CS 583: Approximation Algorithms

Chandra Chekuri[1]

November 18, 2021

[1] Dept. of Computer Science, University of Illinois, Urbana, IL 61820. Email: chekuri@illinois.edu.

Contents

Chapter 1

Introduction

These are lecture notes for a course on approximation algorithms.

Course Objectives

1. To appreciate that not all intractable problems are the same. **NP** optimization problems, identical in terms of exact solvability, can appear very different from the approximation point of view. This sheds light on why, in practice, some optimization problems (such as KNAPSACK) are easy, while others (like CLIQUE) are extremely difficult.

2. To learn techniques for design and analysis of approximation algorithms, via some fundamental problems.

3. To build a toolkit of broadly applicable algorithms/heuristics that can be used to solve a variety of problems.

4. To understand reductions between optimization problems, and to develop the ability to relate new problems to known ones.

The complexity class **P** contains the set of problems that can be solved in polynomial time. From a theoretical viewpoint, this describes the class of tractable problems, that is, problems that can be solved efficiently. The class **NP** is the set of problems that can be solved in *non-deterministic* polynomial time, or equivalently, problems for which a solution can be *verified* in polynomial time. **NP** contains many interesting problems that often arise in practice, but there is good reason to believe **P** ≠ **NP**. That is, it is unlikely that there exist algorithms to solve **NP** optimization problems efficiently, and so we often resort to heuristic methods to solve these problems.

Heuristic approaches include backtrack search and its variants, mathematical programming methods, local seach, genetic algorithms, tabu search, simulated annealing etc. Some methods are guaranteed to find an optimal solution, though they may take exponential time; others are guaranteed to run in polynomial time, though they may not return a (optimal) solution. Approximation algorithms are (typically) polynomial time heuristics that do not always find an optimal solution but they are distinguished from general heuristics in providing guarantees on the *quality* of the solution they output.

Approximation Ratio: To give a guarantee on solution quality, one must first define what we mean by the quality of a solution. We discuss this more carefully later. For now, note that each *instance* of an optimization problem has a set of feasible solutions. The optimization problems we consider have an **objective function** which assigns a (real/rational) number/value to each feasible solution of each instance I. The goal is to find a feasible solution with minimum objective function value or maximum objective function value. The former problems are minimization problems and the latter are maximization problems.

For each instance I of a problem, let $\text{OPT}(I)$ denote the value of an optimal solution to instance I. We say that an algorithm $\mathcal{A}$ is an α-approximation algorithm for a problem if, for *every* instance I, the value of the feasible solution returned by $\mathcal{A}$ is within a (multiplicative) factor of α of $\text{OPT}(I)$. Equivalently, we say that $\mathcal{A}$ is an approximation algorithm with *approximation ratio α*. For a minimization problem we would have $\alpha \geq 1$ and for a maximization problem we would have $\alpha \leq 1$. However, it is not uncommon to find in the literature a different convention for maximization problems where one says that $\mathcal{A}$ is an α-approximation algorithm if the value of the feasible solution returned by $\mathcal{A}$ is at least $\frac{1}{\alpha} \cdot \text{OPT}(I)$; the reason for using convention is so that approximation ratios for both minimization and maximization problems will be ≥ 1. In this course we will for the most part use the convention that $\alpha \geq 1$ for minimization problems and $\alpha \leq 1$ for maximization problems.

Remarks:

1. The approximation ratio of an algorithm for a minimization problem is the *maximum* (or supremum), over all instances of the problem, of the ratio between the values of solution returned by the algorithm and the optimal solution. Thus, it is a bound on the *worst-case* performance of the algorithm.

2. The approximation ratio α can depend on the size of the instance I, so one should technically write $\alpha(|I|)$.

3. A natural question is whether the approximation ratio should be defined in an *additive* sense. For example, an algorithm has an α-approximation for a minimization problem if it outputs a feasible solution of value at most

OPT(I) + α for all I. This is a valid definition and is the more relevant one in some settings. However, for many **NP** problems it is easy to show that one cannot obtain any interesting additive approximation (unless of course $P = NP$) due to scaling issues. We will illustrate this via an example later.

Pros and cons of the approximation approach: Some advantages to the approximation approach include:

1. It explains why problems can vary considerably in difficulty.
2. The analysis of problems and problem instances distinguishes easy cases from difficult ones.
3. The worst-case ratio is *robust* in many ways. It allows *reductions* between problems.
4. Approximation allgorithmic ideas/tools/relaxations are valuable in developing heuristics, including many that are practical and effective.
5. Quantification of performance via a concrete metric such as the approximation ratio allows for innovation in algorithm design and has led to many new ideas.

As a bonus, many of the ideas are beautiful and sophisticated, and involve connections to other areas of mathematics and computer science.

Disadvantages include:

1. The focus on *worst-case measures* risks ignoring algorithms or heuristics that are practical or perform well *on average*.
2. Unlike, for example, integer programming, there is often no incremental/continuous tradeoff between the running time and quality of solution.
3. Approximation algorithms are often limited to cleanly stated problems.
4. The framework does not (at least directly) apply to decision problems or those that are inapproximable.

Approximation as a broad lens

The use of approximation algorithms is not restricted solely to **NP**-Hard optimization problems. In general, ideas from approximation can be used to solve many problems where finding an exact solution would require too much of any resource.

A resource we are often concerned with is *time*. Solving **NP**-Hard problems exactly would (to the best of our knowledge) require exponential time, and so we may want to use approximation algorithms. However, for large data sets, even polynomial running time is sometimes unacceptable. As an example, the best exact algorithm known for the Matching problem in general graphs requires $O(m\sqrt{n})$ time; on large graphs, this may be not be practical. In contrast, a simple greedy algorithm takes near-linear time and outputs a matching of cardinality at least 1/2 that of the maximum matching; moreover there have been randomized sub-linear time algorithms as well.

Another often limited resource is *space*. In the area of data streams/streaming algorithms, we are often only allowed to read the input in a single pass, and given a small amount of additional storage space. Consider a network switch that wishes to compute statistics about the packets that pass through it. It is easy to exactly compute the average packet length, but one cannot compute the median length exactly. Surprisingly, though, many statistics can be approximately computed.

Other resources include programmer time (as for the Matching problem, the exact algorithm may be significantly more complex than one that returns an approximate solution), or communication requirements (for instance, if the computation is occurring across multiple locations).

1.1 Formal Aspects

1.1.1 NP Optimization Problems

In this section, we cover some formal definitions related to approximation algorithms. We start from the definition of optimization problems. A problem is simply an infinite collection of *instances*. Let Π be an optimization problem. Π can be either a minimization or maximixation problem. Instances I of Π are a subset of Σ^* where Σ is a finite encoding alphabet. For each instance I there is a set of feasible solutions $\mathcal{S}(I)$. We restrict our attention to real/rational-valued optimization problems; in these problems each feasible solution $S \in \mathcal{S}(I)$ has a value $val(S, I)$. For a minimization problem Π the goal is, given I, find $\text{OPT}(I) = \min_{S \in \mathcal{S}(I)} val(S, I)$.

Now let us formally define NP optimization (NPO) which is the class of optimization problems corresponding to NP.

Definition 1.1. Π *is in NPO if*

- *Given $x \in \Sigma^*$, there is a polynomial-time algorithm that decide if x is a valid instance of Π. That is, we can efficiently check if the input string is well-formed. This is a basic requirement that is often not spelled out.*

- *For each I, and $S \in \mathcal{S}(I)$, $|S| \leq poly(|I|)$. That is, the solution are of size polynomial in the input size.*
- *There exists a poly-time decision procedure that for each I and $S \in \Sigma^*$, decides if $S \in \mathcal{S}(I)$. This is the key property of NP; we should be able to* verify *solutions efficiently.*
- *$val(I, S)$ is a polynomial-time computable function.*

We observe that for a minimization NPO problem Π, there is a associated natural decision problem $L(\Pi) = \{(I, B) : \mathrm{OPT}(I) \leq B\}$ which is the following: given instance I of Π and a number B, is the optimal value on I at most B? For maximization problem Π we reverse the inequality in the definition.

Lemma 1.1. *$L(\Pi)$ is in NP if Π is in NPO.*

1.1.2 Relative Approximation

When Π is a minimization problem, recall that we say an approximation algorithm $\mathcal{A}$ is said to have approximation ratio α iff

- $\mathcal{A}$ is a polynomial time algorithm
- for all instance I of Π, $\mathcal{A}$ produces a feasible solution $\mathcal{A}(I)$ s.t. $val(\mathcal{A}(I), I) \leq \alpha\ val\ (\mathrm{OPT}(I), I)$. (Note that $\alpha \geq 1$.)

Approximation algorithms for maximization problems are defined similarly. An approximation algorithm $\mathcal{A}$ is said to have approximation ratio α iff

- $\mathcal{A}$ is a polynomial time algorithm
- for all instance I of Π, $\mathcal{A}$ produces a feasible solution $\mathcal{A}(I)$ s.t. $val(\mathcal{A}(I), I) \geq \alpha\ val\ (\mathrm{OPT}(I), I)$. (Note that $\alpha \leq 1$.)

For maximization problems, it is also common to see use $1/\alpha$ (which must be ≥ 1) as approximation ratio.

1.1.3 Additive Approximation

Note that all the definitions above are about relative approximations; one could also define *additive* approximations. $\mathcal{A}$ is said to be an α-additive approximation algorithm, if for all I, $val(\mathcal{A}(I)) \leq \mathrm{OPT}(I) + \alpha$. Most NPO problems, however, do not allow any additive approximation ratio because $\mathrm{OPT}(I)$ has a scaling property.

To illustrate the scaling property, let us consider Metric-TSP. Given an instance I, let I_β denote the instance obtained by increasing all edge costs by a factor of β. It is easy to observe that for each $S \in \mathcal{S}(I) = \mathcal{S}(I_\beta)$, $val(S, I_\beta) = \beta val(S, I_\beta)$ and $\mathrm{OPT}(I_\beta) = \beta\,\mathrm{OPT}(I)$. Intuitively, scaling edge by a factor of β scales the value by the same factor β. Thus by choosing β sufficiently large, we can essentially make the additive approximation(or error) negligible.

Lemma 1.2. *Metric-TSP does not admit an α additive approximation algorithm for any polynomial-time computable α unless $P = NP$.*

Proof. For simplicity, suppose every edge has integer cost. For the sake of contradiction, suppose there exists an additive α approximation $\mathcal{A}$ for Metric-TSP. Given I, we run the algorithm on I_β and let S be the solution, where $\beta = 2\alpha$. We claim that S is the optimal solution for I. We have $val(S, I) = val(S, I_\beta)/\beta \leq \mathrm{OPT}(I_\beta)/\beta + \alpha/\beta = \mathrm{OPT}(I) + 1/2$, as $\mathcal{A}$ is α-additive approximation. Thus we conclude that $\mathrm{OPT}(I) = val(S, I)$, since $\mathrm{OPT}(I) \leq val(S, I)$, and $\mathrm{OPT}(I)$, $val(S, I)$ are integers. This is impossible unless $P = NP$. ■

Now let us consider two problems which allow additive approximations. In the Planar Graph Coloring, we are given a planar graph $G = (V, E)$. We are asked to color all vertices of the given graph G such that for any $vw \in E$, v and w have different colors. The goal is to minimize the number of different colors. It is known that to decide if a planar graph admits 3-coloring is NP-complete [143], while one can always color any planar graph G with using 4 colors (this is the famous 4-color theorem) [9, 149]. Further, one can efficiently check whether a graph is 2-colorable (that is, if it is bipartite). Thus, the following algorithm is a 1-additive approximation for Planar Graph Coloring: If the graph is bipartite, color it with 2 colors; otherwise, color with 4 colors.

As a second example, consider the Edge Coloring Problem, in which we are asked to color edges of a given graph G with the minimum number of different colors so that no two adjacent edges have different colors. By Vizing's theorem [154], we know that one can color edges with either $\Delta(G)$ or $\Delta(G) + 1$ different colors, where $\Delta(G)$ is the maximum degree of G. Since $\Delta(G)$ is a trivial lower bound on the minimum number, we can say that the Edge Coloring Problem allows a 1-additive approximation. Note that the problem of deciding whether a given graph can be edge colored with $\Delta(G)$ colors is NP-complete [85].

1.1.4 Hardness of Approximation

Now we move to hardness of approximation.

Definition 1.2 (Approximability Threshold). *Given a minimization optimization problem Π, it is said that Π has an approximation threshold $\alpha^*(\Pi)$, if for any $\epsilon > 0$, Π admits a $\alpha^*(\Pi) + \epsilon$ approximation but if it admits a $\alpha^*(\Pi) - \epsilon$ approximation then $P = NP$.*

If $\alpha^*(\Pi) = 1$, it implies that Π is solvable in polynomial time. Many NPO problems Π are known to have $\alpha^*(\Pi) > 1$ assuming that $P \neq NP$. We can say that approximation algorithms try to decrease the upper bound on $\alpha^*(\Pi)$, while hardness of approximation attempts to increase lower bounds on $\alpha^*(\Pi)$.

To prove hardness results on NPO problems in terms of approximation, there are largely two approaches; a direct way by reduction from NP-complete problems and an indirect way via gap reductions. Here let us take a quick look at an example using a reduction from an NP-complete problem.

In the (metric) k-center problem, we are given an undirected graph $G = (V, E)$ and an integer k. We are asked to choose a subset of k vertices from V called centers. The goal is to minimize the maximum distance to a center, i.e. $\min_{S \subseteq V, |S|=k} \max_{v \in V} \text{dist}_G(v, S)$, where $\text{dist}_G(v, S) = \min_{u \in S} \text{dist}_G(u, v)$.

The k-center problem has approximation threshold 2, since there are a few 2-approximation algorithms for k-center and there is no $2 - \epsilon$ approximation algorithm for any $\epsilon > 0$ unless $P = NP$. We can prove the inapproximability using a reduction from the decision version of Dominating Set: Given an undirected graph $G = (V, E)$ and an integer k, does G have a dominating set of size at most k? A set $S \subseteq V$ is said to be a dominating set in G if for all $v \in V$, $v \in S$ or v is adjacent to some u in S. Dominating Set is known to be NP-complete.

Theorem 1.3 ([88]). *Unless $P = NP$, there is no $2 - \epsilon$ approximation for k-center for any fixed $\epsilon > 0$.*

Proof. Let I be an instance of Dominating Set Problem consisting of graph $G = (V, E)$ and integer k. We create an instance I' of k-center while keeping graph G and k the same. If I has a dominating set of size k then $\text{OPT}(I') = 1$, since every vertex can be reachable from the Dominating Set by at most one hop. Otherwise, we claim that $\text{OPT}(I') \geq 2$. This is because if $\text{OPT}(I') < 2$, then every vertex must be within distance 1, which implies the k-center that witnesses $\text{OPT}(I')$ is a dominating set of I. Therefore, the $(2 - \epsilon)$ approximation for k-center can be used to solve the Dominating Set Problem. This is impossible, unless $P = NP$. ■

1.2 Designing Approximation Algorithms

How does one design and more importantly analyze the performance of approximation algorithms? This is a non-trivial task and the main goal of the course is to expose you to basic and advanced techniques as well as central problems. The purpose of this section is to give some high-level insights. We start with how we design polynomial-time algorithms. Note that approximation makes sense mainly in the setting where one can find a feasible solution relatively easily but finding an *optimum* solution is hard. In some cases finding a feasible solution itself may involve some non-trivial algorithm, in which case it is useful to properly understand the structural properties that guarantee feasibility, and then build upon it.

Some of the standard techniques we learn in basic and advanced undergraduate algorithms courses are recursion based methods such as divide and conquer, dynamic programming, greedy, local search, combinatorial optimization via duality, and reductions to existing problems. How do we adapt these to the approximation setting? Note that intractability implies that there are no efficient characterizations of the optimum solution value.

Greedy and related techniques are often fairly natural for many problems and simple heuristic algorithms often suggest themselves for many problems. (Note that the algorithms may depend on being able to solve some existing problem efficiently. Thus, knowing a good collection of general poly-time solvable problems is often important.) The main difficulty is in analyzing their performance. The key challenge here is to identify appropriate *lower bounds* on the optimal value (assuming that the problem is a minimization problem) or *upper bounds* on the optimal value (assuming that the problem is a maximization problem). These bounds allow one to compare the output of the algorithm and prove an approximation bound. In designing poly-time algorithms we often prove that greedy algorithms do not work. We typically do this via examples. This skill is also useful in proving that some candidate algorithm does *not* give a good approximation. Often the bad examples lead one to a new algorithm.

How does one come up with lower or upper bounds on the optimum value? This depends on the problem at hand and knowing some background and related problems. However, one would like to find some automatic ways of obtaining bounds. This is often provided via linear programming relaxations and more advanced convex programming methods including semi-definite programming, lift-and-project hierarchies etc. The basic idea is quite simple. Since integer linear programming is NP-Complete one can formulate most discrete optimization problems easily and "naturally" as an integer program. Note that there may be many different ways of expressing a given problem as an integer program. Of course we cannot solve the integer program but we can solve the linear-programming relaxation which is obtained by removing the integrality constraints on the variables. Thus, for each instance I of a given problem we can obtain an LP relaxation $LP(I)$ which we typically can be solve in polynomial-time. This automatically gives a bound on the optimum value since it is a relaxation. How good is this bound? It depends on the problem, of course, and also the specific LP relaxation. How do we obtain a feasible solution that is close to the bound given by the LP relaxation. The main technique here is to *round* the fractional solution x to an integer feasible solution x' such that x''s value is close to that of x. There are several non-trivial rounding techniques that have been developed over the years that we will explore in the course. We should note that in several cases one can analyze combinatorial algorithms via LP relaxations even though the LP relaxation does not play any direct role in the algorithm itself. Finally, there is the question of which LP relaxation to use. Often it is required to "strengthen" an LP relaxation via addition of constraints to provide better bounds. There are some automatic ways to strengthen any LP and often one also needs problem specific ideas.

Local search is another powerful technique and the analysis here is not obvious. One needs to relate the value of a local optimum to the value of a global optimum via various *exchange* properties which define the local search heuristic. For a formal analysis it is necessary to have a good understanding of the problem structure.

Finally, dynamic programming plays a key role in the following way. Its main use is in solving to optimality a *restricted* version of the given problem or a subroutine that is useful as a building block. How does one obtain a restricted version? This is often done by some clever proprocessing of a given instance.

Reductions play a very important role in both designing approximation algorithms and in proving inapproximability results. Often reductions serve as a starting point in developing a simple and crude heuristic that allows one to understand the structure of a problem which then can lead to further improvements.

Discrete optimization problems are brittle — changing the problem a little can lead to substantial changes in the complexity and approximability. Nevertheless it is useful to understand problems and their structure in broad categories so that existing results can be leveraged quickly and robustly. Thus, some of the emphasis in the course will be on classifying problems and how various parameters influence the approximability.

Chapter 2

Covering Problems

Part of these notes were scribed by Abul Hassan Samee and Lewis Tseng.

