) into \( \mathbb{R}^{O(\log^2 n)} \) such that

\[
\min cd(u,v) \leq \|u - v\|_1 \leq C \log(n) d(u,v) \text{ for all } u, v \in V
\]

for absolute constants \( c, C > 0 \).

**Proof sketch.** Fix \( u, v \in V \). We treat each coordinate difference \( |u_i - v_i| \) (for \( O(\log^2 n) \) coordinates over \( O(\log n) \) independent calls to randomized-Fréchet) as an independent random variable bounded above by \( d(u,v) \). The expected sum of the \( |u_i - v_i|'s \) is \( \Omega(n \log(n)) \). By standard Chernoff inequalities, the sum is strongly concentrated at the mean; scaling down by \( \log n \) (from averaging) gives the desired result.
5.4 Expander lower bounds

Recall that the integrality gap of an LP relaxation is the maximum (multiplicative) difference between then value of the LP and the discrete problem. The rounding algorithms above show that the integrality gap of the metric LP relaxation (5.1) is at most $O(\log n)$ for uniform sparsest cut (or $O(\log k)$ for $k$ commodity pairs). We want to find a lower bound on the integrality gap as tight as possible.

Consider sparsest cut. The sparsest cut is at most a $O(\log n)$-multiplicative factor greater then the sparsity of the sparsest metric. We want to find a family of graphs where the sparsest cut is at least a $\Omega(\log n)$-multiplicative factor greater than the sparsest metric. We will show this for a family of graphs called constant-degree expanders.

An unweighted, undirected graph $G$ is an $\varphi$-expander if for all sets $S \subset V$,

$$|\partial(S)| \geq \varphi \min\{|S|, |\bar{S}|\}.$$  

The definition extends to weighted graphs by replacing the LHS with the weight of the cut. We only need to consider unweighted graphs for our lower bound.

An obvious example of a graph with high expansion is a clique, i.e., a complete graph. The clique on $n$ vertices has expansion $n/2$. An obvious example of a graph with low expansion is a tree; a tree on $n$ vertices has expansion $\Omega(1/n)$. As noted in the introduction, expansion and uniform sparsity are, up to scaling, within a constant factor of one another. Sparse cuts imply low expansion and high expansion imply no sparse cuts.

The clique and tree examples reflects a general intuition that high expansion is correlated with many edges in the graph. But in fact this is not necessarily so: there exists graphs with constant expansion (much higher than a tree) but constant maximum degree (like a tree, and unlike a clique).

Theorem 5.15. There exists constants $c, d$ such that for all sufficiently large $n$, there exists a $c$-expander with $n$ vertices and maximum degree $d$.

The fact that a very small (constant degree) graph can be strongly connected (i.e., have large expansion) has vast and profound consequences. We will see this phenomena realized in a few different places in this course. Below we will prove theorem 5.15 by showing that the union a constant number of uniformly random, perfect matchings is an expander with nonzero probability. First, we use theorem 5.15 to lower bound the integrality gap.

5.4.1 Integrality gap for uniform sparsest cut

Theorem 5.16. Integrality gap for uniform sparsest cut is $O(\log n)$. 

Proof. By theorem 5.15, let $G$ be an $n$-vertex $c$-expander with maximum degree $d$, where $c$ and $d$ are universal constants. For every set of vertices $S$, we have

$$\text{(sparsity of } \partial(S)) = \frac{|\partial(|S|)|}{|S||\bar{S}|} \geq \frac{c \min\{|S|, |\bar{S}|\}}{|S||\bar{S}|} \geq \frac{c}{n}$$

where (a) invokes the definition of a $c$-expander.

By LP duality, the sparsest metric has sparsity equal to the maximum throughput $\lambda$ of any concurrent flow. We claim that any concurrent flow has throughput $\lambda$ at most $O\left(\frac{1}{n \log n}\right)$. Then multiplicative difference between $c/n$ and $O\left(\frac{1}{n \log n}\right)$ is $\Omega\left(\log n\right)$, as desired.

To upper bound $\lambda$, we argue that, since $G$ has constant maximum degree, for almost every $(s, t)$-pair, the length of the shortest $(s, t)$-path is at least (roughly) $\log n$. This implies that all the flow paths in the concurrent flow has at least $\log n$, and almost every $(s, t)$-pair requires a total capacity of $\lambda$. Meanwhile the total capacity of the graph is limited because $G$ is sparse, which leads to an upper bound on $\lambda$.

To this end, fix $s$. Since $G$ has maximum degree $d$, there are at most $d$ vertices at distance 1 from $s$, at most $d^2$ vertices at distance 2 from $s$, and so forth. In general there are $O(d^i)$ vertices at distance $i$ or less from $s$. For $h = \log_d(n)/2$, there are at most $O(\sqrt{n})$ vertices at distance $h$ or less from $i$. This leaves $n - O(\sqrt{n})$ vertices at distance $h = \Omega(\log n)$ or greater from $s$. Over all $s$, there are $(1 - o(1))(\frac{n}{2})$ pairs $(s, t)$ at distance $\Omega(\log n)$.

It follows that any concurrent flow with through $\lambda$ – which, in particular, sends $\lambda$ units of flow between $\Omega(n^2)$ $(s, t)$-pairs at distance $\Omega(\log n)$ – requires a total capacity of $O(n^2 \log(n) \lambda)$. But the total capacity in the constant degree expander is $O(n)$. So $\lambda \leq O\left(\frac{1}{n \log n}\right)$. ■

Remark 5.17. The proof above used to constant maximum degree to argue that most $(s, t)$-pairs are at distance $\Omega(\log n)$. Meanwhile one can show that a graph with constant expansion has diameter $O(\log n)$

5.4.2 A few random matchings make an expander

We now turn to proving theorem 5.15. While a proof of existence would suffice, we will prove theorem 5.15 via a very simple, randomized construction (for $n$ even): namely, for a fixed set of $n$ vertices, the union of a few (independently sampled) uniformly random perfect matchings over these $n$ vertices has constant expansion. We remark that there are many other constructions and in particular there is interest in questions such as deterministic constructions, implicit constructions

\footnote{The diameter of a graph is the maximum $(s, t)$-distance among all $(s, t)$ pairs.}
of very large expanders with compact oracle access, and extreme families such as Ramanujan graphs.

**Theorem 5.18.** Let \( n \) be even and sufficiently large. There exists constants \( c, d > 0 \) such that the union of \( d \) independent and uniformly random matchings of a set of \( n \) vertices form a \( c \)-expander with high probability.

**Proof.** Let \( d \in \mathbb{N} \) be a parameter to be determined, and consider the union of \( d \) perfect matchings over a fixed set of \( n \) vertices. We want to show that for every set \( S \) of size \( |S| \leq n/2 \), \( |\partial(S)| \geq c|S| \) for some constant \( c \).

We reframe this claim as follows. Let \( S \subset V \) be of size \( k \overset{\text{def}}{=} |S| \leq n/2 \). Let \( T \subseteq V \setminus S \) be of size \( |T| = k/6 \).

**Claim.** With nonzero probability, for all such \( S \) and \( T \), some vertex in \( S \) is matched to a vertex outside \( S \cup T \).

This implies that, for all set \( S \) with at most \( n/2 \) vertices, \( |\partial(S)| \geq |N(S)| \geq (7/6)|S| \). We will prove the claim probabilistically: our first goal is to fix \( S \) and \( T \), and bound the probability that all of \( S \) is matched to \( S \cup T \).

To this end, for fixed \( S \) and \( T \), consider the following randomized procedure for generating a perfect matching.

1. Index the vertices \( v_1, \ldots, v_n \) such that \( S = \{v_1, \ldots, v_k\} \) and \( T = \{v_{k+1}, \ldots, v_{7k/6}\} \).
2. Repeat \( n/2 \) times:
   (a) Let \( v_i \) be the unmatched vertex of smallest index
   (b) Sample \( v_j \) among the remaining unmatched vertices uniformly at random.
   (c) Match \( v_i \) with \( v_j \)

While this procedure might appear biased by \( S \) and \( T \), we argue that it is not.

**Claim.** The above steps generate a uniformly random perfect matching.

Indeed, observe that each random choice in the procedure leads to a different matching. There are \((n-1)(n-3)\cdots 1\) possible outcomes and all are equally likely. There are also \((n-1)(n-3)\cdots 1\) possible matchings. This proves the claim.

Thus we may assume that our \( d \) perfect matchings are generated by the procedure above. Now, for a single matching, the probability that \( S \) is matched to \( (S \cup T) \) is bounded above by probability that first \( k/2 \) choices of \( v_j \)'s are in \( S \cup T \). The probability of this is bounded above by
5. Sparsest cut

5.5. Application: Minimum bisection

In (a) we observe that

\[(7/6)k - 2i + 1 \leq k/2 \iff i \geq k/3 + 1/2\]

and that

\[n - 2i + 1 \geq n/2 \iff i \leq n/4 + 1/2, \text{ if } i \leq k/2 + 1/2\]

For \(d\) matchings, the probability that \(S\) is matched to \((S \cup T)\) in all \(d\) matchings is bounded above by \(\binom{n}{k}^{-d/6}\). Now we take a union bound over all choices of \(S\) and \(T\). We have

\[\frac{n}{7k/6} \left( \frac{n}{k} \right)^{-d/6} = \left( \frac{n}{k} \right)^{1-d/6} \leq \left( \frac{n}{k} \right)^{-O(d)}\]

for a sufficiently large constant \(d\). Taking a union bound over \(k = 1, 2, \ldots, n/2\), we conclude that with high probability, all sets \(S\) with \(|S| \leq n/2\) have at least \(|S|/6\) neighbors. So the union of a constant number of random matchings is a \((1/6)\)-expander.

5.5 Application: Minimum bisection

A bisection is a partition of the vertices \(V\) into \((S, \bar{S})\) of (essentially) equal size: \([n/2] \leq |S|, |\bar{S}| \leq [n/2]\). Alternatively a bisection can be defined in terms of cuts as a set of edges whose removal leaves the graph with connected components of at most \([n/2]\) vertices each. In any case the minimum bisection problem is to compute a vertex set \(S \subset V\) of size \(|S| = [n/2]\) minimizing the cost of the cut, \(c(\partial(S))\).

There is a natural connection between minimum bisection and the sparsest cut – the minimum bisection problem can be recast as the restricting the sparsest cut problem to vertex sets with exactly half the vertices.

The following algorithm uses a \(O(\log n)\)-approximation for uniform sparsest to obtain a bicriteria-approximation algorithm. In particular, it returns a \((1/3)\)-balanced partition \((S, \bar{S})\) – that is, \(n/3 \leq |S| \leq 2n/3\) – with cost at most \(O(\log n)\) times the cost of the minimum bisection. The algorithm is very simple. It repeatedly computes the sparsest cut and removes the smaller side from the graph, until the number of vertices removed is at least \(n/3\) (and necessarily at most \(2n/3\)).
1. For $i = 1, 2, \ldots$
   A. $S_i \leftarrow$ smaller side of a $O(\log n)$-approximate uniform sparsest cut.
   B. If $|S_1| \cup \cdots \cup |S_i| \geq n/3$
      1. return $(S_1 \cup \cdots \cup S_i, V - (S_1 \cup \cdots \cup S_i))$.
   C. Else remove $S_i$ and all incident edges form the graph, and repeat.

It is (relatively) easy to see why the algorithm returns a $(1/3)$-balanced cut; it remains to show that the cost is comparable to that of the minimum bisection. The intuition is as follows. Suppose for simplicity we have an exact algorithm for the sparsest cut. The sparsest cut is very close to the minimum (weighted) expansion, which we recall is the cost of the cut divided by the number of vertices on the smaller side of the cut. In particular this ratio for the sparsest cut is no worse than that of the minimum bisection. If the sparsest cut is balanced, then its cost is comparable to the minimum cost bisection. While its not balanced, we can interpret the sparsest cut as removing some vertices from the graph at the cost of the edges being cut. The ratio of vertices removed per unit cost – the bang-for-buck, so to speak – is at least as good. So we are gradually removing vertices while paying a favorable rate compared to the minimum cost bisection.

There are some additional details to take care of – for one, the minimum bisection in the input graph may no longer be a minimum bisection in the residual graphs, although it will still be somewhat balanced as long as we haven’t removed $n/3$ vertices yet. Also we only have a $O(\log n)$-approximation for the sparsest cut, which will imply that we pay an additional $O(\log n)$ factor throughout the argument. exercise 5.3 guides the reader through a formal proof of the argument.

**Theorem 5.19.** In polynomial time, one can compute $(1/3)$-balanced cut with total cost at most a $O(\log n)$-factor greater than the minimum bisection.

### 5.6 Exercises

**Exercise 5.1.** Prove lemma 5.4.

**Exercise 5.2.** Recall the randomized $O(\log n)$ approximation algorithm for sparsest cut based on $L_1$-embeddings. Show how to adjust the algorithm and analysis to obtain a randomized $O(\log k)$ approximation factor where $k$ is the number of commodities with nonzero demand.

**Exercise 5.3.** Recall the bicriteria approximation algorithm for the minimum bisection problem from section 5.5.
1. Show that the algorithm returns a 1/3-balanced cut.

2. For each iteration $i$, w/r/t the graph remaining at iteration $i$, we have

$$\frac{c(\partial(S_i))}{|S_i|} \leq O(\log(n)) \frac{\text{OPT}}{n}.$$

3. Combine the two parts above to prove that the algorithm returns a 1/3-balanced cut of size $O(\log(n)) \text{OPT}$.

Exercise 5.4. One generalization of uniform sparsest cut is to directed graphs. Here one is given a directed graph $G$ with edge costs $c$; the (directed, out-) sparsity of a set $S$ is defined

$$\frac{c(\partial^+(S))}{|S||S|}$$

where we recall that $\partial^+(S)$ is the directed cut of edges leaving $S$. The directed uniform sparsest cut problem is to find the set $S$ that minimized the directed sparsity just defined. Extend Leighton and Rao’s algorithm [LR99] to obtain a $O(\log n)$ approximation for the directed uniform sparsest cut.

Exercise 5.5. One can also consider the bisection problem in directed graphs. Here the goal is to find a vertex set $S$ of size $\lfloor n/2 \rfloor \leq |S| \leq \lceil n/2 \rceil$ minimizing the cost of the directed cut $c(\partial^+(S))$. Suppose one had access to a $O(\log n)$ approximation algorithm for uniform directed sparsest cut (as described in exercise 5.4). Using this as a subroutine, design and analyze an algorithm that obtains a bicriteria approximation algorithm for the minimum directed bisection problem with essentially the same approximation bicriteria for the undirected setting: compute a set $S$ with $n/3 \leq |S| \leq 2n/3$ with cost $c(\partial^+(S))$ at most a $O(\log n)$-factor greater than that of the minimum directed bisection.

Exercise 5.6. Prove that any graph $G$ with constant expansion has diameter $O(\log n)$.

5.7 Additional notes and references

One can do better than a $O(\log n)$ approximation for uniform sparsest cut – Arora, Rao, and Vazirani [ARV09] gave a $O(\sqrt{\log n})$ via semi-definite programming and ideas from high-dimensional geometry. We may discuss this result later in the course; in the meantime we refer the reader to lecture notes by Rothvoss [Rot16].

We refer to Spielman [Spi19] and Trevisan [Tre16] for further background on expanders, especially in the context of spectral graph theory. This follows the proof in [Tre16].

---

This is not an easy exercise.
Chapter 6

Edge-Disjoint Paths and All-or-Nothing Multicommodity Flow

6.1 Edge-Disjoint Paths

We have already discussed several variants of (fractional) multicommodity flow problems – both explicitly, and as they are arise in the duals of NP-hard cut problems such as multicut and sparsest cut. These flow problems are ultimately (compact) LP’s and thus can be solved in polynomial time.

If we require integrality, however, the problems are not necessarily polynomial-time solvable. The simplest example is edge disjoint paths. Let \( G = (V, E) \) be a graph and let \( T = \{(s_1, t_1), \ldots, (s_k, t_k)\} \). The goal is to compute an \((s_i, t_i)\)-path for each \( i \) such that all the paths are mutually edge-disjoint. A generalization, sometimes also called integral multicommodity flow, also includes an integral demand \( b_i \) for each \((s_i, t_i)\) ∈ \( T \), and asks for \( b_i \) paths for each \((s_i, t_i)\), subject to all the paths between edge-disjoint. One can consider the problem in directed and undirected graphs, and with (integral) capacities. One can also consider vertex disjoint paths (or vertex capacities).

**Theorem 6.1** (Fortune, Hopcroft, and Wyllie [FHW80]). Edge-disjoint paths is NP-hard in directed graphs even for 2 commodities that are opposite pairs (of the form \((s, t), (t, s)\)).

Surprisingly, in undirected graphs, if \( k \) is held constant, then there are polynomial time algorithms. This was first shown for \( k = 2 \) [Sey80; Shi80; Tho80], and extended to any constant \( k \) as part of Robertson and Seymour’s seminal work on graph minors.
Theorem 6.2 (Robertson and Seymour [RS95]). For fixed $k$, the $k$ vertex-disjoint paths problem in undirected graphs is polynomial-time solvable.

This in turn implies an algorithm for edge-disjoint paths for fixed $k$. (The reduction is left to the reader in exercise 6.2.)

Corollary 6.3. For fixed $k$, the $k$ edge-disjoint paths problem in undirected graphs is polynomial time solvable.

The dependence on $k$ in both of the results above is very large, and these algorithms are not polynomial time in general when $k$ is part of the input.

Given the intractability of edge-disjoint paths for large $k^1$, we naturally consider approximation algorithms. In fact the edge-disjoint paths problem is extremely difficult - there are lower bounds showing that a $O(n^{1/2-\epsilon})$-approximation factor for the directed version implies $P=NP$ (for all $\epsilon > 0$), and for a long time the best upper bound for undirected graphs was bigger. It was not clear if the undirected case would turn out to be as hard to approximate as the directed setting. We will study a relaxation called the all-or-nothing multicommodity flow problem which was introduced by Chekuri, Mydlarz, and Shepherd [CMS07]; this problem turned out to be an important step towards later developments in approximating edge-disjoint paths.

### 6.2 The All-or-Nothing Multicommodity Flow Problem

Let $G$ be an undirected and edge-capacitated graph. Let $\mathcal{F} = \{\{s_1,t_1\}, \ldots, \{s_k,t_k\}\} \subseteq \binom{V}{2}$ be a set of $k$ commodity pairs. The goal is to compute a subset of commodity pairs $\mathcal{F}' \subseteq \mathcal{F}$, with $|\mathcal{F}'|$ as large as possible, that can be routed by a (fractional) multicommodity flow with congestion 1. Without loss of generality we assume the pairs are disjoint, and each terminal is a leaf in the graph.

Note that for any fixed set of commodities $\mathcal{F}' \subset \mathcal{F}$, a linear program can decide if $\mathcal{F}'$ is feasibly routable in polynomial time. The all-or-nothing flow

---

1. and also to duck the heavy graph theory required for Robertson and Seymour’s results [RS95], which far beyond the scope of this class
2. For a fixed vertex $s$, suppose $s$ is routed to $\ell$ distinct terminals $t_1, \ldots, t_\ell$. We introduce $\ell$ new vertices $s'_1, \ldots, s'_\ell$ each attached as a leaf to $s$ via an edge of infinite capacity. We replace each commodity pair $(s,t_i)$ with the pair $(s'_i,t_i)$.

![Graph Diagram](https://via.placeholder.com/150)
problem – identifying which subset $S'$ to route - is NP-Hard. We will present a $O\left(\log^2 n\right)$-approximation algorithm due to Chekuri, Khanna, and Shepherd [CKS05].

The problem can be further generalized by having varying demands for different pairs. This algorithm leverages our recent discussions about sparsest cut a flow-cut gaps in a divide-and-conquer framework that is of broader interest.

We briefly mention two natural ways to generalize this problem. The first is to introduce demands $b_i$ for each commodity pair $\{s_i, t_i\}$: now we must route $b_i$ units of flow between $s_i$ and $t_i$, or none. It turns out that one can reduce general demands to unit demand with only a constant factor loss via ideas from [CMS03; KS04]. The second generalization is to introduce weights (i.e., profits) for each commodity: the goal becomes to choose the maximum weight set $S$. We leave this as ??.

6.3 Well-linked terminals

Before developing an algorithm for the all-or-nothing multicommodity flow problem, we first introduce the notion of “well-linked terminals” and explore some of their properties.

Let $T \subseteq V$ be a set of terminal vertices. Let $\pi : T \rightarrow \mathbb{R}_{>0}$ be a set of weights over $T$, and let $P = \sum_{t \in T} \pi(t)$. We say that $T$ is $\pi$-well-linked if the demands

$$b(s, t) = \frac{\pi(s)\pi(t)}{T}$$

can be routed with congestion 1.

For example, suppose $\pi = 1_T$ is uniformly 1 for all terminals, and let $k = |T|$ be the number of terminals. Then $T$ being $1_T$-well-linked implies that we can simultaneously route $1/k$ units of flow between all pairs of terminals.

To further interpret this, let $H$ be the complete graph on $T$ with edge weights $1/k$; $H$ is a graphical representation of the demands $b(s, t) = 1/k$. $H$ is an expander has expansion $\geq 1/2$ (Why?) A (feasible) multicommodity flow routing of the demands $b(s, t) = 1/k$ for $s, t \in T$ can be interpreted as an embedding of $H$ into $G$. In a sense, $G$ contains an expander between $T$.

6.3.1 Routing well-linked terminals

The following lemma helps make explicit the intuition that well-linkedness facilitates routing.
Lemma 6.4. Let $T \subset V$ be $\pi$-well-linked for $\pi \in \mathbb{R}_{\geq 0}^T$. Let $b : (T^2) \rightarrow \mathbb{R}_{\geq 0}$ be a set of demands such that for all $s \in T$,

$$\sum_{t \in T} b(s, t) \leq \pi(s).$$

Then $b$ can be routed with congestion 2.

One way to interpret lemma 6.4 is via the following simple application. Let $T$ be 1-well-linked, and let $M \subset \binom{T}{2}$ be any matching on $T$. Then $M$ can be routed\(^3\) with congestion 2.

Proof of lemma 6.4. Decreasing $\pi$ as needed, we may assume that $\sum_{t \in T} b(s, t) = \pi(s)$ for all $s \in T$. Let $P = \sum_s \pi(s)$.

Let $\{g_{s,t} : s,t \in T\}$ by a multicommodity routing $\pi(s)\pi(t)/P$ units of flow for every pair $s, t \in T$. We will use 2 copies of $g$ to reroute $b$.

For every commodity pair $\{s,t\}$ with $b(s,t) > 0$, and every terminal $v \in T$,

consider the $(s,t)$-flow through $v$ where we route $b(s,t)\pi(v)/P$ flow between $s$ and $t$ using the $(s,v)$-flow $g_{s,v}$ and the $(v,t)$-flow $g_{v,t}$ from the $\pi$-product multicommodity flow $g$. For fixed $\{s,t\}$, summing over all $v$, this sends a total of $b(s,t) \sum_v \pi(v)/P = b(s,t)$ flow from $s$ to $t$. It remains to show that, summing over all of $\{s,t\}$, all of this rerouted flow fits in 2 copies of $g$.

For fixed $s$ and $v$, the total amount of flow to/from $s$ to $v$ with $v$ as the intermediary is

$$\sum_t b(s,t)\pi(v)/P = \pi(s)\pi(v)/P = |g_{s,v}|.$$ 

That is, one copy of $g_{s,v}$ fits all of the routed flow from $s$ to various $t$’s with $v$ as the midpoint (in the sense described above). Reversing roles between $v$ and $s$ gives another $\pi(v)\pi(s)/P = |g_{s,v}|$ units of flow along $g_{s,v}$. So in total we are sending $2\pi(s)\pi(v)/P$ units of flow between $s$ and $v$ which fits in two copies of $g_{s,v}$. All put together, we require two copies of the product flow $g$ which incurs congestion at most $2\alpha$.

---

\(^3\)Here "routing $M$" means routing the demands where $b(s,t) = 1$ for every edge $\{s,t\} \in M$, and 0 otherwise.
6.3.2 Matching well-linked terminals

Recall that in all-or-nothing flow, we want to either route a pair completely or not at all. This is similar in spirit to the following lemma, which says that when a set of terminals $T$ is $\pi$-well-linked for some $\pi$ with total vertex weight $P$, and $M$ is any given matching on $T$, one can route about $P$ of the pairs in $M$ with constant congestion. This lemma will be invoked in the base case of an eventual divide-and-conquer algorithm for all-or-nothing flow.

**Lemma 6.5.** Let $G = (V, E)$ be an undirected graph with minimum edge capacity 1, and let suppose $T \subset V$ is $\pi$-well-linked for $\pi : T \rightarrow [0, 1]$. Let $\mathcal{M} \subseteq \binom{T}{2}$ be any matching on $T$, and let $Q = \sum_{\{s,t\} \in \mathcal{M}} \min\{\pi(s), \pi(t)\}$. Then one can compute a subset $\mathcal{M}' \subseteq \mathcal{M}$ of size $\Omega(Q)$ that can be routed with congestion 3.

**Proof.** Without loss of generality we may assume $Q$ is bigger than a fixed constant\(^4\). Consider any spanning tree of this graph and root it arbitrarily. Working from bottom up, we can partition the edges of the tree to form edge disjoint subtrees with the following properties.

1. Each subtree is the endpoint to at most 2 units of flow.
2. All but one subtree is the endpoint to at least 1 units of flow. Henceforth we ignore this one subtree and drop any commodity intersecting with it.

![Diagram](image)

Now, pick any maximal subset of terminal pairs $T' \subseteq T$ so that we select at most one terminal from every subtree. Observe that $|T'| \geq \Omega(Q)$ because each matched pair eliminates at most a constant amount of $\pi$-weight of edges from $\mathcal{M}$ that are incident to their subtrees. Now we will show how to connect all the pairs in $T'$ with congestion at most 3.

Consider any pair $\{s,t\} \in T'$. Suppose $s$ is in a subtree $A$ containing a set of terminals $T_A$, and $t$ is in a subtree $B$ containing a set of terminals $T_B$. The high level idea is to route one unit of flow from $s$ to $t$ as follows. (We will explain and justify each step afterwards.)

1. Disperse 1 unit of flow through the subtree $A$ to the terminals $a \in T_A$, so that each $a \in T_A$ receives at most $F_a$ units of flow.

\(^4\)Otherwise choose any single edge and route it.
2. Send the total of 1 unit of flow from the terminals $T_A$ to the terminals $T_B$ so that the amount of flow received by each $b \in T_B$ is at most $F_b$. Here we will invoke the fact that the terminals are well-linked.

3. Send the 1 unit of flow through the subtree $B$ from the terminals $T_B$ into $t$.

For step 1, recall that by construction, we have $\sum_{a \in T_A} F_a \geq 1$. Thus we can easily allocate 1 unit of flow to $a \in T_A$ such that each $a$ receives at most $F_a$ unit of flow. Similarly, we can allocate one unit of flow to $T_B$ so that each $b \in T_B$ has at most $F_b$ units of flow. Let $p_a \leq F_A$ denote the amount of flow allocated to $a \in A$, and let $p_b \leq F_B$ denote the amount of flow allocated to $b \in B$. We have $\sum_{a \in A} p_a = 1$ and $\sum_{b \in B} p_b = 1$.

To execute step 2, consider the demands between $T_A$ and $T_B$ where each $a \in A$ is required to send $p_a p_b$ units of flow to each $b \in B$. Then each $a \in A$ has a total demand of $p_a \sum_{b \in T_B} p_b = p_a \leq F_a$ and each $b \in B$ has a total demand of $p_b \sum_{a \in T_A} p_a = p_b \leq F_b$.

Above we have described a set of demands for one pair $\{s, t\} \in \mathcal{F}'$. Consider all these demands over all pairs $\{s, t\} \in \mathcal{F}'$, summed together. Since each $\{s, t\} \in \mathcal{F}'$ involves distinct subtrees, we still have that every terminal $t \in T$ has total demand at most $F_t$. Since the terminals $T$ are well-linked, by lemma 6.4, we can simultaneously route, over all $\{s, t\} \in \mathcal{F}'$, the demand described above with congestion 2.

All put together, we route $\mathcal{F}'$ with congestion 3: congestion 1 from routing in the spanning trees, and congestion 2 from routing between the well-linked terminals. ■

We apply lemma 6.6 to round multicommodity flows as follows.

**Lemma 6.6.** Let $\{f_{s,t} : \{s,t\} \in \mathcal{F}\}$ be a multicommodity flow between distinct leaves $T$, and let $\pi : T \rightarrow \mathbb{R}_{\geq 0}$ denote the flow to/from each terminal (per (6.2)). Then one can compute a routable subset $\mathcal{F}' \subseteq \mathcal{F}$ of size $|\mathcal{F}'| \geq \Omega(\sum_{\{s,t\} \in \mathcal{F}} |f_{s,t}|)$.

**Proof.** If the total flow, $\sum_{\{s,t\} \in \mathcal{F}} |f_{s,t}|$, is less than some constant, then we can pick any pair $\{s, t\} \in \mathcal{F}$ and route just that pair. Otherwise, the terminal pairs $\mathcal{F}$ give a matching on $T$, and $T$ is $\pi$-well-linked, so by lemma 6.5, we can routeable set $\mathcal{F}' \subseteq \mathcal{F}$ of size $\Omega(\sum_{\{s,t\} \in \mathcal{F}} |f_{s,t}| - 1) = \Omega(\sum_{\{s,t\} \in \mathcal{F}} |f_{s,t}| - 1)$.

■
Exercise 6.1. Extend the argument of lemma 6.5 to show that for all \( \epsilon \in (0, 1) \), one can connect \( \Omega(\epsilon(P - 1)) \) pairs with congestion \( 1 + 2\epsilon \).

Remark 6.7. Chekuri, Khanna, and Shepherd [CKS13] show how to improve the clustering scheme so that the total congestion is at most 1, instead of \( 1 + 2\epsilon \).

6.3. Well-linked terminals

We move on to a dual perspective on well-linked terminals. If a set of terminals \( T \) is not \( \pi \)-well-linked, then there exists a sparse cut w/r/t the product demands induced by \( \pi \). This sets up an argument where we can partition the graph into well-linked parts so that the edges cut can be (somehow) charged to \( \pi \).

The first lemma is a primitive that analyzes a single sparse cut. It is convenient to instead ask for a \( (\pi/\alpha) \)-well linked set where \( \alpha \geq 1 \) is a parameter under our control.5

Lemma 6.8. Let \( G = (V, E) \) be an undirected graph, \( T \subset V \) a set of \( k \) terminals, and \( \pi : V \rightarrow \mathbb{R}_{>0} \). Let \( \alpha > 0 \) be a fixed parameters. Then one can either:

1. Certify that \( T \) is \( \pi/\alpha \)-well-linked.

2. Compute a partition \((V_1, V_2)\) of \( V \) such that, letting \( T_i = T \cap V_i \) and \( P_i = \sum_{t \in T_i} \pi(t) \) for \( i = 1, 2 \), we have

\[
c(\partial(V_i)) \leq O\left( \frac{\log(k)}{\alpha} \right) \min\{P_1, P_2\}.
\]

Proof. Let \( P = \sum_t \pi(t) \) be the total terminal weight. Let \( b \) be the product demands induced by \( \pi/\alpha \), defined by

\[
b(s, t) = \frac{\pi s \pi t}{\alpha P}.
\]

If \( b \) is routable with congestion \( \leq 1 \), this certifies that \( T \) is \( \pi/\alpha \)-well-linked. Otherwise, there is a cut with sparsity at most \( 1 \) w/r/t the demands \( b \). Recall that we have a \( O(\log k) \)-approximation for sparsest cut; invoking it, let \((V_1, V_2 = V \setminus V_1)\) induces a \( O(\log k) \) sparsest cut. We have

\[
c(\partial(V_1)) \leq O(\log k) \sum_{s \in V_1, t \in V_2} b(s, t)
\leq O(\log k) \frac{P_1 P_2}{\alpha P}
\leq O\left( \frac{\log k}{\alpha} \right) \min\{P_1, P_2\},
\]

as desired. \( \blacksquare \)

5Of course, it does not fundamentally change anything since really we are just rescaling \( \pi \).
Now, given a graph $G$ and vertex weights $\pi$, we can recursively apply lemma 6.8 to $G$ and its subgraphs until (the restriction to) each subgraph is either $(\rho_1/\alpha)$-well-linked or contains a small amount of vertex weight. This gives what Chekuri, Khanna, and Shepherd [CKS05] call a well-linked decomposition. What is most interesting is that the total capacity of edges cut is at most a $\tilde{O}(1/\alpha)$-fraction of the total vertex weight.

**Lemma 6.9 (Well-linked decomposition).** Let $G = (V, E)$ be an undirected graph, $T \subset V$ a set of terminals, and $\pi : V \to \mathbb{R}_{\geq 0}$. Then one can compute a partition $(V_1, \ldots, V_\ell)$ of $G$ with the following properties.

1. For each $i$, $T_i = T \cap V_i$, $\pi_i : T_i \to \mathbb{R}_{> 0}$ restrict $\pi$ to $T_i$, and letting $G_i = G|_{V_i}$ be the subgraph induced by $V_i$, either:
   
   (a) $T_i$ is $\pi_i/\alpha$-well-linked in $T_i$, or
   
   (b) $\sum_{t \in T_i} \pi_i \leq O(1)$.

2. The total capacity of edges cut by $(V_1, \ldots, V_\ell)$ is bounded above by
   
   $O(P \log(k) \log(P)/\alpha)$.

**Proof.** Initially starting from the trivial partition $(V)$ of $V$, we have, any point in time, partition $(V_1, \ldots, V_\ell)$ start with the trivial partition $(V)$. As long as some $V_i$ does not satisfy one of the two base conditions described above, we invoke lemma 6.8 on $G_i$ and $\pi_i$.

It remains to bound the total capacity of edges cut. Whenever we divide some $V_i$ into sets $V_j$ and $V_k$, the capacity of cut between $V_j$ and $V_k$ is at most $O(\log(k)/\alpha)$-times of the vertex weight of both $V_j$ and $V_k$; we charge the cut to the weight on the smaller side of the cut. More precisely, each unit of vertex weight on the smaller side is charged $O(\log(k)/\alpha)$ units of capacity cut.

Now, each unit of vertex weight appears can appear on the smaller side at most $O(\log P)$ times before reaching the base case where the total weight is a constant, since each such instance removes at least half the vertex weight. Thus each unit of vertex weight is charged a total of $O(\log(k) \log(P)/\alpha)$ units of capacity of edges cut. In total, then, at most $O(P \log(k) \log(P)/\alpha)$ capacity of edges are cut.

### 6.4 Approximating All-or-Nothing Flow

#### 6.4.1 An LP for All-or-Nothing Multicommodity Flow

We now begin our discussion specifically about all-or-nothing multicommodity flow. We will develop an LP rounding algorithm, starting from the following LP
6. Edge-Disjoint Paths and All-or-Nothing Multicommodity Flow

6.4. Approximating All-or-Nothing Flow

We point out that the final set of constraints, \( f_{s,t} \leq 1 \) for all \( \{s,t\} \in \mathcal{T} \), is redundant with the capacity constraints above it given that each terminal \( s \) is a leaf in \( G \) connected to the graph by an edge of capacity 1. If we drop these redundant constraints, then \((6.1)\) is exactly the demand multicommodity flow where every \( \{s,t\} \in \mathcal{T} \) has demand \( b(s,t) = 1 \). In particular, the dual LP is a relaxation of a sparsest cut problem where we want to minimize the ratio of capacity cut to divide terminal pairs \( \mathcal{T} \).

The goal now is to take a multicommodity flow \( F = \{f_{s,t}\} \) solving \((6.1)\) extract a set of routable set of terminals pairs \( S \subseteq \mathcal{T} \) of size \(|S|\) comparable to the total flow \( \sum_{\{s,t\} \in \mathcal{T}} |f_{s,t}| \).

6.4.2 A well-linked divide-and-conquer algorithm

We first give a high-level overview of the algorithm. Let \( F = \{f_{s,t}: \{s,t\} \in \mathcal{T}\} \) be an optimum multicommodity flow for \((6.1)\), of total size \( \text{OPT} \). We want to round \( F \) to an all-or-nothing multicommodity flow. This means we want to extract from \( F \) a set \( \mathcal{T}' \subseteq \mathcal{T} \), large as possible, such that \( \mathcal{T}' \) can be routed with congestion 1.

Given \( F \), let \( \pi: T \rightarrow \mathbb{R}_{\geq 0} \) denote the amount of flow routed for each terminal; to wit,

\[ \pi(s) \overset{\text{def}}{=} |f_{s,t}| \text{ where } \{s,t\} \in \mathcal{T} \text{ is the unique commodity for } s. \quad (6.2) \]

Let \( P = \sum_t \pi(t) = 2 \text{OPT} \). Let \( \alpha > 1 \) be a suitably large parameter. Intuitively, we have two cases.

1. \( T \) is \((\pi/\alpha)\)-flow linked, then \( T \) is to some extent well-connected, and this makes it easier to extract an all-or-nothing flow.

---

\(^6\)Here we recall that we can model the constraint that \( f_{s,t} \) is an \((s,t)\)-flow with \( O(n) \) flow conservation constraints. Here we use \( |f_{s,t}| \) to denote the size of the \((s,t)\)-flow \( f_{s,t} \), measured by either the net-flow leaving \( s \) or the net-flow entering \( t \).
2. If $T$ is not $(\pi/\alpha)$-flow linked, then (by flow-cut duality) there is a sparse cut w.r.t the product demands induced by $\pi$. The sparse cut divides the all-or-nothing flow problem into two problems and some flow is lost across the cut. We can try to use the sparsity of the cut to argue that we have not lost too much flow.

In principle, these two ideas play off each other in a divide-and-conquer approach. In the base case, the terminals are well-linked, and routing (rounding) is much easier. If not, then we can find a sparse cut and divide the problem into two smaller ones. We also divide the initial flow $F$ into smaller flows as commodity pairs and flow paths are split up by the smallest cut. Charging off the flow lost from dividing is a little subtle.

Actually, what we will do is a little more direct. We take the vertex weights $\pi$ and compute a well-linked decomposition $(V_1, \ldots, V_\ell)$ of $\pi$ with parameter $\alpha = O(\log(k) \log(\text{OPT}))$. Then the capacity of the cut is at most $\text{OPT}/2$ (say, for a sufficiently large constant hidden in $\alpha$). We will then argue that we can recover a multicommodity flow within each part so that the total flow loss is proportional to the edges cut. That is, we will obtain flows $F_i$ in each subgraph $G_i$ such that the total size of the flow is $\Omega(\text{OPT})$. We then invoke the well-linked-ness within each subgraph to route $\Omega(\text{OPT}/\alpha)$ total pairs.

**Theorem 6.10** (Chekuri, Khanna, and Shepherd [CKS05]). *In polynomial time, one can compute an all-or-nothing multicommodity flow of size $\Omega(\text{OPT}/\log(k) \log(\text{OPT}))$ with constant congestion.*

**Proof.** As described above, let $F$ be an optimum solution to (6.1) of size $\text{OPT}$, let $\pi$ be the vertex weights defined by (6.2), let $\alpha = O(\log(k) \log(\text{OPT}))$, and let $(V_1, \ldots, V_\ell)$ be a well-linked decomposition w.r.t $\pi$ and $\alpha$, per lemma 6.9. We assume the notation of lemma 6.9.

We construct, for each subgraph $G_i$, a multicommodity flow $F_i = \{f'_{i,t} : \{s,t\} \in \mathcal{T}_i\}$ in $G_i$ as follows. Fix a path decomposition of $f_{i,t}$. For each $i$, and each $\{s,t\} \in \mathcal{T}_i$, let $f'_{i,t}$ be the flow obtained from $f_{i,t}$ by retaining only the paths that lie entirely inside $V_i$. Then $F_i = \{f'_{i,t} : \{s,t\} \in \mathcal{T}_i\}$ gives a feasible multicommodity flow in $G_i$ over the terminal pairs in $\mathcal{T}_i$.

Next we relate the total flow in the subgraphs, $\sum_i \sum_{i,t} |f'_{i,t}|$ with the original flow, $|f_{i,t}|$. To this end, observe that every flow that is lost is from a flow path that goes across the multi-cut induced by the partition $(V_1, \ldots, V_\ell)$. In particular the total amount of flow is bounded above by the capacity of the multi-cut, which is at most $\text{OPT}/2$ by lemma 6.9. Thus $\sum_i \sum_{i,t} |f'_{i,t}| \geq \text{OPT}/2$.

Now, for each $i$, $G_i$ is $\pi_i/\alpha$-well-linked. Moreover, for each $\{s,t\} \in \mathcal{T}_i$, we have $|f'_{i,t}| \leq |f_{i,t}| = \pi_i(s) = \pi_i(t)$. By lemma 6.6, we can feasibly route a set $\mathcal{T}_i' \subseteq \mathcal{T}_i$ of
size $\mathcal{T}_i' \geq \Omega\left(\sum_{\{s,t\} \in \mathcal{T}_i} f_{s,t}' / \alpha\right)$ with congestion 3. Taking the union of $\mathcal{T}_i'$ over all $i$ gives a set of $\Omega(OPT / \alpha)$ terminals that can be routed with congestion 3. ■

Remark 6.11. Chekuri, Khanna, and Shepherd [CKS05] actually obtain congestion 1 (hence a proper approximation) but we only present constant congestion for simplicity. The key difference is to strengthen lemma 6.5 to only have congestion 1.

Theorem 6.10 leaves open the question of approximating the edge-disjoint path problem and there was several followup works in this direction. Later on, Chuzhoy [Chu12] obtained a polylogarithmic approximation with constant congestion, and then Chuzhoy and Li [CL12] obtained a polylogarithmic approximation with congestion 2. These works build on the well-linked framework discussed here and also involves significantly more graph theory building on the aforementioned work of Robertson and Seymour among others. See the survey by [Chu14].

6.5 Exercises

Exercise 6.2. Give a polynomial time reduction from $k$ edge-disjoint paths in undirected graphs to $k$ vertex disjoint paths in undirected graphs, thus proving Corollary ??.
Chapter 7

Cut-Matching

In multiple discussions to the sparsest cut problem so far, we had relied on an LP solver to produce a fractional solution to the metric relaxation, which is then rounded to a cut. This suffices to obtain polynomial time algorithms in theory, but sparsest cut is also of very interest. For large networks, the LP solver becomes a computational bottleneck. Meanwhile, in practice and in theory, we have fairly reliable $(s,t)$-flow algorithms. In this chapter we discuss an influential work by Khandekar, Rao, and Vazirani [KRV09] that computes a $O(\log^2 n)$-approximate sparsest cut via $O(\log^2 n)$ $(s,t)$-max flows, via an interesting framework called the cut-matching game. Besides the obvious appeal to efficiency, this forms a new conceptual connection between multicommodity flow and $(s,t)$-flow.

7.1 (Sparse) cuts and (expanding) matchings: some intuition

Speaking at a high-level, we find ourselves in the situation where sparsest cuts and multicommodity flow are slower to compute than $(s,t)$-flow. This is to some extent not surprising; multicommodity flow is computing an $(s,t)$-flow for as many as $\Omega(n^2)$ pairs $(s,t)$. Given the connection via duality, we might expect sparsest cut to be about as difficult as multicommodity flow.

Recall that the edge expansion is (up to scaling by $n$) within a factor 2 of the sparsity. For sake of discussion let us consider the decision version in terms of edge expansion: given a graph $G$ and parameter $\varphi$, we want to decide if the expansion of $G$ least $\varphi$ (at least approximately, up to logarithmic factors). If not, then we should be able to find a sparse cut; namely $S \subset V$ such that
If the graph has edge expansion at least \( \varphi \), how would we prove it? One approach is to compute a uniform multicommodity flow with congestion at most \( \tilde{O}(n\varphi) \). While this would certify expansion, it might be hard to make it fast since at some level we need to compute flows for \( \Omega(n^2) \) pairs. A sparser alternative comes from our randomized construction of expanders. Suppose we randomly sample a constant number of uniformly random perfect matchings \( M_1, \ldots, M_d \subset \binom{V}{2} \), and set the demand \( b(s, t) = 1 \) for any pair \( \{s, t\} \) matched in at least one of the matchings. With high probability, the union of the matchings is an expander (with constant expansion), in which case routing these demands with congestion \( \varphi \) certifies that \( G \) has expansion at least \( \varphi \). The advantage to this approach, compared to uniform demands, is that we are only trying to route \( O(n) \) pairs.

Now, suppose for simplicity that \( G \) has integral capacities and we want to verify if \( G \) has constant expansion (for some convenient constant). Let \( S \subset V \) half exactly half the vertices (\( |S| = n/2 \)). Observe that with a single \( \{s, t\} \)-flow, one can compute a perfect matching between \( (S, \bar{S}) \). Indeed, let us connect an auxiliary source \( s^* \) to every vertex \( s \in S \) with a directed edge \( (s^*, s) \) of unit capacity. Let us also connect, for every \( t \in \bar{S} \), an arc \( (t, t^*) \) with unit capacity. Then the flow paths of integral \( \{s, t\} \)-flow from \( s^* \) to \( t^* \) will implicitly match up vertices in \( S \) to vertices in \( \bar{S} \). If the max flow is less than a perfect matching, then we know that \( V \) is not \( .51 \)-well-linked in \( G \). But then \( G \) has expansion less than some constant.

If the \( \{s, t\} \)-flow does perfectly match \( S \) to \( \bar{S} \), then this only says that \( G \) is not necessarily not an expander. How to proceed? One idea is to keep selecting different sets \( S_1, S_2, S_3 \), each of size \( n/2 \), and using \( f_{i,t} \) to either route a perfect matching between each \( S_i \) and \( \bar{S}_i \), or prove that \( G \) has expansion at most a constant. If – big if – we show that these matchings behave like uniformly random matchings – then after a constant number \( k \) of sets \( S_1, \ldots, S_k \), we will have embedded a random expander with congestion \( d \) into \( G \) – certifying that \( G \) has expansion at least \( 1/k \). However we treat \( \{s, t\} \)-flow as a black box and there is no reason that the matchings should behave like random matchings. Instead, Khandekar, Rao, and Vazirani [KRV09] show how to adaptively select sets \( S_1, \ldots, S_k \) – where \( S_i \) depends on the matchings \( M_1, \ldots, M_{i-1} \) returned in rounds 1 through \( i - 1 \) – such that with high probability, after \( k = O\left(\log^2 n\right) \) rounds, the multigraph induced by the sum \( M_1, \ldots, M_k \) is an expander. This implies that \( G \) has expansion at least \( 1/k = \Omega\left(1/\log^2 n\right) \).

\[^{1}\text{Here } \tilde{O}(\cdots) \text{ hides polylogarithmic factors.}\]
7. Cut-Matching

7.2 Preliminaries

Let $G = (V, E)$ be an undirected graph with positive edge capacities $c : E \to \mathbb{R}_{>0}$. Assume for simplicity that the number of vertices, $n = |V|$, is even.

7.3 Setting up the cut-matching game

Theorem 7.1. Let $k = O\left(\log^2 n\right)$. There is an algorithm that, with $k$ $(s, t)$-max flow computations, with high probability returns either:

1. A cut with edge expansion $\leq 1$.

2. An expander $H$ on $V$ that can be embedded with congestion $O\left(\log^2 n\right)$, certifying that $G$ has expansion $\Omega\left(1/\log^2 n\right)$.

Before we mentioned that, if there is no feasible matching flow between $S$ and $\bar{S}$, then this should indicate that there $G$ is not a very good expander. Indeed, by multicommodity max-flow min-cut, there is a cut with edge expansion at most $O(\log n)$. The following lemma improves this bound to 1.

Lemma 7.2. Suppose there is no feasible matching flow between $(S, \bar{S})$. Then $G$ has expansion at most 1.

Proof. Then we have a minimum $(s^*, t^*)$-cut of size $< n/2$. In this cut, let $a$ be the number of edges from $s^*$, and let $b$ be the number of edges to $t^*$, and let $c$ be the remaining number of edges; we have $a + b + c < n/2$.

The $c$ non-auxiliary edges is also a cut in $G$, separates at least $(n/2 - a)$ vertices in $S$ from $(n/2 - b)$ vertices in $\bar{S}$. The expansion of this cut is at most

$$\frac{c}{\min\{n/2 - a, n/2 - b\}} < \frac{n/2 - a - b}{n/2 - \max\{a, b\}} \leq 1,$$

as desired. \hfill \blacksquare

If $G$ has edge expansion $\geq 1$, then, every bipartition produces a fractional perfect matching $M_i$ via an $\{s^*, t^*\}$-flow. Repeatedly doing so only implies that $G$ is not necessarily not an expander (which isn’t saying much). As a thought experiment, supposed the $M_i$’s behaved like random matchings. As we have seen, a constant number of matchings forms an expander, and embedding each of these matchings in $G$ implies that $G$ is an expander. Unfortunately, we cannot say that the $M_i$’s behave like random matchings since ultimately the matchings are out of our control. However, the following lemma – arguably the key lemma
in the entire argument – says that we can adaptively choose the bipartitions so that the sum of $M_i$’s form a matching after $O\left(\log^2 n\right)$ rounds, as follows.

**Lemma 7.3.** Consider an alternating two-player game where in each iteration $i$, the first player selects a bipartition $(S_i, \bar{S}_i)$ of $V$, and the second player returns a fractional matching $M_i \in [0,1]^{V \times V}$ between $S_i$ and $\bar{S}_i$. Let $k = O\left(\log^2 n\right)$. Then there is a randomized adaptive strategy for player 1 such that with high probability, the sum of the first $k$ matchings $M_1, \ldots, M_k$ forms an expander.

Lemma 7.3 will take some effort to prove. Let us first apply lemma 7.3 to prove theorem 7.1.

**Proof of theorem 7.1** If any $(S_i, \bar{S}_i)$ cannot be feasibly routed, then we have a cut with edge expansion at most 1. Otherwise, with high probability, the matchings induced by the matching flows combine to form an expander $H$. This implies that the sum of the $k$ matching flows (each treated as a weighted, undirected edge set) forms an expander $H$. Each matching flow fits in $G$, so the $k$ matching flows fit in $G$ scaled up by a factor of $k$. This means that $H$ can be embedded in $G$ with congestion $k$. ■

### 7.4 A randomized cut strategy

It remains to prove lemma 7.3. Our goal is to strategically select $(S_1, \bar{S}_1), \ldots, (S_k, \bar{S}_k)$ so that the sum of fractional matchings, $M_1 + \cdots + M_k$, is an expander. But how can we certify that the sum $M_1 + \cdots + M_k$ is an expander? The question might seem cyclical given that the original question is to decide if $G$ is an expander. But it is not cyclical because we will do something very specific to the structure of the matchings.

The idea is to use the $M_1 + \cdots + M_i$ route a (directed) multicommodity flow where we send (about) $1/n$ units of flow between every pair of vertices. This implies that we can embed the weighted clique, $K_n/n$ into $M_1 + \cdots + M_i$. $K_n/n$ has edge expansion 1, which also certifies that $M_1 + \cdots + M_i$.

In the following, we treat a fractional matching $M$ as a weighted set of edges. We let $M_{s,t}$ denote the weight of the edge between $s$ and $t$ (which is 0 when there is no edge).

We build out this multicommodity flow incrementally, one matching at a time. We start with a trivial flow where each vertex sends one unit of flow to itself. Each successive matching $M$ is used to disperse (and hopefully diversify) the flow.

At the beginning of a generic iteration, we have a multicommodity flow for again each vertex sends and receives one unit of flow. Given the new matching
We update the flow as follows. For each pair \( \{s, t\} \) (simultaneously), out of the 1 total unit of flow terminating at \( s \) (from various sources), we send a \( M_{s,t}/2 \)-fraction of each commodity to \( t \); we also do the same for the 1 total unit of flow terminating at \( t \), sending \( M_{s,t}/2 \)-fraction of each commodity at \( s \).

In a sense, we are using the fractional perfect matching the “mix up” the existing flow, hoping to produce a multicommodity flow that is closer to uniform.

We will choose the bipartition to maximize the “mixing”.

Suppose, at the end of a fixed iteration, we let \( b_s \in [0,1]^V \) be the stochastic vector where each \( b_s(t) \) is the total amount of flow from \( t \) to \( s \) routed by the preceding matchings. Ideally we want \( b_s(t) = 1/n \) for all \( s \) and \( t \). Given the next matching \( M \), \( b_s \) is replaced by the weighted average, \( b_s/2 + \sum_t M_{s,t}b_t/2 \). In a sense, we are taking weighted averages of the \( b_s \)'s as dictated by \( M \). In principal, pairing up very different \( b_s \)'s should produce averages closer to uniform.

Let us define a potential function by

\[
\Phi = \sum_{(s,t)} (b_s(t) - 1/n)^2.
\]

\( \Phi \) is a measure of difference between the multicommodity flow routed so far and a fully symmetric multicommodity flow. Observe that if \( \Phi \leq 1/4n^2 \), then \( |g_{s,t}| \geq 1/2n \) for all \( s,t \). This implies that \( K_n/n \) can be embedded in the matchings, and the matchings form an expander.

Abstracting out the graphs and flows, let \( x_s = b_s - 1/n \) for each \( s \). Then \( \sum_s x_s = 0 \) and \( \Phi = \sum_s \|x_s\|^2 \). Given a prefect matching \( M \), we update the \( x_s \)'s to vectors \( y_s \) where

\[
y_s = \frac{1}{2} x_s + \frac{1}{2} \sum_t M_{s,t} x_t.
\]

We want to

(a) Choose a bipartition \((S, \bar{S})\) that minimizes:

(b) the maximum, over all fractional perfects matchings \( M \) of \((S, \bar{S})\), of:

(c) the sum \( \sum_s \|y_s\|^2 \).

To find a good strategy, we work backwards.

1. First we analyze \( \sum_s \|y_s\|^2 \) as a function of \( M \).

2. We consider the simpler case where each \( x_s \) is just a scalar value; then the (intuitive) solution is to split the vertices by the smaller and larger halves. With the help of the first step, we analyze the reduction in \( \sum_s \|y_s\|^2 \) for this special case.
3. We use random projections to (more or less) reduce to the 1-dimensional case.

Here is step 1.

**Lemma 7.4.** For each vertex \( s \in V \), let \( x_s \in \mathbb{R}^k \) be a fixed vector. Let \( M \in [0,1]^{V \times V} \) encode a fractional perfect matching. For each vertex \( s \in V \), let

\[
y_s = \frac{1}{2} x_s + \frac{1}{2} \sum_{t \in V} M_{s,t} x_t.
\]

Then

\[
\sum_s \|y_s\|^2 \leq \sum_s \|x_s\|^2 - \frac{1}{2} \sum_{\{s,t\}} M_{s,t} \|x_s - x_t\|^2.
\]

**Proof.** First consider a fixed \( s \). We have

\[
\|y_s\|^2 = \left\| \frac{1}{2} x_s + \frac{1}{2} \sum_t M_{s,t} x_t \right\|^2
\]

\[=
\frac{1}{4} \|x_s\|^2 + \sum_t \|M_{s,t} x_t\|^2/4 + \frac{1}{2} \sum_t M_{s,t} (x_s, x_t)
\]

\[=
\frac{1}{4} \|x_s\|^2 + \sum_t \|M_{s,t} x_t\|^2/4 + \frac{1}{4} \sum_t M_{s,t} (\|x_s\|^2 + \|x_t\|^2 - \|x_s - x_t\|^2)
\]

\[
\leq (a) \frac{1}{2} \|x_s\|^2 + \frac{1}{2} \sum_t M_{s,t} \|x_t\|^2 - \frac{1}{4} \sum_t M_{s,t} \|x_s - x_t\|^2
\]

Here in (a) we use the facts that \( \sum_t M_{s,t} = 1 \) and \( M_{s,t}^2 \leq M_{s,t} \). Now, summing over \( s \), we have

\[
\sum_s \|y_s\|^2 \leq \frac{1}{2} \sum_s \|x_s\|^2 + \frac{1}{2} \sum_{s,t} M_{s,t} \|x_t\|^2 - \frac{1}{4} \sum_{s,t} M_{s,t} \|x_s - x_t\|^2
\]

\[=
\sum_s \|x_s\|^2 - \frac{1}{4} \sum_{s,t} M_{s,t} \|x_s - x_t\|^2,
\]

as desired. \( \blacksquare \)

Here is step 2.

**Lemma 7.5.** For each vertex \( s \in V \), let \( u_s \in \mathbb{R} \) be a fixed scalar. Let \( (S,T) \) partition \( V \) in half where \( S \) is the set of the smallest \( n/2 \) values and \( T \) is the set of the largest \( n/2 \) values (with ties broken arbitrarily). Then for any fractional perfect matching \( M \) between \( S \) and \( T \),

\[
\sum_{\{s,t\} \in M} M_{s,t} (u_s - u_t)^2 \geq \sum_v u_v^2.
\]
Proof. The claim is translation invariant, so without loss generality, suppose \( u_s \leq 0 \) for all \( s \in S \) and \( u_t \geq 0 \) for all \( t \in T \). For any perfect matching \( M \) between \( S \) and \( T \), we have

\[
\sum_{\{s,t\}} M_{s,t} (u_s - u_t)^2 = \sum_{\{s,t\}} M_{s,t} \left(u_s^2 - 2u_s u_t + u_t^2\right) \\
\overset{(a)}{=} \sum_{v} u_v^2 - 2 \sum_{\{s,t\}} u_s u_t \overset{(b)}{\geq} \sum_{v} u_v^2,
\]

where (a) is because \( \sum_{t} M_{s,t} = 1 \) for all fixed \( s \), and \( u_s u_t \leq 0 \) for all pairs \( \{s,t\} \) with \( M_{s,t} > 0 \).