Packing and Covering problems together capture many important problems in combinatorial optimization. We will discuss several covering problems in this chapter. Two canonical one problems are Minimum Vertex Cover and its generalization Minimum Set Cover. (Typically we will omit the use of the qualifiers minimum and maximum since this is often clear from the definition of the problem and the context.) They play an important role in the study of approximation algorithms.

A *vertex cover* in an undirected graph $G = (V, E)$ is a set $S \subseteq V$ of vertices such that for each edge $e \in E$, at least one of its end points is in S. It is also called a node cover. In the Vertex Cover problem, our goal is to find a smallest vertex cover of G. In the *weighted* version of the problem, a weight function $w : V \rightarrow \mathcal{R}^+$ is given, and our goal is to find a minimum weight vertex cover of G. The unweighted version of the problem is also known as Cardinality Vertex Cover. Note that we are picking vertices to *cover* the edges. Vertex Cover is NP-Hard and is on the list of problems in Karp's list.

In the Set Cover problem the input is a set $\mathcal{U}$ of n elements, and a collection $\mathcal{S} = \{S_1, S_2, \ldots, S_m\}$ of m subsets of $\mathcal{U}$ such that $\bigcup_i S_i = \mathcal{U}$. Our goal in the Set Cover problem is to select as few subsets as possible from $\mathcal{S}$ such that their union covers $\mathcal{U}$. In the weighted version each set S_i has a non-negative weight w_i the goal is to find a set cover of minimim weight. Closely related to the Set Cover problem is the Maximum Coverage problem. In this problem the input is again $\mathcal{U}$ and $\mathcal{S}$ but we are also given an integer $k \leq m$. The goal is to select k subsets from $\mathcal{S}$ such that their union has the maximum cardinality. Note that Set Cover is a minimization problem while Maximum Coverage is a maximization problem. Set Cover is essentially equivalent to the Hitting Set problem. In Hitting Set the input is $\mathcal{U}$ and $\mathcal{S}$ but the goal is to pick the smallest number of elements of $\mathcal{U}$ that cover the given sets in $\mathcal{S}$. In other words we are seeking a set cover in the dual set system. It is easy to see Vertex Cover is a special case of Set Cover .

Set Cover is an important problem because in discrete optimization. In the standard definition the set system is given *explicitly*. In many applications the set system is *implicit*, and often exponential in the explicit part of the input; nevertheless such set systems are ubiquitious and one can often obtain exact or approximation algorithms. As an example consider the well known MST problem in graphs. One way to phrase MST is the following: given an edge-weighted graph $G = (V, E)$ find a minimum cost subset of the edges that *cover* all the cuts of G; by cover a cut $S \subseteq V$ we mean that at least one of the edges in $\delta(S)$ must be chosen. This may appear to be a strange way of looking at the MST problem but this view is useful as we will see later. Another implicit example is the following. Suppose we are given n rectangles in the plane and the goal is to choose a minimum number of points in the plane such that each input rectangle contains one of the chosen points. This is perhaps more natural to view as a special case of the Hitting Set problem. In principle the set of points that we can choose from is infinite but it can be seen that we can confine our attention to vertices in the arrangement of the given rectangles and it is easy to see that there are only $O(n^2)$ vertices — however, explicity computing them may be expensive and one may want to treat the problem as an implicit one for the sake of efficiency. want to think of

Covering problems have the feature that a superset of a feasible solution is also a feasible solution. More abstractly one can cast covering problems as the following. We are given a finite ground set V (vertices in a graph or sets in a set system) and a family of feasible solutions $\mathcal{I} \subseteq 2^V$ where $\mathcal{I}$ is upward closed; by this we mean that if $A \in \mathcal{I}$ and $A \subset B$ then $B \in \mathcal{I}$. The goal is to find the smallest cardinality set A in $\mathcal{I}$. In the weighted case V has weights and the goal is to find a minimum weight set in $\mathcal{I}$. In some case one can also consider more complex non-additive objectives that assign a cost $c(S)$ for each $S \in \mathcal{I}$.

2.1 Greedy for Set Cover and Maximum Coverage

In this section we consider the unweighted version of Set Cover.

2.1.1 Greedy Algorithm

A natural greedy approximation algorithm for these problems is easy to come up with.

```
Greedy Cover($\mathcal{U}, \mathcal{S}$)
1. repeat
    A. pick the set that covers the maximum number of uncovered elements
    B. mark elements in the chosen set as covered
2. until done
```

In case of Set Cover, the algorithm Greedy Cover is *done* when all the elements in set $\mathcal{U}$ have been covered. And in case of Maximum Coverage, the algorithm is *done* when exactly k subsets have been selected from $\mathcal{S}$.

We will prove the following theorem.

Theorem 2.1. Greedy Cover *is a* $1 - (1 - 1/k)^k \geq (1 - \frac{1}{e}) \simeq 0.632$ *approximation for* Maximum Coverage, *and a* $(\ln n + 1)$ *approximation for* Set Cover.

The following theorem due to Feige [56] implies that Greedy Cover is essentially the best possible in terms of the approximation ratio that it guarantees.

Theorem 2.2. *Unless* ***NP*** $\subseteq$ ***DTIME***$(n^{O(\log \log n)})$*, there is no* $(1 - o(1)) \ln n$ *approximation for* Set Cover. *Unless* ***P=NP****, for any fixed* $\epsilon > 0$*, there is no* $(1 - \frac{1}{e} - \epsilon)$ *approximation for* Maximum Coverage.

Recently the preceding theorem has been strengthened so that the hardness holds under the assumption that $NP \neq P$ [124].

2.1.2 Analysis of Greedy Cover

We proceed towards the proof of Theorem 10.3 by providing analysis of Greedy Cover separately for Set Cover and Maximum Coverage.

Analysis for Maximum Coverage

Let OPT denote the value of an optimal solution to the Maximum Coverage problem; this is the maximum number of elements that are covered by k sets in the given set system. Let x_i denote the number of *new* elements covered by the i-th set chosen by Greedy Cover. Also, let $y_i = \sum_{j=1}^{i} x_i$ be the total number of elements covered after i iterations, and $z_i = \text{OPT} - y_i$. Note that, according to our notation, $y_0 = 0$ and y_k is the number of elements chosen by Greedy Cover at the end of the algorithm, and $z_0 = \text{OPT}$. The key to the analysis is the following simple claim.

Claim 2.1.1. *For* $0 \le i < k$, $x_{i+1} \ge \frac{z_i}{k}$.

Proof. Let $F^* \subseteq \mathcal{U}$ be the elements covered by some fixed optimum solution; we have $|F^*| = \mathrm{OPT}$. Consider iteration $i+1$. Greedy Cover selects the subset S_j whose inclusion covers the maximum number of uncovered elements. Since z_i is the total number of elements covered upto iteration i, at least $\mathrm{OPT} - z_i$ elements from F^* are uncovered. Let the set of uncovered elements from F^* at the end of iteration i be F^*_i. Since k sets together cover F^*, and hence F_i* as well, there must be some set in that collection of k sets that covers at least $|F^*_i|/k$ elements. This is a candidate set that can be chosen in iteration $i+1$. Since the algorithm picks the set that covers the maximum number of uncovered elements, the chosen set in iteration $i+1$ covers at least $|F^*_i|/k = \frac{z_i}{k}$ uncovered elements. Hence, $x_{i+1} \ge \frac{z_i}{k}$. ■

Remark 2.1. It is tempting to make a stronger claim that $x_{i+1} \ge \frac{z_i}{k-i}$. This is however false, and it is worthwhile to come up with an example.

By definition we have $y_k = x_1 + x_2 + \ldots + x_k$ is the total number of elements covered by Greedy Cover. To analyze the worst-case we want to make this sum as small as possible given the preceding claim. Heuristically (which one can formalize), one can argue that choosing $x_{i+1} = z_i/k$ minimizes the sum. Using this one can argue that the sum is at least $(1 - (1 - 1/k)^k)\,\mathrm{OPT}$. We give a formal argument now.

Claim 2.1.2. *For* $i \ge 0$, $z_i \le (1 - \frac{1}{k})^i \cdot \mathrm{OPT}$.

Proof. By induction on i. The claim is trivially true for $i = 0$ since $z_0 = \mathrm{OPT}$. We assume inductively that $z_i \le (1 - \frac{1}{k})^i \cdot OPT$. Then

$$\begin{aligned} z_{i+1} &= z_i - x_{i+1} \\ &\le z_i(1 - \frac{1}{k}) \quad \text{[using Claim 2.1.1]} \\ &\le (1 - \frac{1}{k})^{i+1} \cdot \mathrm{OPT}\,. \end{aligned}$$

■

The preceding claims yield the following lemma for algorithm Greedy Cover when applied on Maximum Coverage.

Lemma 2.1. Greedy Cover *is a* $1 - (1 - 1/k)^k \ge 1 - \frac{1}{e}$ *approximation for* Maximum Coverage.

Proof. It follows from Claim 2.1.2 that $z_k \le (1 - \frac{1}{k})^k \cdot \mathrm{OPT} \le \frac{\mathrm{OPT}}{e}$. Hence, $y_k = \mathrm{OPT} - z_k \ge (1 - \frac{1}{e}) \cdot \mathrm{OPT}$. ■

We note that $(1 - 1/e) \simeq 0.632$.

Analysis for Set Cover

Let k^* denote the value of an optimal solution to the Set Cover problem. Then an optimal solution value to the Maximum Coverage problem with the same system and $k = k^*$ would by $n = |\mathcal{U}|$ since it is possible to cover all the n elements in set $\mathcal{U}$ with k^* sets. From our previous analysis, $z_{k^*} \leq \frac{n}{e}$. Therefore, at most $\frac{n}{e}$ elements would remain uncovered after the first k^* steps of Greedy Cover. Similarly, after $2 \cdot k^*$ steps of Greedy Cover, at most $\frac{n}{e^2}$ elements would remain uncovered. This easy intuition convinces us that Greedy Cover is a $(\ln n + 1)$ approximation for the Set Cover problem. A more formal proof is given below.

Lemma 2.2. Greedy Cover *is a* $(\ln n + 1)$ *approximation for* Set Cover.

Proof. Since $z_i \leq (1 - \frac{1}{k^*})^i \cdot n$, after $t = \lceil k^* \ln \frac{n}{k^*} \rceil$ steps,

$$z_t \leq n(1 - 1/k^*)^{k^* \ln \frac{n}{k^*}} \leq n e^{-\ln \frac{n}{k^*}} \leq k^*.$$

Thus, after t steps, at most k^* elements are left to be covered. Since Greedy Cover picks at least one element in each step, it covers all the elements after picking at most $\lceil k^* \ln \frac{n}{k^*} \rceil + k^* \leq k^*(\ln n + 1)$ sets. ■

A useful special case of Set Cover is when all sets are "small". Does the approximation bound for Greedy improve? We can prove the following corollary via Lemma 2.2.

Corollary 2.3. *If each set in the set system has at most d elements, then* Greedy Cover *is a* $(\ln d + 1)$ *approximation for* Set Cover.

Proof. If each set has at most d elements then we have that $k^* \geq \frac{n}{d}$ and hence $\ln \frac{n}{k^*} \leq \ln d$. Then the claim follows from Lemma 2.2. ■

Theorem 10.3 follows directly from Lemma 2.1 and 2.2.

A near-tight example for Greedy Cover **when applied on** Set Cover **:** Let us consider a set $\mathcal{U}$ of n elements along with a collection $\mathcal{S}$ of $k + 2$ subsets $\{R_1, R_2, C_1, C_2, \ldots, C_k\}$ of $\mathcal{U}$. Let us also assume that $|C_i| = 2^i$ and $|R_1 \cap C_i| = |R_2 \cap C_i| = 2^{i-1}$ $(1 \leq i \leq k)$, as illustrated in Fig. 2.1.

Clearly, the optimal solution consists of only two sets, i.e., R_1 and R_2. Hence, OPT = 2. However, Greedy Cover will pick each of the remaining k sets, namely $C_k, C_{k-1}, \ldots, C_1$. Since $n = 2 \cdot \sum_{i=0}^{k-1} 2^i = 2 \cdot (2^k - 1)$, we get $k = \Omega(\log n)$. One can construct tighter examples with more involved analysis.

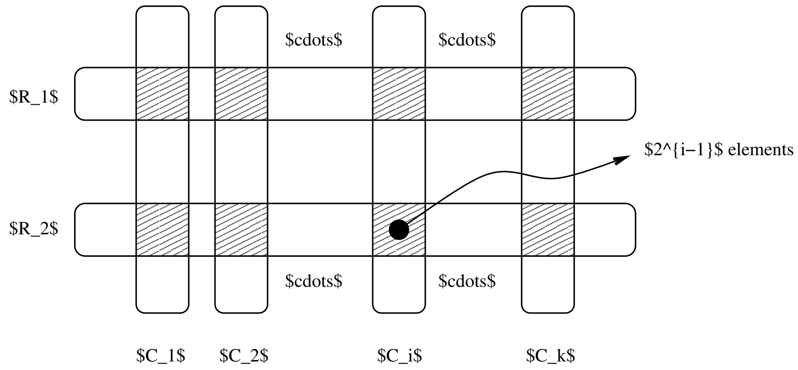


Figure 2.1: A near-tight example for Greedy Cover when applied on Set Cover

Exercise 2.1. Consider the weighted version of the Set Cover problem where a weight function $w : \mathcal{S} \rightarrow \mathcal{R}^+$ is given, and we want to select a collection $\mathcal{S}'$ of subsets from $\mathcal{S}$ such that $\cup_{X \in \mathcal{S}'} X = \mathcal{U}$, and $\sum_{X \in \mathcal{S}'} w(X)$ is the minimum. One can generalize the greedy heuristic in the natural fashion where in each iteration the algorithm picks the set that maximizes the ratio of the number of elements to its weight. Adapt the unweighted analysis to prove that the greedy algorithm yields an $O(\ln n)$ approximation for the weighted version (you can be sloppy with the constant in front of $\ln n$).

2.1.3 Dominating Set

A *dominating set* in a graph $G = (V, E)$ is a set $S \subseteq V$ such that for each $u \in V$, either $u \in S$, or some neighbor v of u is in S. In other words S covers/dominates all the nodes in V. In the Dominating Set problem, the input is a graph G and the goal is to find a smallest sized dominating set in G.

Exercise 2.2. 1. Show that Dominating Set is a special case of Set Cover.

2. What is the greedy heuristi when applied to Dominating Set. Prove that it yields an $(\ln(\Delta + 1) + 1)$ approximation where Δ is the maximum degree in the graph.

3. Show that Set Cover can be reduced in an approximation preserving fashion to Dominating Set. More formally, show that if Dominating Set has an $\alpha(n)$-approximation where n is the number of vertices in the given instance then Set Cover has an $(1 - o(1))\alpha(n)$-approximation.

2.2 Vertex Cover

We have already seen that the Vertex Cover problem is a special case of the Set Cover problem. The Greedy algorithm when specialized to Vertex Cover picks a highest degree vertex, removes it and the covered edges from the graph, and recurses in the remaining graph. It follows that the Greedy algorithm gives an $O(\ln \Delta + 1)$ approximation for the unweighted versions of the Vertex Cover problem. One can wonder wheter the Greedy algorith has a better worst-case for Vertex Cover than the analysis suggests. Unfortunately the answer is negative and there are examples where the algorithm outputs a solution with $\Omega(\log n \cdot \text{OPT})$ vertices.

We sketch the construction. Consider a bipartite graph $G = (U, V, E)$ where $U = \{u_1, u_2, \ldots, u_h\}$. V is partitioned into $S_1, S_2, \ldots, S_h$ where S_i has $\lfloor h/i \rfloor$ vertices. Each vertex v in S_i is connected to exactly *i distinct vertices* of U; thus, any vertex u_j is incident to at most one edge from S_i. It can be seen that the degree of each vertex $u_j \in U$ is roughly h. Clearly U is a vertex cover of G since the graph is bipartite. Convince yourself that the Greedy algorithm will pick all of V starting with the lone vertex in S_h (one may need to break ties to make this happen but the example can be easily perturbed to make this unnecessary). We have $n = \Theta(h \log h)$ and $OPT \leq h$ and Greedy outputs a solution of size $\Omega(h \log h)$.

2.2.1 A 2-approximation for Vertex Cover

There is a very simple 2-approximation algorithm for the Cardinality Vertex Cover problem.

```
Matching-VC($G$)

1. $S \leftarrow \emptyset$
2. Compute a maximal matching $M$ in $G$
3. for each edge $(u, v) \in M$ do
   A. add both $u$ and $v$ to $S$
4. Output $S$
```

Theorem 2.4. *Matching-VC is a 2-approximation algorithm.*

The proof of Theorem 2.4 follows from two simple claims.

Claim 2.2.1. *Let* OPT *be the size of the vertex cover in an optimal solution. Then* OPT $\geq |M|$.

Proof. Any vertex cover must contain at least one end point of each edge in M since no two edges in M intersect. Hence OPT $\geq |M|$. ∎

Claim 2.2.2. *Let $S(M) = \{u, v | (u, v) \in M\}$. Then $S(M)$ is a vertex cover.*

Proof. If $S(M)$ is not a vertex cover, then there must be an edge $e \in E$ such that neither of its endpoints are in M. But then e can be included in M, which contradicts the maximality of M. ∎

We now finish the proof of Theorem 2.4. Since $S(M)$ is a vertex cover, Claim 2.2.1 implies that $|S(M)| = 2 \cdot |M| \leq 2 \cdot \text{OPT}$.

Weighted Vertex Cover: The matching based heuristic does not generalize in a straight forward fashion to the weighted case but 2-approximation algorithms for the Weighted Vertex Cover problem can be designed based on LP rounding.

2.2.2 Set Cover with small frequencies

Vertex Cover is an instance of Set Cover where each element in $\mathcal{U}$ is in at most two sets (in fact, each element was in exactly two sets). This special case of the Set Cover problem admits a 2-approximation algorithm. What would be the case if every element is contained in at most three sets? More generally, given an instance of Set Cover, for each $e \in \mathcal{U}$, let $f(e)$ denote the number of sets containing e. Let $f = \max_e f(e)$, which we call the *maximum frequency*.

Exercise 2.3. Give an f-approximation for Set Cover, where f is the maximum frequency of an element. *Hint:* Follow the approach used for Vertex Cover .

2.3 Vertex Cover via LP

Let $G = (V, E)$ be an undirected graph with arc weights $w : V \rightarrow R^+$. We can formulate Vertex Cover as an integer linear programming problem as follows. For each vertex v we have a variable x_v. We interpret the variable as follows: if $x_v = 1$ if v is chosen to be included in a vertex cover, otherwise $x_v = 0$. With this interprtation we can easily see that the minimum weight vertex cover can be formulated as the following integer linear program.

$$\begin{array}{rcll} \min & \sum_{v \in V} w_v x_v & & \\ \text{subject to} & & & \\ x_u + x_v & \geq & 1 & \forall e = (u,v) \in E \\ x_v & \in & \{0,1\} & \forall v \in V \end{array}$$

However, solving the preceding integer linear program is NP-Hard since it would solve Vertex Cover exactly. Therefore we use Linear Programming (LP) to approximate the optimal solution, OPT(I), for the integer program. First, we can relax the constraint $x_v \in \{0,1\}$ to $x_v \in [0,1]$. It can be further simplified to $x_v \geq 0, \forall v \in V$.

Thus, a linear programming formulation for Vertex Cover is:

$$\begin{array}{rcll} \min & \sum_{v \in V} w_v x_v & & \\ \text{subject to} & & & \\ x_u + x_v & \geq & 1 & \forall e = (u,v) \in E \\ x_v & \geq & 0 & \end{array}$$

We now use the following algorithm:

```
Vertex Cover via LP
1. Solve LP to obtain an optimal fractional solution $x^*$
2. Let $S = \{v \mid x_v^* \geq \frac{1}{2}\}$
3. Output $S$
```

Claim 2.3.1. *S is a vertex cover.*

Proof. Consider any edge, $e = (u,v)$. By feasibility of x^*, $x_u^* + x_v^* \geq 1$, and thus $x_u^* \geq \frac{1}{2}$ or $x_v^* \geq \frac{1}{2}$. Therefore, at least one of u and v will be in S. ■

Claim 2.3.2. *$w(S) \leq 2\,\mathrm{OPT}_{LP}(I)$.*

Proof. $\mathrm{OPT}_{LP}(I) = \sum_v w_v x_v^* \geq \frac{1}{2} \sum_{v \in S} w_v = \frac{1}{2} w(S)$. ■

Therefore, $\mathrm{OPT}_{LP}(I) \geq \frac{\mathrm{OPT}(I)}{2}$ for all instances I.

Remark 2.2. For minimization problems: $\mathrm{OPT}_{LP}(I) \leq \mathrm{OPT}(I)$, where $\mathrm{OPT}_{LP}(I)$ is the optimal solution found by LP; for maximization problems, $\mathrm{OPT}_{LP}(I) \geq \mathrm{OPT}(I)$.

Integrality Gap: We introduce the notion of *integrality gap* to show the best approximation guarantee we can obtain if we only use the LP values as a lower bound.

Definition 2.5. *For a minimization problem Π, the integrality gap for a linear programming relaxation/formulation LP for Π is* $\sup_{I \in \pi} \frac{\text{OPT}(I)}{\text{OPT}_{LP}(I)}$.

That is, the integrality gap is the worst case ratio, over all instances I of Π, of the integral optimal value and the fractional optimal value. Note that different linear programming formulations for the same problem may have different integrality gaps.

Claims 2.3.1 and 2.3.2 show that the integrality gap of the Vertex Cover LP formulation above is *at most* 2.

Question: Is this bound tight for the Vertex Cover LP?

Consider the following example: Take a complete graph, K_n, with n vertices, and each vertex has $w_v = 1$. It is clear that we have to choose $n - 1$ vertices to cover all the edges. Thus, $\text{OPT}(K_n) = n - 1$. However, $x_v = \frac{1}{2}$ for each v is a feasible solution to the LP, which has a total weight of $\frac{n}{2}$. So gap is $2 - \frac{1}{n}$, which tends to 2 as $n \longrightarrow \infty$. One can also prove that the integrality gap is essentially 2 even in a class of sparse graphs.

Exercise 2.4. The vertex cover problem can be solved optimally in polynomial time in bipartite graphs. In fact the LP is integral. Prove this via the maxflow-mincut theorem and the integrality of flows when capacities are integral.

Other Results on Vertex Cover

1. The current best approximation ratio for Vertex Cover is $2 - \Theta(\frac{1}{\sqrt{\log n}})$ [98].

2. It is known that unless $P = NP$ there is α-approximation for Vertex Cover for $\alpha < 1.36$ [50]. Under a stronger hypothesis called the Unique Games Conjecture it is known that there is no $2 - \epsilon$ approximation for any fixed $\epsilon > 0$ [103].

3. There is a polynomial time approximation scheme (PTAS), that is a $(1 + \epsilon)$-approximation for any fixed $\epsilon > 0$, for planar graphs. This follows from a general approach due to Baker [17]. The theorem extends to more general classes of graphs.

2.4 Set Cover via LP

The input to the Set Cover problem consists of a finite set $U = \{1, 2, \ldots, n\}$, and m subsets of U, $S_1, S_2, \ldots, S_m$. Each set S_j has a non-negative weigh w_j and the goal is to find the minimum weight collection of sets which cover all elements in U (in other words their union is U).

A linear programming relaxation for Set Cover is:

$$\begin{array}{rcll} \min & \sum_{j} w_j x_j & & \\ \text{subject to} & & & \\ \sum_{j: i \in S_j} x_j & \geq & 1 & \forall i \in \{1, 2, \ldots, n\} \\ x_j & \geq & 0 & 1 \leq j \leq m \end{array}$$

And its dual is:

$$\begin{array}{rcll} \max & \sum_{i=1}^{n} y_i & & \\ \text{subject to} & & & \\ \sum_{i \in S_j} y_i & \leq & w_j & \forall j \in \{1, 2, \ldots, m\} \\ y_i & \geq & 0 & \forall i \in 1, 2, \ldots, n \end{array}$$

We give several algorithms for Set Cover based on this primal/dual pair LPs.

2.4.1 Deterministic Rounding

```
SET COVER VIA LP

1. Solve LP to obtain an optimal solution $x^*$, which contains fractional
   numbers.
2. Let $P = \{i \mid x_i^* > 0\}$
3. Output $\{S_j \mid j \in P\}$
```

Note that the above algorithm, even when specialized to VERTEX COVER is different from the one we saw earlier. It includes all sets which have a strictly positive value in an *optimum* solution to the LP.

Let x^* be an optimal solution to the primal LP, y^* be an optimum solution to the dual, and let $P = \{j \mid x^*_j > 0\}$. First, note that by strong duality, $\sum_j w_j x^*_j = \sum_i y^*_i$. Second, by complementary slackness if $x^*_j > 0$ then the corresponding dual constraint is tight, that is $\sum_{i \in S_j} y^*_i = w_j$.

Claim 2.4.1. *The output of the algorithm is a feasible set cover for the given instance.*

Proof. Exercise. ■

Claim 2.4.2. $\sum_{j \in P} w_j \leq f \sum_j w_j x^*_j = \text{OPT}_{LP}$.

Proof.

$$\sum_{j \in P} w_j = \sum_{j : x^*_j > 0} w_j = \sum_{j : x^*_j > 0} \left(\sum_{i \in S_j} y^*_i \right) = \sum_i y^*_i \left(\sum_{j : i \in S_j, x^*_j > 0} 1 \right) \leq f \sum_i y^*_i = f\,\text{OPT}_{LP}(I).$$

. ■

Notice that the the second equality is due to complementary slackness conditions (if $x^*_j > 0$, the corresponding dual constraint is tight), the penultimate inequality uses the definition of f, and the last inequality follows from weak duality (a feasible solution for the dual problem is a lower bound on the optimal primal solution).

Therefore we have that the algorithm outputs a cover of weight at most $f\,\text{OPT}_{LP}$. We note that f can be as large as n in which case the bound given by the algorithm is quite weak. In fact, it is not hard to construct examples that demonstrate the tightness of the analysis.

Remark 2.3. The analysis cruically uses the fact that x^* is an optimal solution. On the other hand the algorithm for Vertex Cover is more robust and works with any feasible solution x. It is easy to generalize the earlier rounding for Vertex Cover to obtain an f-approximation. The point of the above rounding is to illustrate the utility of complementary slackness.

2.4.2 Randomized Rounding

Now we describe a different rounding that yields an approximation bound that does not depend on f.

```
Set Cover via Randomized Rounding

1. $A = \emptyset$, and let $x^*$ be an optimal solution to the LP
2. for $k = 1$ to $2 \ln n$ do
    A. pick each $S_j$ independently with probability $x^*_j$
    B. if $S_j$ is picked, $A = A \cup \{j\}$
3. Output the sets with indices in $A$
```

Claim 2.4.3. $\mathbf{P}[$*i is not covered in an iteration*$] = \prod_{j: i \in S_j} (1 - x^*_j) \leq \frac{1}{e}$.

Intuition: We know that $\sum_{j: i \in S_j} x^*_j \geq 1$. Subject to this constraint, if we want to minimize the probability that element i is covered, one can see that the minimum is achieved with $x^*_j = 1/\ell$ for each set S_j that covers i; here ℓ is the number of sets that cover i. Then the probability is $(1 - \frac{1}{\ell})^\ell$.

Proof. We use the inequality $(1 - x) \leq e^{-x}$ for all $x \in [0, 1]$.

$$\mathbf{P}[i \text{ is not covered in an iteration}] = \prod_{j: i \in S_j} (1 - x^*_j) \leq \prod_{j: i \in S_j} e^{-x^*_j} \leq e^{-\sum_{j: i \in S_j} x^*_j} \leq \frac{1}{e}.$$

■

We then obtain the following corollaries:

Corollary 2.6. $\mathbf{P}[$*i is not covered at the end of the algorithm*$] \leq e^{-2 \log n} \leq \frac{1}{n^2}$.

Corollary 2.7. $\mathbf{P}[$*all elements are covered, after the algorithm stops*$] \geq 1 - \frac{1}{n}$.

Proof. Via the union bound. The probability that i is not covered is at most $1/n^2$, hence the probability that there is some i that is not covered is at most $n \cdot 1/n^2 \leq 1/n$. ■

Now we bound the expected cost of the algorithm. Let C_t = cost of sets picked in iteration t, then $\mathbf{E}[[C_t] = \sum_{j=1}^m w_j x^*_j$, where $\mathbf{E}[X]$ denotes the expectation of a random variable X. Then, let $C = \sum_{t=1}^{2 \ln n} C_t$; we have $\mathbf{E}[C] = \sum_{t=1}^{2 \ln n} \mathbf{E}[C_t] \leq 2 \ln n \, \mathrm{OPT}_{LP}$. By Markov's inequality, $\mathbf{P}[C > 2\, \mathbf{E}[C]] \leq \frac{1}{2}$, hence $\mathbf{P}[C \leq 4 \ln n \, \mathrm{OPT}_{LP}] \geq \frac{1}{2}$. Therefore, $\mathbf{P}[C \leq 4 \ln n \, \mathrm{OPT}_{LP}$ and all items are covered$] \geq \frac{1}{2} - \frac{1}{n}$. Thus, the randomized rounding algorithm, with probability close to 1/2 succeeds in giving a feasible solution of cost $O(\log n)\,\mathrm{OPT}_{LP}$. Note that we can check whether the solution satisfies the desired properties (feasibility and cost) and repeat the algorithm if it does not.

1. We can check if solution after rounding satisfies the desired properties, such as all elements are covered, or cost at most $2c\log n\,\mathrm{OPT}_{LP}$. If not, repeat rounding. Expected number of iterations to succeed is a constant.

2. We can also use Chernoff bounds (large deviation bounds) to show that a single rounding succeeds with high probability (probability at least $1 - \frac{1}{poly(n)}$).

3. The algorithm can be *derandomized*. Derandomization is a technique of removing randomness or using as little randomness as possible. There are many derandomization techniques, such as the method of conditional expectation, discrepancy theory, and expander graphs.

4. After a few rounds, select the cheapest set that covers each uncovered element. This has low expected cost. This algorithm ensures feasibility but guarantees cost only in the expected sense. We will see a variant on the homework.

Randomized Rounding with Alteration: In the preceding analysis we had to worry about the probability of covering all the elements and the expected cost of the solution. Here we illustrate a simple yet powerful technique of *alteration* in randomized algorithms and analysis. Let d be the maximum set size.

```
Set Cover: Randomized Rounding with Alteration

1. $A = \emptyset$, and let $x^*$ be an optimal solution to the LP
2. Add to $A$ each $S_j$ independently with probability $\min\{1, \ln d \cdot x_j^*\}$
3. Let $\mathcal{U}'$ be the elements uncovered by the chosen sets in $A$
4. For each uncovered element $i \in \mathcal{U}'$ do
   A. Add to $A$ the cheapest set that covers $i$
5. Output the sets with indices in $A$
```

The algorithm has two phases. A randomized phase and a fixing/altering phase. In the second phase we apply a naive algorithm that may have a high cost in the worst case but we will bound its expected cost appropriately. The algorithm deterministically guarantees that all elements will be covered, and hence we only need to focus on the expected cost of the chosen sets. Let C_1 be the random cost of the sets chosen in the first phase and let C_2 be the random cost of the sets chosen in the second phase. It is easy to see that $\mathbf{E}[C_1] = \ln d \sum_j w_j x_j^* = \ln d \, \mathrm{OPT}_{LP}$. Let $\mathcal{E}_i$ be the event that element i is not covered after the first randomized phase.

Exercise 2.5. $\mathbf{P}[\mathcal{E}_i] \le e^{-\ln d} \le 1/d$.

The worst case second phase cost can be upper bounded via the next lemma.

Lemma 2.3. *Let β_i be the cost of the cheapest set covering i. Then $\sum_i \beta_i \le d \, \mathrm{OPT}_{LP}$.*

Proof. Consider an element i. We have the constraint that $\sum_{j:i\in S_j} x_j^* \ge 1$. Since each set covering i has cost at least β_i, we have $\sum_{j:i\in S_j} c_j x_j^* \ge \beta_i \sum_{j:i\in S_j} x_j^* \ge \beta_i$. Thus,

$$\sum_i \beta_i \le \sum_i \sum_{j:i\in S_j} c_j x_j^* \le \sum_j c_j x_j^* |S_j| \le d \sum_j c_j x_j^* = d \, \mathrm{OPT}_{LP} .$$

■

Now we bound the *expected* second phase cost.

Lemma 2.4. $\mathbf{E}[C_2] \le \mathrm{OPT}_{LP}$.

Proof. We pay for a set to cover element i in the second phase only if it is not covered in the first phase. Hence $C_2 = \sum_i \mathcal{E}_i \beta_i$. Note that the events $\mathcal{E}_i$ for different elements i are not necessarily independent, however, we can apply linearity of expectation.

$$\mathbf{E}[C_2] = \sum_i \mathbf{E}[\mathcal{E}_i]\beta_i = \sum_i \mathbf{P}[\mathcal{E}_i]\beta_i \le 1/d \sum_i \beta_i \le \mathrm{OPT}_{LP} .$$

■

Combining the expected costs of the two phases we obtain the following theorem.

Theorem 2.8. *Randomized rounding with alteration outputs a feasible solution of expected cost* $(1 + \ln d) \, \mathrm{OPT}_{LP}$.

Note that the simplicity of the algorithm and tightness of the bound.

Remark 2.4. If $d = 2$ the Set Cover problem becomes the Edge Cover problem in a graph which is the following. Given an edge-weighted graph $G = (V, E)$, find the minimum weight subset of edges such that each vertex is covered. Edge Cover admits a polynomial-time algorithm via a reduction to the minimum-cost matching problem in a general graph. However $d = 3$ for Set Cover is NP-Hard via a reduction from 3-D Matching.

2.4.3 Dual-fitting

In this section, we introduce the technique of dual-fitting for the analysis of approximation algorithms. At a high-level the approach is the following:

1. Consider an algorithm that one wants to analyze.
2. Construct a feasible solution to the dual LP based on the structure of the algorithm.
3. Show that the cost of the solution returned by the algorithm can be bounded in terms of the value of the dual solution.

Note that the algorithm itself need not be LP based. Here, we use Set Cover as an example. See the previous section for the primal and dual LP formulations for Set Cover .

We can interpret the dual as follows: Think of y_i as how much element i is willing to pay to be covered; the dual maximizes the total payment, subject to the constraint that for each set, the total payment of elements in that set is at most the cost of the set.

We rewrite the Greedy algorithm for Weighted Set Cover.

```
Greedy Set Cover

1. $Covered = \emptyset$
2. $A = \emptyset$;
3. While $Covered \neq U$ do
    A. $j \leftarrow \arg\min_k (\frac{w_k}{|S_k \cap \ Uncovered|})$;
    B. $Covered = Covered \cup S_j$;
    C. $A = A \cup \{j\}$.
4. end while;
5. Output sets in $A$ as cover
```

Let $H_k = 1 + 1/2 + \ldots + 1/k$ be the $k$the Harmonic number. It is well known that $H_k \leq 1 + \ln k$.

Theorem 2.9. Greedy Set Cover *picks a solution of cost* $\leq H_d \cdot \mathrm{OPT}_{LP}$, *where* d *is the maximum set size, i.e.,* $d = \max_j |S_j|$.

To prove this, we augment the algorithm to keep track of some additional information.

```
AUGMENTED GREEDY ALGORITHM OF WEIGHTED SET COVER
1. $Covered = \emptyset$
2. While $Covered \neq U$ do
    A. $j \leftarrow \arg\min_k(\frac{w_k}{|S_k \cap \ Uncovered|})$
    B. if $i$ is uncovered and $i \in S_j$, set $p_i = \frac{w_j}{|S_j \cap \ Uncovered|}$;
    C. $Covered = Covered \cup S_j$
    D. $A = A \cup \{j\}$.
3. Output sets in $A$ as cover
```

It is easy to see that the algorithm outputs a feasible cover.

Claim 2.4.4. $\sum_{j \in A} w_j = \sum_i p_i$.

Proof. Consider when j is added to A. Let $S'_j \subseteq S_j$ be the elements that are uncovered before j is added. For each $i \in S'_j$ the algorithm sets $p_i = w_j/|S'_j|$. Hence, $\sum_{i \in S'_j} p_i = w_j$. Moreover, it is easy to see that the sets S'_j, $j \in A$ are disjoint and together partition U. Therefore,

$$\sum_{j \in A} w_j = \sum_{j \in A} \sum_{i \in S'_j} p_i = \sum_{i \in U} p_i.$$

■

For each i, let $y'_i = \frac{1}{H_d} p_i$.

Claim 2.4.5. *y' is a feasible solution for the dual LP.*

Suppose the claim is true, then the cost of GREEDY SET COVER's solution = $\sum_i p_i = H_d \sum_i y'_i \leq H_d \, \mathrm{OPT}_{LP}$. The last step is because any feasible solution for the dual problem is a lower bound on the value of the primal LP (weak duality).

Now, we prove the claim. Let S_j be an arbitrary set, and let $|S_j| = t \leq d$. Let $S_j = \{i_1, i_2, ..., i_t\}$, where we the elements are ordered such that i_1 is covered by Greedy no-later than i_2, and i_2 is covered no later than i_3 and so on.

Claim 2.4.6. *For* $1 \leq h \leq t$, $p_{i_h} \leq \frac{w_j}{t-h+1}$.