Now, for step 3, we require the following lemma about Gaussian vectors. The first property is really the fact that the variances of Gaussians scale quadratically, and that the variance of independent Gaussians sum nicely. The second is a tail inequality for the Gaussian distribution.

**Lemma 7.6.** Let \( p \in \mathcal{N}^n(0,1) \) be a random Gaussian vector. For all \( x \in \mathbb{R}^n \), \( \langle x, p \rangle \) is a Gaussian with mean 0 and variance \( \|x\|^2 \). Consequently:

1. For all \( x \in \mathbb{R}^n \), \( \mathbb{E} \left[ \sum_{s} y_s^2 \right] \geq \|x\|^2 \).

2. For all \( x \in \mathbb{R}^n \), with high probability, \( \langle p, x \rangle^2 \geq \|x\|^2 / c \log n \) for some constant \( c > 1 \).

To complete the proof, let \( \alpha = \Omega(1/\log n) \). We have

\[
\mathbb{E} \left[ \sum_{\{s,t\}} M_{s,t} \|x_s - x_t\|^2 \right] \geq \alpha \mathbb{E} \left[ \sum_{\{s,t\}} \|u_s - u_t\|^2 \right] \geq \alpha \mathbb{E} \left[ \sum_{v} \|u_v\|^2 \right] = \alpha \sum_{v} \|x_v\|^2.
\]

In turn we have

\[
\mathbb{E} \left[ \sum_{s} \|y_s\|^2 \right] \leq (1 - \alpha) \sum_{s} \|x_s\|^2.
\]

It follows that we need \( \log_{1+\alpha}(\text{poly}(n)) = O\left(\log^2 n\right) \) iterations to reduce the potential from \( O(n) \) to \( 1/n^2 \).
Chapter 8

Tree Metrics

8.1 Introduction

All of our problems involve paths in graphs, whether packing paths in flow, cutting paths in cuts, or evaluating shortest paths in fractional cuts. Obviously there are many paths between any two points in a graph and this makes all these problems nontrivial. An extremely simple setting, then, would be a graph where there is a unique path between any two vertices: by definition, a tree. Any of our graph problems would have been trivial in a tree. Here we will study a bold approach to graph algorithms based on this idea: processing the input graph $G$ to produce a tree $T$ that somehow reflects $G$ and preserves its salient properties, solve the problem on $T$, and transfer the solution back to $G$. Of course a tree $T$ cannot preserve all of $G$, so we will only preserve specific properties and only approximately at that, in such a way that is appropriate to the problem at hand.

In this discussion, we will focus on preserving the shortest path metric of a graph. Let $G$ be an undirected graph $G$, $d_G$ denote the shortest path metric in $G$. Let $T$ be a spanning tree of $T$ (with the same edge weights), and let $d_T$ denote the shortest path metric in $T$. (Of course, the shortest path in $T$ is also the only path.) Ideally, we want $d_T(u,v)$ to resemble $d_G(u,v)$ as much as possible for each edge $e = \{u,v\}$. In general, we have

$$d_G(u,v) \leq d_T(u,v) \text{ for all } u,v \in V$$

simply because $T$ is a subgraph of $G$. For an edge $e = \{u,v\}$, we say that the stretch of $e$ is defined as the ratio

$$(\text{stretch } e) = \frac{d_T(u,v)}{d_G(u,v)}.$$

We say that $d_T$ has uniform stretch (at most) $\alpha$, for $\alpha \geq 1$, if every edge has stretch at most $\alpha$. 

114
A natural goal is to obtain a spanning tree with small uniform stretch, but the $n$-vertex cycle $C_n$ presents a lower bound of $n - 1$. Indeed, any spanning tree $T$ of $C_n$ is obtained by dropping one edge $e$; this edge $e$ is stretched around the cycle, so to speak, and has stretch $n - 1$.

The $n$-vertex cycle $C_n$ indicates that we cannot, in the worst-case, find spanning trees with uniform stretch better than $n - 1$. (We leave it as ?? to obtain a matching upper bound.) The rest of this chapter discusses two different approaches that obtain better bounds for relaxations of this problem.

**Low-stretch spanning trees.** Alon, Karp, Peleg, and West [AKPW95] showed how to compute a spanning tree $T$ where the average stretch among all edges is $n^{o(1)}$, in the sense that

$$\frac{1}{|E|} \sum_{e \in E} (\text{stretch } e) \leq n^{o(1)}$$

**Dominating tree metrics.** Bartal [Bar96; Bar98] ignored the requirement that $T$ is a spanning tree of $G$; more general he sought auxiliary trees $T$ where the vertex of $G$ correspond to the leaves of $G$, while retaining the property $T$ that $d_T \geq d_G$. He produced randomized trees where for each edge the average stretch was $\text{polylog}(n)$:

$$\mathbb{E}_T[(\text{stretch } e)] \leq O(\text{polylog}(n)) \text{ for all } e \in E.$$ 

Note that this a different sense of “average stretch” then above. We will present an algorithm of [FRT04] building on [Bar98] to obtain (per-edge) average stretch of $O(\log n)$.

### 8.2 Low-Stretch Spanning Trees

We first present the low-stretch spanning trees of [AKPW95]. Here we recall that we want to compute a spanning tree

Suppose our goal was to obtain average stretch (roughly) $D$ for a parameter $D > 0$. Let us partition the graph in vertex-disjoint subgraphs each with radius at most $D/2$ from some center vertex, and compute a shortest path tree from each center. Then every edge within a neighborhood has stretch at most $D$; it remains to address the edges that are cut by the partition. We can try to address these edges recursively by contracting every subgraph/subtree into a single vertex, leaving a multigraph $G'$ consisting of the cut edges, and recursing on this graph. This produces a tree $T'$ on the contracted multigraph $G'$; expanding out the vertices of $T'$ by the underlying shortest path trees gives a spanning tree $T$. 


[AKPW95] algorithm for $m^{o(1)}$-average stretch spanning trees:

1. **Compute a low diameter decomposition:** repeatedly, until no vertices remain:
   (a) Select a remaining vertex $v$ and compute a shortest path tree of $v$ in the remaining graph.
   (b) Remove the set $C_v$ all vertices (remaining) in $V$ within some distance $R' \leq R$ such that
   \[ |\partial(C_v)| \leq O(\log(m)) (1 + |E[C_v]|) \]
   where $E[C_v]$ denotes the set of all (remaining) edges incident to some vertex in $v$.

2. **Recurse:** Let $G'$ be the multi-graph obtained from the input graph by contracting each $C_v$ to a single vertex. Recurse on $G'$ to obtain a spanning tree $T'$, and return the tree $T$ obtained by replacing each $C_v$ with the corresponding shortest path tree.

Recursively, we might expect that every edge cut edge $e$ has stretch $O(R)$ in $T'$, but this expands out to stretch $O(R^2)$ with respect to $T$ because passing through a vertex in $T'$ actually corresponds to traversing a path of length $O(R)$ in the underlying tree.

So we have a problem where each step of the recursion induces an additional factor of $R$. Now, recall that we want to preserve average stretch. Let $f(m)$ be the stretch obtained by the recursive approach. We have

\[ f(m) \leq Dm + (D + 1)f(c \log m) \]

Studying this recursion, we can see that if the number of external edges was extremely small, then we might hope that it can offset the extra factor of $D$. How can we minimize the number of external, or cut, edges? Region growing! Low diameter decompositions!

If we partition the graph by region growing techniques, we can ensure that

\[ (# \text{ external edges}) \leq \frac{O(\log m)}{R} m. \]

This revises the recursive bound as

\[ f(m) \leq Dm + (D + 1)f(\epsilon \log m) \]

for a constant $\epsilon > 0$. For $D = \epsilon^\sqrt{\log(m)/\log \log m}$ the recursion is bounded by

\[ f(m) = \epsilon^O(\sqrt{\log m \log \log m}) \]

(The calculations are given below.)
Theorem 8.1. The AKPW algorithm returns a spanning tree with average stretch $e^{O(\sqrt{\log m \log \log m})}$.

**Solving the recursion.** We have

$$f(m) \leq Dm + (D + 1)f(c_0 \log(m)m/D)$$

where $\epsilon = c_0 \log(m)/D$ for a constant $c_0$. The height of the recursion is

$$h = O\left(\log_{D/c_0 \log(m)} m\right) = O\left(\log(m)/\log(D/c_0 \log(m))\right) = O\left(\frac{\log(m)}{\log(D)}\right)$$

assuming $D = \Omega(\log m)$. Unrolling the recursion gives

$$f(m) \leq O(Dm) \sum_{i=0}^{h} (1 + 1/D)c_0 \log(m))^i \leq Dme^{O(h \log \log m)}.$$  

To minimize the RHS, we can instead minimize the logarithm of the RHS, $\log(m) + \log(D) + O(h \log \log m)$. Choosing $D$ to make the last two terms (roughly) equal, we have

$$\log(D) = \frac{\log(m) \log \log(m)}{\log(D)},$$

hence

$$\log(D) = \sqrt{\log(m) \log \log(m)}.$$  

Then $D = e^{\sqrt{\log(m) \log \log(m)}}$ gives

$$f(m) \leq me^{O(\sqrt{\log(m) \log \log(m)})},$$

as desired.

### 8.3 Hierarchical Tree Metrics

We now discuss...

The [AKPW95] algorithm was able to obtain low stretch *in total*. This means that a few unlucky edges might have extremely high stretch. In this section we want every edge to have low stretch, in some sense. Unfortunately we already know that it is impossible to guarantee $o(n)$ stretch for every edge simultaneously.
But a different possibility when we allow for randomization. Perhaps we can output a randomized tree $T$, such that for each edge $e$, the expected value of $e$’s stretch is $o(n)$. Such a claim does not contradict the lower bound for the $n$-vertex cycle; in fact, one can get constant stretch for the cycle which we leave as $\square$.

To build some intuition, let us start from the [AKPW95] algorithm for inspiration (even though ultimately we will not produce a spanning tree). A high level goal is to inject randomization so that every edge has a decent chance at having low stretch. To this end there are at least two natural ways to introduce randomization into [AKPW95].

1. We can make the radii of the clusters randomized, instead of a deterministic function of the total number of edges cut.

2. Second, the order of vertices that center the clusters can be randomized, which would seem most equitable.

Both of these ideas will be reflected at a high level in the following algorithm which we now present.

The algorithm we present will produced a randomized hierarchical tree metric over $V$. This means that the tree $T$ will be rooted, with $V$ at the leaves, and the edges between height $i$ and height $i-1$ have length $\alpha^i$ for a fixed constant $\alpha$. Here we choose $\alpha = 4$ to simplify calculations, though we note that $\alpha = 2$ is more common, and the analysis can be adjusted to accommodate any fixed constant. The convenience of a hierarchical tree $T$ is that the tree distance $d_T(u,v)$ is entirely determined by the height of their least common ancestor, and within a factor of the biggest edges at the top of the corresponding subtree. This additional structure turns out to be useful for several other problems.

We now present [FRT04]’s randomized algorithm. We assume the input is an edge-weighted graph where the minimum edge length is normalized to 1. We let $D = \max_{u,v} d(u,v)$ denote the diameter of the graph. Below we describe the algorithm in detail and first we give a high level description. For every radius of the form $\alpha 4^i$, we are randomly scooping out balls of size $\alpha 4^i$, where $\alpha \in [1,2]$ is drawn uniformly at random, and the vertices at the center of the balls are in random order. A key and subtle point is that we use the same (random) $\alpha$ and ordering for every $i$. the intersections of these balls (across $i$’s) induce a laminar family of sets over $V$ which are arranged as a tree.

[FRT04]’s algorithm producing a randomized hierarchical tree metric.

1. Let $v_1, \ldots, v_n$ be a uniformly random ordering of $V$. Let $L = \lceil \log_4 D \rceil$. Let $\alpha \in [1,2]$ be drawn uniformly at random.
2. For $i$ from $L$ down to 0,
   (a) For each vertex $v_j$ in order,
      i. Let $C_{i,j}$ be the set of vertices at distance at most $\alpha 4^{i-1}$
         from $v_j$, excluding any vertex already included by the
         cluster of a previous $C_{i,j}$.
   3. We use the $C_{i,j}$’s to arrange the vertices as leaves in a tree $T$
      hierarchically as follows. For each intermediate node $x$ at height
      $i$, the leaves in the subtree rooted at $x$ corresponds to a set
      of vertices with diameter at most $4^i$. The root at height $L+1$
      corresponds to $V$. The nodes at height $L$ correspond to the
      clusters $C_{L,j}$. In general, for a node $x$ at height $i$ corresponding
      to a set $S \subseteq V$, its children correspond to the (nonempty)
      intersections of $S$ with clusters $C_{i,j}$. (Here a cluster center
      $v_j$ may not be in $S$.) Observe the leaves (at height 0) each
      correspond to a single vertex $V$ because $C_{0,j} = \{v_j\}$ for all $j$.
      Each edge descending from height $i$ is given weight $4^i$.

So much for the algorithm. Here, then, is the key claim.

**Theorem 8.2.** [FRT04]’s algorithm produces a randomized hierarchical tree $T$
such that for each edge $e$, $\mathbb{E}[(\text{stretch } e)] \leq O(\log n)$.

To prove the theorem, fix an edge $e = \{u, v\}$. Recall that $d(u, v)$ is decided,
up to a constant factor, by the height $k$ where $u$ and $v$ are first separated. (Then
$d(u, v) = O(4^k)$.) When this occurs, $u$ and $v$ are separated in particular by a
cluster of radius $\alpha 4^k$ centered at some vertex $w$; in this event, we say that $w$
“contributes” $4^k$ to $d(u, v)$ (which upper bounds the diameter of the remaining
vertices). By this terminology, we have

$$d_T(u, v) \leq \sum_w O(1)(\text{contrib. of } w \text{ to } d_T(u, v)). \quad (8.1)$$

Now, fix a vertex $w$. Suppose that $w$ was the $\ell$th closest vertex to $u$ or $v$ (i.e.,
with respect to $\min\{d(w, u), d(w, v)\}$). The key lemma, which we analyze below,
is that

$$\mathbb{E}[\text{contrib. from } w \text{ to } d_T(u, v)] \leq O(1/\ell) d(u, v).$$

Taking expectations of eq. (8.1) and applying the bound above to each $w$ gives
$O(\log n)$ stretch, as desired.

It remains to prove the key lemma, as follows.
Lemma 8.3. Let \( w \) be the \( \ell \)th closest vertex to \( u \) or \( v \). Then

\[
\mathbb{E}[\text{contrib. from } w \text{ to } d_T(u,v)] \leq 4d(u,v)/\ell.
\]

Proof. We first observe that \( w \) contributes to \( d_T(u,v) \) only if the following two events both occur.

\( E_1 \). \( d(w,u) \leq \alpha 4^k \leq d(w,v) \) for some \( k \).

\( E_2 \). \( w \) is ordered before any of the \( \ell - 1 \) vertices that are closer to \( u \) or \( v \).

Indeed, the necessity of the first condition is clear. The second is necessary because if any closer vertex would otherwise cluster either \( u \) or \( v \) (or both) before \( w \).

We also observe that the above conditions are independent, since the first event depends (only) on \( \alpha \) and the second event depends on the random ordering, which are independent. It is also clear that the second event \( E_2 \) occurs with probability \( 1/\ell \). It remains to analyze \( E_1 \). We have two cases.

Case 1: \( d(u,v) \geq 4d(w,u) \). Then for any \( k \) satisfying the inequality in \( E_1 \), this inequality and the triangle inequality imply that

\[
4^{k+1} \leq 2d(w,v) \leq 2(d(w,u) + d(u,v)) \leq (5/2)d(u,v).
\]

Thus the contribution from \( w \) is at most \( O(d(u,v)) \). It follows that

\[
\mathbb{E}[\text{contrib. from } w \text{ to } d_T(u,v)] \leq 2.5d(u,v)\mathbb{P}[E_2] = 2.5d(u,v)/\ell,
\]

as desired.

Case 2: \( d(u,v) \leq d(w,u) \). We first note that there may not be any \( k \) in item \( E_1 \) satisfying inequality \( E_1 \); if not, then the claim is immediate. Henceforth we assume such a \( k \) exists. We claim that the choice of \( k \) is unique. Indeed, suppose there exists some choice of \( k \) and \( \alpha \in [1,2] \) such that the inequality in \( E_1 \) holds. We have

\[
d(w,u) \geq d(w,v) - d(u,v) \overset{(a)}{=} 4^k - 4^k/2 > 24^{k-1}
\]

which rules out smaller values of \( k \). Here (a) is by the the triangle inequality and (b) is by assumption on \( \alpha \). To rule out larger values of \( k \), we have

\[
d(w,v) \leq d(w,u) + d(u,v) < 4^{k+1}
\]

by similar reasoning.
Thus the choice of $k$ in $E_1$ is unique; fix $k$ as such. Now we have

$$P[E_1] \leq \frac{|d(w, u), d(w, v)|}{|4^k, 24^k|} \leq \frac{d(u, v)}{4^k}$$

by (c) the triangle inequality. Thus

$$E[\text{contrib. from } w \ldots] \leq 4^{k-1} P[E_1] P[E_2] \leq 4d(u, v)/\ell,$$

as desired.

### 8.4 Exercises

**Exercise 8.1.** Design and analyze an algorithm that computes a spanning tree with uniform stretch $n - 1$ (matching the lower bound induced by the cycle).

**Exercise 8.2.** Recall that the low-stretch spanning tree of [AKPW95] obtained an average stretch $n^{o(1)}$. We consider extensions to the weighted average. Let $w(e) \in \mathbb{R}_{>0}$ be a positive weight for every edge, and let $W = \sum_{e \in E} w(e)$ be the total weight. We assume for simplicity that the weights are between 1 and poly$(n)$. We still treat the edges as unit length edges. Design and analyze and algorithm to compute a spanning tree $T$ such that

$$\frac{1}{W} \sum_{e \in E} w(e)(\text{stretch } e) \leq n^{o(1)}.$$

**Exercise 8.3.** Show how to use the randomized tree metric to randomly round the sparsest cut LP and obtain $O(\log n)$-approximation for sparsest cut.

**Exercise 8.4.** Prove that the $O(\log n)$ bound is tight for tree metrics (up to constants).

**Exercise 8.5.** The randomized tree metric produces a distribution such that each edge has randomized stretch that is $O(\log n)$ in expectation. Design an algorithm that produces a distribution over spanning trees such that each edge has stretch $n^{o(1)}$ in expectation.\(^1\)

\(^1\)Hint: Consider the techniques of chapter 9.
Chapter 9

LP duality by approximation

9.1 Introduction

Throughout our discussions we have often looked at LP duality for inspiration and insight. We briefly recall the setting. Let \( A \in \mathbb{R}^{m \times n} \), \( b \in \mathbb{R}^n \), and \( c \in \mathbb{R}^m \). These induce a (pure) packing LP of the form

\[
\text{maximize } \langle b, x \rangle \text{ over } x \in \mathbb{R}^n_{\geq 0} \text{ s.t. } Ax \leq c, \quad (P)
\]

and a dual (pure) covering LP of the form

\[
\text{minimize } \langle c, y \rangle \text{ over } y \in \mathbb{R}^m_{\geq 0} \text{ s.t. } A^T x \geq b. \quad (C)
\]

Abusing notation, we let \((P)\) and \((C)\) also denote the optimum values of the corresponding problems. It is easy to see that \((P) \leq (C)\); LP duality gives the opposite inequality which is nontrivial.

**Theorem 9.1** (LP Duality Theorem). \((P) = (C)\).

In this chapter we will prove the LP duality theorem. Our proof will be (essentially) algorithmic; more precisely, it will lead to an efficient approximation algorithm for solving \((P)\) and \((C)\) as well.

9.2 Lagrangian Relaxations

There are multiple ways to prove LP duality and our presentation will conceptually center on Lagrangian relaxations\(^1\). Given an instance of packing, and a set of

\(^1\)We note that there are other types of Lagrangian relaxations.
nonnegative weights \( w \in \mathbb{R}^m_{\geq 0} \) over the constraints, consider the following one-dimensional packing problem:

\[
\text{maximize } \langle b, x \rangle \text{ over } x \in \mathbb{R}^n_{\geq 0} \text{ s.t. } \langle w, Ax \rangle \leq \langle w, c \rangle. 
\]

As a relaxation of (P), we have (L) \( \geq \) (P), and in general the gap is unbounded. The advantage of (L) is in the simplicity of the constraints. Taking the transpose, we have \( \langle w, Ax \rangle = A^T w, x \), and each \( (A^T w)_i \) can be interpreted as the cost of coordinate \( i \). Meanwhile each \( b_i \) can be understood as the profit of coordinate \( i \), and \( \langle w, c \rangle \) represents a budget. To maximize the total profit subject to a single cost constraint, one simple greedy solution is to identify the coordinate \( i \) maximizing the “bang-for-buck” ratio \( b_i / (A^T w)_i \), and taking as much as possible that fits the budget. This is in contrast to the original problem (P) which seems hard to compute.

We already know that (P) \( \leq \) (C) and (P) \( \leq \) (L) and these inequalities are not difficult to show. The connection between (C) and (L) is not so clear.

**Lemma 9.2.** For all \( w \in \mathbb{R}^m_{\geq 0} \), (L) \( \geq \) (C).

**Proof.** Rescaling, we can assume that \( b = 1, c = 1 \). By rescaling we may also assume that \( \langle 1, w \rangle = 1 \), in which case \( \langle w, c \rangle = 1 \). Now, let \( \lambda = (L) \). We claim that \( \lambda w \) is a feasible solution for (C), which implies that that \( \lambda = \langle \lambda w, c \rangle \geq (C) \), as desired.

Fix \( i \in [m] \); we need to show that \( \lambda w \) covers the \( i \)th constraint. To this end, let \( x = \lambda e_i \). Since \( \langle 1, x \rangle = \lambda \), we must have \( \langle p, Ax \rangle \geq \langle w, x \rangle \). (Otherwise scaling up \( x \) slightly gives a better-than-opt solution.) Now we have

\[
(A^T (\lambda w))_i = \lambda \langle w, A e_i \rangle = \langle p, Ax \rangle \geq 1
\]
as desired. \( \square \)

Lemma 9.2 is interesting. We know of course that (C) \( \geq \) (P), and we don’t know how to show that (C) \( \geq \) (P). Meanwhile lemma 9.2 says that for every relaxation (L) of (P), (L) \( \geq \) (C). Question: does (L) \( \geq \) (C) for all relaxations (L) of (P) imply that (P) \( \geq \) (C)?

### 9.3 A continuous algorithm

The next part of our discussion presents an idealized, “continuous” algorithm for solving (P). We will show that the solution produced by this algorithm has

---

2 Ultimately we replace each \( A_{ij} \) with \( A_{ij} / c_i b_j \). This does not affect (P), (C), or (L).

3 Here \( e_i \) is the \( i \)th standard basis vector, where the \( i \)th coordinate is 1 and all other coordinates are 0.
value (arbitrarily close to) \((C)\), proving LP modulo some mathematical details to remove the ideal assumptions.

To simplify our discussion, we assume the problems are rescaled so that \(b = 1\) and \(c = 1\). This assumption can be without loss of generality for the sake of proving LP duality. Hopefully the presentation will be simple enough that the reader can reproduce the algorithm and analysis without the normalization, in case that is convenient for concrete problems of interest.

We now describe the general continuous framework. We have a time variable \(t\) that goes starts at \(t = 0\) and increases continuously \(t = 1\). We maintain a feasible solution \(x\) that starts at \(x = 0\) at time \(t = 0\) and increases continuously until \(t = 1\). We let \(x(t)\) denote the value of \(x\) at time \(t\). At each time instance (in this idealized model of computation), the algorithm must decide the derivative \(dx(t)/dt\) that (by definition of the gradient) dictates the direction in which \(x(t)\) will grow. In addition to preserving feasibility (i.e., \(Ax \leq 1\)), we will select \(dx(t)/dt\) to be nonnegative and satisfy \(\langle 1, dx(t)/dt \rangle \geq (C)\) for all \(t\). (In fact, each \(dx(t)/dt\) will be the solution of some relaxation \((L)\) of \((P)\), which is at least \((C)\) by lemma 9.2.) In principle, at time \(t = 1\), \(x(1)\) is a feasible solution with \(\langle 1, x(1) \rangle = \langle 1, x(0) \rangle + \int_0^1 \langle 1, dx(t)/dt \rangle dt \geq \int_0^1 (C) dt = (C)\).

But then, as \(x(1)\) is a feasible solution to \((P)\), we also have \((P) \geq \langle 1, x(1) \rangle \geq (C)\). (1)

The primary goal of this continuous algorithm is to serve as a prototype for discrete algorithm later. Consequently we ignore for the time being some formal technicalities. For example, the integral over \(dx(t)/dt\) is not well defined when \(dx(t)/dt\) is not continuous in \(t\). But this issue will dissolve automatically when we discretize the continuous algorithm and \(x(t)\) becomes a stepwise linear function of \(t\).

All that said, the main design decision within this framework is in choosing a direction \(dx(t)/dt \geq 0\) at each time \(t\). Here we want to balance two requirements:

1. \(\langle 1, dx(t)/dt \rangle \geq (C)\).

2. \(dx(t)/dt\) keeps \(x(t)\) on track to satisfy \(Ax(1) \leq 1\)

Should such a direction exist? We point that, letting \(x^*\) denote the optimum solution of \((P)\), then \(dx(t)/dt = x^*\) would satisfy the above except its objective value is \((P)\) rather than \((C)\). (And in hindsight we will understand that \((P) = (C)\).) Of course if we could identify \(x^*\) then we will have immediately solved \((P)\). The requirements of \(dx(t)/dt\) are easier than solving \((P)\), since we only need to satisfy
the all packing constraints in the aggregate (over all \( t \)) rather than at any instance of \( t \).

That said, consider the second requirement, that \( dx^{(t)}/dt \) is trending towards \( Ax^{(1)} \leq 1 \). Informally speaking, ideally we could choose \( dx^{(t)}/dt \) so that

\[
\frac{d}{dt} \max_i (Ax^{(t)})_i \leq 1. \tag{9.1}
\]

After all, if our goal is to satisfy all the constraints, then at an instance in time we should be most concerned with the constraint closest to being violated. Now there are at least two issues with ??.

1. The first issue, more technical, is that \( \max_i y_i \) is not differentiable.

2. The second issue, more conceptual, is that in the event of a tie – \( \max_i (Ax^{(t)})_i \) is attained by several indices \( i \) – it is not clear how to prioritize one over the other. Trying to satisfy multiple constraints simultaneously, of course, is essentially the same problem we started with. \( \max_i (Ax^{(t)})_i \) also does not recognize any constraint that are very close to the maximum, offering no distinction from constraints where \( (Ax^{(t)})_i \) is very small (and less important) and where \( (Ax^{(t)})_i \) is very close to the maximum and intuitively more important.

Both these points are pointing to the discrete, all-or-nothing nature of taking the maximum.

**Imagining a smooth max.** We now make a leap of faith and presuppose that these issues can be overcome by a “smooth” function \( \pi : \mathbb{R}^m \to \mathbb{R} \) that models \( \max_i y_i \). More precisely, suppose there exists a function \( \pi(y) \) with the following (informal) properties.

1. \( \pi(y) \approx \max_i y_i \) for all \( y \)

2. \( \pi \) is smoothly differentiable, and \( \pi'(y) \in \Delta^m \) for all \( y \).

Later we will elaborate on each of the above points later. At this preliminary stage where we are still developing ideas, one should treat \( \pi(y) \) synonymously as \( \max_i y_i \).

Assuming such a function \( \pi \), we have

\[
\max_i (Ax^{(t)})_i \approx \pi(Ax^{(t)}),
\]

and overall we want \( \pi(Ax^{(1)}) \leq 1 \) at time \( t = 1 \). Revisiting eq. (9.1) above we now want \( (d/dt)\pi(Ax^{(t)}) \leq 1 \). This time \( \pi \) is differentiable, and by the chain
To identify the good $dx^{(t)}/dt$, consider the problem

$\langle 1, y \rangle \text{ over } y \in \mathbb{R}^n_{\geq 0} \text{ s.t. } \langle \pi', Ay \rangle \leq 1,$

where we abbreviate $\pi' \overset{\text{def}}{=} \pi'(Ax^{(t)})$ for ease of notation. We first observe that the optimum solution $x^*$ for $(P)$ is a feasible solution is objective value $(P)$. But we can make a stronger observation: the problem described above is a Lagrangian relaxation (L) of $(P)$, with weights given by $\pi$. So in fact there is a solution $y$, hence a direction $dx^{(t)}/dt$, with objective value $\geq (C)$.