Proof. Let $S_{j'}$ be the set that covers i_h in Greedy. When Greedy picked $S_{j'}$ the elements $i_h, i_{h+1}, \ldots, i_t$ from S_j were uncovered and hence Greedy could have picked S_j as well. This implies that the density of $S_{j'}$ when it was picked was no more than $\frac{w_j}{t-h+1}$. Therefore p_{i_h} which is set to the density of $S_{j'}$ is at most $\frac{w_j}{t-h+1}$. ■

From the above claim, we have

$$\sum_{1 \le h \le t} p_{i_h} \le \sum_{1 \le h \le t} \frac{w_j}{t-h+1} = w_j H_t \le w_j H_d.$$

Thus, the setting of y'_i to be p_i scaled down by a factor of H_d gives a feasible solution.

2.4.4 Greedy for implicit instances of Set Cover

Set Cover and the Greedy heuristic are quite useful in applications because many instances are *implicit*, nevertheless, the algorithm and the analysis applies. That is, the universe $\mathcal{U}$ of elements and the collection $\mathcal{S}$ of subsets of $\mathcal{U}$ need not be restricted to be finite or explicitly enumerated in the Set Cover problem. For instance, a problem could require covering a finite set of points in the plane using disks of unit radius. There is an infinite set of such disks, but the Greedy approximation algorithm can still be applied. For such implicit instances, the Greedy algorithm can be used if we have access to an *oracle*, which, at each iteration, selects a set having the optimal density. However, an oracle may not always be capable of selecting an optimal set. In some cases it may have to make the selections *approximately*. We call an oracle an *α-approximate oracle* for some $\alpha \ge 1$ if, at each iteration, it selects a set S such that $\frac{w(S)}{S} \le \alpha \min_{A \text{ in collection}} \frac{A}{w(A)}$.

Exercise 2.6. Prove that the approximation guarantee of Greedy with an α-approximate oracle would be $\alpha(\ln n + 1)$ for Set Cover, and $(1 - \frac{1}{e^{\alpha}})$ for Maximum Coverage.

We will see several examples of implicit use of the greedy analysis in the course.

2.5 Submodularity

Set Cover turns out to be a special case of a more general problem called Submodular Set Cover. The Greedy algorithm and analysis applies in this more generality. Submodularity is a fundamental notion with many applications in combinatorial optimization and else where. Here we take the opportunity to provide some definitions and a few results.

Definition 2.10. *Given a finite set E, a real-valued set function $f : 2^E \rightarrow \mathbb{R}$ is* submodular *iff*

$$f(A) + f(B) \geq f(A \cup B) + f(A \cap B) \quad \forall A, B \subseteq E.$$

Alternatively, f is a submodular function iff

$$f(A \cup \{i\}) - f(A) \geq f(B \cup \{i\}) - f(B) \quad \forall A \subset B, i \in E \setminus B.$$

The second characterization shows that submodularity is based on *decreasing marginal utility* property in the discrete setting. Adding element i to a set A will help at least as much as adding it to to a (larger) set $B \supset A$. It is common to use $A + i$ to denote $A \cup \{i\}$ and $A - i$ to denote $A \setminus \{i\}$.

Exercise 2.7. Prove that the two characterizations of submodular functions are equivalent.

Many application of submodular functions are when f is a non-negative function though there are several important applications when f can be negative. A submodular function $f(\cdot)$ is *monotone* if $f(A + i) \geq f(A)$ for all $i \in E$ and $A \subseteq E$. Typically one assumes that f is normalized by which we mean that $f(\emptyset) = 0$; this can always be done by shifting the function by $f(\emptyset)$. f is *symmetric* if $f(A) = f(E \setminus A)$ for all $A \subseteq E$. Submodular set functions arise in a large number of fields including combinatorial optimization, probability, and geometry. Examples include rank function of a matroid, the sizes of cutsets in a directed or undirected graph, the probability that a subset of events do not occur simultaneously, entropy of random variables, etc. In the following we show that the Set Cover and Maximum Coverage problems can be easily formulated in terms of submodular set functions.

Exercise 2.8. Let $\mathcal{U}$ be a set and let $\mathcal{S} = \{S_1, S_2, \ldots, S_m\}$ be a finite collection of subsets of $\mathcal{U}$. Let $N = \{1, 2, \ldots, m\}$, and define $f : 2^N \rightarrow \mathbb{R}$ as: $f(A) = |\cup_{i \in A} S_i|$ for $A \subseteq E$. Show that f is a monotone non-negative submodular set function.

Exercise 2.9. Let $G = (V, E)$ be a directed graph and let $f : 2^V \rightarrow \mathbb{R}$ where $f(S) = |\delta^+(S)$ is the number of arcs leaving S. Prove that f is submodular. Is the function monotone?

2.5.1 Submodular Set Cover

When formulated in terms of submodular set functions, the Set Cover problem is the following. Given a monotone submodular function f (whose value would be computed by an oracle) on $N = \{1, 2, \ldots, m\}$, find the smallest set $S \subseteq N$ such that $f(S) = f(N)$. Our previous greedy approximation can be applied to this formulation as follows.

```
GREEDY SUBMODULAR ($f, N$)
1. $S \leftarrow \emptyset$
2. While $f(S) \neq f(N)$ do
    A. find $i$ to maximize $f(S + i) - f(S)$
    B. $S \leftarrow S \cup \{i\}$
3. Output $S$
```

Not so easy exercise.

Exercise 2.10. 1. Prove that the greedy algorithm is a $1 + \ln(f(N))$ approximation for SUBMODULAR SET COVER?

2. Prove that the greedy algorithm is a $1 + \ln(\max_i f(i))$ approximation for SUBMODULAR SET COVER.

The above results were first obtained by Wolsey [158].

2.5.2 SUBMODULAR MAXIMUM COVERAGE

By formulating the MAXIMUM COVERAGE problem in terms of submodular functions, we seek to maximize $f(S)$ such that $|S| \leq k$. We can apply algorithm GREEDY SUBMODULAR for this problem by changing the condition in line 2 to be: **while** $|S| \leq k$.

Exercise 2.11. Prove that greedy gives a $(1 - 1/e)$-approximation for SUBMODULAR MAXIMUM COVERAGE problem when f is monotone and non-negative. *Hint:* Generalize the main claim that we used for MAXIMUM COVERAGE .

The above and many related results were shown in the influential papers of Fisher, Nemhauser and Wolsey [61, 128].

2.6 Covering Integer Programs (CIPs)

There are several extensions of SET COVER that are interesting and useful. SUBMODULAR SET COVER is a very general problem while there are intermediate problems of interest such as Set Mmulticover. We refer to the reader to the relevant chapters in the two reference books. Here we refer to a general problem called Covering Integer Programs (CIPs for short). The goal is to solve the following *integer* program where $A \in \mathbb{R}_{+}^{n \times m}$ is a *non-negative* matrix. We can assume without loss of generality that w and b are also non-negative.

$$\begin{array}{rcll} \min & \displaystyle\sum_{j=1}^{n} w_j x_j & & \\ \text{subject to} & & & \\ Ax & \geq & b & \\ x_j & \leq & d_j & 1 \leq j \leq m \\ x_j & \geq & 0 & 1 \leq j \leq m \\ x_j & \in & \mathbb{Z} & 1 \leq j \leq m \end{array}$$

$Ax \geq b$ model covering constraints and $x_j \leq d_j$ models multiplicity constraints. Note that Set Cover is a special case where A is simply the incidence matrix of the sets and elements (the columns correspond to sets and the rows to elements) and $d_j = 1$ for all j. What are CIPs modeling? It is a generalization of Set Cover . To see this, assume, without loss of generality, that A, b are integer matrices. For each element corresponding to row i the quantity b_i corresponds to the requirement of how many times i needs to be covered. A_{ij} corresponds to the number of times set S_j covers element i. d_j is an upper bound on the number of copies of set S_j that are allowed to be picked.

Exercise 2.12. Prove that CIPs are a special case of Submodular Set Cover.

One can apply the Greedy algorithm to the above problem and the standard analysis shows that the approximation ratio obtained is $O(\log B)$ where $B = \sum_i b_i$ (assuming that they are integers). Even though this is reasonable we would prefer a strongly polynomial bound. In fact there are instances where B is exponential in n and the worst-case approximation ratio can be poor. The natural LP relaxation of the above integer program has a large integrality gap in constrat to the case of Set Cover . One needs to *strengthen* the LP relaxation via what are known as *knapsack cover inequalities*. We refer the reader to the paper of Kolliopoulos and Young [106] and recent one by Chekuri and Quanrud [40] for more on this problem.

Chapter 3

Knapsack

In this lecture we explore the Knapsack problem. This problem provides a good basis for learning some important procedures used for approximation algorithms that give better solutions at the cost of higher running time.

3.1 The Knapsack Problem

In the Knapsack problem we are given a number (knapsack capacity) $B \geq 0$, and a set N of n items; each item i has a given size $s_i \geq 0$ and a profit $p_i \geq 0$. We will assume that all the input numbers are integers (or more generally rationals). Given a subset of the items $A \subseteq N$, we define two functions, $s(A) = \sum_{i \in A} s_i$ and $p(A) = \sum_{i \in A} p_i$, representing the total size and profit of the group of items respectively. The goal is to choose a subset of the items, A, such that $s(A) \leq B$ and $p(A)$ is maximized. We will assume, without loss of generality, that $s_i \leq B$ for all i; we can discard items that do not satisfy this constraint.

It is not difficult to see that if all the profits are identical (say 1), the natural greedy algorithm that inserts items in the order of non-increasing sizes yields. Assuming the profits and sizes are integral, we can still find an optimal solution to the problem using dynamic programming in either $O(nB)$ or $O(nP)$ time, where $P = \sum_{i=1}^{n} p_i$. These are standard exercises. While these algorithms appear to run in polynomial time, it should be noted that B and P can be exponential in the size of the input written in binary. We call such algorithms *pseudo-polynomial time algorithms* as their running times are polynomial when numbers in the input are given in unary. Knapsack is a classical NP-Hard problem and these results (and the proof of NP-Hardness) show that the hardness manifests itself when the numbers are large (exponential in n which means that the number of bits in the size or profit are polynomial in n).

3.1.1 A Greedy Algorithm

Consider the following greedy algorithm for the Knapsack problem which we will refer to as GreedyKnapsack. We sort all the items by the ratio of their profits to their sizes so that $\frac{p_1}{s_1} \geq \frac{p_2}{s_2} \geq \cdots \geq \frac{p_n}{s_n}$. Afterward, we greedily take items in this order as long as adding an item to our collection does not exceed the capacity of the knapsack. It turns out that this algorithm can be arbitrarily bad. Suppose we only have two items in N. Let $s_1 = 1$, $p_1 = 2$, $s_2 = B$, and $p_2 = B$. GreedyKnapsack will take only item 1, but taking only item 2 would be a better solution and the ratio of the profits in the two cases is $2/B$ which can be made arbitrarily small. As it turns out, we can easily modify this algorithm to provide a 2-approximation by simply taking the best of GreedyKnapsack's solution or the most profitable item. We will call this new algorithm ModifiedGreedy.

Theorem 3.1. ModifiedGreedy *has an approximation ratio of* $1/2$ *for the* Knapsack *problem.*

Proof. Let k be the index of the first item that is not accepted by GreedyKnapsack. Consider the following claim:

Claim 3.1.1. $p_1 + p_2 + \ldots p_k \geq \text{OPT}$. *In fact,* $p_1 + p_2 + \cdots + \alpha p_k \geq \text{OPT}$ *where* $\alpha = \frac{B-(s_1+s_2+\cdots+s_{k-1})}{s_k}$ *is the fraction of item* k *that can still fit in the knapsack after packing the first* $k-1$ *items.*

The proof of Theorem 3.1 follows immediately from the claim. In particular, either $p_1 + p_2 + \cdots + p_{k-1}$ or p_k must be at least OPT/2. We now only have to prove Claim 3.1.1. We give an LP relaxation of the Knapsack problem as follows: Here, $x_i \in [0,1]$ denotes the fraction of item i packed in the knapsack.

$$\begin{aligned} \text{maximize} \quad & \sum_{i=1}^{n} p_i x_i \\ \text{subject to} \quad & \sum_{i=1}^{n} s_i x_i \leq B \\ & x_i \leq 1 \text{ for all } i \text{ in } \{1 \ldots n\} \\ & x_i \geq 0 \text{ for all } i \text{ in } \{1 \ldots n\} \end{aligned}$$

Let OPT_{LP} be the optimal value of the objective function in this linear programming instance. Any solution to Knapsack is a feasible solution to the LP and both problems share the same objective function, so $\text{OPT}_{LP} \geq \text{OPT}$. Now set $x_1 = x_2 = \cdots = x_{k-1} = 1$, $x_k = \alpha$, and $x_i = 0$ for all $i > k$. This is a feasible solution to the LP. We leave it as an exercise to the reader to argue that is an optimum solution. Therefore, $p_1 + p_2 + \cdots + \alpha p_k = \text{OPT}' \geq \text{OPT}$. The first statement of the lemma follows from the second as $\alpha \leq 1$. ■

3.1.2 A Polynomial Time Approximation Scheme

Using the results from the last section, we make a few simple observations. Some of these lead to a better approximation.

Observation 3.2. *If for all i, $p_i \leq \epsilon \, \mathrm{OPT}$,* GreedyKnapsack *gives a $(1-\epsilon)$ approximation.*

Proof. Follows easily from Claim 3.1.1. ■

Observation 3.3. *There are at most $\lceil \frac{1}{\epsilon} \rceil$ items with profit at least $\epsilon \, \mathrm{OPT}$ in any optimal solution.*

The next claim is perhaps more interesting and captures the intuition that the bad case for greedy happens only when there are "big" items.

Claim 3.1.2. *If for all i, $s_i \leq \epsilon B$,* GreedyKnapsack *gives a $(1-\epsilon)$ approximation.*

Proof. We give a proof sketch via the LP relaxation. Recall that k is the first item that did not fit in the knapsack. We make the following observation. Recall that OPT_{LP} is the optimum value of LP relaxation. Suppose we reduce the knapsack capacity to $B' = s_1 + s_2 + \ldots s_{k-1}$ while keeping all the items the same. Let OPT'_{LP} be the value for the new size. We claim that $\mathrm{OPT}'_{LP} \geq \frac{B'}{B} \mathrm{OPT}_{LP}$ — this is because we can take any feasible solution to the original LP and scale each variable by B'/B to obtain a feasible solution with the new capacity. What is OPT'_{LP}? We note that Greedy will fill B' to capacity with the first $k-1$ items and hence, $\mathrm{OPT}'_{LP} = p_1 + \ldots + p_{k-1}$. Combining, we obtain that

$$p_1 + \ldots + p_{k-1} \geq \frac{B'}{B} \mathrm{OPT}_{LP} \geq \frac{B'}{B} \mathrm{OPT}\,.$$

We note that $B' + s_k \geq B$ since item k did not fit, and hence $B' \geq B - s_k \geq B - \epsilon B \geq (1-\epsilon)B$. Therefore $B'/B \geq (1-\epsilon)$ and this finishes the proof. ■

We may now describe the following algorithm. Let $\epsilon \in (0,1)$ be a fixed constant and let $h = \lceil \frac{1}{\epsilon} \rceil$. We will try to guess the h most profitable items in an optimal solution and pack the rest greedily.

```
GUESS H + GREEDY($N$, $B$)
1. For each $S \subseteq N$ such that $|S| \leq h$ and $s(S) \leq B$ do
    A. Pack $S$ in knapsack of size at most $B$
    B. Let $i$ be the least profitable item in $S$. Remove all items $j \in N - S$
       where $p_j > p_i$.
    C. Run GREEDYKNAPSACK on remaining items with remaining capacity
       $B - \sum_{i \in S} s_i$
2. Output best solution from above
```

Theorem 3.4. *Guess h + Greedy gives a* $(1-\epsilon)$ *approximation and runs in* $O(n^{\lceil 1/\epsilon \rceil + 1})$ *time.*

Proof. For the running time, observe that there are $O(n^h)$ subsets of N. For each subset, we spend linear time greedily packing the remaining items. The time initially spent sorting the items can be ignored thanks to the rest of the running time.

For the approximation ratio, consider a run of the loop where S actually is the h most profitable items in an optimal solution and the algorithm's greedy stage packs the set of items $A' \subseteq (N - S)$. Let OPT′ be the optimal way to pack the smaller items in $N - S$ so that $\text{OPT} = p(S) + \text{OPT}'$. Let item k be the first item rejected by the greedy packing of $N - S$. We know $p_k \leq \epsilon \, \text{OPT}$ so by Claim 3.1.1 $p(A') \geq \text{OPT}' - \epsilon \, \text{OPT}$. This means the total profit found in that run of the loop is $p(S) + p(A') \geq (1 - \epsilon) \, \text{OPT}$. ■

Note that for any fixed choice of $\epsilon > 0$, the preceding algorithm runs in polynomial time. This type of algorithm is known as a *polynomial time approximation scheme* or PTAS. The term "scheme" refers to the fact that the algorithm varies with ϵ. We say a maximization problem Π has a PTAS if for all $\epsilon > 0$, there exists a polynomial time algorithm that gives a $(1 - \epsilon)$ approximation ($(1 + \epsilon)$ for minimization problems). In general, one can often find a PTAS for a problem by greedily filling in a solution after first searching for a good basis on which to work. As described below, KNAPSACK actually has something stronger known as a *fully polynomial time approximation scheme* or FPTAS. A maximization problem Π has a FPTAS if for all $\epsilon > 0$, there exists an algorithm that gives a $(1 - \epsilon)$ approximation ($(1 + \epsilon)$ for minimization problems) and runs in time polynomial in both the input size and $1/\epsilon$.

3.1.3 Rounding and Scaling

Earlier we mentioned exact algorithms based on dynamic programming that run in $O(nB)$ and $O(nP)$ time but noted that B and P may be prohibitively large. If we could somehow decrease one of those to be polynomial in n without losing too much information, we might be able to find an approximation based on one of these algorithms. Let $p_{\max} = \max_i p_i$ and note the following.