Putting everything together, we have a continuous algorithm where $x^{(t)}$ for $t \in [0,1]$ is defined by

$x^{(0)} = 0$ and $\frac{dx^{(t)}}{dt}$ maximizing $\left\langle 1, \frac{dx^{(t)}}{dt} \right\rangle \text{ s.t. } \frac{d}{dt}\pi(Ax^{(t)}) \leq 1;$

noting that the RHS expands into a relaxation (L) of $(P)$. We have

$max_i (Ax^{(1)})_i \approx \pi(Ax^{(1)}) - \pi(Ax^{(0)}) = \int_0^1 \frac{d}{dt}\pi(Ax^{(t)}) \, dt \leq \int_0^1 1 \, dt = 1,$

so $x^{(1)}$ is (essentially) feasible. Moreover, we have

$(P) \geq \langle 1, x^{(1)} \rangle = \langle 1, x^{(1)} \rangle - \langle 1, x^{(0)} \rangle = \int_0^1 \left\langle 1, \frac{dx^{(t)}}{dt} \right\rangle \, dt \geq \int_0^1 (C) \, dt \geq (C). \quad (!)$

So up to actually defining such a function $\pi$, we’ve shown that $(P) \geq (C)$.

**A smooth max.** Consider the function $\pi$ defined by

$\pi(y) = \log \left( \sum_i e^{y_i} \right).$

$\pi$ has several names in the literature including the “soft max” and the “partition” function. Intuitively it is clear how $\pi(y)$ tries to identify the max: taking the exponent amplifies the biggest coordinates, and dominates the sum inside the log. More precisely, we have

$max_i y_i \leq \log \left( \sum_i e^{y_i} \right) \leq max_i y_i + \log (m); \quad (9.2)$
the RHS is tight when all coordinates attain the maximum. Note that we have an additive approximation, which is unusual for us. Another interesting property of \( \pi \) comes from its gradient: we have
\[
\pi'(y) = \frac{e^{y_i}}{\sum_j e^{y_j}}
\]
for each coordinate \( i \). In particular, \( \pi'(y) \) is nonnegative and the sum of coordinates is 1, as desired.

If we drop in \( \pi \) to the description above, elaborating on the \( \approx \) in ?? leads instead to the conclusion that
\[
\max_i (Ax^{(1)})_i \leq 1 + \pi(0) = 1 + \log(m).
\]
The \( \log(m) \) additive factor is problematic; from this we can only conclude that \( x^{(1)}/(1 + \log(m)) \) is feasible, and \( (1 + \log m)(P) \geq (C) \). Suppose we want to reduce the \( \log(m) \) factor to \( \epsilon > 0 \), for an input parameter \( \epsilon \). In particular we want to reduce the additive gap in eq. (9.2) from \( \log(m) \) to \( \epsilon \). Here we will seize on the fact that the additive error is \textit{relatively small} when \( y_i \) is large. Consider the rescaled function
\[
f(y) \overset{\text{def}}{=} \frac{1}{\eta} \pi(\eta y) \text{ where } \eta = \log(m)/\epsilon.
\]
Then we have
\[
\max_i y_i \leq f(y) \leq \max_i y_i + \epsilon
\]
for all \( y \). \( f'(y) \) still has the convient property of being nonnegative and summing to one; indeed, we have
\[
f'(y) = \pi'(\eta y) = \frac{e^{\eta y_i}}{\sum_j e^{\eta y_j}}.
\]
Dropping this function into our continuous algorithm above, ?? leads to
\[
Ax^{(1)} \leq (1 + \epsilon)\mathbb{1},
\]
hence \( (1 + \epsilon)(P) \geq (C) \). Now taking the limit \( \epsilon \to 0 \) gives \( (P) \geq (C) \).

**Discretization.** Next we convert our continuous algorithm to a discrete one. This will address the remaining technical issue in our proof, which is that \( xt' \) may not be integral. It will also produce The high-level overview is simple. Suppose at time \( t_0 \) we have a direction \( dx^{(t_0)}/dt = y \) that solves the relaxation (L) induced by
f′(Ax(t₀)). Then for a sufficiently small but discrete amount of time, f′(Ax(t)) remains similar to f′(Ax(t₀)), and y continues to be the approximation solution to the new relaxation (L), which is suffices for the proof. So instead of revising dx(t)/dt every moment, we keep dx(t)/dt for conservatively small time steps.

We now make this argument precise. Suppose at time t₀ we compute a direction dx(t₀)/dt = y that solves the Lagrangian relaxation (L) induced by f′(Ax(t₀)). Now suppose we fix dx(t)/dt = y for t ∈ [t₀, t₀ + δ] for δ > 0 sufficiently small such that

\[ f′(Ax(t)) \leq (1 + \epsilon)f′(Ax(t₀)) \]

for all such t. Then, for all t ∈ [t₀, t₀ + δ], we have

\[ \frac{d}{dt}f(Ax(t)) = \langle f′(Ax(t)), Ay \rangle \leq (1 + \epsilon)\langle f′(Ax(t₀)), Ay \rangle \leq 1 + \epsilon. \]

To wit, for t close to t₀, the weights f′(Ax(t)) to the relaxation (L) have not really changed, and y remains an approximately good solution to (L). This approximation would introduce an additional ε in the overall bound.

How big is δ? We observe that for x(t) = x(0) + δy, we have

\[ f′_i(A(x(t))) = \frac{e^{\eta(A(x(0) + \delta y))} e^{\delta \eta(Ay)}}{\sum_j e^{\eta(A(x(0) + \delta y))} e^{\delta \eta(Ay)}}. \]

In particular, if \( \delta \eta(Ay)_i \leq \epsilon \), and \( \epsilon \) is sufficiently small, then

\[ f′(A(x(t₀) + \delta y)) \leq e^\epsilon \cdot f′(Ax(t₀)) \leq (1 + 2\epsilon)f′(Ax(t₀)), \]

as desired (up to constants). Any choice of δ satisfying these inequalities will lead to Ax(t) ≤ (1 + 3\epsilon)1, hence (1 + 3\epsilon)(P) ≥ (C), as desired.

Now, subject to \( \delta \eta \min_i(Ay)_i \leq \epsilon \), we might as well take δ as large as possible. Suppose we set

\[ \delta = \frac{\epsilon}{\eta \max_i(Ay)_i}. \]

each iteration. Let us call the constraint i maximizing (Ay)_i the “bottleneck” constraint. For this choice of δ, (Ax(t))_i will increase by exactly \( \epsilon/\eta \). Meanwhile, we already know that Ax(t) ≤ (1 + 3\epsilon)1 for all t. So a constraint i can only be the bottleneck O(\( \eta/\epsilon \)) = O(log(m)/\( \epsilon^2 \)) times. There are m constraints, so choosing δ maximally in this fashion, we have at most O(m log(m)/\( \epsilon^2 \)) iterations!

Here is the final algorithm that computes a point x ∈ R^n such that Ax ≤ (1 + 3\epsilon)1 and \( \langle 1, x \rangle \geq (C) \). This implies that (1 + 3\epsilon)(P) ≥ (C) for all \( \epsilon > 0 \), hence (P) ≥ (C), as desired.
1. Set $x = 0$ and $t = 0$.

2. Until $t = 1$:
   (a) Let $y$ solve the relaxation $(L)$ induced by $f'(Ax)$.
   (b) Let $\delta = \min\left\{1 - t, \frac{\epsilon}{\max_i (Ay)}\right\}$.
   (c) Set $x = x + \delta y$ and $t = t + \delta$

3. Return $x$.

We present two additional observations are left as exercises. The first shows that we are also implicitly the algorithm also (approximately) solves $(C)$.

**Corollary 9.3.** In the algorithm above, there exists an iteration such that $f'(Ax)$ (up to scaling) is a $(1 - O(\epsilon))$-approximation to the covering problem $(C)$.

The second observation says that there exists weights for which $(L) = (P)$.

**Corollary 9.4.** There exists a set of weights $w$ such that $(L) = (P)$.

### 9.4 Exercises

**Exercise 9.1.** Prove corollary 9.3.

**Exercise 9.2.** Prove corollary 9.4.\(^4\)

---

\(^4\)It may be helpful to focus on proving that $(L) \leq (1 + \epsilon)(P)$ for some weights $w$, for arbitrarily small $\epsilon$.  

129
Chapter 10

Oblivious Routing

Recall the concurrent multicommodity flow problem, where given a graph \( G = (V,E) \) with edge capacities \( c : E \to \mathbb{R}_{\geq 0} \) and demands \( d : V \times V \to \mathbb{R}_{>0} \), the goal is to simultaneously route \( d(s,t) \) units of flow from \( s \) to \( t \) for all \( s,t \in V \), with minimum congestion. Here we consider multicommodity flow problems in undirected graphs.

Oblivious routing is the idea that one can take the graph \( G \) and, before knowing the demands \( d \), precomputing flows for every \( \{s,t\} \) pair that induces a competitive multicommodity flow for any given demands \( d \). More explicitly, an oblivious routing scheme consists of a unit \( \{s,t\} \)-flow \( f_{s,t} \) for every \( s,t \in V \). Given demands \( d : \binom{V}{2} \to \mathbb{R}_{\geq 0} \) and an oblivious routing scheme \( F = \{f_{s,t} : s,t \in V\} \), we can route \( d(s,t) \) units of flow along \( f_{s,t} \) for all \( s,t \in V \) to obtain a multicommodity flow. For a parameter \( \alpha \geq 1 \), we say that \( F \) is an \( \alpha \)-competitive routing scheme if for any set of demands \( d : \binom{V}{2} \to \mathbb{R}_{\geq 0} \), the multicommodity flow \( \{d(s,t)f_{s,t} : s,t \in V\} \) has congestion within an \( \alpha \)-approximate factor of the optimum congestion. That is, \( F \) gives an \( \alpha \)-approximate solution for any set of demands \( d \).

Our goal is to construct an \( \alpha \)-competitive routing scheme. Note that the minimum congestion \( s \to t \) flow for all \( s \) and \( t \) gives a \( \binom{n}{2} \)-competitive routing scheme. A priori, there is no reason why anything better should be possible. A very low congestion (e.g., \( \text{polylog}(n) \)) would not only be useful but also indicate a lot of symmetry underlying undirected multicommodity flow problems.

10.1 Optimal oblivious routing

We can express the oblivious routing as an LP. Recall that the goal is to choose a unit flow for every \( \{s,t\} \)-pair that minimizes the congestion of the multicommodity flow over all demands that can be routed with congestion 1. Flow constraints
and congestion can both be expressed by linear constraints. Of course there are infinitely many possible demands to consider but that does not prevent us from first writing the LP that considers all of these.

\[
\begin{align*}
\text{minimize } & \lambda \text{ over } \lambda > 0 \text{ and } f_{st} : E \to \mathbb{R}_{\geq 0} \text{ for each } \{s,t\} \subseteq V \\
\text{s.t. } & f_{st} \text{ is a unit } \{s,t\}\text{-flow for all } \{s,t\}, \\
& \sum_{\{s,t\}} b(s,t) f_{st}(e) \leq \lambda c(e) \text{ for all } e \in E, \text{ feasible } b : \left(\frac{V}{2}\right) \to \mathbb{R}_{\geq 0}
\end{align*}
\]

(10.1)

Here, \(f_{st}(e)\) denotes the (nonnegative) amount of flow on edge \(e\) (in either direction). A set of demands \(b\) is called feasible if there is a multicommodity flow routing it with congestion 1.

The solution to the LP above gives an optimum oblivious routing. The only issue in computing the LP above is that there are infinitely many constraints, as there are infinitely many feasible sets of demands. However there is a separation oracle for this LP. We leave it to the reader (exercise 10.1) to obtain a separation oracle for the LP above. (cf. [ACF+03].)

So in fact we can obtain the optimal oblivious routing as an LP. However this does not tell us why the optimal oblivious routing is particularly good at all. We address this in the following section. The difference here is the difference between computability and structure of the problem; only understanding the latter will motivate the former.

### 10.2 \(O(\log n)\)-congestion tree routing

We now present a result by Räcke [Räc08] that develops routing schemes of the following structured type. A routing tree is a routing scheme obtained by embedding a tree with leaves corresponding to \(V\) into \(G\). More precisely, a routing tree consists of an auxiliary tree \(T\) where each vertex in the tree corresponds to a node in the graph and each edge in the tree corresponds to a path in the graph with the following properties:

1. If \(e = \{u,v\} \in T\) is an edge in the tree, then \(e\) corresponds to a path between the vertices corresponding to \(u\) and \(v\) in \(G\).

2. The leaves of \(T\) are in one-to-one correspondance with the vertices \(V\).

A routing tree gives the following routing scheme. For every pair of vertices \(s,t \in V\), there are two leaves in \(T\) corresponding to \(s\) and \(t\). The unique path in
10. Oblivious Routing

10.2. $O(\log n)$-congestion tree routing

Kent Quanrud
Fall 2021

\[ T \text{ between these leaves implies a walk from } s \text{ to } t \text{ in } G, \text{ where we concatenate the paths in } G \text{ corresponding to the edges in } T \text{ along the } s\text{-leaf to } t\text{-leaf path in } T. \]

**Theorem 10.1** (Räcke [Räc08]). In randomized polynomial time, one can compute a convex combination of routing trees that form a $O(\log n)$-competitive oblivious routing scheme.

To present and analyze the algorithm we introduce the following notation. For a routing tree $T$, and a pair of vertices $s, t \in V$, let $T_{s,t}$ be the \{s, t\}-walk in $G$ given by $T$. For any edge $e = \{u, v\} \in E$, we denote $T_e = T_{u,v}$. For a walk $T_{s,t}$ and an edge $e \in E$, let $T_{s,t}(e)$ be the number of times $e$ appears in the walk $T_{s,t}$. We write $\sum_{e \in T_{s,t}}$ to denote a sum over the edges in the walk $T_{s,t}$ with multiplicity. Let $\mathcal{T}$ denote the family of all routing trees in $G$. Now, consider the following LP.

\[
\begin{align*}
\text{maximize } & \sum_{T \in \mathcal{T}} x_T \text{ over } x : \mathcal{T} & \rightarrow \mathbb{R}_{\geq 0} \\
\text{s.t. } & x_T \sum_{e \in E} c(d) T_{d}(e) \leq c(e) \text{ for all } e \in E.
\end{align*}
\]

This LP is quite different then the LP (10.1) above. In particular there is no role played be “feasible demands”. Rather the LP is trying to reroute the capacity of the edges of $G$ through the trees. Indeed the packing constraint for an edge $e$ is tracking the total amount of capacity pushed by the routing trees through $e$ in the auxiliary walks for each other edge $d$. If the edges of $G$ can be rerouted through $T$ with congestion $\alpha$, then certainly any feasibly flow can be routed through $T$ with congestion $\alpha$ since such a flow satisfies the capacities in $G$. So in a sense this LP is more ambitious then (10.1). The following lemma formalizes the relationship to oblivious routing and the proof expands on the discussion above in greater detail.

**Lemma 10.2.** Suppose $x : \mathcal{T} \rightarrow \mathbb{R}_{\geq 0}$ is a feasible solution to (P), packing $\alpha$ fractional routing trees into $G$. Then $x/\alpha$ is a convex combination of routing trees that give a $(1/\alpha)$-competitive oblivious routing scheme.

**Proof.** Suppose we simultaneously route, for every routing tree $T \in \mathcal{T}$ and every edge $e = \{s, t\} \in E$, $x_T c(e)$ units of flow from $s$ to $t$ along the walk $T(s,t)$. Because $x$ is a feasible solution to (P), this routes a total of $\alpha c(e)$ units of flow from $s$ to $t$ for each edge $e = \{s, t\} \in E$ with congestion 1. By scaling, for any $\gamma > 0$, we can simultaneously route $\gamma c(e)$ units of flow from $s$ to $t$ for every edge $e = \{s, t\}$ with congestion $\gamma/\alpha$.

Now, given any demands $d : V \rightarrow \mathbb{R}_{\geq 0}$ that can be routed in $G$ with congestion $\gamma$, consider the multicommodity flow where we route, for every $s, t \in V$ and $T \in \mathcal{T}$, $(x_T/\alpha) d(s, t)$ units of flow from $s$ to $t$ along the walk $T(s,t)$. Since
the demands \( d \) can be routed with congestion \( \gamma \), the congestion induced by the fractional tree routing is no worse then the congestion induced by routing \( \gamma c(e) \) units of flow from \( s \) to \( t \) through the trees for every edge \( e = \{s, t\} \in E \). By the preceding discussion, this gives congestion at most \( \gamma/\alpha \).

Recall that the algorithm in chapter 9 is based on solving a sequence of Lagrangian relaxations, that form packing problems with only a single packing constraint. Given a probability distribution \( p \in \Delta_m \) over \( E \), the Lagrangian relaxation of (P) has the form

\[
\begin{align*}
\text{maximize} & \quad \sum_{T \in \mathcal{T}} x_T \text{ over } x : \mathcal{T} \rightarrow \mathbb{R}_{\geq 0} \\
\text{s.t.} & \quad \sum_{T \in \mathcal{T}} x_T \text{cost}(T \mid p) \leq B \\
\end{align*}
\]

where \( \text{cost}(T \mid p) = \sum_{d \in E} c(d) \sum_{e \in T_d} p_e \) and \( B = \sum_{e} p_c(e) \)

(Some rearrangement of terms is required to obtain \( \text{cost}(T \mid p) \) as described above.) The key to the relaxation is understanding the cost of a tree, \( \text{cost}(T \mid p) \), relative to the budget \( B \). The cost sums, for each edge \( d \in E \), the sum of weights \( p_e \) over the walk \( T_d \). That is, recasting the weights \( p_e \) as \textit{edge lengths}, the length of the walk \( T_d \). Now, recall the randomized tree metric of section 8.3, applied to edge lengths \( p_e \). This produces an auxiliary tree \( T \) in which each edge \( d \) has stretch \( O(\log n) \) in expectation – more explicitly, in our setting,

\[
\mathbb{E} \left[ \sum_{e \in T_d} p_e \right] \leq O(\log n) p_d
\]

for all \( d \in E \). In particular this randomized tree has expected cost

\[
\mathbb{E}[\text{cost}(T \mid p)] = \sum_{d \in E} c_d \mathbb{E} \left[ \sum_{e \in T_d} p_e \right] \leq O(\log n) \sum_{d \in E} c_d p_d = O(\log n) B.
\]

This gives us a solution to the relaxation of value \( \Omega(1/\log n) \) for any set of Lagrangian weights \( p \).

To complete the proof of theorem 10.1, we apply the discrete LP algorithm from chapter 9 to the packing LP (P) with a constant value of \( \epsilon \). For every Lagrangian relaxation induced by \( p \in \Delta_E \), the randomized tree metric of section 8.3 produces a fractional solution of value \( \geq \Omega(1/\log n) \). Thus we obtain feasible solution to (P) of value \( \Omega(1/\log n) \). This in turn implies a \( O(\log n) \)-competitive oblivious routing scheme, as desired.
10.3 Exercises

Exercise 10.1. Give a separation oracle for (10.1).

10.4 Additional notes and references

Above we saw that a distribution of tree routing schemes obtains $O(\log n)$ congestion. However, even before this result, Räcke [Räc02] showed that a single tree can obtain $O(\text{polylog}(n))$ congestion. (We chose to present [Räc08] for simplicity and for its technical connections to chapter 9.) There are several follow up works balancing the running time against the congestion for either approach [BKR03; HHR03; Mad10; RST14].
Chapter 11

Conductance

11.1 Introduction

Recall our randomized construction of a constant degree expander, as the union of a constant number of uniformly random matchings. With high probability this produces an expander. But given such a randomized graph, how can we verify and know for certain that it has constant expansion? We can obtain a $O(\log n)$-approximation by sparsest cut, but this approximation bound is too rough to decide if the expansion is closer to 1 or $1/\log(n)$.

We will present an algorithm that can certify a constant degree expander. The algorithm is really an approximation algorithm for the conductance of a graph, a different but related notion to sparsity that coincides with sparsity for constant degree graphs. The approximation bound we get will be somewhat unusual but it will suffice for constant expansion.

Just as important is how we approximate the conductance. We model the input graph as a symmetric matrix called the “Laplacian”, and study its eigenvalues. There turns out to be a strong relation between the second smallest eigenvalue and the conductance via what is called “Cheeger’s inequality”. The Laplacian has many other applications and we will discuss more in later chapters. The general study of graphs via their Laplacian’s is called spectral graph theory and this approach has yielded many exciting algorithms.

11.2 The Laplacian of a graph

As mentioned above, we are interested in a certain matrix representation of a graph called the Laplacian. We first show how to model a single edge by a matrix; an entire graph is then modeled by the corresponding weighted sum over its edges. We will work in the $n$-dimensional vector space $\mathbb{R}^n$ – one coordinate per
vertex. Edges, representing pairwise relations between vertices, are modeled as matrices in $\mathbb{R}^{V \times V}$.

The **Laplacian of an (unweighted) edge** $e = \{u, v\}$ is the rank-1 matrix

$$L_e = (1_u - 1_v) \otimes (1_u - 1_v)$$

where $1_u \in \{0, 1\}^V$ denotes the indicator vector\(^1\) for $u$. Here $a \otimes b$ denotes the outer product of two vectors $a, b$, defined by $\langle x, (a \otimes b) y \rangle = \langle a, x \rangle \langle b, y \rangle$. Note that the expression for $L_e$ is indifferent to whether we wrote $1_u - 1_v$ or $1_v - 1_u$, as long as it is symmetric. For any input vector $x \in \mathbb{R}^V$, we have

$$\langle x, L_e x \rangle = (x_u - x_v)^2.$$  

For an undirected graph $G = (V, E)$ with positive edge weights $w : E \to \mathbb{R}_{>0}$, the **Laplacian of the graph** is the corresponding weighted sum of Laplacians of the edges,

$$L = \sum_e w(e) L_e.$$  

Given an input vector $x \in \mathbb{R}^V$, we have

$$\langle x, L x \rangle = \sum_{e \in E} w(e) \langle x, L_e x \rangle = \sum_{e=\{u,v\} \in E} w(e) (x_u - x_v)^2.$$  

That is, $L$ induces a simple sum of squared differences on $x$, based on the edges of the graph. We will see that $\langle x, L x \rangle$ encodes a lot of information about $G$. One example is that $L$ encodes the weight of all the cuts. Given a set $S \subset V$, if we let $\mathbb{1}_S$ denote the indicator vector of $S$, we have

$$\langle \mathbb{1}_S, L \mathbb{1}_S \rangle = \sum_{e \in \partial(S)} w(e).$$  

Another interpretation of $\langle x, L x \rangle$ that we discuss in greater detail later is based on electrical networks. $\langle x, L x \rangle$ is the electrical energy of an electrical network where the edge weights give resistances and $x$ is a set of vertex potentials.

Recall that a linear operator $A : \mathbb{R}^n \to \mathbb{R}^n$ is **symmetric** if $A = A^T$. It is easy to see that the Laplacian $L$ is symmetric: each $L_e$ is symmetric since in general $(a \otimes b)^T = (b \otimes a)$, and $L$ is a positively weighted combination of $L_e$’s. Another salient property of $L$ is that, as a sum of squares,

$$\langle x, L x \rangle \geq 0 \text{ for all } x \in \mathbb{R}^n.$$  

These two properties make $L$ a member of the following very importance class of linear operators.

\(^1\)We are avoiding the conventional notation $e_u$ for the standard basis vectors because $e$ is so frequently used for edges.
Definition 11.1. A linear operator \( A : \mathbb{R}^n \to \mathbb{R}^n \) is a **positive semi-definite** linear operator if

1. \( A \) is symmetric.
2. \( \langle x, Ax \rangle \geq 0 \) for all \( x \in \mathbb{R}^n \).

\( A \) is **(strictly) positive definite** if in addition to being positive semi-definite,

3. \( A \) is invertible.

The Laplacian \( L \) is not invertible: \( L \mathbb{I} = 0 \). If \( G \) is connected, and we restrict to the \( n-1 \) space \( \mathbb{R}^{V/1} \), then \( L \) is invertible and (strictly) positive definite (see ??).

### 11.3 The Spectral Theorem for Symmetric Maps

Let \( A : \mathbb{R}^n \to \mathbb{R}^n \) be a linear map. Recall that a vector \( x \in \mathbb{R}^n \) is an **eigenvector** of \( A \) with **eigenvalue** \( \lambda \in \mathbb{C} \) of \( Ax = \lambda x \). The following fact applies generally to all linear maps.

**Fact 11.2.** Let \( A : \mathbb{R}^n \to \mathbb{R}^n \) be a linear map. Then \( A \) has an eigenvalue \( \lambda \in \mathbb{C} \) and eigenvector \( x \in \mathbb{C}^n \).

We omit the proof because it is redundant with the following stronger claim when \( A \) is symmetric. When \( A \) is symmetric, there is a real-valued eigenvalue and eigenvector with real-valued coordinates, and moreover they can be obtained by optimization.

**Lemma 11.3.** Let \( L : X \to X \) be a symmetric linear map in a vector space \( X \) over \( \mathbb{R} \). Let \( x \) maximize \( \langle x, Lx \rangle \) subject to \( \|x\| = 1 \). Then \( Lx = \lambda x \) for \( \lambda = \langle x, Lx \rangle \).

**Proof.** We claim that for any \( u \in X \) with \( \|u\| = 1 \) and \( \langle u, x \rangle = 0 \), \( \langle u, Lx \rangle = 0 \). If \( Lx \) is orthogonal to \( u \) for every \( u \) orthogonal to \( x \), then we must have \( Lx \in \text{span}(x) \); i.e., \( Lx = \lambda x \) for some \( \lambda \in \mathbb{R} \). Upon inspection, \( \lambda = \lambda \langle x, x \rangle = \langle x, Lx \rangle \), as claimed.

Let \( u \in X \) with \( \|u\| = 1 \) and \( \langle u, x \rangle = 0 \). Define

\[
f(\epsilon) = \frac{x + \epsilon u}{\sqrt{1 + \epsilon^2}} L \left( \frac{x + \epsilon u}{\sqrt{1 + \epsilon^2}} \right) = \frac{\langle x + \epsilon u, L(x + \epsilon u) \rangle}{1 + \epsilon^2}.
\]

\( f(\epsilon) \) can be interpreted as perturbing \( x \) slightly in the direction of \( u \) and renormalizing, and then computing the inner product over \( L \). Note that \( \|x + \epsilon u\|^2 = \|x\|^2 + \epsilon^2\|u\|^2 = 1 + \epsilon^2 \), so \( \frac{x + \epsilon u}{\sqrt{1 + \epsilon^2}} \) is indeed a normal vector that competes with \( x \) in maximizing \( \langle x, Lx \rangle \). In particular, by choice of \( x \), \( f(\epsilon) \) is maximized at \( f(0) = \langle x, Lx \rangle \). Optimality at 0 implies that \( f'(0) = 0 \). Expanding out \( f'(0) \), we find that \( \langle u, Lx \rangle = 0 \), as desired. (See exercise 11.2). \[ \square \]
Remark 11.4. An alternative proof starts from the fact there exists a complex
eigenvalue and eigenvector, and goes on to show that this eigenvalue must be real-
valued and that there is a corresponding eigenvector with real-valued coordinates.

The above lemma implies that beyond simply having an eigenvalue, a sym-
metric map has a real-valued eigenvalue without extending the field to include
complex values. This simple fact leads to the following sweeping spectral theorem
about symmetric matrices.

**Theorem 11.5.** Let $X$ be an $n$-dimensional vector space over $\mathbb{R}$. Let $A : X \rightarrow X$ be a
symmetric linear map. Then there exists an orthonormal basis $u_1, \ldots, u_n$ of $X$ and $n$
scalar values $\lambda_1, \ldots, \lambda_n \in \mathbb{R}$ such that

$$A = \lambda_1 (u_1 \otimes u_1) + \cdots + \lambda_n (u_n \otimes u_n).$$

**Proof.** If $n = 0$, then the claim is tautological, as $X$ is the trivial vector space $\{0\}$
and $A$ can be expressed as an empty sum. Suppose $n \geq 1$. By lemma 11.3, $A$ has
a real-valued eigenvalue $\lambda$ with a corresponding eigenvector $u \in X$. By scaling $u$,
we may assume $\|u\| = 1$. Consider the map $B = A - \lambda (u \otimes u)$. $B$ is also symmetric,
and maps the space $\text{span}(x) = \{\alpha x : \alpha \in \mathbb{R}\}$ to $0$. Let $Y = \{y \in X : \langle x, y \rangle = 0\}$ be
the subspace of $X$ orthogonal to $x$. We have $\dim(Y) = n - 1$.

We claim that $B$ maps $Y$ into $Y$. Indeed, for any $y \in Y$, we have

$$\langle x, By \rangle = \langle Bx, y \rangle = \langle 0, y \rangle = 0,$$

so $By \in Y$.