Observation 3.5. $p_{\max} \leq \text{OPT} \leq np_{\max}$

Now, fix some $\epsilon \in (0, 1)$. We want to scale the profits and round them to be integers so we may use the $O(nP)$ algorithm efficiently while still keeping enough information in the numbers to allow for an accurate approximation. For each i, let $p'_i = \lfloor \frac{n}{\epsilon} \frac{1}{p_{\max}} p_i \rfloor$. Observe that $p'_i \leq \frac{n}{\epsilon}$ so now the sum of the profits P' is at most $\frac{n^2}{\epsilon}$. Also, note that we lost at most n profit from the scaled optimal solution during the rounding, but the scaled down OPT is still at least $\frac{n}{\epsilon}$. We have only lost an ϵ fraction of the solution. This process of rounding and scaling values for use in exact algorithms has use in a large number of other maximization problems. We now formally state the algorithm Round&Scale and prove its correctness and running time.

```
Round&Scale($N$, $B$)

1. For each $i$ set $p'_i = \lfloor \frac{n}{\epsilon} \frac{1}{p_{\max}} p_i \rfloor$

2. Run exact algorithm with run time $O(nP')$ to obtain $A$

3. Output $A$
```

Theorem 3.6. Round&Scale *gives a* $(1 - \epsilon)$ *approximation and runs in* $O(\frac{n^3}{\epsilon})$ *time.*

Proof. The rounding can be done in linear time and as $P' = O(\frac{n^2}{\epsilon})$, the dynamic programing portion of the algorithm runs in $O(\frac{n^3}{\epsilon})$ time. To show the approximation ratio, let $\alpha = \frac{n}{\epsilon} \frac{1}{p_{\max}}$ and let A be the solution returned by the algorithm and A^* be the optimal solution. Observe that for all $X \subseteq N$, $\alpha p(X) - |X| \leq p'(X) \leq \alpha p(X)$ as the rounding lowers each scaled profit by at most 1. The algorithm returns the best choice for A given the scaled and rounded values, so we know $p'(A) \geq p'(A^*)$.

$$p(A) \geq \frac{1}{\alpha} p'(A) \geq \frac{1}{\alpha} p'(A^*) \geq p(A^*) - \frac{n}{\alpha} = \text{OPT} - \epsilon p_{\max} \geq (1 - \epsilon)\,\text{OPT}$$

■

It should be noted that this is not the best FPTAS known for Knapsack. In particular, [111] shows a FPTAS that runs in $O(n \log(1/\epsilon) + 1/\epsilon^4)$ time. There have been several improvements and we refer the reader to Chan's paper for the latest [32].

3.2 Other Problems

There are many variants of Knapsack and it is a fundamental problem of interest in integer programming as well in several other areas. One can find a book length treatment in [102]. We close with an interesting variant.

Multiple Knapsack: The input now consists of m knapsacks with capacities $B_1, B_2, \ldots, B_m$ and n items with sizes and profits as in Knapsack. We again wish to pack items to obtains as large a profit as possible, except now we have more than one knapsack with which to do so. An interesting special case is when all the knapsack capacities are the same quantity B which is related to the well known Bin Packing problem.

Chapter 4

Packing Problems

In the previous lecture we discussed the Knapsack problem. In this lecture we discuss other *packing* and *independent set* problems. We first discuss an abstract model of packing problems. Let N be a finite ground set. A collection of $\mathcal{I} \subset 2^N$ of subsets of N is said to be *down closed* if the following property is true: $A \in \mathcal{I}$ implies that for all $B \subset A$, $B \in \mathcal{I}$. A down closed collection is also often called and *independence* system. The sets in $\mathcal{I}$ are called independent sets. Given an independence family $(N, \mathcal{I})$ and a non-negative weight function $w : N \to \mathbb{R}^+$ the maximum weight independent set problem is to find $\max_{S \in \mathcal{I}} w(S)$. That is, find an independent set in $\mathcal{I}$ of maximum weight. Often we may be interested in the setting where all weights are 1 in which case we wish to find the maximum cardinality independent set. We discuss some canonical examples.

Example 4.1. Independent sets in graphs: Given a graph $G = (V, E)$ $\mathcal{I} = \{S \subseteq V \mid \text{there are no edges between nodes in } S\}$. Here the ground set is V. There are many interesting special cases of the graph problem. For instance problems arising from geometric objects such as intervals, rectangles, disks and others.

Example 4.2. Matchings in graphs: Given a graph $G = (V, E)$ let $\mathcal{I} = \{M \subseteq E \mid M \text{ is a matching in } G\}$. Here the ground set is E.

Example 4.3. Matroids: A matroid $\mathcal{M} = (N, \mathcal{I})$ is defined as a system where $\mathcal{I}$ is down closed and in addition satisfies the following key property: if $A, B \in \mathcal{I}$ and $|B| > |A|$ then there is an element $e \in B \setminus A$ such that $A \cup \{e\} \in \mathcal{I}$. There are many examples of matroids. We will not go into details here.

Example 4.4. Intersections of independence systems: given some k independence systems on the same ground set $(N, \mathcal{I}_1), (N, \mathcal{I}_2), \ldots, (N, \mathcal{I}_k)$ the system defined by $(N, \mathcal{I}_1 \cap \mathcal{I}_2 \ldots \cap \mathcal{I}_k)$ is also an independence system. Well-known examples include intersections of matroids.

4.1 Maximum Independent Set Problem in Graphs

A basic graph optimization problem with many applications is the *maximum (weighted) independent set problem* (MIS) in graphs.

Definition 4.1. *Given an undirected graph $G = (V, E)$ a subset of nodes $S \subseteq V$ is an independent set (stable set) iff there is no edge in E between any two nodes in S. A subset of nodes S is a* clique *if every pair of nodes in S have an edge between them in G.*

The MIS problem is the following: given a graph $G = (V, E)$ find an independent set in G of maximum cardinality. In the weighted case, each node $v \in V$ has an associated non-negative weight $w(v)$ and the goal is to find a maximum weight independent set. This problem is NP-Hard and it is natural to ask for approximation algorithms. Unfortunately, as the famous theorem below shows, the problem is extremely hard to approximate.

Theorem 4.2 (Håstad [80]). *Unless $P = NP$ there is no $\frac{1}{n^{1-\epsilon}}$-approximation for MIS for any fixed $\epsilon > 0$ where n is the number of nodes in the given graph.*

Remark 4.1. The maximum clique problem is to find the maximum cardinality clique in a given graph. It is approximation-equivalent to the MIS problem; simply complement the graph.

The theorem basically says the following: there are a class of graphs in which the maximum independent set size is either less than n^{δ} or greater than $n^{1-\delta}$ and it is NP-Complete to decide whether a given graph falls into the former category or the latter.

The lower bound result suggests that one should focus on special cases, and several interesting positive results are known. First, we consider a simple greedy algorithm for the unweighted problem.

```
GREEDY($G$)
1. $S \leftarrow \emptyset$
2. While $G$ is not empty do
    A. Let $v$ be a node of minimum degree in $G$
    B. $S \leftarrow S \cup \{v\}$
    C. Remove $v$ and its neighbors from $G$
3. Output $S$
```

Theorem 4.3. *Greedy outputs an independent set S such that $|S| \geq n/(\Delta+1)$ where Δ is the maximum degree of any node in the graph. Moreover $|S| \geq \alpha(G)/\Delta$ where $\alpha(G)$ is the cardinality of the largest independent set. Thus Greedy is a $1/\Delta$ approximation.*

Proof. We upper bound the number of nodes in $V \setminus S$ as follows. A node u is in $V \setminus S$ because it is removed as a neighbor of some node $v \in S$ when Greedy added v to S. Charge u to v. A node $v \in S$ can be charged at most Δ times since it has at most Δ neighbors. Hence we have that $|V \setminus S| \leq \Delta|S|$. Since every node is either in S or $V \setminus S$ we have $|S| + |V \setminus S| = n$ and therefore $(\Delta+1)|S| \geq n$ which implies that $|S| \geq n/(\Delta+1)$.

We now argue that $|S| \geq \alpha(G)/\Delta$. Let S^* be a largest independent set in G. As in the above proof we can charge each node v in $S^* \setminus S$ to a node $u \in S \setminus S^*$ which is a neighbor of v. The number of nodes charged to a node $u \in S \setminus S^*$ is at most Δ. Thus $|S^* \setminus S| \leq \Delta|S \setminus S^*|$.

■

Exercise 4.1. Show that Greedy outputs an independent set of size at least $\frac{n}{2(d+1)}$ where d is the *average* degree of G.

Remark 4.2. The well-known Turan's theorem shows via a clever argument that there is always an independent set of size $\frac{n}{(d+1)}$ where d is the average degree of G.

Remark 4.3. For the case of unweighted graphs one can obtain an approximation ratio of $\Omega(\frac{\log d}{d \log\log d})$ where d is the average degree. Surprisingly, under a complexity theory conjecture called the Unique-Games conjecture it is known to be NP-Hard to approximate MIS to within a factor of $O(\frac{\log^2 \Delta}{\Delta})$ in graphs with maximum degree Δ when Δ is sufficiently large.

Exercise 4.2. Consider the weigthed MIS problem on graphs of maximum degree Δ. Alter Greedy to sort the nodes in non-increasing order of the weight and show that it gives a $\frac{1}{\Delta}$-approximation. Can one obtain an $\Omega(1/d)$-approximation for the weighted case where d is the average degree?

LP Relaxation: One can formulate a simple linear-programming relaxation for the (weighted) MIS problem where we have a variable $x(v)$ for each node $v \in V$ indicating whether v is chosen in the independent set or not. We have constraints which state that for each edge (u,v) only one of u or v can be chosen.

$$\begin{aligned} \text{maximize} \sum_{v \in V} w(v)x(v) & \\ \text{subject to } x(u) + x(v) \leq 1 & \qquad (u,v) \in E \\ x(v) \in [0,1] & \qquad v \in V \end{aligned}$$

Although the above is a valid integer programming relaxation of MIS when the variables are constrained to be in $\{0, 1\}$, it is not a particularly useful formulation for the following simple reason.

Claim 4.1.1. *For* any *graph the optimum value of the above LP relaxation is at least $w(V)/2$. In particular, for the unweighted case it is at least $n/2$.*

Simply set each $x(v)$ to $1/2$!

One can obtain a *strengthened* formulation below by observing that if S is clique in G then any independent set can pick at most one node from S.

$$\begin{aligned} \text{maximize} \sum_{v \in V} w(v)x(v) & \\ \text{subject to} \sum_{v \in S} x(v) \leq 1 & \qquad S \text{ is a clique in } G \\ x(v) \in [0, 1] & \qquad v \in V \end{aligned}$$

The above linear program has an exponential number of constraints, and it cannot be solved in polynomial time in general, but for some special cases of interest the above linear program can indeed be solved (or approximately solved) in polynomial time and leads to either exact algorithms or good approximation bounds.

Approximability of Vertex Cover **and** MIS**:** The following is a basic fact and is easy to prove.

Fact 4.1. *In any graph $G = (V, E)$, S is a vertex cover in G if and only if $V \setminus S$ is an independent set in G. Thus $\alpha(G) + \beta(G) = |V|$ where $\alpha(G)$ is the size of a maximum independent set in G and $\beta(G)$ is the size of a minimum vertex cover in G.*

The above shows that if one of Vertex Cover or MIS is NP-Hard then the other is as well. We have seen that Vertex Cover admits a 2-approximation while MIS admits no constant factor approximation. It is useful to see why a 2-approximation for Vertex Cover does not give any useful information for MIS even though $\alpha(G) + \beta(G) = |V|$. Suppose S^* is an optimal vertex cover and has size $\geq |V|/2$. Then a 2-approximation algorithm is only guaranteed to give a vertex cover of size $|V|$! Hence one does not obtain a non-trivial independent set by complementing the approximate vertex cover.

Some special cases of MIS**:** We mention some special cases of MIS that have been considered in the literature, this is by no means an exhaustive list.

- Interval graphs; these are intersection graphs of intervals on a line. An exact algorithm can be obtained via dynamic programming and one can solve more general versions via linear programming methods.

- Note that a maximum (weight) matching in a graph G can be viewed as a maximum (weight) independent set in the line-graph of G and can be solved exactly in polynomial time. This has been extended to what are known as claw-free graphs.

- Planar graphs and generalizations to bounded-genus graphs, and graphs that exclude a fixed minor. For such graphs one can obtain a PTAS due to ideas originally from Brenda Baker.

- Geometric intersection graphs. For example, given n disks on the plane find a maximum number of disks that do not overlap. One could consider other (convex) shapes such as axis parallel rectangles, line segments, pseudo-disks etc. A number of results are known. For example a PTAS is known for disks in the plane. An $\Omega(\frac{1}{\log n})$-approximation for axis-parallel rectangles in the plane when the rectangles are weighted and an $\Omega(\frac{1}{\log \log n})$-approximation for the unweighted case. For the unweighted case, very recently, Mitchell obtained a constant factor approximation!

4.1.1 Elimination Orders and MIS

We have seen that a simple Greedy algorithm gives a Δ-approximation for MIS in graphs with max degree Δ. One can also get a Δ approximation for a larger class of Δ-degenerate graphs. To motivate degenerate graphs consider the class of planar graphs. The maximum degree of a planar graph need not be small. Nevertheless, via Euler's theorem, we know that every planar graph has a vertex of degree at most 5 since the maximum number of edges in a planar graph is at most $3n - 6$. Moreover, every subgraph of a planar graph is planar, and hence the Greedy algorithm will repeatedly find a vertex of degree at most 5 in each iteration. From this one can show that Greedy gives a $1/5$-approximation for MIS in planar graphs. Now consider the intersection graph of a collection of intervals on the real line. That is, we are given n intervals $I_1, I_2, \ldots, I_n$ where each $I_i = [a_i, b_i]$ for real numbers $a_i \leq b_i$. The goal is to find a maximum number of the intervals in the given set of intervals which do not overlap. This is the same as finding MIS in the *intersection graph* of the intervals - the graph is obtained by creating a vertex v_i for each I_i, and by adding edges $v_i v_j$ if I_i and I_j overlap. It is well-known that greedily picking intervals in earliest finish time order (ordering them according to b_i values) is optimal; the reader should try to prove this. Can one understand the analysis of all these examples in a unified fashion? Yes. For this purpose we consider the class of inductive k-independent graphs considered by by Akcoglu et al. [7] and later again by Ye and Borodin [159].

For a vertex v in a graph we use $N(v)$ denote the neighbors of v (not including v itself). For a graph $G = (V, E)$ and $S \subset V$ we use $G[S]$ to denote the subgraph of G induced by S.

Definition 4.4. *An undirected graph $G = (V, E)$ is inductive k-independent if there is an ordering of the vertices $v_1, v_2, \dots, v_n$ such that for $1 \leq i \leq n$, $\alpha(G[N(v_i) \cap \{v_{i+1}, \dots, v_n\}]) \leq k$.*

Graphs which are inductively 1-independent have a *perfect* elimination ordering and are called chordal graphs because they have an alternate characterization. A graph is chordal iff each cycle C in G has a chord (an edge connecting two nodes of C which is not an edge of C), or in other words there is no induced cycle of length more than 3.

Exercise 4.3. Prove that the intersection graph of intervals is chordal.

Exercise 4.4. Prove that if $\Delta(G) \leq k$ then G is inductively k-independent. Prove that if G is k-degenerate then G is inductively k-independent.

The preceding shows that planar graphs are inductively 5-independent. In fact, one can show something stronger, that they are inductively 3-independent. Given a graph G one can ask whether there is an algorithm that checks whether G is inductively k-independent. There is such an algorithm that runs in time $O(k^2 n^{k+2})$ [159]. A classical result shows how to recognize chordal graphs ($k = 1$) in linear time. However, most of the useful applications arise by showing that a certain class of graphs are inductively k-independent for some small value of k. See [159] for several examples.

Exercise 4.5. Prove that the Greedy algorithm that considers the vertices in the inductive k-independent order gives a $\frac{1}{k}$-approximation for MIS.

Interestingly one can obtain a $\frac{1}{k}$-approximation for the maximum weight independent set problem in inductively k-independent graphs. The algorithm is simple and runs in linear time but is not obvious. To see this consider the weighted problem for intervals. The standard algorithm to solve this is via dynamic programming. However, one can obtain an optimum solution for all chordal graphs (given the ordering). We refer the reader to [159] for the algorithm and proof (originally from [7]). Showing a $\Omega(1/k)$-approximation is easier.

4.2 The efficacy of the Greedy algorithm for a class of Independence Families

The Greedy algorithm can be defined easily for an arbitrary independence system. It iteratively adds the best element to the current independent set while maintaining feasibility. Note that the implementation of the algorithm requires having an oracle to find the best element to add to a current independent set S.

```
GREEDY($N, \mathcal{I}$)
1. $S \leftarrow \emptyset$
2. While (TRUE)
    A. Let $A \leftarrow \{e \in N \setminus S \mid S + e \in \mathcal{I}\}$
    B. If $A = \emptyset$ break
    C. $e \leftarrow \text{argmax}_{e \in A} w(e)$
    D. $S \leftarrow S \cup \{e\}$
3. Output $S$
```

Exercise 4.6. Prove that the Greedy algorithm gives a 1/2-approximation for the maximum weight matching problem in a general graph. Also prove that this bound is tight even in bipartite graphs. Note that max weight matching can be solved exactly in polynomial time.

Remark 4.4. It is well-known that the Greedy algorithm gives an optimum solution when $(N, \mathcal{I})$ is a matroid. Kruskal's algorithm for min/max weight spanning tree is a special case of this fact.

It is easy to see that Greedy does poorly for MIS problem in general graphs. A natural question is what properties of $\mathcal{I}$ enable some reasonable performance guarantee for Greedy. A very general result in this context has been established due to Jenkyn's generalizing several previous results. In order to state the result we set up some notation. Given an independence system $(N, \mathcal{I})$ we say that a set $A \in \mathcal{I}$ is a *base* if it is a *maximal* independent set. It is well-known that in a matroid $\mathcal{M}$ all bases have the same cardinality. However this is not true in general independence system.

Definition 4.5. *An independence system $(N, \mathcal{I})$ is a k-system if for any two bases $A, B \in \mathcal{I}$, $|A| \leq k|B|$. That is, the ratio of the cardinality of a maximum base and the cardinality of a minimum base is at most k.*

The following theorem is not too difficult but not so obvious either.

Theorem 4.6. *Greedy gives a $1/k$-approximation for the maximum weight independent set problem in a k-system.*

The above theorem generalizes and unifies several examples that we have seen so far including MIS in bounded degree graphs, matchings, matroids etc. How does one see that a given independence system is indeed a k-system for some parameter k? For instance matchings in graphs form a 2-system. The following simple lemma gives an easy way to argue that a given system is a k-system.

Lemma 4.1. *Suppose $(N, \mathcal{I})$ is an independence system with the following property: for any $A \in \mathcal{I}$ and $e \in N \setminus A$ there is a set $Y \subset A$ such that $|Y| \leq k$ and $(A \setminus Y) \cup \{e\} \in \mathcal{I}$. Then $\mathcal{I}$ is a k-system.*

We leave the proof of the above as an exercise.

We refer the reader to [59, 120] for analysis of Greedy in k-systems and other special cases.