Thus $B$ restricts to a linear and symmetric operator on $Y$. By induction on $n$,
there is an orthonormal basis $u_1, \ldots, u_{n-1}$ of $Y$ and scalar values $\lambda_1, \ldots, \lambda_{n-1} \in \mathbb{R}$
such that

$$B = \lambda_1 (u_1 \otimes u_1) + \cdots + \lambda_{n-1} (u_{n-1} \otimes u_{n-1}).$$

Let $\lambda_n = \lambda$ and $u_n = u$. Observe that $u_1, \ldots, u_n$ is an orthonormal basis of $X$. We have

$$\lambda_1 (u_1 \otimes u_1) + \cdots + \lambda_n (u_n \otimes u_n) = B + \lambda_n (u_n \otimes u_n) = A,$$

as desired. ■

Theorem 11.5 makes the structure of any symmetric map $A : \mathbb{R}^n \rightarrow \mathbb{R}^n$
extremely transparent. By theorem 11.5, let $u_1, \ldots, u_n \in \mathbb{R}^n$ and $\lambda_1, \ldots, \lambda_n \in \mathbb{R}$
be such that

$$A = \lambda_1 (u_1 \otimes u_1) + \cdots + \lambda_n (u_n \otimes u_n).$$
It will be convenient to assume that all \( \lambda_i \)'s are in increasing order: \( \lambda_1 \leq \lambda_2 \leq \cdots \leq \lambda_n \).

For any input vector \( x \in \mathbb{R}^n \), we can write \( x \) uniquely in the basis \( \{u_1, \ldots, u_n\} \) as
\[
x = \alpha_1 u_1 + \cdots + \alpha_n u_n,
\]
where \( \alpha_i = \langle x, u_i \rangle \). Then we have
\[
Ax = \alpha_1 \lambda_1 u_1 + \cdots + \alpha_n \lambda_n u_n.
\]
That is, in the basis \( \{u_1, \ldots, u_n\} \), \( A \) simply rescales the \( i \)th coordinate by a factor of \( \lambda_i \). That is to say:

*Every symmetric matrix is a diagonal matrix up to rotation (i.e., change in basis).*

We can see from the construction in the proof that the \( \lambda_i \)'s are the eigenvalues of \( A \) and the \( u_i \)'s are eigenvectors. But this fact is even more obvious in hindsight given the representation. Indeed, for each \( i \), we have
\[
Au_i = \lambda_i (u_i \otimes u_i) u_i = \lambda_i u_i,
\]
by orthonormality of the \( u_i \)'s. The following theorem gives a min-max characterization of the eigenvalues and follows immediately from the spectral theorem. It is often called the *Courant-Fischer minimax theorem.*

**Theorem 11.6.** Let \( A : \mathbb{R}^n \rightarrow \mathbb{R}^n \) be a symmetric linear operator. Let \( \lambda_1 \leq \cdots \leq \lambda_n \) be the \( n \) eigenvalues of \( A \) (with multiplicity) in increasing order. Then
\[
\lambda_{n-k} = \min_{S : \dim(S) = k} \max_{x \in X \setminus S} \frac{\langle x, Lx \rangle}{\langle x, x \rangle}.
\]

### 11.4 Sparse cuts

Recall that the *sparsity* of a cut \( \partial(S) \) (where \( S \subset V \)), which we denote \( \Phi(S) \), was the ratio
\[
\Phi(S) \overset{\text{def}}{=} \frac{\bar{\omega}(S)}{\min\{|S|, |\bar{S}|\}},
\]
where \( \bar{S} = V \setminus S \). The sparsity of the graph \( G \) is defined as the sparsity of the sparsest cut,
\[
\Phi(G) \overset{\text{def}}{=} \min_{S \subset V} \Phi(S).
\]
To relate sparsity to the Laplacian, note that for any nonempty set $S \subset V$ with at most $n/2$ vertices, we have

$$\Phi(S) = \frac{\bar{w}(S)}{|S|} = \frac{\langle 1_S, L 1_S \rangle}{\langle 1_S, 1_S \rangle},$$

(11.1)

where $1_S$ is the $\{0,1\}$-indicator vector for $S$.

Now we make a deeper connection to the eigenvectors of $L$. As the positive semi-definite matrix $L$, $L$ has nonnegative eigenvalues. Moreover, we know that $1$ is an eigenvector with eigenvalue of $0$ – this gives us our smallest eigenvalue. The eigenvector corresponding to the second smallest eigenvalue, denoted $\lambda_2$, is given by

$$\lambda_2 = \min_{x: \langle 1, x \rangle = 0} \frac{\langle x, Lx \rangle}{\langle x, x \rangle}.$$

Now, consider any cut $1_S$, and let $x$ be the orthogonal projection from $1$; namely,

$$x = 1_S - \alpha 1$$

where $\alpha = \langle 1_S, 1 \rangle / (\langle 1, 1 \rangle) = |S| / n$

Observe that

$$\langle x, x \rangle = \langle x, 1_S \rangle = (1 - \alpha)|S| = (n - |S|)|S| / n.$$

Thus

$$\lambda_2 \leq \frac{\langle x, Lx \rangle}{\langle x, x \rangle} = \frac{n \sum_{e \in \partial(S)} w(e)}{|S|(n - |S|)} \leq 2\Phi(S).$$

Taking the minimum over all sets $S$, we obtain the following.

**Theorem 11.7.** Let $G = (V,E)$ be an undirected graph with $m$ edges and $n$ vertices, and positive edge weights $w : E \to \mathbb{R}_{>0}$. Let $L : \mathbb{R}^V \to \mathbb{R}^V$ be the Laplacian of $G$. Let $\lambda_2$ be the second smallest eigenvalue of $L$ and let $\Phi(G)$ be the sparsity of $G$. Then

$$\lambda_2 \leq n\Phi(G).$$

### 11.5 Conductance

We now turn to an alternative to sparsity called the **conductance**. For a set of vertices $S$, the **volume** of $S$, denoted $\text{vol}(S)$, is the sum of degrees of vertices in $S$:

$$\text{vol}(S) = \sum_{v \in S} \text{deg}(v).$$
The **conductance** of a set $S$, denoted $\Psi(S)$, is defined as

$$\Psi(S) = \frac{\bar{\omega}(S)}{\min\{\text{vol}(S), \text{vol}(\bar{S})\}}$$

Note that $\Psi(S)$ is always positive (for a connected graph) and at most 1. There is a clear resemblance between conductance and sparsity except here the vertices in the denominator are weighted by their degree. Similarly to sparsity, we define the conductance of a graph as the minimum conductance of any cut:

$$\Psi(G) = \min_{\emptyset \subset S \subset V} \Psi(S).$$

Like sparsity, conductance is also useful for divide and conquer. The sparest cut is more suited for divide and conquer on vertices, while conductance, where vertices are weighted by their degree, is more conducive to divide and conquer on edges. Recall that sparsity was naturally motivated by its connection to multicommodity flow. On the other hand, conductance is strongly connected to random walks. Indeed, for any set $S$, the stationary distribution is in $S$ with probability proportional to $\text{vol}(S)$. To continue this analogy, the conductance of a (small) set $S$ models the amount of probability mass that enters and leaves $S$ in each step at the stationary distribution. Figure 11.1 gives some examples of graphs with different levels of sparsity and conductance.

We would like to express conductance in algebraic terms, similar to sparsity in (11.1). While the numerator in (11.1) seems appropriate, the denominator does not capture the volume. Instead, consider the following quotient:

$$\frac{\langle x, Lx \rangle}{\langle x, Dx \rangle} \quad \text{where} \quad x \in \mathbb{R}^V. \quad (11.2)$$

For any set $S$ with at most half the total volume, we have

$$\Psi(S) = \frac{\bar{\omega}(S)}{\text{vol}(S)} = \frac{\langle 1_S, L1_S \rangle}{\langle 1_S, D1_S \rangle}.$$
That said, the quotient (11.2) does not have a direct connection to the Laplacian $L$ in the same way as sparsity did. However, it is connected to the normalized Laplacian, which is the map $M : \mathbb{R}^V \to \mathbb{R}^V$ defined by

$$M = D^{-1/2}LD^{-1/2}.$$  

For any vector $x$, letting $y = D^{1/2}x$, we have

$$\frac{\langle x, Lx \rangle}{\langle x, Dx \rangle} = \frac{\langle y, My \rangle}{\langle y, y \rangle}.$$  

Since the normalized Laplacian $M$ is also symmetric, the RHS models the eigenvalues of $M$. In today’s discussion, we will study the eigenvalues of $M$ and relate it to the condutance of the graph.

We first point out that there are some similarities (in the linear-algebraic sense) to other matrices that we have studied. Let $R = AD^{-1} : \mathbb{R}^V \to \mathbb{R}^V$ denote the random walk map. We define the normalized random walk matrix $Q$ as

$$Q = D^{-1/2}RD^{1/2} = D^{-1/2}AD^{-1/2}.$$  

To draw the connection to $M$, if we expand $L = D - A$, then we have

$$M = D^{-1/2}(D - A)D^{-1/2} = I - Q = D^{-1/2}(I - R)D^{1/2}.$$  

**Theorem 11.8.** Let $G = (V,E)$ be an undirected graph with $m$ edges and $n$ vertices, and positive edge weights $w : E \to \mathbb{R}_{>0}$. Let $M : \mathbb{R}^V \to \mathbb{R}^V$ be the normalized Laplacian and $R : \mathbb{R}^V \to \mathbb{R}^V$ the random walk matrix. Then $M$ is similar to $I - R$, and (equivalently) $I - M$ is similar to $R$.

Recall that similarity preserves eigenvalues. Since $R$ has its eigenvalues in $[-1,1]$ and 1 with multiplicity 1, $M$ has its eigenvalues in $[0,2]$ and eigenvalue 0 with multiplicity 1.

**Cheeger’s inequality** We now relate the eigenvalues of $M$ to the conductance of $G$. The following inequality is called Cheeger’s inequality due to an analogous bound by Jeff Cheeger for continuous manifolds.

**Theorem 11.9.** Let $M$ be the normalized Laplacian of an undirected graph $G$, and let $\lambda_2$ be the second smallest eigenvalue of $M$. Then

$$\frac{\lambda_2}{2} \leq \Psi(G) \leq \sqrt{2\lambda_2}.$$
The presence of the $\sqrt{\cdot}$ on the RHS is unusual for us, and leads to a lot of tricky situations. However, for the sake of constant degree expanders – where the conductance equals the sparsity up to a constant, and where we are interested in constant sparsity/conductance – Cheeger’s inequality implies that the expansion and $\lambda_2$ are within a constant. An algorithmic proof of theorem 11.9 will give the verification algorithm we seek. We note that the LHS is more straightforward than the RHS, and left to the reader in exercise 11.3. It remains to prove the upper bound.

11.6 Fiedler’s algorithm: the upper bound

In this section we present an algorithmic proof of the upper bound, $\Psi(G) \leq \lambda$, due to Fiedler [Fie73]. Beyond the surprising connection to the eigenvalues of $M$, the algorithm is fairly simple, and based on previous discussions, the reader might be able to guess it.

Recall that

$$\lambda_2 = \min_{y: \langle d, y \rangle = 0} \frac{\langle y, My \rangle}{\langle y, y \rangle} = \min_{x: \langle d, x \rangle = 0} \frac{\langle x, Lx \rangle}{\langle x, Dx \rangle}.$$ 

Let $x \in \mathbb{R}^V$ with $\langle d, x \rangle = 0$ attain $\lambda_2$ on the RHS. (We note that eigenvectors, hence $x$, can be computed.) $x$ is orthogonal to $d$ and, assuming that we have normalized $x$ such that $\langle x, Dx \rangle = 1$, $x$ has a “fractional cut value” of $\langle x, Lx \rangle = \lambda_2$. Our goal is to “round” the “fractional cut” $x \in \mathbb{R}^V$ to a set $S$ without loosing too much on the conductance. How?

As an additional hint, we point out that a similar setup arose before for minimum $(s,t)$-cut and sparsest cut. In each case we had a “fractional cut” from the LP and wanted to produce a discrete one.

The answer given here is to output the best cut along the line embedding $x$ – yet again! This is called Fiedler’s algorithm and pseudocode is given below.

1. Let $x$ minimize $\frac{\langle x, Lx \rangle}{\langle x, Dx \rangle}$ s.t. $\langle x, d \rangle = 0$ and $x \neq 0$.
2. For $t \in \mathbb{R}$, let $S_t = \{v \in V : x_v \geq t\}$. Return the set $S_t$ of minimum conductance.

As with $(s,t)$-cut and sparsest cut before, the analysis is probabilistic. We will show that that a randomized variant returns a satisfying cut with nonzero probability. For an appropriate distribution of random cuts alone the line embedding $x$, we show that a random cut is good in expectation. More precisely, for $t \in \mathbb{R}$, let $S_t = \{v \in V : x_v \geq t\}$. We will find a distribution over $t \in \mathbb{R}$ such that a random cut $S_t$ has nonzero chance of good sparsity. The distribution will be
more involved than the simple distribution for \((s, t)\)-cut, which was simply the uniform distribution.

**Recentering.** Before defining the distribution, it is convenient to recenter \(x\) so that 0 is a weighted median with respect to degrees. Explicitly, let \(y = x - \mu \mathbb{1}\) where \(\mu \in \mathbb{R}\) is a weighted median with respect to the degrees; i.e., \(\sum_{v : x_v \leq 0} d_v \geq W\) and \(\sum_{v : x_v \geq \mu} d_v \geq W\) where \(W\) is the total sum of edge weights. Note \(y\) is not orthogonal to \(d\). But the following lemma shows that the affine shift by \(\mu \mathbb{1}\) only decreases the Rayleigh quotient. This is simply because translation by 1 does not affect the numerator, and can only increase the denominator.

**Lemma 11.10.** Let \(\langle x, d \rangle = 0\). For any \(\alpha \in \mathbb{R}\),
\[
\frac{\langle x + \alpha \mathbb{1}, L(x + \alpha \mathbb{1}) \rangle}{\langle x + \alpha \mathbb{1}, D(x + \alpha \mathbb{1}) \rangle} \leq \frac{\langle x, Lx \rangle}{\langle x, Dx \rangle}.
\]

*Proof.* We have \(\langle x, Lx \rangle = \langle x + \alpha \mathbb{1}, L(x + \alpha \mathbb{1}) \rangle\) because \(\mathbb{1} \in \text{ker}(L)\). Meanwhile, consider the function
\[
f(\alpha) = \langle x + \alpha \mathbb{1}, D(x + \alpha \mathbb{1}) \rangle \overset{(a)}{=} \langle x, Dx \rangle + \alpha^2 \langle d, \mathbb{1} \rangle,
\]
where for (a) we recall that \(\langle d, x \rangle = 0\). \(f(\alpha)\) is convex with derivative \(2\alpha \langle d, \mathbb{1} \rangle\), so it is minimized at \(\alpha = 0\). \(\square\)

**The distribution.** Let \(t_0 = \min_v y_v\) be the smallest coordinate value in \(y\) and let \(t_1 = \max_v y_v\) be the largest coordinate value. Scaling \(y\) if necessary, we assume that \(t_0^2 + t_1^2 = 1\). The distribution over \(t \in [t_0, t_1]\) is defined by
\[
P[\alpha t \leq \beta] = \alpha^2 + \beta^2
\]
for \(t_0 \leq \alpha \leq 0 \leq \beta \leq t_1\). For \(t \in \mathbb{R}\), let \(S_t = \{v : y_v \leq t\}\). We will show that with nonzero probability, \(\Psi(S_t) \leq \sqrt{2\lambda_2}\).

### 11.6.1 High level overview of the proof

In this section, we give a high-level overview of the analysis. In particular, we isolate two key lemma’s, and then use them to prove the overall theorem. These lemma’s are proven in subsequent sections.

The overall structure is similar to our previous discussion on sparsest cut. There we had a randomized ratio and we analyzed the expected value of the numerator and denominator separately. These were then combined to show that there exists a good cut. Similarly we start with a lemma addressing the numerator; i.e., the expected weight of edges cut by \(S_t\).

---

\(^2\)We note that specifying the probability of \(t\) being in any given closed interval suffices to describe a continuous distribution \(t \in \mathbb{R}\). Note that the rescaling sets \(P[t_0 \leq t \leq t_1] = 1\).
Lemma 11.11. $E[\bar{w}(S_t)] \leq \sqrt{2\langle y, Ly \rangle \langle y, Dy \rangle}$.

Note the $\sqrt{\cdots}$ on the RHS, which will lead to the $\sqrt{\cdots}$ term in the final bound. The next lemma addresses the denominator; i.e., the volume on the smaller side of the cut.

Lemma 11.12. $E[\min\{\text{vol}(S_t), \text{vol}(\bar{S}_t)\}] = \langle y, Dy \rangle$.

We now have bounds on the expectation of the numerator and denominator of $\Psi(S_t)$. As we showed in our discussion of sparsest cut, this implies there exists a cut $S_t$ whose conductance matches these expected values.

Lemma 11.13. $P[\Psi(S_t) \leq \sqrt{2\langle y, Ly \rangle \langle y, Dy \rangle}] > 0$.

All put together, we obtain the following.

Theorem 11.14. In polynomial time, Fiedler’s algorithm computes $S$ such that $\Psi(S) \leq 2\sqrt{\Psi(G)}$.

Proof. We have

$$P[\Psi(S_t) \leq \sqrt{2\langle y, Ly \rangle \langle y, Dy \rangle}] \geq P[\Psi(S_t) \leq \frac{E[\bar{w}(S_t)]}{E[\min\{\text{vol}(S_t), \text{vol}(\bar{S}_t)\}]})] > 0,$$

as desired. Here (a) is by lemmas 11.11 and 11.12 and (b) is by lemma 11.13.

11.6.2 Expected size of the cut

We first analyze the expected size of the random cut. The first probability gives an upper bound on the problem of an edge $e = \{u, v\}$, in terms of the coordinates $y_u$ and $y_v$.

Lemma 11.15. For an edge $e = \{u, v\} \in E$,

$$P[e \in \partial(S_t)] \leq |y_u - y_v|(|y_u| + |y_v|).$$

Proof. We have three different cases, depending on where $y_u$ and $y_v$ lie relative to $\mu$. We assume without loss of generality that $y_u \leq y_v$.

Case 1. Suppose $y_u \leq 0 \leq y_v$.

Then

$$P[e \in \partial(S_t)] = P[y_u \leq t \leq y_v] = y_u^2 + y_v^2 \overset{(a)}\leq |y_u - y_v|(|y_u| + |y_v|),$$

where (a) observes that $|y_u| \leq |y_u - y_v|$ and $|y_v| \leq |y_u - y_v|$.
Case 2. Suppose $0 \leq y_u \leq y_v$

Then

$$P[e \in \partial(S_t)] = y_v^2 - y_u^2 = (|y_v| + |y_u|) (|y_v| - |y_u|) \overset{(b)}{=} |y_v - y_u|(|y_v| + |y_u|),$$

where (b) observes that $|y_v| = y_v$, $|y_u| = y_u$, and $y_v - y_u = |y_v - y_u|$.

Case 3. Suppose $y_u \leq y_v \leq 0$.

Then

$$P[e \in \partial(S_t)] = y_v^2 - y_u^2 = (|y_v| + |y_u|) (|y_v| - |y_u|) \overset{(c)}{=} |y_v - y_u|(|y_v| + |y_u|),$$

where (c) observes that $|y_v| = y_v$, $|y_u| = y_u$, and $y_v - y_u = |y_v - y_u|$.

The next lemma bounds the expected weight of the cut.

**Lemma 11.11.** $E[\tilde{w}(S_t)] \leq \sqrt{2\langle y, Ly \rangle \langle y, Dy \rangle}$.

**Proof.** We have,

$$E[\tilde{w}(S_t)] = \sum_{\{u,v\} \in E} w(e) P[e \in \partial(S)]$$

$$\leq \sum_{\{u,v\} \in E} w(e) |y_v - y_u|(|y_v| + |y_u|)$$

$$\overset{(a)}{\leq} \sqrt{\sum_{\{u,v\} \in E} w(e)(y_v - y_u)^2} \sqrt{\sum_{\{u,v\} \in E} w(e)(|y_v| + |y_u|)^2}.$$

by (a) Cauchy-Schwartz. For the first term, we have

$$\sum_{\{u,v\} \in E} w(e)(y_v - y_u)^2 = \langle y, Ly \rangle.$$

For the second term, we have

$$\sum_{\{u,v\} \in E} w(e)(|y_v| + |y_u|)^2 \overset{(b)}{\leq} 2 \sum_{\{u,v\} \in E} w(e)(y_v^2 + y_u^2) = 2 \sum_{u} \text{vol}(u)y_u^2 = \langle y, Dy \rangle,$$

where (b) applies the inequality $2ab \leq a^2 + b^2$. 

Kent Quanrud

Fall 2021
11.6.3 Expected volume

It remains to analyze the expected volume, as follows.

**Lemma 11.12.** $\mathbb{E}[\min(\text{vol}(S_t), \text{vol}(\bar{S}_t))] = \langle y, Dy \rangle$.

**Proof.** If $t \geq 0$, then $\text{vol}(S_t) \leq \text{vol}(\bar{S}_t)$, and if $t \leq 0$, then $\text{vol}(\bar{S}_t) \leq \text{vol}(S_t)$. We have $P[t \geq 0] = (t_1)^2$ and

$$\mathbb{E}[\text{vol}(S_t) | t \geq 0] = \frac{1}{t_1^2} \sum_{y_v \geq 0} d^2_v y_v^2.$$  

Similarly, we have $P[t \leq 0] = t_0^2$ and

$$\mathbb{E}[\text{vol}(\bar{S}_t) | t \leq 0] = \frac{1}{t_0^2} \sum_{y_v \leq 0} d^2_v y_v^2.$$  

By conditioning on whether $t$ is $\geq$ or $\leq 0$, we have

$$\mathbb{E}[\min(\text{vol}(S_t), \text{vol}(\bar{S}_t))] = \mathbb{E}[\text{vol}(S_t) | t \geq 0] P[t \geq 0] + \mathbb{E}[\text{vol}(S_t) | t \leq 0] P[t \leq 0] = \sum_{y_v} d^2_v y_v^2 = 2 \langle y, Dy \rangle,$$  

as desired. ■

Additional notes and references

We refer the reader to [Spi19; Tre16] for more on spectral graph theory.

11.7 Exercises

**Exercise 11.1.** Let $G = (V,E)$ be an undirected graph with $m$ edges and $n$ vertices, and positive edge weights $w : E \to \mathbb{R}_{>0}$. Let $L : \mathbb{R}^V \to \mathbb{R}^V$ be the Laplacian of $G$. Prove that $G$ is connected iff for any $x \notin \text{span}(\mathbb{1})$, we have $\langle x, Lx \rangle > 0$.

**Exercise 11.2.** Finish the proof of lemma 11.3, by deriving the derivative $f'(\epsilon)$ and showing that $f'(0) = 0$ implies that $\langle x, Lu \rangle = 0$. Where do we use the assumption that $L$ is symmetric?
Exercise 11.3. Here we prove the lower bound in Cheeger’s inequality (Theorem 11.9), $\Psi(G) \geq \lambda_2/2$. Let $S \subseteq V$ induced the minimum conductance cut; i.e., $\text{vol}(S) \leq \text{vol}(V)/2$ and $\Psi(G) = \Psi(S)$. Consider the vector $x = D^{1/2}\mathbb{1}_S$ and let $y = D^{1/2}\mathbb{1}_{\bar{S}}$.

1. Show that
   $$\frac{\langle x, Mx \rangle}{\langle x, x \rangle} = \frac{\langle \mathbb{1}_S, L\mathbb{1}_S \rangle}{\langle \mathbb{1}_S, D\mathbb{1}_S \rangle}, \quad \frac{\langle y, My \rangle}{\langle y, y \rangle} = \frac{\langle \mathbb{1}_{\bar{S}}, L\mathbb{1}_{\bar{S}} \rangle}{\langle \mathbb{1}_{\bar{S}}, D\mathbb{1}_{\bar{S}} \rangle},$$
   and $\langle x, y \rangle = 0$.

2. Show that for any $\alpha, \beta \neq 0$, we have
   $$\frac{\langle \alpha x + \beta y, M(\alpha x + \beta y) \rangle}{\langle \alpha x + \beta y, \alpha x + \beta y \rangle} \leq 2\Psi(G).$$

3. Argue that one can choose $\alpha, \beta \neq 0$ such that $\langle d, \alpha x + \beta y \rangle = 0$.

4. Finally, prove that the second smallest eigenvector of $M$ is at most $2\Psi(G)$. 
Chapter 12

Electrical flow

12.1 Electrical networks

For the sake of our discussion, an electrical network is an undirected graph $G = (V,E)$ with positive edge weights $r : E \rightarrow \mathbb{R}_{>0}$ called resistances. If one attaches a battery to two vertices $s$ and $t$, it induces a current that (in our discussion) is a unit flow from $s$ to $t$. As the electricity flows from $s$ to $t$, it is said to take the path of least resistance? What is the path of least resistance? A computer scientist or operations researcher might suggest this is the shortest path from $s$ to $t$ with respect to the resistance. But physics does not do combinatorial optimization; physics does calculus. From a calculus point of view, it is more natural to minimize a sum of squares, than just the sum.

To formalize the model, fix an orientation on the edges. We identify each flow with a vector $f \in \mathbb{R}^E$. For each edge $e$, a positive value $f(e) > 0$ means that $f(e)$ units of flow are routed in the same direction as the orientation of $e$. A negative value $f(e) < 0$ means that $|f(e)|$ units of flow are routed in the opposite direction. Then the electrical flow is the
12. Electrical flow

12.1. Electrical networks

\((s,t)\)-flow \(f\) minimizing the electric energy \(\sum_e r_e f_e^2\).

That is, the electrical flow is the solution of a constrained optimization problem. The squared terms encourage the flow to spread out. While there is still some preference to shorter paths, rather than put all the flow along the shortest \((s,t)\)-path, the electrical flow will spread out such as in the following computer. (The following is not the truly optimum electrical flow).

Let us define a linear map

\[ B : \mathbb{R}^E \rightarrow \mathbb{R}^V \]

that maps flows to the net flows at each vertex. That is, for a flow \(f\), and a vertex \(v\), we have

\[(Bf)_v = \text{net flow of } f \text{ at } v.\]

The above definition is linear in \(f\), so \(B\) is a linear map. Let \(d \in \mathbb{R}^V\) be the demands of our problem; namely,

\[ d_v = \begin{cases} 
1 & \text{if } v = s \\
-1 & \text{if } v = t \\
0 & \text{otherwise.}
\end{cases} \]

A flow \(f\) is a unit \((s,t)\)-flow iff \(Bf = d\). Now, while we are principally interested in \((s,t)\)-flow, the following discussion extends to any set of flow demands \(d \in \mathbb{R}^V\) (the only requirement being that \(\langle 1, d \rangle = 0\)). In the above algebraic notation, the electrical flow is obtained as the solution to the following optimization problem:

\[
\text{minimize } \langle f, Rf \rangle = \sum_{e \in E} r_e f_e^2 \text{ over } f \in \mathbb{R}^E \text{ s.t. } Bf = d. \quad (12.1)
\]

For a fixed electrical network, the quantity above is a function of \(d\). In general, for \(d \in \mathbb{R}^V\) with \(\langle 1, d \rangle = 0\), the effective resistance of \(d\) is the minimizing potential obtained by the electrical flow routing \(d\).

The rest of this discussion is broadly organized into two parts.
12. Electrical flow

12.2. Structure of electrical flows

Kent Quanrud

1. The first part is about understanding the structure of an electrical flow. This is based on studying the first-order optimality conditions of (12.1).

2. The second part is about computing electrical flows. Remarkably, one can compute electrical flows in nearly linear time!

12.2 Structure of electrical flows

We have seen the electrical flow minimizes a sum of squares subject to linear constraints. This already endows a lot of structure to electrical flows by understanding the optimality conditions of such a problem. The connection to graphs then leads to further interpretations of these conditions.

12.2.1 Convex optimization s.t. linear constraints

The reader may recall that for unconstrained convex minimization problems, a point $x$ is a minimum solution iff the derivative of the objective function is 0. This is no necessarily true in constrained optimization. For linear constraints, however, we know the following.

**Theorem 12.1.** Consider a minimization problem of the form

$$\text{minimize } \varphi(x) \text{ over } x \in \mathbb{R}^m \text{ s.t. } Ax = b,$$

where $\varphi : X \to \mathbb{R}$ is a convex and smoothly differentiable function over a vector space $X$, $A : X \to Y$ is a linear map, and $b \in Y$ is a vector. Let $x$ be a optimum solution to the problem. Then

$$\varphi'(x) = A^T y \text{ for some } y \in \mathbb{R}^n.$$

**Proof.** Let $\ker(A) = \{ x : Ax = 0 \}$ denote the kernel of $A$; i.e., the set of vectors that map to 0.

Claim. $\varphi'(x)$ is orthogonal to $\ker(A)$. Suppose not. Then there exists $z \in \ker(A)$ such that

$$\langle \varphi'(x), z \rangle < 0.$$

But then for sufficiently small $t > 0$,

$$\varphi'(x + tz) \approx \varphi(x) + t \langle \varphi'(x), z \rangle < \varphi(x),$$
while

\[ A(x + tz) = Ax + tAz = Ax = b. \]

Then \( x + tz \) is not optimal, a contradiction, and proving the claim.

Now, recall that the image of \( A \), denoted by \( \text{im}(A) \), is the set

\[ \text{im}(A) \overset{\text{def}}{=} \{ Ax : x \in X \}, \]

and that the coimage of \( A \), denoted \( \text{coim}(A) \), is the subspace of \( X \) orthogonal to \( \ker(A) \):

\[ \text{coim}(A) \overset{\text{def}}{=} X/\ker(A) = \{ x \in X : \langle x, y \rangle = 0 \text{ for all } y \in \ker(A) \}. \]

Note that we have shown that \( \varphi'(x) \) is orthogonal to \( A \). Basic linear algebra (sometimes called the “fundamental theorem of linear algebra”) states that \( A \) and \( A^T \) both induce isomorphisms (i.e., one-to-one linear mappings) between \( \text{coim}(A) \) and \( \text{im}(A) \). The one-to-one mapping \( A^T : \text{im}(A) \rightarrow \text{coim}(A) \) implies that there exists \( y \in Y \) such that \( A^T y = \varphi'(x) \).

\[ \square \]

### 12.2.2 Ohm’s Law

If we apply theorem 12.1 to the electrical flow problem, then we obtain the following identity called **Ohm’s law**. The vector \( p \in \mathbb{R}^V \) in the following theorem is called the **electric potentials** induced by \( d \).

**Theorem 12.2.** A flow \( f \in \mathbb{R}^E \) subject to demands \( d \) is the electrical flow iff there exists \( p \in \mathbb{R}^V \) such that \( f = R^{-1}B^Tp \).

**Proof.** Suppose \( f \) is the electrical flow. Observe that the gradient of the objective function is \( Rf \). By theorem 12.1, there exists \( q \in \mathbb{R}^V \) such that \( 2Rf = Bq \); hence \( p = q/2 \) is the desired set of electrical potentials.

Conversely, suppose \( f = R^{-1}B^Tp \) for some \( p \in \mathbb{R}^V \). Let \( g \) be any other flow with \( Bg = d \). Recall that for any convex function \( \varphi \), we have

\[ \varphi(y) \geq \varphi(x) + \langle \varphi'(x), y - x \rangle. \]

For our convex function \( \varphi(f) = \langle f, Rf \rangle / 2 \), we have

\[
\langle g, Rg \rangle - \langle f, Rf \rangle \overset{(a)}{\geq} 2\langle Rf, g - f \rangle \overset{(b)}{=} 2\langle B^Tp, g - f \rangle = 2\langle p, B(g - f) \rangle = 2\langle p, d - d \rangle = 0.
\]

Here (a) applies convexity of \( \varphi(x) \), and (b) substitutes \( r = R^{-1}Bp \).  \[ \square \]
12. Electrical flow

12.3 Effective conductance

Recall that the Laplacian of a graph $G$ with edge weights $w(e)$ is the symmetric matrix $L : \mathbb{R}^V \to \mathbb{R}^V$ defined by

$$\langle x, Lx \rangle = \sum_{e = \{u, v\} \in E} w(e)(x_u - x_v)^2.$$ 

**Lemma 12.3.** Let $w(e) = 1/r_e$ for all $e$. Then $L = BR^{-1}B^T$.

To prove lemma 12.3, since both matrices are symmetric, it suffices to show that $\langle x, Lx \rangle = \langle x, BR^{-1}B^T x \rangle$ for all $x$. We leave this calculation to the reader as exercise 12.1.

Recall that the effective resistance of $d$ is the minimum energy attained by the electrical flow. The effective resistance has the following closed form, drawing a direct connection between the electrical flow and the pseudoinverse of $L$. Note that $L^{-1}d$ is well-defined because we assume $G$ is connected and $d$ is orthogonal to the kernel of $L$; i.e., $\perp$.

**Theorem 12.4.** Given a connected electrical network with resistances $r$, let $L$ be the Laplacian of the corresponding undirected graph with weights $1/r$. Let $d$ be a fixed set of demands inducing an electrical flow $f$ with electrical potentials $p$. We have the following.

1. $Lp = d$.
2. (effective resistance of $d$) $= \langle d, L^{-1}d \rangle = \langle p, Lp \rangle = \langle p, d \rangle$.

**Proof.** Let $f$ be the electrical flow and $p$ the electrical potentials with respect to $d$. For the first claim, we have

$$Lp = BR^{-1}B^T p = Bf = d.$$ 

For the second, we have

$$\langle f, Rf \rangle \overset{(a)}{=} \langle R^{-1}B^T p, RR^{-1}B^T p \rangle = \langle p, Lp \rangle = \langle Lp, L^{-1}Lp \rangle \overset{(b)}{=} \langle d, L^{-1}d \rangle$$

where (a) is by Ohm’s Law.

### 12.3 Effective conductance

Consider the following optimization problem.

$$\text{minimize } \langle p, Lp \rangle \text{ over } \langle p, d \rangle = 1. \quad (12.2)$$
In the special case of $d = 1_t - 1_s$, we are seeking the potentials $p$ minimizing $\langle p, Lp \rangle$ subject to $s$ and $t$ being separated by 1 unit. The optimum value to $(12.2)$ is sometimes called the effective conductance.

The first order conditions tell us that the optimum solution $p^*$ satisfies $2Lp^* = \lambda d$, hence $p^* = (\lambda/2)L^{-1}d$, for some scalar $\lambda$. To identify $\lambda$, we plug into $\langle p^*, d \rangle = 1$: We have

$$1 = \langle p^*, d \rangle = \frac{\lambda}{2} \langle d, L^{-1}d \rangle,$$

hence

$$\lambda = \frac{2}{\langle d, L^{-1}d \rangle}.$$

Note that $2/\lambda = \langle d, L^{-1}d \rangle$ is the effective resistance of $d$. Returning to $(12.2)$, we have

$$\langle p^*, Lp^* \rangle = \langle Lp^*, L^{-1}Lp^* \rangle = \frac{\lambda^2}{4} \langle d, L^{-1}d \rangle = \frac{1}{\langle d, L^{-1}d \rangle}.$$ 

The following theorem summarizes our developments.

**Theorem 12.5.** $(12.2)$ has optimum solution $p^* = L^{-1}d/2\langle d, L^{-1}d \rangle$ and optimum value $1/\langle d, L^{-1}d \rangle$.

In particular we have the following symmetry between the effective resistance and the effective conductance of a demand vector $d$.

**Corollary 12.6.** For any demands $d$, the effective resistance of $d$ is the reciprocal of the effective conductance of $d$.

An alternative interpretation of this symmetry is as follows.

**Corollary 12.7.** For any $d$-flow $f$, and any potentials $p$ with $\langle p, d \rangle = 1$, we have

$$\langle f, Rf \rangle \langle p, Lp \rangle \geq 1.$$

The inequality is tight for a unique $f$ and $p$ (modulo $1$).

**12.4 Computing electrical flows**

We now turn to the algorithmic question of computing electrical flows. Let $d$ be a fixed set of demands. We now know that the electrical flow $f$ goes hand-in-hand
with the electrical potentials $p$, by Ohm’s law. We also know that $p$ is given by $p = L^{-1}d$; equivalently, by the solution $p$ to the system of equations

$$Lp = d.$$  

So we are interested in solving a system of linear equations.

One way to solve a linear system such as the above is by algebraic techniques such as Gaussian elimination. These algorithms give exact solutions but take $O(n^3)$ time for an $n \times n$ system of equations. An alternative approach is by iterative methods that obtain (additive) approximations to the system of equations. Meaning that, to solve for $x$ in a system $Ax = b$, they might try to find a point $x$ such that

$$\|Ax - b\| \leq \epsilon$$

for some error parameter $\epsilon$. (Iterative solvers may use other geometric measures of error besides this one as we will see.)

We will study iterative solvers for computing electrical flows. Amazing they can be solved in nearly linear time as first shown in a breakthrough work by [ST04]. We will not necessarily be able to present a linear solver in full but hopefully we can isolate some of the enabling techniques, and the interesting connections to combinatorial techniques we have already discussed.

### 12.4.1 Iterative solvers

Consider a system of equations of the form

$$Lx = d,$$

where we assume $L$ is an $n \times n$ positive semi-definite. (One might imagine $L$ as the Laplacian of a graph but we will not leverage the additional structure until the following section). In fact by restricting ourselves to the image/cokernel of $L$ we may assume $L$ is positive definite. Let $0 < \lambda_1 \leq \cdots \leq \lambda_n$ be the eigenvalues of $L$ in strictly increasing order. (For connected graph Laplacian we will work in $n - 1$ dimensions orthogonal to $1$.) Here we will take an approach based on random walks called the Richardson iteration [Ric11].

To motivate the idea, let us first assume that $\lambda_n \leq 1$. (This is the case for random walks.) If $Lx^* = d$, then we also have

$$x^* = (I - L)x^* + d;$$

that is, $x^*$ is a fixed point of the process mapping $x$ to $(I - Lx) + d$. Thus consider an iterative sequence of points $x_0, x_1, \cdots \in \mathbb{R}^n$ defined by $x_0 = 0$ and

$$x_t = (I - L)x_{t-1} + d$$
thereafter. Then
\[ x^* - x_t = (I - L)(x^* - x_{t-1}) \]
for each \( t > 0 \); unrolling (and recalling that \( x_0 = 0 \)) we have
\[ x^* - x_t = (I - L)^t x^*. \]

Now consider \( \|x^* - x_t\| \), which we treat as a measure of error. Then
\[
\|x^* - x_t\|^2 = \|(I - L)^t x^*\|^2 \langle x^*, (I - L)^{2t} x^* \rangle.
\]
Consider the RHS. We first point out that \( I - L \), hence \( (I - L)^{2t} \), is symmetric.
It is also positive semi-definite because \( \langle x, Lx \rangle \leq 1 \) for all \( x \). Thus the RHS is
bounded above by
\[ \mu_{2t} \|x^*\|^2 \]
where \( \mu_{2t} \) is the maximum eigenvalue of \( (I - L)^{2t} \).

Next we point out that the eigenvalues of \( I - L \) are \( 1 - \lambda_i \) for the eigenvalues \( \lambda_i \)
of \( L \), which follows easily from the spectral decomposition of \( L \) (or the Courant-
Fischer minimax, etc). In particular the eigenvalues of \( I - L \) lies in the range
\( [1 - \lambda_n, 1 - \lambda_1] \); that is, \( \mu_1 \leq 1 - \lambda_1 \). Moreover, the eigenvalues of \( (1 - L)^{2t} \) are the
\((2t)\)th powers of the eigenvalue of \( (I - L)^{2t} \) (which is easy to see from spectral
theorem). This gives
\[ \mu_{2t} = \mu_1^{2t} = (1 - \lambda_1)^{2t}. \]

Thus we have an \textit{exponential} convergence rate determined by the eigenvalues,
\[ \|x^* - x_t\| \leq (1 - \lambda_1)^t \|x^*\| \leq e^{-\lambda_1 t} \|x^*\|; \]
e.g., \( t \geq \log(1/\epsilon)/\lambda_1 \) implies \( \|x^* - x_t\| \leq \epsilon \|x^*\|.

Of course we made the assumption that the eigenvalues are bounded above
by 1, which may not hold for general \( L \). But of course \( Lx = d \) is the same as
\[ \frac{1}{\lambda_n} Lx = \frac{1}{\lambda_n} d; \]
the scaled operator \( L' \) operator \( L' \equiv L/\lambda_n \) now has eigenvalues between
\( \lambda_1/\lambda_n \) and 1. We apply the same procedure now to \( L' \) and obtain a convergence rate of
\[ \|x^* - x_t\| \leq \epsilon^{-\rho t} \|x^*\| \]
where \( \rho = \frac{\lambda_1}{\lambda_n} \).
Alternative error bounds. Besides bounding the distance between $x_t$ and $x^*$, which we cannot actually measure without knowing the solution $x^*$, one can also measure the difference between $Lx_t$ and $d$, which we can measure. To this end we have

$$d - Lx_t = L(x^* - x_t) = L(I - L)(x^* - x_{t-1}) = L(I - L)^t x^* \equiv (I - L)^t Lx^* = (I - L)^t d.$$  

Here in (b) we observe that $L$ commutes with $I - L$. In the simpler case where $\lambda_n \leq 1$, we similarly conclude that

$$\|d - Lx_t\| \leq e^{-\lambda_1 t} \|d\|.$$  

In the general case we rescale $L$ and have

$$\|d - Lx_t\| \leq e^{-\rho t} \|d\| \text{ where } \rho = \frac{\lambda_1}{\lambda_n}. $$

**Better iterative methods.** The algorithm we described is extremely simple, and there are more clever ways to iterate which will lead to an exponential rate with coefficient roughly $\sqrt{\lambda_1/\lambda_n}$, which is better, such as via Chebyshev polynomials or the conjugate gradient method. The more important point is the link we have established between the convergence rate and the ratio $\lambda_1/\lambda_n$ between the smallest and largest eigenvalues of $L$.

### 12.4.2 Preconditioning

The techniques of the previous section have drawn our attention to the (normalized) spectral radius, $\lambda_n/\lambda_1$, of the linear system $L$. The next idea we discuss – called **preconditioning** – tries to decrease this ratio.

At a high level, we want to invert $L$ on a fixed input $d$, and computing $L^{-1}$ is computationally expensive. Suppose we had access to a (positive definite) matrix $K$ and its inverse $K^{-1}$ where $K$ is closely approximates $L$. One might hope to use $K$ and $K^{-1}$.

Suppose $K$ and $L$ are related by

$$K \leq L \leq \alpha K \quad (12.3)$$

for some $\alpha > 1$. This means that for all $x$, we have

$$\langle x, Kx \rangle \leq \langle x, Lx \rangle \leq \alpha \langle x, Kx \rangle.$$  

In this case we call $K$ an $\alpha$-approximation of $L$.  

157
Consider the matrix $K^{-1}L$. $K^{-1}L$ is not positive definite but it is similar, hence has the same eigenvalues, as $L^{1/2}K^{-1}L^{1/2}$. It follows from (12.3) that
\[
I \leq K^{-1/2}LK^{-1/2} \leq \alpha I,
\]
hence all the eigenvalues lie between 1 and $\alpha$. That is, its spectral radius is only $\alpha$!

Applying the Richardson iteration technique above to $L^{1/2}K^{-1}L^{1/2}$ results in a number of iterations proportional to $\alpha$. However we do not have $L^{1/2}$ in hand. We will briefly repeat the analysis for $K^{-1}L$ but first comment that morally, the analysis is based on powers of $L^{1/2}K^{-1}L^{1/2}$ and $K^{-1}L$, and the $i$th powers of these two matrices are similar, as
\[
(L^{1/2}K^{-1}L^{1/2})^i = L^{1/2}(K^{-1}L)^i L^{-1/2}.
\]
In particular they have the same eigenvalues and this is really at the core of the convergence.

Let $M = 1/\alpha K^{-1}L$. We have
\[
Mx^* = d' \text{ where } d' = \frac{1}{\alpha} K^{-1}d.
\]
Then
\[
x^* = (I - M)x^* + d',
\]
which motivates the iterative sequence defined by $x_0 = 0$ and
\[
x_t = (I - M)x_{t-1} + d',
\]
which implies
\[
x^* - x_t = (I - M)^tx^*.
\]
This time, rather than bounding $\|x^* - x_t\|$, we bound the “$L$-norm” $\|x^* - x_t\|_L$.
(Here $\|y\|_L = \langle y, Ly \rangle$). We have
\[
\|x^* - x_t\|_L^2 = \langle (I - M)^tx^*, (I - M)^tx^* \rangle,
\]
\[
\overset{(d)}{=} \langle L^{1/2}x^*, (I - L^{1/2}K^{-1}L^{1/2})^2t L^{1/2}x^* \rangle
\]
\[
\leq (1 - 1/\alpha)^2 \|x^*\|_L^2.
\]
Here for (c) we observe that
\[
L^{1/2}M^i = L^{1/2}(K^{-1}L)^i = L^{1/2}(L^{1/2}K^{-1}L^{1/2})^i L^{1/2}
\]
12. Electrical flow

12.4. Computing electrical flows

for every power $i$.

To sum up, then, given the inverse $K^{-1}$ of an $\alpha$-approximation $K$ of $L$, we can converge to $x^*$ at a rate of

$$\|x^* - x_t\|_L \leq e^{-t/\alpha}\|x^*\|$$

One can apply the same technique to the more aggressive methods such as Chebyshev polynomials and conjugate gradient, to reduce the dependence to $\sqrt{\alpha}$.

12.4.3 Preconditioning graphs

Having developed some more general techniques let us now return to graphs, where $L$ is the graph Laplacian. What makes for a good preconditioned $K$ of $L$?

1. $K$ should approximate $L$: e.g., $K \leq L \leq \alpha K$ for some $\alpha > 1$, the smaller the better.

2. We should be able to obtain the inverse of $K$, $K^{-1}$.

For the second point we note that we don’t have to compute $K^{-1}$ explicitly, so long as we can compute $K^{-1}y$ for a given input $x$. That is to say that we should be able to solve the system $Kx = y$ fairly easily.

One way to construct such a $K$, pioneered by [Vai90], is to take $K$ to be the Laplacian of a subgraph of $H$. The subgraph aspect ensures that $K \leq L$. Meanwhile we also want $K$ to be easy to invert. The easiest type of Laplacian system to solve is a tree. Indeed, we have seen that Laplacian systems are solving electrical flow problems, and flow problems on trees are trivial. So now let $K$ be the Laplacian of a spanning tree $T$ of $G$. It remains to bound the maximum eigenvalue of $K^{-1}L$. We will do so by bounding $\text{trace}(K^{-1}L)$ from above.

Let us first try to understand $K^{-1}$ a little bit better when $K$ is the Laplacian of a spanning tree $T$. Let $x = 1_t - 1_s$ represent a unit $(s,t)$-demand for some $s,t \in V$. Then $\langle x, K^{-1}x \rangle$ is the effective resistance from $s$ to $t$. We also know that the electrical $(s,t)$-flow in $K$ is simply the unique path in $T$ from $s$ to $t$. Let us denote this path $T_{s,t}$. Recalling that the resistances of an electrical network are the reciprocals of the edge weights of the Laplacian we have $\langle x, K^{-1}x \rangle = \sum_{e \in T_{s,t}} \frac{1}{w(e)}$, that is, the length of the $(s,t)$-path in $T$ with respect to the edge lengths $1/w(e)$.

Now, fix an orientation of every edge in $G$. For each arc $e = (u,v)$, let $1_e = 1_v - 1_u$ represent the demands of a unit $(u,v)$-flow. Recall that the Laplacian $L$ has the form

$$L = \sum_e w(e)(1_e \otimes 1_e).$$
We now have
\[ \text{trace}(K^{-1}L) = \sum_{\epsilon} w(\epsilon) \text{trace}(K^{-1}(1_\epsilon \otimes 1_\epsilon)) = \sum_{\epsilon} w(\epsilon) \langle 1_\epsilon, K^{-1}1_\epsilon \rangle \]
\[ = \sum_{\epsilon} w(\epsilon) \sum_{f \in T_\epsilon} \frac{1}{w(f)}. \]  \hspace{1cm} (12.4)

Now, for each $\epsilon$, the summand
\[ w(\epsilon) \sum_{f \in T_\epsilon} \frac{1}{w(f)} = \frac{\sum_{f \in T_\epsilon} \frac{1}{w(f)}}{1/w(\epsilon)} \]
is precisely the stretch of edge $\epsilon$ in $T$ with respect to the edge lengths $w(\epsilon)$. The RHS of (12.4) is the total stretch of the spanning tree $T$. To minimize the RHS, then, we want a low-stretch spanning tree! Recall that in section 8.2 we presented an $\alpha$-stretch spanning tree with stretch $\alpha = n^{o(1)}$ due to [AKPW95]. This gives us the bound
\[ \text{trace}(K^{-1}L) \leq \alpha m \leq m^{1+o(1)}. \]

We point out that there are randomized low-stretch spanning trees with $\alpha = O(\log n \log \log n)$ [AN19]; this gives us a bound of $E[\text{trace}(K^{-1}L)] \leq O(m \log n \log \log n)$.

Putting everything together, applying the low-stretch preconditioner and applying the simple Richardson iteration means that we need about $\tilde{O}(m)$; or, with Chebyshev polynomials, $\tilde{O}(\sqrt{m})$. In particular there is no long a (potentially unbounded) dependence on the eigenvalues of $L$. This connection – using low-stretch spanning trees as a preconditioner – is due to [BH01].

### 12.4.4 Towards a nearly linear running time

By now we have established an $\tilde{O}(m^{3/2})$ running time. This can be improved even further to nearly linear time as first shown by Spielman and Teng [ST04]. We refer to Spielman [Spi19] for these developments.

### 12.5 Exercises

**Exercise 12.1.** Complete the proof of lemma 12.3.

### 12.6 Additional notes and references

There are nice connections between the electrical flow and random walks; see [DS84]. We refer the reader to [Spi19] for more on this and many related topics.
Chapter 13

Max flow via electrical flow

In 2013, Madry [Mad13] obtained a $\tilde{O}(m^{10/7})$ running time for maximum flow in uncapacitated directed graphs – the first improvement since Even and Tarjan [ET75] (cf. section 2.4). Then Lee and Sidford [LS14] obtained an $O(m\sqrt{n})$ running time for polynomial capacities, the first improvement in this regime since Goldberg and Rao [GR98] (cf. section 2.6). By now there are many more papers in this line of work. Besides the obvious appeal of improved running times, we highlight the conceptually un-traditional approach (which has some precursors in [DS08; KV96; Vai89]): speeding up interior point LP solvers via electrical flows and additional ideas therein.

Here we will discuss a follow up result by [Mad16] that simplifies [Mad13]. In our discussion we will focus on maximum flow in unit capacity undirected graphs, allowing for parallel edges. It is known that exact maximum flow in directed graph can be reduced to exact maximum flow in undirected graphs. We should point out that the reduction changes the size of the maximum flow to $\Omega(m)$. We should also mention that [Mad16]'s result extends to integer capacities with a polynomial dependence on the maximum capacity; we are assuming unit capacities only for simplicity.

We will first present a running time of $\tilde{O}(m^{3/2})$, then make the additional adjustments to improve the running time to $\tilde{O}(m^{10/7})$.

13.1 Augmenting electrical flows, continuously

Let $G = (V, E)$ be an undirected graph with $m$ edges and $n$ vertices, and let $s, t \in V$ be a fixed source and sink, respectively. The goal is to compute the $(s, t)$-max flow. We may assume without loss of generality that $(2/3)$rd of the edges are of the form \{s, t\}, which is of technical convenience.

In the ensuing, we fix an orientation of the edges, and represent a flow by
a vector \( f \in \mathbb{R}^E \), where \( f_e \) represents the flow on \( e \) in the orientation of \( e \). It is convenient to assume that all \( \{s,t\} \)-edges are directed from \( s \) to \( t \).

As mentioned above, we will compute the maximum flow as a sum of electrical flows (plus some minor modifications). To build out the high-level ideas we will first develop a continuous algorithm, that adds electrical flows over time. Formally, let \( t \geq 0 \) represent the time starting from \( t = 0 \). We let \( f^{(t)}(t) \) denote the total flow computed at time \( t \) where initially \( f^{(0)} = 0 \). Every time step \( t \) we propose some resistances \( r^{(t)} \) and compute an electrical \((s,t)\)-flow \( g^{(t)} \), with the additional constraint that \( g^{(t)} \) is feasible in the residual graph of \( f^{(t)} \). We then augment \( f^{(t)} \) by \( g^{(t)} \) in the continuous sense by setting \( \frac{d}{dt}f^{(t)} = g^{(t)} \). The goal is to converge to a maximum flow in the minimum amount of time.

The main design decision in this continuous framework is to propose resistances \( r^{(t)} \) at each time step \( t \). The resistances \( r^{(t)} \) determine an electrical \((s,t)\)-flow \( g^{(t)} \) of (say) unit size. Given \( g^{(t)} \), we might as well set \( \frac{d}{dt}f^{(t)} = \delta^{(t)} g^{(t)} \) for \( \delta^{(t)} > 0 \) as large as possible subject to \( g^{(t)} \) being feasible in the residual graph. That is to say that we make progress inversely proportional to the congestion of \( g^{(t)} \) with respect to the residual graph.

To design \( r^{(t)} \) to reduce the congestion of \( g^{(t)} \), we ought to make an edge’s resistance \( r^{(t)}_e \) decreasing in the residual capacity of the edge \( e \); e.g., \( 1 - f^{(t)}_e \). Here, however, we point out that residual graphs are inherently directed graphs while electrical networks are always undirected graphs. So \( r^{(t)}_e \) should somehow be decreasing in the residual capacity of both directions. Let us set

\[
 r^{(t)}_e = \frac{1}{(1 - f^{(t)}_e)^2} + \frac{1}{(1 + f^{(t)}_e)^2}
\]

The first term is inversely proportional to the square of the residual capacity in the direction of \( e \); the second is for the opposite orientation. Note that \( r^{(t)}_e \) is dominated by the term for the smaller of the two residual capacities. To make this point clearer let us denote the minimum residual capacity of an edge \( e \) by

\[
 u^{(t)}_e \overset{\text{def}}{=} \min \{1 - f^{(t)}_e, 1 + f^{(t)}_e\}.
\]

Then we have

\[
 \frac{1}{(u^{(t)}_e)^2} \leq r^{(t)}_e \leq \frac{2}{(u^{(t)}_e)^2}.
\]

It is helpful to think of \( r^{(t)}_e \) as essentially \( 1/(u^{(t)}_e)^2 \).

Henceforth at time \( t \) let \( g^{(t)} \) be the unit electrical \((s,t)\)-flow with respect to the resistances \( r^{(t)}_e \) defined above. Let \( R^{(t)} \) denote the effective resistance from \( s \)
to $t$, which we recall is precisely the energy of $g^{(t)}$:

$$R^{(t)} = \sum_{e} r_e^{(t)} (g_e^{(t)})^2.$$ 

Recall that we are interested in the congestion. To this end, let $\rho^{(t)}$ denote the absolute congestions with respect to $u_e^{(t)}$, defined by

$$\rho_e^{(t)} = \frac{|g_e^{(t)}|}{u_e^{(t)}}.$$ 

Note that $\rho^{(t)}$ overestimates the actual congestion of $g^{(t)}$ in the residual graph of $f^{(t)}$, as we are pessimistically assuming the flow is in the direction of the tighter residual edge. Thus we have $\delta^{(t)} \geq 1/\|\rho^{(t)}\|_{\infty}$. To bound the RHS, reconsider the effective resistance $R^{(t)}$. Substituting $r_e^{(t)} \leq 2/(u_e^{(t)})^2$, we have

$$R^{(t)} \leq \sum_{e} (\rho_e^{(t)})^2 = \|\rho^{(t)}\|_{2}^2 \geq \|\rho^{(t)}\|_{\infty}^2.$$ 

This gives us the following relation between $\delta^{(t)}$ and the effective resistance:

$$\delta^{(t)} \geq \frac{1}{\sqrt{R^{(t)}}};$$

the next task is to bound the effective resistance $R^{(t)}$.

Recall Ohm’s law: for every electrical flow $g^{(t)}$, there are vertex potentials $x^{(t)}$ such that

$$r_e^{(t)} g_e^{(t)} = x_v^{(t)} - x_u^{(t)} \quad \text{for every arc } (u,v)$$

As we augment $f^{(t)}$ along electrical flows $g^{(t)}$, let us also set add up the vertex potentials $x^{(t)}$ into a vector $y^{(t)} \in \mathbb{R}^V$. We initially set $y^{(0)} = \emptyset$, and then

$$\frac{d}{dt} y^{(t)} = x^{(t)} \quad \quad \text{at time } t.$$ 

In particular for every arc $e = (u,v)$ we have

$$\frac{d}{dt} y^{(t)}_v - \frac{d}{dt} y^{(t)}_u = r_e^{(t)} g_e^{(t)} = r_e^{(t)} \frac{d}{dt} f_e^{(t)}.$$ 

Let $q_e^{(t)}$ be the antiderivative of $r_e^{(t)}$ as a function of $f_e^{(t)}$: namely,

$$q_e^{(t)} = \frac{1}{1 - f_e^{(t)}} - \frac{1}{1 + f_e^{(t)}}.$$ 

163
To help interpret \( q_e^{(t)} \), observe that we have \( q_e^{(t)} = f_e^{(t)}/u_e \) up to a factor of 2. Now, we have

\[
\frac{d}{dt} q_e^{(t)} = r_e^{(t)} \quad \text{and} \quad \frac{d}{dt} q_e^{(t)} = r_e^{(t)} \frac{d}{dt} f_e^{(t)}.
\]

That is,

\[
\frac{d}{dt} y_v^{(t)} - \frac{d}{dt} y_u^{(t)} = r_e^{(t)} \frac{d}{dt} f_e^{(t)} = \frac{d}{dt} q_e^{(t)}
\]

for every arc \( e = (u, v) \). Initially we also have

\[
y_v^{(0)} - y_u^{(0)} = 0 = q_e^{(0)}.
\]

Thus we also have the dual correspondence

\[
q_e^{(t)} = y_v^{(t)} - y_u^{(t)}
\]

for all arcs \( e = (u, v) \) at all times \( t \).