4.3 Randomized Rounding with Alteration for Packing Problems

The purpose of this section to highlight a technique for rounding LP relaxations for packing problems. We will consider a simple example, namely the maximum weight independent set problem in interval graphs. Recall that we are given n intervals $I_1, I_2, \ldots, I_n$ with non-negative weights $w_1, \ldots, w_n$ and the goal is to find a maximum weight subset of them which do not overlap. Let $I_i = [a_i, b_i]$ and let $p_1, p_2, \ldots, p_m$ be the collection of end points of the intervals. We can write a simple LP relaxation for this problem. For each interval i we have a variable $x_i \in [0, 1]$ to indicate whether I_i is chosen or not. For each point p_j, among all intervals that contain it, at most one can be chosen. These are clique constraints in the underlying interval graph.

$$\begin{aligned}
\text{maximize} \quad & \sum_{i=1}^{n} w_i x_i \\
\text{subject to} \quad & \sum_{i: p_j \in I_i} x_i \leq 1 \qquad 1 \leq j \leq m \\
& x_i \in [0, 1] \qquad 1 \leq i \leq n
\end{aligned}$$

Note that it is important to retain the constraint that $x_i \leq 1$. Interestingly it is known that the LP relaxation defines an integer polytope and hence one can solve the integer program by solving the LP relaxation! This is because the incidence matrix defining the LP is totally unimodular (TUM). We refer the reader to books on combinatorial optimization for further background on this topic. Here we assume that we do not know the integer properties of the LP. We will round it via a technique that is powerful and generalizes to NP-Hard variants of the interval scheduling problem among many others.

Suppose we solve the LP and obtain an optimum fraction solution x^*. We have $\sum_i w_i x_i^* \geq OPT$. How do we round to obtain an integer solution whose value is close to that of OPT? Suppose we randomly choose I_i with probablility cx_i^* for some $c \leq 1$. Let R be the random set of chosen intervals. Then the expected weight of R, by linearity of expectation, is $c \sum_i w_i x_i^* \geq c \cdot OPT$. However, it is highly likely that the random solution R is not going to be feasible. Some constraint will be violated. The question is how we can *fix* or *alter* R to find a subset $R' \subseteq R$ such that R' is a feasible solution *and* the expected value of R' is not too much smaller than that of R. This depends on the independence structure.

Here we illustrate this via the interval problem. Without loss of generality we assume that $I_1, \ldots, I_n$ are sorted by their right end point. In other words the order is a perfect elimination order for the underlying interval graph.

```
ROUNDING-WITH-ALTERATION

1. Let $x$ be an optimum fractional solution
2. Round each $i$ to $1$ independently with probability $x_i/2$. Let $x'$ be rounded
   solution.
3. $R \leftarrow \{i \mid x'_i = 1\}$
4. $S \leftarrow \emptyset$
5. For $i = n$ down to $1$ do
   A. If $(i \in R)$ and $(S \cup \{i\}$ is feasible) then $S \leftarrow S \cup \{i\}$
6. Output feasible solution $S$
```

The algorithm consists of two phases. The first phase is a simple selection phase via independent randomized rounding. The second phase is deterministic and is a greedy pruning step in the *reverse* elimination order. To analyze the expected value of S we consider two binary random variables for each i, Y_i and Z_i. Y_i is 1 if $i \in R$ and 0 otherwise. Z_i is 1 if $i \in S$ and 0 otherwise.

By linearity of expectation,

Claim 4.3.1. $\mathbb{E}[w(S)] = \sum_i w_i \mathbb{E}[Z_i] = \sum_i w_i \, \mathbf{P}[Z_i = 1]$.

Via the independent randomized rounding in the algorithm.

Claim 4.3.2. $\mathbf{P}[Y_i = 1] = x_i/2$.

How do we analyze $\mathbf{P}[Z_i = 1]$? The random variables $Z_1, \ldots, Z_n$ are not independent and could be highly correlated even though $Y_1, \ldots, Y_n$ are independent. For this purpose we try to understand $\mathbf{P}[Z_i = 0 \mid Y_i = 1]$ which is the conditional probability that an interval I_i that is chosen in the first step is rejected in the pruning phase. We often would not be able to get an exact estimate of this quantity but we can upper bound it as follows. Here the ordering plays a crucial role. Why would I_i be rejected in the pruning phase? Note that when I_i is considered in the pruning phase, the only intervals that have been considered have their right end points after the right end point of I_i. Let $A_i = \{j \mid j > i \text{ and } I_j \text{ and } I_i \text{ intersect at } b_i\}$ be the potential set of intervals that can cause i to be rejected. Recall that the LP implies the following constraint:

$$x_i + \sum_{j \in A} x_j \leq 1$$

at the point b_j. Let $\mathcal{E}_1$ be the event that I_i is rejected in the pruning phase. Let $\mathcal{E}_\in$ be the event that at least one of the intervals in A is selected in the *first phase*. Note that $\mathcal{E}_1$ can happen only if $\mathcal{E}_2$ happens. Thus $\mathbf{P}[\mathcal{E}_1] \leq \mathbf{P}[\mathcal{E}_2]$. In general we try to upper bound $\mathbf{P}[\mathcal{E}_2]$. In this simple case we have an exact formula for it.

$$\mathbf{P}[\mathcal{E}_2] = 1 - \prod_{j \in A} \mathbf{P}[Y_j = 0] = 1 - \prod_{j \in A}(1 - x_j/2).$$

We claim that $\mathbf{P}[\mathcal{E}_2] \leq \sum_{j \in A} x_j/2 \leq 1/2$. One can derive this by showing that $\prod_{j \in A}(1 - x_j/2)$ subject to $\sum_{j \in A} x_j/2 \leq 1/2$ is at least $1/2$. Another way of doing this is via Markov's inequality. Let $T = \sum_{j \in A} Y_j$ be the number of intervals from A selected in the first phase. $E[T] \leq \sum_{j \in A} x_j/2 < 1/2$. By Markov's inequality $\mathbf{P}[T \geq 2E[T]] \leq 1/2$. $\mathcal{E}_2$ is the event that $\mathbf{P}[T \geq 1]$.

Using the claim,

$$\mathbf{P}[Z_i = 1 \mid Y_i = 1] = 1 - \mathbf{P}[Z_i = 0 | Y_i = 1] \geq 1/2.$$

This allows us to lower bound the expected weight of the solution output by the algorithm, and yields a randomized $1/4$ approximation.

Claim 4.3.3. $\mathbb{E}[w(S)] \geq \sum_i w_i x_i/4$.

Proof. We have

$$\mathbb{E}[w(S)] = \sum_i w_i \,\mathbf{P}[Z_i = 1] = \sum_i w_i \,\mathbf{P}[Y_i = 1]\,\mathbf{P}[Z_i = 1 \mid Y_i = 1] \geq \sum_i w_i(\frac{x_i}{4} \cdot \frac{1}{2}) \geq \sum_i w_i x_i/4.$$

■

This type of rounding has applications to a variety of settings - see **[CVZ]** for applications and the general framework called *contention resolution schemes*.

4.4 Packing Integer Programs (PIPs)

We can express the Knapsack problem as the following integer program. We scaled the knapsack capacity to 1 without loss of generality.

$$\begin{aligned} \text{maximize } & \sum_{i=1}^{n} p_i x_i \\ \text{subject to } & \sum_{i} s_i x_i \leq 1 \\ & x_i \in \{0,1\} \qquad 1 \leq i \leq n \end{aligned}$$

More generally if have multiple linear constraints on the "items" we obtain the following integer program.

Definition 4.7. *A packing integer program (PIP) is an integer program of the form* $\max\{wx \mid Ax \leq 1, x \in \{0,1\}^n\}$ *where w is a* $1 \times n$ *non-negative vector and A is a* $m \times n$ *matrix with entries in* $[0,1]$. *We call it a* $\{0,1\}$*-PIP if all entries are in* $\{0,1\}$.

In some cases it is useful/natural to define the problem as $\max\{wx \mid Ax \leq b, x \in \{0,1\}^n\}$ where entries in A and b are required to rational/integer valued. We can convert it into the above form by dividing each row of A by b_i.

When m the number of rows of A (equivalently the constraints) is small the problem is tractable. It is some times called the m-dimensional knapsack and one can obtain a PTAS for any fixed constant m. However, when m is large we observe that MIS can be cast as a special case of $\{0,1\}$-PIP. It corresponds exactly to the simple integer/linear program that we saw in the previous section. Therefore the problem is at least as hard to approximate as MIS. Here we show via a clever LP-rounding idea that one can generalize the notion of bounded-degree to *column-sparsity* in PIPs and obtain a related approximation. We will then introduce the notion of *width* of the constraints and show how it allows for improved bounds.

Definition 4.8. *A PIP is k-column-sparse if the number of non-zero entries in each column of A is at most k. A PIP has width W if* $\max_{i,j} A_{ij}/b_i \leq 1/W$.

4.4.1 Randomized Rounding with Alteration for PIPs

We saw that randomized rounding gave an $O(\log n)$ approximation algorithm for the Set Cover problem which is a canonical covering problem. Here we will consider the use of randomized rounding for packing problems. Let x be an optimum fractional solution to the natural LP relaxation of a PIP where we replace the constraint $x \in \{0,1\}^n$ by $x \in [0,1]^n$. Suppose we apply independent randomized rounding where we set x'_i to 1 with probability x_i. Let x' be the resulting integer solution. The expected weight of this solution is exactly $\sum_i w_i x_i$ which is the LP solution value. However, x' may not satisfy the constraints given by $Ax \leq b$. A natural strategy to try to satisfy the constraints is to set x'_1 to 1 with probability cx_i where $c < 1$ is some scaling constant. This may help in satisfying the constraints because the scaling creates some room in the constraints; we now have that the expected solution value is $c \sum_i w_i x_i$, a loss of a factor of c. Scaling by itself does not allow us to claim that all constraints are satisfied with good probability. A very useful technique in this context is the technique of *alteration*; we judiciously fix/alter the rounded solution x' to force it to satisfy the constraints by setting some of the variables that are 1 in x' to 0. The trick is to do this in such a way as to have a handle on the final probability that a variable is set to 1. We will illustrate this for the KNAPSACK problem and then generalize the idea to k-sparse PIPs. The algorithms we present are from [18]. See **[CVZ]** for further applications and related problems.

Rounding for Knapsack: Consider the KNAPSACK problem. It is convenient to think of this in the context of PIPs. So we have $ax \leq 1$ where a_i now represents the size of item i and the knapsack capacity is 1; w_i is the weight of item. Suppose x is a fractional solution. Call an item i "big" if $a_i > 1/2$ and otherwise it is "small". Let S be the indices of small items and B the indices of the big items. Consider the following rounding algorithm.

```
ROUNDING-WITH-ALTERATION FOR KNAPSACK

1. Let $x$ be an optimum fractional solution
2. Round each $i$ to $1$ independently with probability $x_i/4$. Let $x'$ be rounded
   solution.
3. $x'' = x'$
4. If ($x'_i = 1$ for exactly one big item $i$)
   A. For each $j \neq i$ set $x''_j = 0$
5. Else If ($\sum_{i \in S} s_i x'_i > 1$ or two or more big items are chosen in $x'$)
   A. For each $j$ set $x''_j = 0$
6. Output feasible solution $x''$
```

In words, the algorithm alters the rounded solution x' as follows. If exactly one big item is chosen in x' then the algorithm retains that item and rejects all the other small items. Otherwise, the algorithm rejects all items if two or more big items are chosen in x' or if the total size of all small items chosen in x' exceeds the capacity.

The following claim is easy to verify.

Claim 4.4.1. *The integer solution x'' is feasible.*

Now let us analyze the probability of an item i being present in the final solution. Let $\mathcal{E}_1$ be the event that $\sum_{i\in S} a_i x'_i > 1$, that is the sum of the sizes of the small items chose in x' exceeds the capacity. Let $\mathcal{E}_2$ be the event that at least one big item is chosen in x'.

Claim 4.4.2. $\mathbf{P}[\mathcal{E}_1] \leq 1/4$.

Proof. Let $X_s = \sum_{i\in S} a_i x'_i$ be the random variable that measures the sum of the sizes of the small items chosen. We have, by linearity of expectation, that

$$\mathbb{E}[X_s] = \sum_{i\in S} a_i \mathbb{E}[x'_i] = \sum_{i\in S} a_i x_i/4 \leq 1/4.$$

By Markov's inequality, $\mathbf{P}[X_s > 1] \leq \mathbb{E}[X_s]/1 \leq 1/4$. ■

Claim 4.4.3. $\mathbf{P}[\mathcal{E}_2] \leq 1/2$.

Proof. Since the size of each big item in B is at least $1/2$, we have $1 \geq \sum_{i\in B} a_i x_i \geq \sum_{i\in B} x_i/2$. Therefore $\sum_{i\in B} x_i/4 \leq 1/2$. Event $\mathcal{E}_2$ happens if some item $i \in B$ is chosen in the random selection. Since i is chosen with probability $x_i/4$, by the union bound, $\mathbf{P}[\mathcal{E}_2] \leq \sum_{i\in B} x_i/4 \leq 1/2$. ■

Lemma 4.2. *Let Z_i be the indicator random variable that is 1 if $x''_i = 1$ and 0 otherwise. Then $\mathbb{E}[Z_i] = \mathbf{P}[Z_i = 1] \geq x_i/16$.*

Proof. We consider the binary random variable X_i which is 1 if $x'_i = 1$. We have $\mathbb{E}[X_i] = \mathbf{P}[X_i = 1] = x_i/4$. We write

$$\mathbf{P}[Z_i = 1] = \mathbf{P}[X_i = 1] \cdot \mathbf{P}[Z_i = 1 \mid X_i = 1] = \frac{x_i}{4}\mathbf{P}[Z_i = 1 \mid X_i = 1].$$

To lower bound $\mathbf{P}[Z_i = 1 \mid X_i = 1]$ we upper bound the probability $\mathbf{P}[Z_i = 0|X_i = 1]$, that is, the probability that we reject i conditioned on the fact that it is chosen in the random solution x'.

First consider a big item i that is chosen in x'. Then i is rejected iff if *another* big item is chosen in x'; the probability of this can be upper bounded by $\mathbf{P}[\mathcal{E}_1]$.

If item i is small then it is rejected if and only if $\mathcal{E}_2$ happens or if a big item is chosen which happens with $\mathbf{P}[\mathcal{E}_1]$. In either case

$$\mathbf{P}[Z_i = 0 | X_i = 1] \leq \mathbf{P}[\mathcal{E}_1] + \mathbf{P}[\mathcal{E}_2] \leq 1/4 + 1/2 = 3/4.$$

Thus,

$$\mathbf{P}[Z_i = 1] = \mathbf{P}[X_i = 1] \cdot \mathbf{P}[Z_i = 1 \mid X_i = 1] = \frac{x_i}{4}(1 - \mathbf{P}[Z_i = 0 \mid X_i = 1]) \geq \frac{x_i}{16}.$$

■

One can improve the above analysis to show that $\mathbf{P}[Z_i = 1] \geq x_i/8$.

Theorem 4.9. *The randomized algorithm outputs a feasible solution of expected weight at least $\sum_{i=1}^{n} w_i x_i/16$.*

Proof. The expected weight of the output is

$$\mathbb{E}[\sum_i w_i x''_i] = \sum_i w_i \mathbb{E}[Z_i] \geq \sum_i w_i x_i/16$$

where we used the previous lemma to lower bound $\mathbb{E}[Z_i]$. ■

Rounding for k-sparse PIPs: We now extend the rounding algorithm and analysis above to k-sparse PIPs. Let x be a feasible fractional solution to $\max\{wx \mid Ax \leq 1, x \in [0,1]^n\}$. For a column index i we let $N(i) = \{j \mid A_{j,i} > 0\}$ be the indices of the rows in which i has a non-zero entry. Since A is k-column-sparse we have that $|N(i)| \leq k$ for $1 \leq i \leq n$. When we have more than one constraint we cannot classify an item/index i as big or small since it may be big for some constraints and small for others. We say that i is small for constraint $j \in N(i)$ if $A_{j,i} \leq 1/2$ otherwise i is big for constraint j. Let $S_j = \{i \mid j \in N(i)$, and i small for $j\}$ be the set of all small columns for j and $B_j = \{i \mid j \in N(i)$, and i small for $j\}$ be the set of all big columns for j. Note that $S_j \cap B_j$ is the set of all i with $A_{j,i} > 0$.

```
Rounding-with-Alteration for $k$-sparse PIPs

1. Let $x$ be an optimum fractional solution
2. Round each $i$ to $1$ independently with probability $x_i/(4k)$. Let $x'$ be
   rounded solution.
3. $x'' = x'$
4. For $j = 1$ to $m$ do
   A. If ($x'_i = 1$ for exactly one $i \in B_j$)
      1. For each $h \in S_j \cup B_j$ and $h \neq i$ set $x''_h = 0$
   B. Else If ($\sum_{i \in S_j} A_{j,i} x'_i > 1$ or two or more items from $B_j$ are chosen in $x'$)
      1. For each $h \in S_j \cup B_j$ set $x''_h = 0$
5. Output feasible solution $x''$
```

The algorithm, after picking the random solution x', alters it as follows: it applies the previous algorithm's strategy to each constraint j separately. Thus an element i can be rejected at different constraints $j \in N(i)$. We need to bound the total probability of rejection. As before, the following claim is easy to verify.

Claim 4.4.4. *The integer solution x'' is feasible.*

Now let us analyze the probability of an item i being present in the final solution. Let $\mathcal{E}_1(j)$ be the event that $\sum_{i \in S_j} A_{j,i} x'_i > 1$, that is the sum of the sizes of the items that are small for j in x' exceed the capacity. Let $\mathcal{E}_2(j)$ be the event that at least one big item for j is chosen in x'. The following claims follow from the same reasoning as the ones before with the only change being the scaling factor.

Claim 4.4.5. $\mathbf{P}[\mathcal{E}_1(j)] \leq 1/(4k)$.

Claim 4.4.6. $\mathbf{P}[\mathcal{E}_2(j)] \leq 1/(2k)$.

Lemma 4.3. *Let Z_i be the indicator random variable that is 1 if $x''_i = 1$ and 0 otherwise. Then $\mathbb{E}[Z_i] = \mathbf{P}[Z_i = 1] \geq x_i/(16k)$.*

Proof. We consider the binary random variable X_i which is 1 if $x'_i = 1$ after the randomized rounding. We have $\mathbb{E}[X_i] = \mathbf{P}[X_i = 1] = x_i/(4k)$. We write

$$\mathbf{P}[Z_i = 1] = \mathbf{P}[X_i = 1] \cdot \mathbf{P}[Z_i = 1 \mid X_i = 1] = \frac{x_i}{4k} \mathbf{P}[Z_i = 1 \mid X_i = 1].$$

We upper bound the probability $\mathbf{P}[Z_i = 0|X_i = 1]$, that is, the probability that we reject i conditioned on the fact that it is chosen in the random solution x'. We observe that

$$\mathbf{P}[Z_i = 0|X_i = 1] \leq \sum_{j \in N(i)} (\mathbf{P}[\mathcal{E}_1(j)] + \mathbf{P}[\mathcal{E}_2(j)] \leq k(1/(4k) + 1/(2k)) \leq 3/4.$$

We used the fact that $N(i) \leq k$ and the claims above. Therefore,

$$\mathbf{P}[Z_i = 1] = \mathbf{P}[X_i = 1] \cdot \mathbf{P}[Z_i = 1 \mid X_i = 1] = \frac{x_i}{4k}(1 - \mathbf{P}[Z_i = 0 \mid X_i = 1]) \geq \frac{x_i}{16k}.$$

■

The theorem below follows by using the above lemma and linearity of expectation to compare the expected weight of the output of the randomized algorithm with that of the fractional solution.