The vertex potentials \( y_v^{(t)} \) act like distance labels; we may even assume that \( y_s^{(t)} = 0 \) for all \( t \) in which case \( y_v^{(t)} \) can be interpreted as a distance from \( s \). To elaborate further, for an arc \( e = (u, v) \), we can interpret \( q_e^{(t)} \) as an edge length of \( q_e^{(t)} \) from \( u \) to \( v \) and an edge length of \(-q_e^{(t)}\) from \( v \) to \( u \). Then \( y_v^{(t)} - y_u^{(t)} \) is always the length of the shortest path from \( s \) to \( v \) with respect to these edge lengths. In previous flow algorithms we have seen that distances from \( s \) (in some sense of distance) measure our progress towards a maximum flow. The following lemma is very similar (even if the calculations appear different).

**Lemma 13.1.** The residual flow at time \( t \) has size at most \( m/\left(y_v^{(t)} - y_s^{(t)}\right) \).

**Proof.** Let \( h \) be a residual max flow of size \( F' = |h| \). Consider the quantity

\[
\sum_e h_e q_e^{(t)} = \sum_{e=(u,v)} h_e (y_v^{(t)} - y_u^{(t)})
\]

where the equality is via the dual correspondence inherited from Ohm’s law. First consider the LHS. The reader may verify that for every edge \( e \) we have \( h_e q_e^{(t)} \leq 1 \). Consequently we have

\[
\sum_e h_e q_e^{(t)} \leq m.
\]

For the RHS, we have

\[
\sum_{e=(u,v)} h_e (y_v^{(t)} - y_u^{(t)}) = F' (y_v^{(t)} - y_s^{(t)})
\]

by conservation of flow. This gives our desired inequality. \( \blacksquare \)
We see then that \( y^{(t)} \) – and in particular \( y^{(t)}_t - y^{(t)}_s \) – is giving us a measure of progress, and the algorithm terminates before \( y^{(t)}_t = y^{(t)}_s + m \). Meanwhile \( y^{(t)}_t - y^{(t)}_s = q^{(t)}_e \) for any arc of the form \( e = (s, t) \), and \( q^{(t)}_e \) is roughly inversely proportional to \( u_e \). We also know the sign of \( q^{(t)}_e \) because the flows \( g^{(t)}_e \) will only route flow in the direction of \( e \), hence \( f^{(t)}_e \geq 0 \) and \( q^{(t)}_e \geq 0 \). It is easy to see that either \( u_e \geq 1/2 \) or \( q^{(t)}_e \geq u_e / 3 \). Connecting back to lemma 13.1 gives us the following lemma.

**Lemma 13.2.** Every \((s, t)-arc\) has residual capacity at least a \((1/3m)\) fraction of the residual maximum flow.

Now recall our assumption that a full \((2/3)\)rd of the arcs are of the form \((s, t)\). The flow saturating every \((s, t)-arc\) sends \( \Omega(F') \) units of flow from \( s \) to \( t \) and has total electrical energy \( O(m) \). Scaling down by \( \Omega(F') \) gives a unit \((s, t)-flow\) with total electrical energy at most \( O(m/(F')^2) \). This gives an upper bound on the effective resistance of

\[
R^{(t)} \leq O\left(\frac{m}{(F')^2}\right).
\]

In turn we have

\[
\delta^{(t)} \geq \Omega\left(\frac{F' / \sqrt{m}}{1}\right).
\]

This leads to a continuous algorithm running in \( \tilde{O}(\sqrt{m}) \) units of time, so to speak.

Next we will turn to developing a discrete algorithm that approximates the continuous one above. The \( \tilde{O}(\sqrt{m}) \)-time bound developed here will translate to \( \tilde{O}(\sqrt{m}) \) iterations each taking nearly linear time to compute an electrical flow. We will carry the high-level structure of the continuous algorithm, but will have to deal with lower order errors that arise from the discretization error.

### 13.2 A discrete max flow algorithm

We now move to developing a discrete analogue of the continuous algorithm above. The algorithm we present will work in iterations where each iteration augments one electrical flow (in the same way as the continuous algorithm) besides some additional work that we introduce later. Most of our discussion will be framed to analyze a single iteration, and we no longer annotate the objects by a time index \((t)\). We let \( f \) denote the flow at the beginning of a fixed iteration, likewise for \( y \). It is convenient to treat \( r \), \( q \), and the alike as functions: e.g., \( r(x) \overset{\text{def}}{=} 1/(1-x)^2 + 1/(1 + x)^2 \), and so forth, so we can analyze their changes in greater detail. We abbreviate the values with respect to the flow \( f \) at the beginning of the iteration by \( r_e \overset{\text{def}}{=} r(f_e) \), \( q_e \overset{\text{def}}{=} q(f_e) \), and so forth.
13.2.1 Discretization error

Let us first understand where the error is introduced and its extent. Now, recall the dual vertex potentials $y$. Because we electrical flows and potentials continuously along Ohm’s law we maintained the correspondence

$$q_e = y_v - y_u$$

for all arcs $e = (u, v)$ exactly, which in turn allowed us to measure our progress towards a maximum flow. In the discrete setting, however, there will be some error entering the correspondence above.

Suppose we augment our flow $f$ and potentials $y$ by an electrical flow $g$ with potentials $x$. Fix an arc $e = (u, v)$; by Ohm’s law we have $r_e g_e = x_v - x_u$. For simplicity (and without loss of generality) let us assume $g_e \geq 0$. Clearly $y_v - y_u$ increases by $x_v - x_u$. However the increase from $q_e$ to $q_e + g_e$ is not exactly $r_e g_e = x_v - x_u$, since $q(x)$ is not linear. Instead, Taylor’s remainder theorem offers the bound

$$|q(f_e + g_e) - q_e - (x_v - x_u)| = |q(f_e + g_e) - q_e - r_e g_e|$$

$$\leq \sup_{0 \leq h \leq g_e} \frac{r'(f_e + h) |g_e|^2}{2}; \quad (13.1)$$

here $r'(x)$ denotes the derivative of $r(x)$. Now,

$$r'(x) = \frac{1}{(1 - x)^3} - \frac{1}{(1 + x)^3};$$

and in particular, we have

$$\frac{1}{u^3(x)} \leq |r'(x)| \leq \frac{2}{u^3(x)}.$$

We now obtain

$$(13.1) \leq \sup_{0 \leq h \leq g_e} \frac{g_e^2}{u^3(f_e + h)} \leq \frac{g_e^2}{(u_e - g_e)^3}.$$  

Let $\rho_e = \frac{|g_e|}{u_e}$ denote the congestion on edge $e$ with respect to $u$ (just as above).

$$|q(f_e + g_e) - q_e - r_e g_e| \leq \frac{g_e^2}{(1 - \rho_e)^3 u_e^3} = \left( \frac{\rho_e^2}{(1 - \rho_e)^3 u_e} \right) \leq (1 + O(\rho_e)) \frac{\rho_e^2}{u_e},$$

assuming $\rho_e$ is small than 1. This establishes an upper bound on the error introduced by a discrete augmentation, as a quadratic function of the congestion.
Lemma 13.3. Let \( f, y, g, x, \rho \) be as described above. If \( \rho < 1 \), then when augmenting \( f \) by \( g \) and \( y \) by \( x \), for each arc \( e = (u, b) \), the absolute difference \( |f_e - (y_v - y_u)| \) increases by at most \( (1 + O(\rho_e)) \rho_e^2 / u_e \).

Curiously our error is inversely proportional to \( u_e \). It turns out to be helpful to understand the error inversely proportional to \( u_e \). Let us define a nonnegative function \( \xi(f, y) \in \mathbb{R}_{\geq 0}^E \) by

\[
\xi_e(f, y) = \left| q_e \right| f_e - (y_v - y_u) u_e
\]

for each arc \( e = (u, v) \). As usual we omit the arguments \( (f, y) \) when they refer to the values at the beginning of the current iteration. We think of \( \xi \) as an error vector. Now lemma 13.3 can be rephrased in terms of \( \xi \) as follows.

Lemma 13.4. Under the same conditions as lemma 13.3, for each arc \( e = (u, b) \), \( \xi_e \) increases by at most \( (1 + O(\rho_e)) \rho_e^2 \).

13.2.2 Recentering / correcting

It is not hard to believe that the error \( \xi \) hurts our analysis – the exact tolerance to \( \xi \) will be discussed later. For the time being it is clear that we would like to reduce \( \xi \) if possible. Such an error-reduction method is provided by the following lemma.

Lemma 13.5. There exists constants \( c_1, c_2 > 0 \) for which the following holds. Let \( (f, y) \) have error vector \( \xi \), with \( \|\xi\|_2 \leq c_1 \). Then one can compute \( (f', y') \) with error \( \xi' \) such that:

1. \( f \) and \( f' \) have the same net flows at each vertex.
2. \( \|\xi'\|_2 \leq c_2 \|\xi\|_2^2 \).

Proof. The high level idea is to build a circulation out of two flows, \( g \) and \( h \). \( g \) is designed to aggressively reduce \( \xi \), but is not an \((s,t)\)-flow. More precisely, \( g \) is defined by

\[
g_e = \frac{y_u - y_u - q_e}{r_e}
\]

for \( e = (u, v) \), which is motivated by the first order approximation \( q(f_e + g_e) \approx q_e + r_e g_e \). \( h \) is an electric flow matching the change in net flow from \( g \); i.e., \( g + h \) is a circulation. Let \( x \) be the electrical potentials for \( h \); we augment \( y \) with \( x \) as well. \((h, x)\) reintroduce some error but we will see that it is relatively small compared to the error reduced by \( g \).
13. Max flow via electrical flow

13.2. A discrete max flow algorithm

Kent Quanrud
Fall 2021

There are two main observations driving the proof. The first is that $g$ has low congestion as long as $\xi$ is small: $|g_e| \leq ||\xi||_2 u_e$ for all $e$. In particular for sufficiently small $||\xi||_2$ the Taylor approximation is still very good (lemma 13.4). The second is that $-g$ upper bounds the electrical energy of $h$, which allows us to argue that $|h_e| \leq ||\xi||_2 u_e$ for each $e$ as well. Hence lemma 13.4 is also applicable to $h$. Since lemma 13.4 roughly squares the error we obtain the claim.

We now formalize the argument. To set up the analysis, we have

$$\xi' = \left| q\left(f_e + g_e + h_e\right) - (y_o + x_o - (y_u - x_u) \right|$$

$$\leq \left| q\left(f_e + g_e + h_e\right) - q\left(f_e + g_e\right) - (x_o - x_u) \right| + \left| q\left(f_e + g_e\right) - (y_o - y_u) \right|.$$ 

We analyze each term in the RHS separately, starting with the second. Observe that $g_e = \frac{\xi_e}{u_e} \leq \frac{\xi_e u_e}{u_e} \leq ||\xi||_2 u_e$.

That is, if $\xi$ is small, then $g$ has low congestion. We have

$$|g_e| = \frac{\xi_e}{u_e r_e} \leq \xi_e u_e \leq ||\xi||_2 u_e.$$

The electrical energy of $g$ is bounded above by that of $-g$, hence

$$\sum_e s_e h^2_e \leq 0 \left( \sum_e r_e g^2_e \right) \leq 0 \left( \sum_e r_e \xi_e^2 u^2_e \right) \leq O\left(||\xi||_2^2\right).$$

In particular, for every edge $e$, we have

$$\frac{|h_e|}{u_e} \leq \sqrt{\sum_e q_e h^2_e} \leq O\left(||\xi||_2\right).$$

Therefore

$$|q\left(f_e + g_e + h_e\right) - q\left(f_e + g_e\right) - (x_o - x_u) \leq O\left(||\xi||_2^2\right)/u_e$$

by lemma 13.3.
13.2.3 An $m^{3/2}$ running time

We now have the algorithmic ingredients for a discrete running time. Previously, we continuously augmented by an electric $(s,t)$-flow. This time, we will discretely add an electric $(s,t)$-flow—potentially increasing $\|\xi\|_2$—and then add a correcting circulation to reduce $\|\xi\|_2$. We will always keep $\|\xi\|_2$ small enough so that the quadratic reduction in lemma 13.5 dominates the increase from the flow.

To bound the running time we need to argue that we can add a large $(s,t)$-electrical flow in each iteration. Previously, in the continuous setting, we leveraged the dual potentials $y_v$ to argue that the $(s,t)$-arcs alone constituted a large flow. The analysis was clean because the vertex potentials were perfectly aligned with the arc lengths $q_e$. Here we extend the analysis to account for $\xi$.

Lemma 13.6. Suppose $\|\xi\|_\infty \leq 1$. Then

$$F'(y_t - y_s) \leq m + \|\xi\|_1 \leq m + \sqrt{m}\|\xi\|_2.$$  

Proof. Just as before we have

$$F'(y_t - y_s) = \sum_e f'_e y_e \leq \sum_e f'_e q_e + |f'_e|\frac{\xi_e}{u_e}. \quad (13.2)$$

We claim that for each edge $e$, $f'_e q_e + |f'_e|\frac{\xi_e}{u_e} \leq 1 + \xi_e$. If so, then the proof concludes with

$$(13.2) \leq \sum_e 1 + \xi_e \leq 1 + \sqrt{m}\|\xi\|_2.$$  

Now fix an edge $e$. We have

$$f'_e q_e + |f'_e|\frac{\xi_e}{u_e} = \frac{f'_e}{1 - f_e} - \frac{f'_e}{1 - f_e} + |f'_e|\frac{\xi_e}{u_e}.$$  

The claim is immediate if $|f'_e| \leq u_e$. Otherwise $f'_e$ must flow in the larger direction (of capacity $1 + u_e$), and

$$\frac{f'_e}{1 - f_e} - \frac{f'_e}{1 + f_e} + |f'_e|\frac{\xi_e}{u_e} \leq 1 - \frac{(1 - \xi_e)}{u_e} \leq 1$$

because $\xi_e \leq 1$.

Lemma 13.7. Every $(s,t)$-arc $e$ has residual capacity at least a $\Omega(1/(1 + \|\xi\|_2) m)$-fraction of the residual max flow.

Proof sketch. Recall the proof of lemma 13.2. The difference now is just that $q_e \geq (y_t - y_s) - \xi_e u_e$, which leads to the claim.
We now complete the analysis of an $m^{3/2}$-half algorithm. Initially we have $\zeta = 0$, and for a sufficiently small constant $c > 0$, we can preserve $\|\zeta\|_2 \leq c$ at the beginning of each iteration by following up each augmenting electrical flow with a correcting circulation. Each iteration, the electrical flow must have congestion (relative to $u_e$) at most a fixed constant. Since we are maintaining $\|\zeta\|_2$ small, by virtue of lemma 13.7, we can always augment a $\Omega(1/\sqrt{m})$-fraction of the residual flow with sufficiently small congestion. This gives us $\tilde{O}(\sqrt{m})$ iterations where each iteration requires two electrical flows. Electrical flows take $\tilde{O}(m)$ time and this gives an $\tilde{O}(m^{3/2})$ running time overall.

Before proceeding to even faster algorithms, we remark that it is already quite interesting to shoehorn electrical flows into a maximum flow algorithm that can match $[ET75]$.

### 13.3 Boosting to an $m^{10/7}$ running time

We have arrived to the finale of our discussion, where we improve the running time from $\tilde{O}(m^{3/2})$ to $\tilde{O}(m^{10/7})$. While some of the following calculations might seem more complicated, let us suggest that the biggest conceptual step – establishing a reasonably good max flow algorithm via electrical flows – is already done. Everything that follows is still within this overall framework.

We will focus on reducing the number of iterations from $\tilde{O}(\sqrt{m})$ to $m^{1/2-\epsilon}$, for some $\epsilon > 0$. (Each iteration will still consist of roughly an electrical flow.) We localize our discussion further and let $F'$ be a fixed upper bound on the residual maximum flow, and focus the number of iterations required to decrease $F'$ by a constant factor. We will assume that $F'$ is at least $O(m^{1/2-\epsilon})$, since one can otherwise round the existing flow to a discrete one and complete the maximum flow with $O(m^{1/2-\epsilon})$ augmenting paths. When we refer to the $(s, t)$-electrical flow, we specifically mean the one of size $O(F')$.

Now, the dominating factor in analyzing the progress was the maximum congestion. More specifically, we argued that there is an electrical flow of size roughly $F'$ and the maximum congestion with respect to $u_e$ is at most $O(\sqrt{m})$. (We would prefer $O(m^{1/2-\epsilon})$). That argument was by comparison to the flow saturating all $(s, t)$-arcs, which has size $\Omega(F')$ and total electrical energy $O(m)$. The same argument also implies that the average congestion of the electrical flow was $\leq 1$. So we are being limited by edges whose congestion far exceeds the average.

Consider the extreme case where there is just one congested edge $e$ holding us back. (Such an edge is necessarily not an $(s, t)$-arc.) Then one idea is to artificially increase (say, double) the resistance on $e$. Then the electrical flow would want to send less flow along that edge. This approach has some technical
issues. If we artificially increase $r_e$, what happens to $\xi_e$? How do we charge off the computation for the electrical flow that identified $e$ as a bottleneck?

**Boosting.** Let us first address the first question – how to artificially increase the resistance $r_e$ of an edge $e = (u,v)$ while still preserving the connection between $q_e$ and $y_v - y_u$. One idea might be to replace $e$ with a path of two arcs with the same resistance. However this would disrupts the connection with $y$ and increase $\xi_e$. Loosely speaking, we can correct this by adding roughly $1/u_e$ edges with negative length, to obtain the following.

**Lemma 13.8.** Let $e$ be any edge. We can replace $e$ with a path of $O(1/u_e)$ edges (with $q_e$ and $y_v$’s appropriately configured) such that (a) the total resistance along the path is at least double the resistance of $e$, and (b) all new edges have $\xi_e = 0$.

We defer the proof of lemma 13.8 to later and continue with our high-level discussion. Now, let us assume a policy where we apply lemma 13.8 to any edge with congestion at least $m^{1/2} - \epsilon$. This gives the first step of our algorithm, as follows.

1. If there is an arc $e$ with congestion $\rho_e \geq m^{1/2} - \epsilon$, (effectively) double its resistance.
2. ...

What are the consequences of boosting? The flow on the boosted edge should go down, and the effective resistance should go up. We also increase edges in inverse proportion to the capacity – how many edges? The first lemma translates the congestion $\rho_e$ to $u_e$.

**Lemma 13.9.** For each edge $e$, $u(e) \geq \rho_e / m^{1/2+\epsilon}$.

The following lemma gives a bound on how much the congestion goes up. Anticipating later developments, the lemma more broadly analyzes what happens when we boost a collection of edges $H$.

**Lemma 13.10.** Suppose we boost a set of edges $H$. Then the effective resistance increases by at least

$$\frac{1}{2} \sum_{e \in H} \rho_e^2.$$

When we effectively double the resistance of an edge with congestion $\rho_e \geq m^{1/2} - \epsilon$, we introduce $m^{2\epsilon}$ edges, and increase the effective resistance by about $\rho_e^2 \geq m^{1-2\epsilon}$. Meanwhile the effective resistance is bounded above by $O(m)$. So
we can only execute step 1 above about $O(m^{2\epsilon})$ times in a row, before we are guaranteed the congestion is at most $m^{1/2-\epsilon}$, and we can augment with a step size of $\delta = \Omega(m^{1/2+\epsilon})$. The problem now is that each iteration of step 1 still takes nearly linear time to identify a large congestion edge, which (more than) offsets the gain from taking a larger step size from the lower congestion. One might hope that the effects of boosting might carry over to after the augmentation. After all, the $O(m)$ upper bound holds after the augmentation too. To this end, we seek a lower bound on the decrease to the effective resistance from an augmentation. The following lemma (proven later) provides such a bound as long as the step size is inversely proportional to the $\ell_3$-norm of the congestion vector.

**Lemma 13.11.** For a step size $\delta \leq O(1/\|\rho\|_3)$, the effective resistance from augmenting (and correcting) decreases by at most $O(1 + O(\|\rho\|_3^2/R))$-multiplicative factor.

In particular, to evenly offset the increase from boosting above, we can augment as long as $\|\rho\|_3 \leq O(m^{1/2-\epsilon})$. Let us add this to our developing algorithm, which now stands as follows.

1. If there is an arc $e$ with congestion $\rho_e \geq m^{1/2-\epsilon}$, (effectively) double its resistance.
2. If $\|\rho\|_3 \leq O(m^{1/2-\epsilon})$, then augment the flow and correct $\xi$ with a circulation.
3. ...

The two steps above are not exhaustive. It is possible that all arcs have congestion $\rho_e \leq m^{1/2-\epsilon}$, but also $\|\rho\|_3 \geq O(m^{1/2-\epsilon})$. In that case, there must still be a large number of arcs with congestion nearly $m^{1/2-\epsilon}$. Instead of boosting one arc with extremely high congestion, the following lemma boosts a moderate number of arcs with still high congestion.

**Lemma 13.12.** Suppose $\|\rho\|_3 \geq \Omega(m^{1/2-\epsilon})$, and $\|\rho\|_{\infty} \leq m^{1/2-\epsilon}$. Let $H$ be a maximal set of up to $m^{4\epsilon}$ edges each of congestion at least $m^{1/2-3\epsilon}$. Then boosting all of these edges introduces $m^{8\epsilon}$ edges and increases the effective resistance by at least $\Omega(m^{1-2\epsilon})$.

We now include this as our third and final option in the following algorithm. In hindsight, one could omit the first step, as it is subsumed by the third.

1. If there is an arc $e$ with congestion $\rho_e \geq m^{1/2-\epsilon}$, (effectively) double its resistance.
2. If $\|\rho\|_3 \leq O(m^{1/2-\epsilon})$, then augment the flow and correct the duals with a circulation.
3. Otherwise boost up to $m^{4\epsilon}$ edges of congestion at least $m^{1/2-3\epsilon}$. 

172
Now each iteration either boosts some edges, or augments up to $\Omega(F'm^{\epsilon-1/2})$ units of flow. The boosting is amortized against the augmentation (plus $m^{2\epsilon}$ additional iterations). Consequently we need at most $m^{1/2-\epsilon}$ iterations to reduce the residual flow by half. It remains to choose $\epsilon$. One constraint is that we can only introduce at most $m$ (or so) edges if we want to keep the remaining analysis intact. Each boosting iteration introduces at most $m^{8\epsilon}$ edges. Then $m^{1/2-\epsilon} \cdot m^{8\epsilon} \leq m$ implies $\epsilon \leq 1/12$. Note that for $\epsilon = 1/14$ we also have $m^{2\epsilon} \leq m^{1/2-\epsilon}$ so the number of augmentations remains the bottleneck. This gives a total of $m^{3/7}$ iterations.

This completes our high-level analysis. Up to proving the lemma’s introduced above, we have the desired $O(m^{10/7})$ running time.

13.3.1 Boosting an edge: proof of lemma 13.8

The construction is as follows. Let $e = (u, v)$ be a fixed arc; we assume without loss of generality that $f_e > 0$. We will replace $e$ with a path from $u$ to $v$, where the total length is the same as $e$, all carrying $f_e$ units of flow from $u$ to $v$.

1. Two of the edges are identical with $e$. Note that we now exceed the required length by a factor of 2.

2. The remaining edges are artificially designed to satisfy the following requirements.
   
   (a) They have at least unit capacity in both directions, so the maximum flow does not change.
   
   (b) They have negative length (in the $(u, v)$-direction) and their sum of lengths negates the length of $e$.

Until this point, we have assumed that every edge had unit capacity, and defined the edge lengths and resistances accordingly. We now introduce edges that, so to speak, have infinite capacity in one direction and some finite capacity $\alpha \geq 1$ in the reverse direction. Let $\alpha \geq 1$ be a parameter TBD. Such an edge has length function

$$q(x) = -\frac{1}{\alpha + x},$$

where $x$ is the flow in the forwards direction. The resistance and the rest for such an edge follow accordingly. Note that such an edge is to some extent easier to work with, since the minimum capacity is always given by the backwards direction.

Returning to our boosting, we require $\alpha \geq 1$ and $\frac{1}{\alpha + f_e}$ to be a multiple of the original length of $e$. Now the length of $e$ is roughly $u_e$, and so there is always
some choice of $\alpha = O(1)$ for which add $O(1/u_e)$ new edges and make the lengths add up exactly.

This proves lemma 13.8.

13.3.2 $\rho_e$ vs. $u_e$: proof of lemma 13.9

In this section we prove lemma 13.9, which bounds $u_e$ of any highly congested edge $e$. This lemma is motivated by the fact the number of edges introduced by boosting a highly congested edge is inversely proportional to $u_e$.

Let $e = (u, v)$ be a fixed arc. Let $d = (s, t)$ denote an $(s, t)$-arc. We have

$$\frac{\rho_e}{u_e} \leq |g_e| r_e = |x_u - x_v| \leq |x_t - x_s| = |g_d| r_d \leq \frac{1}{u_d} \leq O\left(\frac{m}{F}\right) \leq m^{1/2+\epsilon}.$$  

(a) is because the optimum potentials for $(s, t)$-demands always set $|x_v - x_u| \leq |x_t - x_s|$. (b) is because $F' \geq m^{1/2} - \epsilon$.

13.3.3 Boosting and effective resistance: proof of lemma 13.10

In this section we prove lemma 13.10, regarding the increase to the effective resistance when boosting a set of edges. Let $x_e$ denote the vertex potentials for $g$. Suppose we double the resistance of a set $H$ of edges. Let $s_e$ denote the new resistances (doubling $r_e$ for $e \in H$) and let $S$ denote the effective resistance with the doubled resistance. We have $S \geq R$ by monotonicity, and we want to analyze how much the effective resistance has increased. Recall that the effective conductance is the reciprocal of the effective resistance, and that the potentials $x/R$ solve the effective conductance problem with respect to the resistances $r_e$ (cf. section 12.3). Moreover, $x/R$ can be used to bound the effective conductance with respect to the revised resistances; in particular, we have

$$\frac{1}{S} \leq \sum_{e \in \{u, v\}} \frac{(x_u - x_v)}{R^2 s_e}$$  

by this method. Since $s_e = 2r_e$ for $e \in H$ and $s_e = r_e$ for $e \notin H$ we have

$$\frac{R^2}{S} \leq R - \sum_{e \in \{u, v\} \in H} \frac{(x_u - x_v)^2}{2r_e} = R - \sum_{e \in H} \frac{r_e f_e^2}{2} \leq R - \sum_{e \in H} \frac{f_e^2}{2u_e^2} = R - \frac{1}{2} \sum_{e \in H} \rho_e^2.$$  

Rearranging, we have

$$S \geq R - \frac{S}{2R} \sum_{e \in H} \rho_e^2 \geq R - \frac{1}{2} \sum_{e \in H} \rho_e^2,$$

as desired.
13.3.4 Augmentation for small $\|\rho\|_3$: proof of lemma 13.11

Now we prove lemma 13.11. We briefly recall the motivation. We currently have a tool, boosting, for increasing the resistance of high-congestion edges, which increases the effective resistance. Now we want to bound the amount that the effective resistance decreases when doing an augmentation. If this quantity is comparable to the energy gain from boosting, then we can (essentially) charge the number of boosting iterations to the number of augmentations. In this analysis we specifically consider a step size $\delta \leq O(1/\|\rho\|_3)$.

We assume the same notation as for lemma 13.10. Let $x_v$ denote the vertex potentials for $g$, $s_e$ denote the resistance of an edge $e$ after augmenting by $g$ and adding a correcting circulation. We let $\nu_e$ denote the congestion with respect to the correcting circulation and $u_e$. Let $R$ denote the effective $(s,t)$-resistance with respect to $r$, and let $S$ denote the effective resistance with respect to $s$. As in the proof of lemma 13.10, we have

$$\frac{1}{S} \leq \sum_{e=\{u,v\}} \frac{(x_v - x_u)^2}{s_e R^2 s_e}.$$  

We also have

$$s_e \geq (1 - O(\delta \rho_e + \nu_e)) r_e$$

for each edge $e$. Altogether, we have

$$\frac{R^2}{S} \leq \sum_{e=\{u,v\}} \frac{(x_v - x_u)^2}{s_e} \leq \sum_{e} (1 + O(\delta \rho_e + \nu_e)) r_e g_e^2$$

$$= R + O\left(\delta \sum_{e} \rho_e r_e g_e^2 + O\left(\sum_{e} \nu_e r_e g_e^2\right)\right)$$

$$\leq R + O\left(\delta \sum_{e} \rho_e^3 + O\left(\sum_{e} \nu_e \rho_e^2\right)\right)$$

$$\leq R + O\left(\delta \|\rho\|_3^3 + \|\rho\|_4^2 \|\nu\|_3\right) \leq R + O\left(\|\rho\|_3^2\right).$$

Here (c) is by Cauchy-Schwartz and (d) is because $\|\nu\|_2 \leq 1$, $\|\rho\|_4 \leq \|\rho\|_3$, and $\delta \leq 1/\|\rho\|_3$. Rearranging we have

$$\frac{R}{S} \leq 1 + O\left(\frac{\delta \|\rho\|_3^3}{R}\right).$$

This completes the proof of lemma 13.11.
13.5 Boosting for large $\|\rho\|_3$: proof of lemma 13.12

This brings us to the final proof of our analysis. We recall the context. We have $\|\rho\|_\infty \leq O(m^{1/2-\epsilon})$ as well as $\|\rho\|_3 \geq m^{1/2-\epsilon}$. Although the maximum congestion is not useful for boosting, as we will see, $\|\rho\|_3 \geq m^{1/2-\epsilon}$ implies that there are still many edges with pretty high congestion, and show that boosting many such edges will still give the desired increase in effective resistance. Let $H$ be a maximal set of up to $m^{4\epsilon}$ edges with congestion $\rho_e \geq cm^{1/2-3\epsilon}$ for some constant $c$. Now, the bound on the total number of edges introduced by boosting follows from lemma 13.8 and lemma 13.9. For the increase in effective resistance, in light of lemma 13.10, it suffices to show that

$$\sum_{e \in H} \rho_e^2 \geq m^{1-2\epsilon}.$$  

Note that this bound is already met if $|H| \geq m^{4\epsilon}$; since $m^{4\epsilon} \cdot m^{1-6\epsilon} = m^{1-2\epsilon}$. So we assume that $|H| < m^{4\epsilon}$ in which case all edges outside $H$ have $\rho_e < cm^{1/2-3\epsilon}$ for a small constant $c$. This limits their contribution to $\|\rho\|_3^3$: we have

$$\sum_{e \notin H} \rho_e^3 \leq cm^{1/2-3\epsilon} \sum_{e \notin H} \rho_e^2 \leq cm^{3/2-3\epsilon} \leq \|\rho\|_3^3/2.$$  

In turn $H$ contributes at least half of $\|\rho\|_3^3$, so

$$\sum_{e \in H} \rho_e^3 \geq \Omega(\|\rho\|_3^3) \geq \Omega(m^{1-2\epsilon}).$$  

Finally, since the maximum congestion is at most $m^{1/2-\epsilon}$, we have

$$\sum_{e \in H} \rho_e^2 \geq \frac{\sum_{e \in H} \rho_e^3}{m^{1/2-\epsilon}} \geq \Omega(m^{1-2\epsilon}),$$  

as desired.
Bibliography


<table>
<thead>
<tr>
<th>Reference</th>
<th>Authors</th>
<th>Title</th>
<th>Conference/Book</th>
<th>Pages</th>
<th>Notes</th>
</tr>
</thead>
</table>


[Che13] Chandra Chekuri. CS583: Approximation algorithms. Fall 2013. URL: http://courses.engr.illinois.edu/cs583/fa2013/.


[Che18b] Chandra Chekuri. CS583: Approximation algorithms. Spring 2018. URL: https://courses.engr.illinois.edu/cs583/sp2018/.


[Ric11] Lewis Fry Richardson. “IX. The approximate arithmetical solution by finite differences of physical problems involving differential equations, with an application to the stresses in a masonry dam”. In:


186


Part II

Additional Background
Probability Theory

Probability theory is simple and intuitive. Let us start with a casual example.

Suppose we flip a coin in the air, and press pause. Will it land heads or tails? Obviously, we don’t know yet. But we can state without ambiguity that half the time it will land heads, and half the time it will land tails. What does it mean to say that half the time it will land heads? There is of course only one coin, and we can’t split the coin in half. We are imagining that, if we repeat the experiment many times, we would expect half the coin tosses to come up heads.

This simple example, which we all understand thoroughly, points to a deeper feature of probability: probability allows us to interpret fractional values as discrete ones. Here, “half heads” does not mean that “half the coin will come up heads”, which is total nonsense; rather it means that half the time the coin will come up heads.

The formal rules of probability are simple. (The only tricky part is sticking to them!) We assume the reader has some acquaintance already with random events and variables, but we will still review the basics. Probability theory assumes an uncertain world where events occur with fixed probabilities. Each event \( A \) has a probability between 0 and 1, denoted

\[
P[A] \in [0,1].
\]

For every event \( A \), there is the complementary event, \( \bar{A} \), of \( A \) not occurring. We always have

\[
P[A] + P[\bar{A}] = 1.
\]

For any two events \( A \) and \( B \), one can define the event that both \( A \) and \( B \) occur: denoted

“\( A \land B \)” or “\( A \cap B \)” or “\( A \) and \( B \)”.

190
We also have the event that either \( A \) or \( B \) occurs, denoted 
\[ A \lor B \] or \( A \cup B \) or \( A \ or \ B \).

Their probabilities satisfy
\[
P[A \land B] \leq \min\{P[A], P[B]\}
\]
and
\[
P[A \lor B] \leq P[A] + P[B].
\]

The second inequality, called the union bound, is surprisingly useful. One can also consider intersections and unions of more than two variables, which lead to the same inequalities as above.

A finite random variable is where a random variable \( X \) takes one of a finite set of values, \( \{x_1, \ldots, x_k\} \).\(^1\) For each outcome \( x_i \), “\( X \) equals \( x_i \)” is an event, with a fixed probability, denoted \( P[X = x_i] \). These probabilities sum to 1:
\[
\sum_{i=1}^{k} P[X = x_i] = 1.
\]

For example, we can describe a coin toss as a random variable \( X \in \{\text{heads, tails}\} \). Let a fair coin be tossed. If the coin comes up heads, then \( X = \text{heads} \). If the coin comes up tails, then \( X = \text{tails} \). We have
\[
P[X = \text{heads}] = P[X = \text{tails}] = \frac{1}{2}.
\]
Observe that these probabilities sum to 1.

If we have two random variables \( X \in \{x_1, \ldots, x_k\} \) and \( Y \in \{y_1, \ldots, y_\ell\} \), then their product \((X, Y)\) forms a random variable in the set \( \{(x_i, y_j) : i = 1, \ldots, k, j = 1, \ldots, \ell\} \). We have probabilities of the form
\[
P[X = x_i, Y = y_j]
\]
that gives the probability that \((X, Y) = (x_i, y_j)\). It is possible, but not necessarily the case, that
\[
P[X = x_i, Y = y_\ell] = P[X = x_i] P[Y = y_\ell]
\]
In the special case where the above holds for all \( x_i \) and \( y_j \), then \( X \) and \( Y \) are independent.

\(^1\)One can also define continuous variable (e.g., that take values continuously between 0 and 1), where sums are replaced by variables.
A. Probability Theory

A.1. Expectations and linearity

For example, suppose $X, Y \in \{\text{heads, tails}\}$ describe coin tosses. If they described different coin tosses, then they would be independent random variables, and each combination of heads and tails would occur with probability $0.25$. That is,

$$P[X = \text{heads}, Y = \text{heads}] = P[X = \text{heads}, Y = \text{tails}] = P[X = \text{tails}, Y = \text{heads}] = P[X = \text{tails}, Y = \text{tails}] = \frac{1}{4},$$

Thus $X$ and $Y$ are independent random variables. If they described the same coin, then we would have

$$P[X = \text{heads}, Y = \text{heads}] = P[X = \text{tails}, Y = \text{tails}] = \frac{1}{2},$$

while

$$P[X = \text{heads}, Y = \text{tails}] = P[X = \text{tails}, Y = \text{heads}] = 0.$$  

Here, $X$ and $Y$ are not independent.

A.1 Expectations and linearity

When a random variable $X$ takes on real values, we can have a well-defined and quantitative notion of “averages”, called the expected value.

**Definition A.1.** Let $X \in \mathbb{R}$ be a real-valued random variable that has a finite set of possible values. Then the expected value of $X$, denoted $E[X]$, is the weighted sum

$$E[X] \overset{\text{def}}{=} \sum_x P[X = x] \cdot x,$$

where the sum is over all values of $x$ where $P[X = x] > 0$.

For continuous random variables, the sum would be replaced by an integral. The average quantity of a random variable is very intuitive.

**Theorem A.2** (Linearity of expectation). Let $X, Y \in \mathbb{R}$ be two random variables. Then

$$E[X + Y] = E[X] + E[Y].$$

The proof of linearity of expectation is left as exercise A.1. The reader may want to first consider the simple case where $X \in \{x_1, x_2\}$ takes on exactly two
values, and \( Y \in \{y_1, y_2\} \) takes on exactly two values. One can generalize to generally finite sets from there.

Observe that linearity of expectation does not make any assumptions about how \( X \) and \( Y \) are structured or related. This makes linearity of expectation extremely useful and often leads to surprising observations.

A simple example of linearity of expectation is as follows. Consider a population of people with various heights. Let \( X \) and \( Y \) be two quantities obtained by the following experiment. Draw one person uniformly at random. Let \( X \) be the length from the waist of this person to the top of their head. Let \( Y \) be the length from the waist of this person to the ground. \( X + Y \) gives to the total height of the person. Note that \( X \) and \( Y \) are highly dependent, since they both measure the same (randomly drawn) person. Linearity of expectation says:

\[
\frac{\text{average height}}{E[X+Y]} = \left( \frac{\text{average length from waist up}}{E[X]} \right) + \left( \frac{\text{average length from waist down}}{E[Y]} \right).
\]

Of course, this makes total sense.

## A.2 Additional references

This note was adapted from a longer note from the Spring 2021 CS580 course [Qua21c] which introduces randomized algorithms. Further notes on randomized algorithms by the author are available from a Fall 2020 course on Randomized Algorithms [raf21]. Another note on background for probability theory is given by [Che20]. [MR95] is a popular textbook on randomized algorithms.

## A.3 Exercises

**Exercise A.1.** Prove linearity of expectation (theorem A.2).
Appendix B

Linear algebra

A matrix $A \in \mathbb{R}^{m \times n}$ is defined by coordinates $A_{i,j} \in \mathbb{R}$ for $i \in [m]$ and $j \in [n]$. A matrix $A$ can be understood as a linear map $A: \mathbb{R}^n \to \mathbb{R}^m$ defined by

$$(Ax)_i = \sum_{j=1}^{n} A_{i,j} x_j$$

for all $i \in [m]$. Conversely every linear map from $\mathbb{R}^n$ to $\mathbb{R}^m$ is defined by a matrix.

The transpose of a matrix $A \in \mathbb{R}^{m \times n}$ is the matrix $A^T \in \mathbb{R}^{n \times m}$ uniquely defined by

$$\langle y, Ax \rangle = \langle A^T y, x \rangle$$

for all $x \in \mathbb{R}^n$ and $y \in \mathbb{R}^m$. The coordinates of the transpose $A^T \in \mathbb{R}^{n \times m}$ are given by

$$A^T_{i,j} = A_{j,i}$$

for all $i \in [n]$ and $j \in [m]$.

(To be continued...)

---

1For a more comprehensive overview, we recommend the following book which is available online in the Purdue library.


2Recall that the inner product $\langle \cdot, \cdot \rangle$ in $\mathbb{R}^n$ is defined by $\langle x, y \rangle = \sum_{i=1}^{n} x_i y_i$. 
Part III

Course Information
Appendix C

Homework assignments

C.1 Homework 1

• Due before class on Friday, September 17.
• Please see appendix D.7 for homework policies.
• Recall that some student submissions will be selected and distributed to the class as solutions (see appendix D.7.7). If you would like your to be excluded from consideration, please state so explicitly at the top of each problem.
• Please do not include your student ID number (which Gradescope is already aware of.)
• An entry for homework 1 is now available on gradescope.com.
• Be aware that problems are untested, and the first homework is often poorly calibrated. Let’s see how this one goes, and we will adjust accordingly going forward.

Problems

1. Exercise 1.1.
2. Exercise 1.2.
4. Exercise 2.2.
5. Exercise 2.3.
C.2 Homework 2

- (!) Due on Monday, October 4, at 6:00 PM.

- Please see appendix D.7 for homework policies.

- Recall that some student submissions will be selected and distributed to the class as solutions (see appendix D.7.7). If you would like your homework to be excluded from consideration, please state so explicitly at the top of each problem.

- Also, if you submit your problems on different pages, it makes it a little easier for me to extract solutions.

- Please do not include your student ID number (which Gradescope is already aware of.)

- All books listed below can be found online. ([Wil19; WS11] have their own websites; [Vaz01] is available via the Purdue library.)

- These problems might be a little harder than last time, particularly since we have only just begun to discuss approximation algorithms and this is a different way of thinking. Just try your best; I would be surprised if everyone solved all of them completely.

Problems

1. Exercise 3.1.

2. Exercise 2.15 in Williamson’s book [Wil19].

3. Exercise 20.8 in Vazirani’s book [Vaz01].

C.3 Homework 3

- Due on Monday, October 25\(^1\), at 6:00 PM.
- Please see appendix D.7 for homework policies.
- Recall that some student submissions will be selected and distributed to the class as solutions (see appendix D.7.7). If you would like your homework to be excluded from consideration, please state so explicitly at the top of each problem.
- Also, if you submit your problems on different pages, it makes it a little easier for me to extract solutions.
- Please do not include your student ID number (which Gradescope is already aware of.)

1. Exercise 4.4.
2. Exercise 5.1.
3. Exercise 5.2.
4. Exercise 5.3.
5. Exercise 5.5.

\(^1\)Pushed back from October 18.
C.4 Homework 4

- This is the last homework!
- Due Wednesday, December 15th\(^2\), at 6:00 PM.
- Here we listed many more problems then usual, and don’t expect you to solve all of them. (Although I encourage you to try all of them!) It would be good to submit solutions to, say, at least 6 of them.
- Please see appendix D.7 for homework policies.
- We probably won’t do the regrade system for this homework because it is so late in the semester.
- Also, if you submit your problems on different pages, it makes it a little easier for me to extract solutions.
- Please do not include your student ID number (which Gradescope is already aware of).
1. Exercise 8.1.
2. Exercise 8.2.
3. Exercise 8.3.
4. Exercise 8.4.
5. Exercise 8.5.
10. Exercise 11.3.

\(^2\)This leaves me a little time before the grade deadline.
Appendix D

Syllabus

D.1 Dates and times

Lectures. Friday’s, 11:30 to 2:20 in the Recitation Building, #123.

Office hours. After class (as long as needed), or by appointment.

Class schedule. See page 2.

D.2 Course description

This is a special topics course on advanced topics on algorithms. We will largely develop algorithms for fundamental problems in graph problems and combinatorial optimization with an infusion of modern techniques and perspectives, such as: randomization, approximation, continuous techniques, advanced data structures, spectral techniques, and more. The goal for most problems is to obtain fast and scalable algorithms for basic problems that (for the sake of efficiency) we allow to be randomized and/or approximate. There is particular emphasis on the interplay of these techniques. While the problems we consider are basic enough to be broadly applicable, we also hope to develop useful and versatile techniques that can transfer readily to diverse applications outside theoretical computer science that need very good algorithms. CS580 or equivalent background is expected. We expect the course to accessible to ambitious undergraduates who have taken the undergraduate algorithms course and have some mathematical maturity.

Topics. A tentative list of topics includes (1) advanced flow algorithms, (2) sparsest cut, (3) the cut-matching game for sparsest cut, (4) local cut algorithms,
D. Syllabus

D.3 Learning outcomes

(5) algorithms for pagerank and personal pagerank, (6) nearly linear time algorithms for electrical flow, (7) network design and optimization, (8) spectral sparsification, (9) randomized rounding, (10) oblivious routing and applications, (11) Nash-Williams tree-packing theorem and other minimax theorems in combinatorial optimization, (12) Edmonds’ polymatroid theorem and applications, (13) multiplicative weight updates, randomized multiplicative, weight updates, and fast approximate LP solvers, (14) submodular optimization, (15) isolating cuts, (16) expander decompositions, (17) primal-dual rounding, (18) randomized minimum cut, (19) matroid optimization, (20) matroid union and intersection, (21) hypergraphs, (22) interior point solvers and applications in combinatorial optimization, (23) first-order optimization and applications to flow problems, (24) nearly linear time algorithms for optimal transport, (25) maximum flow via continuous optimization, (26) dynamic graph data structures and algorithms, and (27) matchings and the matching polytope. These topics have many connections and unifying themes which the course will highlight.

D.3 Learning outcomes

1. A wide acquaintance with the state-of-the-art for graph algorithms and combinatorial optimization.

2. Understand deep connections across problems and theorems in combinatorial optimization.

3. Fluency with randomized techniques and the ability to apply them in new contexts.

4. Comfort with and ability to perform approximation analyses of inexact algorithms.

5. Modeling and solving discrete problems via continuous techniques.

6. The broad ability the synthesize and mix different techniques and perspectives - discrete and continuous, deterministic (e.g., data structures) and randomized, exact and approximate - while staying disciplined in the design and completely rigorous in the analysis of the algorithms.

7. For some, a fruitful research direction in theoretical or applied algorithm design based on the topics and techniques in this course.
D.4 Textbooks

There is no textbook for this course due to the emphasis on current techniques. Lecture notes and pointers to papers and other references on the web will be provided (in this same document). The papers we discuss are often available by open access online; otherwise, they can be obtained (online) via the school library. Some helpful background references for classical topics are:


D.5 Correspondence

The course website is

http://www.maxflowmincut.com,

which should load this document. This document is meant to contain everything related to the course.

D.5.1 Piazza

There is a Piazza for the course at the following address.

piazza.com/purdue/fall2021/cs583

The first goal of Piazza is to increase interactions among the students, and the students are strongly encouraged to help one another. The instructor will check Piazza (somewhat) regularly (and not continuously).

D.5.2 Slack

There is also a slack channel at the following address.

https://join.slack.com/t/cs593ata-f21/signup

The goal of the slack channel, as with Piazza, is to encourage interaction among the students.
D.5.3 Email

The instructor can be reached his their @purdue.edu email addresses. The instructor admits up front that he processes his emails periodically in batch, so emails may not get immediate responses. If the email concerns a question that could benefit others in the class, please post the question to Piazza or Slack instead.

D.6 Grading

- 60% Homework (spread over 5 assignments, each with about 4 word problems)
- 30% Final project
- 10% Class participation

D.7 Homework

This course has regular homework assignments consisting a several word problems due once every two weeks. They will always be due before the corresponding lecture. Homework’s will be posted in Gradescope.

D.7.1 Typesetting

Homework submissions that are not typeset in LaTeX or equivalent will not be graded. Some tips on typesetting are listed below.

D.7.2 On writing

The onus is on the student to make the arguments in their solution clear, and points will be docked if the grader cannot easily verify that the solution is correct. The class is as much about communicating complicated ideas as solving problems and applying techniques. Particularly clear exposition may be selected as homework solutions which is rewarded with extra credit (see below).

D.7.3 Gradescope

Homework will be collected online at gradescope.com. You may already be added to the course via brightspace; otherwise you can use the code RWBBBN to enroll.
D.7.4 Collaboration

Collaboration is allowed and interaction among students is encouraged. Currently we are allowing an unlimited number of students per submission for the word problems\(^1\). If you collaborate with others, we expect it to be earnest. Collaborators should discuss all the problems, together – as opposed to just divvying up the problems to work on separately. Everyone is expected to give real thought to all of the problems; otherwise, they will not absorb the class material, and may fall behind.

Please also indicate any other students that you may have worked on the problems with.

D.7.5 Dropping scores

In the overall homework grade, the bottom 10\% of word problem scores will be dropped. More precisely, if there are \(n\) total word problems assigned in homework, then the \(\lceil n/10 \rceil\) lowest scores will be dropped. This is largely to help catch the arbitrary exceptions that arise throughout a semester.

D.7.6 Late policy

For word problems, we have a simple late policy where you can submit:

1. Up to 5 minutes late for 5\% off.
2. Up to 1 week late for 25\% off.

We try not to make exceptions for minor things and expect the late policies and score-dropping policy to handle these cases. Of course when real and formal exceptions arise we will adjust accordingly on an individual basis. In this case, please email the staff ASAP (and not, say, after or just before the deadline.)

D.7.7 Solutions

The staff will select exemplary submissions and publish them on Piazza as solutions. If you have a strong preference to be excluded from consideration for a particular homework problem, please indicate it clearly and explicitly at the top of your submission (for each problem). If you a strong preference to be anonymous if your homework is selected, please indicate that on your document.

Selected solutions will get 10\% extra credit.

We plan to put up the solutions quickly, such as the day after the homework is collected.

\(^{1}\text{We reserve the right to change this if things get out of hand}\)
D.7.8 Resubmitting homework

You might have noticed that there is both a late policy and a plan to post solutions on the same day as the submission deadline. You can take advantage of this by comparing the answer key to your own submission, and possibly resubmit your homework late even with the benefit of the answer key. If you do use the posted answer key in a resubmission, we expect you to cite it accordingly, and still express the solution in your own words.

D.7.9 IDK

One may simply write “I don’t know” or “IDK” and automatically get 25% of the possible points.

D.7.10 Typesetting tips

- The standard for typesetting mathematical and scientific articles is LaTeX. Even if you do not know LaTeX now, you will definitely have to learn it sooner or later in your graduate studies.
- The instructor uses emacs to write LaTeX, but any editor will do\(^2\). There is also a website called overleaf.com for typesetting LaTeX.
- Alternatively, the software typora allows one to write LaTeX within a markdown document, which is particularly easy to use.
- LyX is another popular latex editor that is WYSIWYG.
- There are several apps for scanning documents (e.g., when inserting pictures) that are much better than taking a photo. The instructor uses scanbot, and other popular apps include microsoft office lens, camscanner, and evernote scannable.

D.8 Academic integrity

Behavior consistent with cheating, copying, and academic dishonesty is not tolerated. Depending on the severity, this may result in a zero score on the assignment or exam, and could result in a failing grade for the class or even expulsion. Purdue prohibits “dishonesty in connection with any University activity. Cheating, plagiarism, or knowingly furnishing false information to the University are examples of dishonesty.” (Part 5, Section III-B-2-a, University Regulations) Furthermore,\(^2\)But really, just use emacs.
the University Senate has stipulated that “the commitment of acts of cheating, lying, and deceit in any of their diverse forms (such as the use of substitutes for taking examinations, the use of illegal cribs, plagiarism, and copying during examinations) is dishonest and must not be tolerated. Moreover, knowingly to aid and abet, directly or indirectly, other parties in committing dishonest acts is in itself dishonest.” (University Senate Document 7218, December 15, 1972)
You are expected to read both Purdue’s guide to academic integrity (http://www.purdue.edu/purdue/about/integrity_statement.html) and Prof. Gene’s Spafford’s guide (http://spaf.cerias.purdue.edu/integrity.html) as well. You are responsible for understanding their contents and how it applies to this class.

D.9 Posting Class Material

Posting material associated with this class (e.g., solutions to homework sets or exams) without the written permission of the instructor is forbidden and may be a violation of copyright.

D.10 Purdue’s Honor Pledge

As a boilermaker pursuing academic excellence, I pledge to be honest and true in all that I do. Accountable together - we are Purdue. https://www.purdue.edu/provost/teachinglearning/honor-pledge.html

D.11 Grief Absence Policy

Purdue University recognizes that a time of bereavement is very difficult for a student. The University therefore provides the following rights to students facing the loss of a family member through the Grief Absence Policy for Students (GAPS). According to GAPS Policy, students will be excused for funeral leave and given the opportunity to earn equivalent credit and to demonstrate evidence of meeting the learning outcomes for missed assignments or assessments in the event of the death of a member of the student’s family.

D.12 Conduct and Courtesy

Students are expected to maintain a professional and respectful classroom environment. This includes: silencing cellular phones, arriving on time for class, speaking respectfully to others and participating in class discussion. You may
use non-disruptive personal electronics for the purpose class participation (e.g., taking notes).

**D.13 Students with Disabilities**

Purdue University is required to respond to the needs of the students with disabilities as outlined in both the Rehabilitation Act of 1973 and the Americans with Disabilities Act of 1990 through the provision of auxiliary aids and services that allow a student with a disability to fully access and participate in the programs, services, and activities at Purdue University. If you have a disability that requires special academic accommodation, please make an appointment to speak with the instructor within the first three (3) weeks of the semester in order to discuss any adjustments.

It is the student’s responsibility to notify the Disability Resource Center (http://www.purdue.edu/drc) of an impairment/condition that may require accommodations and/or classroom modifications. We cannot arrange special accommodations without confirmation from the Disability Resource Center.

**D.14 Emergencies**

In the event of a major campus emergency, course requirements, deadlines and grading percentages are subject to changes that may be necessitated by a revised semester calendar or other circumstances beyond the instructor’s control. Relevant changes to this course will be posted onto the course website and/or announced via email. You are expected to read your purdue.edu email on a frequent basis. Emergency Preparedness: Emergency notification procedures are based on a simple concept: If you hear an alarm inside, proceed outside. If you hear a siren outside, proceed inside. Indoor Fire Alarms are mean to stop class or research and immediately evacuate the building. Proceed to your Emergency Assembly Area away from building doors. Remain outside until police, fire, or other emergency response personnel provide additional guidance or tell you it is safe to leave. All Hazards Outdoor Emergency Warning sirens mean to immediately seek shelter (Shelter in Place) in a safe location within the closest building. “Shelter in place” means seeking immediate shelter inside a building or University residence. This course of action may need to be taken during a tornado, a civil disturbance including a shooting or release of hazardous materials in the outside air. Once safely inside, find out more details about the emergency. Remain in place until police, fire, or other emergency response personnel provide additional guidance or tell you it is safe to leave. In both cases, you should seek additional clarifying information by all means possible: Purdue Home
page, email alert, TV, radio, etc. Review the Purdue Emergency Warning Notification System multi-communication layers at http://www.purdue.edu/ehps/emergencypreparedness/warning-system.html. Please review the Emergency Response Procedures at https://www.purdue.edu/emergencypreparedness/flipchart/index.html. Please review the evacuation routes, exit points, emergency assembly area and shelter in place procedures and locations for the building. Video resources include a 20-minute active shooter awareness video that illustrates what to look for and how to prepare and react to this type of incident. See http://www.purdue.edu/securepurdue/police/video/.

D.15 Violent Behavior Policy

Purdue University is committed to providing a safe and secure campus environment for members of the university community. Purdue strives to create an educational environment for students and a work environment for employees that promote educational and career goals. Violent Behavior impedes such goals. Therefore, Violent Behavior is prohibited in or on any University Facility or while participating in any university activity.

D.16 Mental Health and Wellness

If you find yourself beginning to feel some stress, anxiety and/or feeling slightly overwhelmed, try WellTrack (https://purdue.welltrack.com). Sign in and find information and tools at your fingertips, available to you at any time.

If you need support and information about options and resources, please contact or see the Office of the Dean of Students (www.purdue.edu/odos). Call 765-494-1747. Hours of operation are M-F, 8 am-5 pm.

If you find yourself struggling to find a healthy balance between academics, social life, stress, etc., sign up for free one-on-one virtual or in-person sessions with a Purdue Wellness Coach at RecWell (https://www.purdue.edu/recwell/fitness-wellness/wellness/coaching/wellness-coaching.php). Student coaches can help you navigate through barriers and challenges toward your goals throughout the semester. Sign up is completely free and can be done on BoilerConnect. If you have any questions, please contact Purdue Wellness at evans240@purdue.edu.

If you’re struggling and need mental health services: Purdue University is committed to advancing the mental health and well-being of its students. If you or someone you know is feeling overwhelmed, depressed, and/or in need of
mental health support, services are available. For help, such individuals should contact Counseling and Psychological Services (CAPS) (https://www.purdue.edu/caps/) at 765-494-6995 during and after hours, on weekends and holidays, or by going to the CAPS office on the second floor of the Purdue University Student Health Center (PUSH) during business hours.

Purdue University is committed to advancing the mental health and well-being of its students. If you or someone you know is feeling overwhelmed, depressed, and/or in need of support, services are available. For help, such individuals should contact Counseling and Psychological Services (CAPS) at (765) 494-6995 and http://www.purdue.edu/caps/ during and after hours, on weekends and holidays, or through its counselors physically located in the Purdue University Student Health Center (PUSH) during business hours.

### D.17 Nondiscrimination

Purdue University is committed to maintaining a community which recognizes and values the inherent worth and dignity of every person; fosters tolerance, sensitivity, understanding, and mutual respect among its members; and encourages each individual to strive to reach his or her own potential. In pursuit of its goal of academic excellence, the University seeks to develop and nurture diversity. The University believes that diversity among its many members strengthens the institution, stimulates creativity, promotes the exchange of ideas, and enriches campus life. Purdue University prohibits discrimination against any member of the University community on the basis of race, religion, color, sex, age, national origin or ancestry, marital status, parental status, sexual orientation, disability, or status as a veteran. The University will conduct its programs, services and activities consistent with applicable federal, state and local laws, regulations and orders and in conformance with the procedures and limitations as set forth in Executive Memorandum No. D-1, which provides specific contractual rights and remedies.

### D.18 Privacy

The Federal Educational Records Privacy Act (FERPA) protects information about students, such as grades. If you apply for a job and wish to use the instructor as a reference, you should tell the instructor beforehand. Otherwise, the instructor cannot say anything about you to a prospective employer who might call. The instructor is happy to provide references and to write letters of recommendation for his students as needed.
D.19 Changes to the syllabus

This syllabus is subject to change and changes will be announced appropriately.