Theorem 4.10. *The randomized algorithm outputs a feasible solution of expected weight at least $\sum_{i=1}^{n} w_i x_i/(16k)$. There is $1/(16k)$-approximation for k-sparse PIPs.*

Larger width helps: We saw during the discussion on the Knapsack problem that if all items are small with respect to the capacity constraint then one can obtain better approximations. For PIPs we defined the *width* of a given instance as W if $\max_{i,j} A_{ij}/b_i \leq 1/W$; in other words no single item is more than $1/W$ times the capacity of any constraint. One can show using a very similar algorithm and anaylisis as above that the approximation bound improves to $\Omega(1/k^{\lceil W \rceil})$ for instance with width W. Thus if $W = 2$ we get a $\Omega(1/\sqrt{k})$ approximation instead of $\Omega(1/k)$-approximation. More generally when $W \geq c \log k/\epsilon$ for some sufficiently large constant c we can get a $(1 - \epsilon)$-approximation. Thus, in the setting with multiple knapsack constraints, the notion of small with respect to capacities is that in each constraint the size of the item is $\leq \frac{c\epsilon}{\log k}$ times the capacity of that constraint.

Chapter 5

Load Balancing and Bin Packing

This chapter is based on notes first scribed by Rachit Agarwal.

In the last lecture, we studied the Knapsack problem. The Knapsack problem is an NP-hard problem but does admit a *pseudo-polynomial time* algorithm and can be solved efficiently if the input size is small. We used this pseudo-polynomial time algorithm to obtain an FPTAS for Knapsack. In this lecture, we study another class of problems, known as *strongly NP-hard* problems.

Definition 5.1 (Strongly NP-hard Problems)**.** *An NPO problem π is said to be strongly NP-hard if it is NP-hard even if the inputs are polynomially bounded in combinatorial size of the problem* [1]*.*

Many NP-hard problems are in fact strongly NP-hard. If a problem Π is strongly NP-hard, then Π does not admit a pseudo-polynomial time algorithm. We study two such problems in this lecture, Multiprocessor Scheduling and Bin Packing.

5.1 Load Balancing / MultiProcessor Scheduling

A central problem in scheduling theory is to design a schedule such that the finishing time of the last jobs (also called *makespan*) is minimized. This problem is often referred to as the Load Balancing, the Minimum Makespan Scheduling or Multiprocessor Scheduling problem.

[1] An alternative definition: A problem π is strongly NP-hard if every problem in NP can be polynomially reduced to π in such a way that numbers in the reduced instance are always written in unary

5.1.1 Problem Description

In the Multiprocessor scheduling problem, we are given m identical machines $M_1, \dots, M_m$ and n jobs $J_1, J_2, \dots, J_n$. Job J_i has a processing time $p_i \geq 0$ and the goal is to assign jobs to the machines so as to minimize the maximum load[2].

5.1.2 Greedy Algorithm

Consider the following greedy algorithm for the Multiprocessor Scheduling problem which we will call Greedy Multiprocessor Scheduling.

```
Greedy Multiprocessor Scheduling:
Order (list) the jobs arbitrarily
For $i = 1$ to $n$ do
    Assign Job $J_i$ to the machine with least current load
    Update load of the machine that receives job $J_i$
```

This algorithm is also called a *list scheduling* algorithm following Graham's terminology from his paper from 1966 [68]. The list is the order in which the jobs are processed and changing it creates different schedules. We prove that the Greedy Multiprocessor Scheduling algorithm gives a $(2 - 1/m)$-approximation.

Theorem 5.2. Greedy Multiprocessor Scheduling *algorithm gives a* $(2 - \frac{1}{m})$-*approximation for any list.*

To prove the theorem, we will make use of two lower bounds on the length of the optimal schedule which we denote by OPT.

Observation 5.3. $\text{OPT} \geq \dfrac{\sum_i p_i}{m}$.

Observation 5.4. $\text{OPT} \geq \max_i p_i$.

We leave the proofs of the observations as easy exercises.

Proof of Theorem 5.2. Fix the list and let L denote the makespan of the Greedy Multiprocessor Scheduling algorithm. Let L_i denote the load of machine M_i and let M_{i^*} be the most heavily loaded machine in the schedule by Greedy Multiprocessor Scheduling algorithm.

Let J_k be the last job assigned to M_{i^*}. Since Greedy Multiprocessor Scheduling algorithm assigns a job to the machine that is least loaded, all machines must be loaded at least $L - p_k$ at the time of assigning J_k. Hence, we have:

$$\left(\sum_{i=1}^{n} p_i\right) - p_k \geq m\,(L - p_k) \tag{5.1}$$

[2] The load of a machine is defined as the sum of the processing times of jobs that are assigned to that machine.

which implies:

$$\begin{aligned} L - p_k &\leq \frac{(\sum_{i=1}^{n} p_i) - p_k}{m} \\ \text{hence} & \\ L &\leq \frac{(\sum_{i=1}^{n} p_i)}{m} + p_k\left(1 - \frac{1}{m}\right) \\ &\leq \text{OPT} + \text{OPT}\left(1 - \frac{1}{m}\right) \\ &= \text{OPT}\left(2 - \frac{1}{m}\right) \end{aligned}$$

where the third step follows from the two lower bounds on OPT. ■

The above analysis is tight, *i.e.,* there exist instances where the greedy algorithm produces a schedule which has a makespan $(2 - 1/m)$ times the optimal. Consider the following instance: $m(m-1)$ jobs with unit processing time and a single job with processing time m. Suppose the greedy algorithm schedules all the short jobs before the long job, then the makespan of the schedule obtained is $(2m - 1)$ while the optimal makespan is m. Hence the algorithm gives a schedule which has makespan $2 - 1/m$ times the optimal.

It may seem from the tight example above that an approximation ratio $\alpha < (2 - 1/m)$ could be achieved if the jobs are sorted before processing, which indeed is the case. The following algorithm, due to [69], sorts the jobs in decreasing order of processing time prior to running Greedy Multiprocessor Scheduling algorithm.

```
Modified Greedy Multiprocessor Scheduling:
Sort the jobs in decreasing order of processing times
For $i = 1$ to $n$ do
    Assign Job $J_i$ to the machine with least current load
    Update load of the machine that receives job $J_i$
```

Graham [69] proved the following tight bound.

Theorem 5.5. Modified Greedy Multiprocessor Scheduling *algorithm gives a $(4/3 - 1/3m)$-approximation for the* Multiprocessor Scheduling *problem.*

We will not prove the preceding theorem which requires some careful case analysis. Instead we will show how one can obtain an easier bound of 3/2 via the following claim.

Claim 5.1.1. *Suppose $p_i \geq p_j$ for all $i > j$ and $n > m$. Then,* $\text{OPT} \geq p_m + p_{m+1}$.

Proof. Since $n > m$ and the processing times are sorted in decreasing order, some two of the $(m+1)$ largest jobs must be scheduled on the same machine. Notice that the load of this machine is at least $p_m + p_{m+1}$. ∎

Exercise 5.1. Prove that Modified Greedy Multiprocessor Scheduling gives a $(3/2 - 1/2m)$-approximation using the preceding claim and the other two lower bounds on OPT that we have seen already.

Before going to the description of a PTAS for Multiprocessor Scheduling problem, we discuss the case when the processing times of the jobs are bounded from above.

Claim 5.1.2. *If* $p_i \leq \epsilon \cdot \mathrm{OPT}, \forall i$, *then* Modified Greedy Multiprocessor Scheduling *gives a* $(1+\epsilon)$*-approximation.*

5.1.3 A PTAS for Multi-Processor Scheduling

We will now give a PTAS for the problem of scheduling jobs on identical machines. We would like to use the same set of ideas that were used for the Knapsack problem (see Lecture 4): that is, given an explicit time T we would like to round the job lengths and use dynamic programming to see if they will fit within time T. Then the unrounded job lengths should fit within time $T(1+\epsilon)$.

Big Jobs, Small Jobs and Rounding Big Jobs: For the discussion that follows, we assume that all the processing times have been scaled so that OPT = 1 and hence, $p_{max} \leq 1$.

Given all the jobs, we partition the jobs into two sets: *Big jobs* and *Small jobs*. We call a job J_i "big" if $p_i \geq \epsilon$. Let $\mathcal{B}$ and $\mathcal{S}$ denote the set of big jobs and small jobs respectively, *i.e.*, $\mathcal{B} = \{J_i \;:\; p_i \geq \epsilon\}$ and $\mathcal{S} = \{J_i \;:\; p_i < \epsilon\}$. The significance of such a partition is that once we pack the jobs in set $\mathcal{B}$, the jobs in set $\mathcal{S}$ can be greedily packed using list scheduling.

Claim 5.1.3. *If there is an assignment of jobs in* $\mathcal{B}$ *to the machines with load* L, *then greedily scheduling jobs of* $\mathcal{S}$ *on top gives a schedule of value no greater than* $\max\{L, (1+\epsilon)\,\mathrm{OPT}\}$.

Proof. Consider scheduling the jobs in $\mathcal{S}$ after all the jobs in $\mathcal{B}$ have been scheduled (with load L). If all of these jobs in $\mathcal{S}$ finish processing by time L, the total load is clearly no greater than L.

If the jobs in $\mathcal{S}$ can *not* be scheduled within time L, consider the last job to finish (after scheduling the small jobs). Suppose this job starts at time T'. All the machines must have been fully loaded up to T', which gives $\mathrm{OPT} \geq T'$. Since, for all jobs in $\mathcal{S}$, we have $p_i \leq \epsilon \cdot \mathrm{OPT}$, this job finishes at $T' + \epsilon \cdot \mathrm{OPT}$. Hence, the schedule can be no more than $T' + \epsilon \cdot \mathrm{OPT} \leq (1+\epsilon)\,\mathrm{OPT}$, settling the claim. ∎

Scheduling Big Jobs: We concentrate on scheduling the jobs in $\mathcal{B}$. We round the sizes of all jobs in $\mathcal{B}$ using geometrically increasing interval sizes using the following procedure:

```
ROUNDING JOBS:
For each big job $i$ do
  If $p_i \in (\epsilon(1+\epsilon)^j, \epsilon(1+\epsilon)^{j+1}]$
      Set $p_i = \epsilon(1+\epsilon)^{j+1}$
```

Let $\mathcal{B}'$ be the set of new jobs.

Claim 5.1.4. *If jobs in $\mathcal{B}$ can be scheduled with load 1, then the rounded jobs in $\mathcal{B}'$ can be scheduled with load $(1+\epsilon)$.*

Claim 5.1.5. *The number of distinct big job sizes after rounding is $O(\frac{\ln(1/\epsilon)}{\epsilon})$.*

Proof. Notice that due to scaling, we have $p_i \leq 1$ for all jobs J_i. Since the job sizes are between ϵ and 1 the number of geometric powers of $(1+\epsilon)$ required is k where

$$\begin{aligned} \epsilon(1+\epsilon)^k &\leq 1 \\ \Rightarrow k &\leq \ln_{(1+\epsilon)}\frac{1}{\epsilon} = O(\frac{\ln(1/\epsilon)}{\epsilon}). \end{aligned}$$

■

Lemma 5.1. *If the number of distinct job sizes is k, then there is an exact algorithm that returns the schedule (if there is one) and runs in time $O(n^{2k})$.*

Proof. Use Dynamic Programming. ■

Corollary 5.6. *Big Jobs can be scheduled (if possible) with load $(1+\epsilon)$ in time $n^{O(\frac{\ln(1/\epsilon)}{\epsilon})}$.*

Once we have scheduled the jobs in $\mathcal{B}$, using Claim 5.1.3, we can pack small items using greedy list scheduling on top of them. The overall algorithm is then given as:

```
PTAS MULTIPROCESSOR SCHEDULING:
1. Guess OPT
2. Define $\mathcal{B}$ and $\mathcal{S}$
3. Round $\mathcal{B}$ to $\mathcal{B}'$
4. If jobs in $\mathcal{B}'$ can be scheduled in $(1+\epsilon)$ OPT
       Greedily pack $\mathcal{S}$ on top
   Else
       Modify the guess and Repeat.
```

In the following subsection, we comment on the guessing process.

Guessing: We define a $(1+\epsilon)$-relaxed decision procedure:

Definition 5.7. *Given $\epsilon > 0$ and a time T, a $(1+\epsilon)$-relaxed decision procedure returns:*

- *Output a schedule (if there is one) with makespan load $(1+\epsilon) \cdot T$ or*
- *Output correctly that there is no schedule with makespan T.*

Define

$$L = \max\left\{\max_j p_j, \frac{1}{m}\sum_j p_j\right\}$$

L is a lower bound on OPT as we saw earlier. Furthermore, an upper bound on OPT is given by the Greedy Multiprocessor Scheduling algorithm, which is $2L$. Consider running the decision procedure with guess $L + i\epsilon L$ for each integer $i \in [\lceil 2/\epsilon \rceil]$. We will choose the schedule with the best makespan among all the successful runs. If L^* is the optimum load then the algorithm will try the decision procedure with $L^* + \epsilon L \leq (1+\epsilon)L^*$. For this guess we are guaranteed a solution and the decision procedure will succeed in outputting a schedule with load $(1+\epsilon)(1+\epsilon)L^* \leq (1+3\epsilon)L^*$ for sufficiently small ϵ. We run the decision procedure $O(1/\epsilon)$ times. This gives us the desired PTAS.

Remark 5.1. A PTAS indicates that the problem can approximated arbitrarily well in polynomial time. However, a running time of the form $n^{f(\epsilon)}$ is typically not very interesting. We have seen that an FPTAS is ruled out for the makespan minimization problem. However, it does admit what is now called an *Efficient* PTAS (EPTAS) whose running time is $2^{O(1/\epsilon^2 \cdot (\log(1/\epsilon))^3)} + \text{poly}(n)$. See [93].

5.1.4 Section Notes

Multiprocessor Scheduling is NP-hard as we can reduce 2-Partition to Multiprocessor Scheduling on two machines. Note that this reduction only proves that Multiprocessor Scheduling is weakly NP-hard. When m is a fixed constant Horowitz and Sahni [87] give an FPTAS. However Multiprocessor Scheduling problem is strongly NP-hard when m is part of the input (by a reduction from 3-Partition [62]). Thus, there can not exist an FPTAS for the Multiprocessor Scheduling problem in general, unless $P = NP$. However, Hochbaum and Shmoys [83] gave a PTAS which is the one we described. EPTASes have been developed for several problems and a key technique is the use of integer linear programming solvers with a small number of variables.

5.2 Bin Packing

5.2.1 Problem Description

In the Bin Packing problem, we are given a set of n items $\{1, 2, \ldots, n\}$. Item i has size $s_i \in (0, 1]$. The goal is to find a minimum number of bins of capacity 1 into which all the items can be packed.

One could also formulate the problem as partitioning $\{1, 2, \ldots, n\}$ into k sets $\mathcal{B}_1, \mathcal{B}_2, \ldots, \mathcal{B}_k$ such that $\sum_{i \in \mathcal{B}_j} s_i \leq 1$ and k is minimum.

5.2.2 Greedy Approaches

Consider the following greedy algorithm for bin packing:

```
Greedy Bin Packing:
Order items in some way
For $i = 1$ to $n$
  If item $i$ can be packed in some open bin
      Pack it
  Else
      Open a new bin and pack $i$ in the new bin
```

In Greedy Bin Packing algorithm, a new bin is opened only if the item can not be packed in any of the already opened bins. However, there might be several opened bins in which the item i could be packed. Several rules could be formulated in such a scenario:

- *First Fit*: Pack item in the earliest opened bin
- *Last Fit*: Pack item in the last opened bin
- *Best Fit*: Pack item in the bin that would have least amount of space left after packing the item
- *Worst Fit*: Pack item in the bin that would have most amount of space left after packing the item

Irrespective of what strategy is chosen to pack an item in the opened bins, one could get the following result:

Theorem 5.8. *Any greedy rule yields a 2-approximation.*

Observation 5.9. OPT $\geq \sum_i s_i$.

We call a bin *α-full* if items occupy space at most α.

Claim 5.2.1. *Greedy has at most* 1 *bin that is* $\frac{1}{2}$*-full.*

Proof. For the sake of contradiction, assume that there are two bins B_i and B_j that are $\frac{1}{2}$-full. WLOG, assume that GREEDY BIN PACKING algorithm opened bin B_i before B_j. Then, the first item that the algorithm packed into B_j must be of size at most $\frac{1}{2}$. However, this item could have been packed into B_i since B_i is $\frac{1}{2}$-full. This is a contradiction to the fact that GREEDY BIN PACKING algorithm opens a new bin if and only if the item can not be packed in any of the opened bins. ■

Proof of Theorem 5.8. Let m be the number of bins opened by GREEDY BIN PACKING algorithm. From Claim 5.2.1, we have:

$$\sum_i s_i > \frac{m-1}{2}$$

Using the observation that OPT $\geq \sum_i s_i$, we get:

$$\text{OPT} > \frac{m-1}{2}$$

which gives us:

$$\begin{aligned} m &< 2 \cdot \text{OPT} + 1 \\ \Rightarrow m &\leq 2 \cdot \text{OPT} \end{aligned}$$

■

5.2.3 (Asymptotic) PTAS for Bin Packing

A natural question follows the discussion above: Can BIN PACKING have a PTAS? In this subsection, we settle this question in negative. In particular, we give a reduction from an NP-complete problem to the BIN PACKING problem and show that a PTAS for the BIN PACKING problem will give us an exact solution for the NP-complete problem in polynomial time. We consider the PARTITION problem:

In the PARTITION problem, we are given a set of items $\{1, 2, \ldots, n\}$. Item i has a size s_i. The goal is to partition the $\{1, 2, \ldots, n\}$ into two sets $\mathcal{A}$ and $\mathcal{B}$ such that $\sum_{i \in \mathcal{A}} s_i = \sum_{j \in \mathcal{B}} s_j$.

Claim 5.2.2. *If* BIN PACKING *has a* $(\frac{3}{2} - \epsilon)$*-approximation for any* $\epsilon > 0$*, the* PARTITION *problem can be solved exactly in polynomial time.*

Proof. Given an instance I of the Partition problem, we construct an instance of the Bin Packing problem as follows: Scale the size of the items such that $\sum_i s_i = 2$. Consider the scaled sizes of the items as an instance I' of the Bin Packing problem. If all items of I' can be packed in 2 bins, then we have an "yes" answer to I. Otherwise, the items of I' need 3 bins and the answer to I is "no".

OPT for I' is 2 or 3. Hence, if there is a $(\frac{3}{2} - \epsilon)$-approximation algorithm for the Bin Packing problem, we can determine the value of OPT which in turn implies that we can solve I. Thus, there can not exist a $(\frac{3}{2} - \epsilon)$-approximation algorithm for the Bin Packing problem, unless P = NP. ■

Recall the scaling property where we discussed why many optimization problems do not admit additive approximations. We notice that the Bin Packing problem does not have the scaling property. Hence it may be possible to find an additive approximation algorithms. We state some of the results in this context:

Theorem 5.10 (Johnson '74 [96]). *There exists a polynomial time algorithm such $\mathcal{A}_J$ such that:*

$$\mathcal{A}_J(I) \leq \frac{11}{9}\,\mathrm{OPT}(I) + 4$$

for all instances I of the Bin Packing *problem.*

Theorem 5.11 (de la Vega, Lueker '81 [48]). *For any fixed $\epsilon > 0$ there exists a polynomial time algorithm such $\mathcal{A}_{FL}$ such that:*

$$\mathcal{A}_{FL}(I) \leq (1 + \epsilon)\,\mathrm{OPT}(I) + 1$$

for all instances I of the Bin Packing *problem.*

Theorem 5.12 (Karmarkar, Karp '82 [100]). *There exists a polynomial time algorithm such $\mathcal{A}_{KK}$ such that:*

$$\mathcal{A}_{KK}(I) \leq \mathrm{OPT}(I) + O(\log^2(\mathrm{OPT}(I)))$$

for all instances I of the Bin Packing *problem.*

This has been improved recently.

Theorem 5.13 (Hoberg and Rothvoss 2017 [82]). *There exists a polynomial time algorithm such $\mathcal{A}_{HT}$ such that:*

$$\mathcal{A}_{KK}(I) \leq \mathrm{OPT}(I) + O(\log(\mathrm{OPT}(I)))$$

for all instances I of the Bin Packing *problem.*

A major open problem is the following.

Open Question 5.14. *Is there a polynomial-time algorithm $\mathcal{A}$ such that $\mathcal{A}(I) \leq \mathrm{OPT}(I) + c$, for some fixed constant c? In particular is $c = 1$?*

Exercise 5.2. Show that *First Fit* greedy rule yields a $\frac{3}{2}\,\mathrm{OPT} + 1$-approximation.

5.2.4 Asymptotic PTAS for Bin Packing

A recurring theme in last two lectures has been the rounding of jobs/tasks/items. To construct an asymptotic PTAS for Bin Packing problem, we use the same set of ideas with simple in retrospect but non-obvious modifications. In particular, we divide the set of items into *big* and *small* items and concentrate on packing the big items first. We show that such a technique results in an asymptotic PTAS for the Bin Packing problem.

Consider the set of items, $s_1, s_2, \ldots, s_n$. We divide the items into two sets, $\mathcal{B} = \{i \ : \ s_i \geq \epsilon\}$ and $\mathcal{S} = \{j \ : \ s_j < \epsilon\}$. Similar to the Multiprocessor Scheduling problem, where we rounded up the processing times of the jobs, we round up the sizes of the items in the Bin Packing problem. Again, we concentrate only on the items in $\mathcal{B}$. Let $n' = |\mathcal{B}|$ be the number of big items.

We observed earlier that OPT $\geq \sum_i s_i$ and hence OPT $\geq \epsilon \cdot n'$ by just considering the big items.

Claim 5.2.3. *Suppose $n' > 4/\epsilon^2$. Then* OPT $\geq 4/\epsilon$.

If there are very few big items one can solve the problem by brute force.

Claim 5.2.4. *Suppose $n' < 4/\epsilon^2$. An optimal solution for the* Bin Packing *problem can be computed in $2^{O(1/\epsilon^4)}$ time.*

Proof Sketch. If the number of big items is small, one can find the optimal solution using brute force search. ■

The following gives a procedure to round up the items in $\mathcal{B}$:

Rounding Item Sizes:
Sort the items such that $s_1 \geq s_2 \geq \cdots \geq s_{n'}$
Group items in $k = 2/\epsilon^2$ groups $\mathcal{B}_1, \ldots, \mathcal{B}_k$ such that each group has $\lfloor n'/k \rfloor$ items
Round the size of all the items in group $\mathcal{B}_i$ to the size of the smallest item in $\mathcal{B}_{i-1}$

Lemma 5.2. *Consider the restriction of the bin packing problem to instances in which the number of distinct item sizes is k. There is an $n^{O(k)}$-time algorithm that outputs the optimum solution.*

Proof Sketch. Use Dynamic Programming. ■

Claim 5.2.5. *The items in $\mathcal{B}$ can be packed in* OPT $+|\mathcal{B}_1|$ *bins in time $n^{O(1/\epsilon^2)}$.*

Proof. Using Rounding Item Sizes, we have restricted all items but those in $\mathcal{B}_1$ to have one of the $k-1$ distinct sizes. Using lemma 5.2, these items can be packed efficiently in OPT. Furthermore, the items in $\mathcal{B}_1$ can always be packed in $|\mathcal{B}_1|$ bins (one per bin). Hence, the total number of bins is OPT $+|\mathcal{B}_1|$.

The running time of the algorithm follows since $k = O(1/\epsilon^2)$. ■

Lemma 5.3. *Let $\epsilon > 0$ be fixed. Consider the restriction of the bin packing problem to instances in which each items is of size at least ϵ. There is a polynomial time algorithm that solves this restricted problem within a factor of $(1 + \epsilon)$.*

Proof. Using Claim 5.2.5, we can pack $\mathcal{B}$ in OPT $+|\mathcal{B}_1|$ bins. Recall that $|\mathcal{B}_1| = \lfloor n'/k \rfloor \le \epsilon^2 \cdot n'/2 \le \epsilon \cdot \mathrm{OPT}/8$ where, we have used Claim 5.2.3 to reach the final expression. ∎

Theorem 5.15. *For any ϵ, $0 < \epsilon < 1/2$, there is an algorithm $\mathcal{A}_\epsilon$ that runs in time polynomial in n and finds a packing using at most $(1 + 2\epsilon)\,\mathrm{OPT} + 1$ bins.*

Proof. Assume that the number of bins used to pack items in $\mathcal{B}$ is m and the total number of bins used after packing items in $\mathcal{S}$ is m'. Clearly

$$m' \le \max\left\{m, \left\lceil \frac{(\sum_i s_i)}{(1-\epsilon)} \right\rceil\right\}$$

since at most one bin must be $(1 - \epsilon)$ full using an argument in Greedy Bin Packing. Furthermore,

$$\left\lceil \frac{(\sum_i s_i)}{(1-\epsilon)} \right\rceil \le \left(\sum_i s_i\right)(1 + 2\epsilon) + 1$$

for $\epsilon < 1/2$. This gives the required expression. ∎

The algorithm is summarized below:

```
Asymptotic Ptas Bin Packing:
Split the items in $\mathcal{B}$ (big items) and $\mathcal{S}$ (small items)
Round the sizes of the items in $\mathcal{B}$ to obtain constant number of item sizes
Find optimal packing for items with rounded sizes
Use this packing for original items in $\mathcal{B}$
Pack items in $\mathcal{S}$ using Greedy Bin Packing.
```

5.2.5 Section Notes

An excellent but perhaps somewhat dated survey on approximation algorithms for Bin Packing problem is [97]. See [82] for some pointers to more recent work. There has also been substantial recent work on various generalizations to multiple dimensions.

Chapter 6

Unrelated Machine Scheduling and Generalized Assignment

This chapter is based on notes first scribed by Alina Ene.

6.1 Scheduling on Unrelated Parallel Machines

We have a set J of n jobs, and a set M of m machines. The processing time of job i is p_{ij} on machine j. Let $f : J \to M$ be a function that assigns each job to exactly one machine. The *makespan* of f is $\max_{1 \le j \le m} \sum_{i: f(i)=j} p_{ij}$, where $\sum_{i: f(i)=j} p_{ij}$ is the total processing time of the jobs that are assigned to machine j. In the Scheduling on Unrelated Parallel Machines problem, the goal is to find an assignment of jobs to machines of minimum makespan.

We can write an LP for the problem that is very similar to the routing LP from the previous lecture. For each job i and each machine j, we have a variable x_{ij} that denotes whether job i is assigned to machine j. We also have a variable λ for the makespan. We have a constraint for each job that ensures that the job is assigned to some machine, and we have a constraint for each machine that ensures that the total processing time of jobs assigned to the machines is at most the makespan λ.

$$\begin{aligned}
\text{minimize} \quad & \lambda & \\
\text{subject to} \quad & \sum_{j \in M} x_{ij} = 1 & \forall i \in J \\
& \sum_{i \in J} x_{ij} p_{ij} \leq \lambda & \forall j \in M \\
& x_{ij} \geq 0 & \forall i \in J, j \in M
\end{aligned}$$

The above LP is very natural, but unfortunately it has unbounded integrality gap. Suppose that we have a single job that has processing time T on each of the machines. Clearly, the optimal schedule has makespan T. However, the LP can schedule the job to the extend of $1/m$ on each of the machines, i.e., it can set $x_{1j} = 1/m$ for all j, and the makespan of the resulting fractional schedule is only T/m.

To overcome this difficulty, we modify the LP slightly. Suppose we knew that the makespan of the optimal solution is equal to λ, where λ is some fixed number. If the processing time p_{ij} of job i on machine j is greater than λ, job i is *not* scheduled on machine j, and we can strengthen the LP by setting x_{ij} to 0 or equivalently, by removing the variable. More precisely, let $\mathcal{S}_\lambda = \{(i, j) \mid i \in J, j \in M, p_{ij} \leq \lambda\}$. Given a value λ, we can write the following LP for the problem.

LP(λ)

$$\begin{aligned}
& \sum_{j:\ (i,j) \in \mathcal{S}_\lambda} x_{ij} = 1 & \forall i \in J \\
& \sum_{i:\ (i,j) \in \mathcal{S}_\lambda} x_{ij} p_{ij} \leq \lambda & \forall j \in M \\
& x_{ij} \geq 0 & \forall (i, j) \in \mathcal{S}_\lambda
\end{aligned}$$

Note that the LP above does *not* have an objective function. In the following, we are only interested in whether the LP is feasible, i.e, whether there is an assignment that satisfies all the constraints. Also, we can think of λ as a parameter and **LP**(λ) as a family of LPs, one for each value of the parameter. A useful observation is that, if λ is a lower bound on the makespan of the optimal schedule, **LP**(λ) is feasible and it is a valid relaxation for the Scheduling on Unrelated Parallel Machines problem.

Lemma 6.1. *Let λ^* be the minimum value of the parameter λ such that* **LP**(λ) *is feasible. We can find λ^* in polynomial time.*

Proof. For any fixed value of λ, we can check whether $\mathbf{LP}(\lambda)$ is feasible using a polynomial-time algorithm for solving LPs. Thus we can find λ^* using binary search starting with the interval $[0, \sum_{i,j} p_{ij}]$. ■

In the following, we will show how to round a solution to $\mathbf{LP}(\lambda^*)$ in order to get a schedule with makespan at most $2\lambda^*$. As we will see shortly, it will help to round a solution to $\mathbf{LP}(\lambda^*)$ that is a *vertex* solution.

Let x be a vertex solution to $\mathbf{LP}(\lambda^*)$. Let G be a bipartite graph on the vertex set $J \cup M$ that has an edge ij for each variable $x_{ij} \neq 0$. We say that job i is fractionally set if $x_{ij} \in (0, 1)$ for some j. Let F be the set of all jobs that are fractionally set, and let H be a bipartite graph on the vertex set $F \cup M$ that has an edge ij for each variable $x_{ij} \in (0, 1)$; note that H is the induced subgraph of G on $F \cup M$. As shown in Lemma 6.2, the graph H has a matching that matches every job in F to a machine, and we will it in the rounding algorithm.

Lemma 6.2. *The graph G has a matching that matches every job in F to a machine.*

We are now ready to give the rounding algorithm.

```
SUPM-Rounding
  Find $\lambda^*$
  Find a vertex solution $x$ to $\mathbf{LP}(\lambda^*)$
  For each $i$ and $j$ such that $x_{ij} = 1$, assign job $i$ to machine $j$
  Construct the graph $H$
  Find a maximum matching $\mathcal{M}$ in $H$
  Assign the fractionally set jobs according to the matching $\mathcal{M}$
```

Theorem 6.1. *Consider the assignment constructed by SUPM-Rounding. Each job is assigned to a machine, and the makespan of the schedule is at most $2\lambda^*$.*

Proof. By Lemma 6.2, the matching $\mathcal{M}$ matches every fractionally set job to a machine and therefore all of the jobs are assigned. After assigning all of the integrally set jobs, the makespan (of the partial schedule) is at most λ^*. Since $\mathcal{M}$ is a matching, each machine receives at most one additional job. Let i be a fractionally set job, and suppose that i is matched (in $\mathcal{M}$) to machine j. Since the pair (i, j) is in $\mathcal{S}_{\lambda^*}$, the processing time p_{ij} is at most λ^*, and therefore the total processing time of machine j increases by at most λ after assigning the fractionally set jobs. Therefore the makespan of the final schedule is at most $2\lambda^*$. ■

Exercise 6.1. Give an example that shows that Theorem 6.1 is tight. That is, give an instance and a vertex solution such that the makespan of the schedule SUPM-Rounding is at least $(2 - o(1))\lambda^*$.

Since λ^* is a lower bound on the makespan of the optimal schedule, we get the following corollary.

Corollary 6.2. *SUPM-Rounding achieves a* 2*-approximation.*

Now we turn our attention to Lemma 6.2 and some other properties of vertex solutions to **LP**(λ). The following can be derived from the rank lemma which is described in Chapter A. Here we give a self-contained proof.

Lemma 6.3. *If **LP**(λ) is feasible, any vertex solution has at most $m + n$ non-zero variables and it sets at least $n - m$ of the jobs integrally.*

Proof. Let x be a vertex solution to **LP**(λ). Let r denote the number of pairs in $\mathcal{S}_\lambda$. Note that **LP**(λ) has r variables, one for each pair $(i, j) \in \mathcal{S}_\lambda$. If x is a vertex solution, it satisfies r of the constraints of **LP**(λ) with equality. The first set of constraints consists of n constraints, and the second set of constraints consists of m constraints. Therefore at least $r - (m + n)$ of the tight constraints are from the third set of constraints, i.e., at least $r - (m + n)$ of the variables are set to zero.

We say that job i is set fractionally if $x_{ij} \in (0, 1)$ for some j; job i is set integrally if $x_{ij} \in \{0, 1\}$ for all j. Let I and F be the set of jobs that are set integrally and fractionally (respectively). Clearly, $|I| + |F| = n$. Any job i that is fractionally set is assigned (fractionally) to at least two machines, i.e., there exist $j \neq \ell$ such that $x_{ij} \in (0, 1)$ and $x_{i\ell} \in (0, 1)$. Therefore there are at least $2|F|$ distinct non-zero variables corresponding to jobs that are fractionally set. Additionally, for each job i that is integrally set, there is a variable x_{ij} that is non-zero. Thus the number of non-zero variables is at least $|I| + 2|F|$. Hence $|I| + |F| = n$ and $|I| + 2|F| \leq m + n$, which give us that $|I|$ is at least $n - m$. ■

Definition 6.3. *A connected graph is a **pseudo-tree** if the number of edges is at most the number of vertices. A graph is a **pseudo-forest** if each of its connected components is a pseudo-tree.*

Lemma 6.4. *The graph G is a pseudo-forest.*

Proof. Let C be a connected component of G. We restrict **LP**(λ) and x to the jobs and machines in C to get **LP**$'$(λ) and x'. Note that x' is a feasible solution to **LP**$'$(λ). Additionally, x' is a vertex solution to **LP**$'$(λ). If not, x' is a convex combination of two feasible solutions x'_1 and x'_2 to **LP**$'$(λ). We can extend x'_1 and x'_2 to two solutions x_1 and x_2 to **LP**(λ) using the entries of x that are not in x'. By construction, x_1 and x_2 are feasible solutions to **LP**(λ). Additionally, x is a convex combination of x_1 and x_2, which contradicts the fact that x is a vertex solution. Thus x' is a vertex solution to **LP**$'$(λ) and, by Lemma 6.3, x' has at most $n' + m'$ non-zero variables, where n' and m' are the number of jobs and machines in C. Thus C has $n' + m'$ vertices and at most $n' + m'$ edges, and therefore it is a pseudo-tree. ■

Proof. of Lemma 6.2 Note that each job that is integrally set has degree one in G. We remove each integrally set job from G; note that the resulting graph is H. Since we removed an equal number of vertices and edges from G, it follows that H is a pseudo-forest as well. Now we construct a matching $\mathcal{M}$ as follows.

Note that every job vertex has degree at least 2, since the job is fractionally assigned to at least two machines. Thus all of the leaves (degree-one vertices) of H are machines. While H has at least one leaf, we add the edge incident to the leaf to the matching and we remove both of its endpoints from the graph. If H does not have any leaves, H is a collection of vertex-disjoint cycles, since it is a pseudo-forest. Moreover, each cycle has even length, since H is bipartite. We construct a perfect matching for each cycle (by taking alternate edges), and we add it to our matching. ■

Exercise 6.2. (Exercise 17.1 in [152]) Give a proof of Lemma 6.2 using Hall's theorem.

6.2 Generalized Assignment Problem

The Generalized Assignment problem is a generalization of the Scheduling on Unrelated Parallel Machines problem in which there are costs associated with each job-machine pair, in addition to a processing time. More precisely, we have a set J of n jobs, a set M of m machines, and a target λ. The processing time of job i is p_{ij} on machine j, and the cost of assigning job i to machine j is c_{ij}. Let $f : J \rightarrow M$ be a function that assigns each job to exactly one machine. The assignment f is *feasible* if its makespan is at most λ (recall that λ is part of the input), and its cost is $\sum_i c_{if(i)}$. In the Generalized Assignment problem, the goal is to construct a minimum cost assignment f that is *feasible*, provided that there is a feasible assignment.

Remark 6.1. We could allow each machine M_j to have a different capacity c_j which is more natural in certain settings. However, since p_{ij} values are allowed to depend on j we can scale them to ensure that $c_j = \lambda$ for every j without loss of generality.

In the following, we will show that, if there is an assignment of cost C and makespan at most λ, then we can construct a schedule of cost at most C and makespan at most 2λ. In fact the assignment will have a stronger property that the load on a a machine exceeds λ due to at most one job.

As before, we let $\mathcal{S}_\lambda$ denote the set of all pairs (i, j) such that $p_{ij} \leq \lambda$. We can generalize the relaxation $\mathbf{LP}(\lambda)$ from the previous section to the following LP.

GAP-LP

$$\begin{aligned}
\text{min} \quad & \sum_{(i,j)\in\mathcal{S}_\lambda} x_{ij}c_{ij} & \\
\text{subject to} \quad & \sum_{j:\,(i,j)\in\mathcal{S}_\lambda} x_{ij} = 1 & \forall i \in J \\
& \sum_{i:\,(i,j)\in\mathcal{S}_\lambda} x_{ij}p_{ij} \leq \lambda & \forall j \in M \\
& x_{ij} \geq 0 & \forall (i,j) \in \mathcal{S}_\lambda
\end{aligned}$$

Since we also need to preserve the costs, we can no longer use the previous rounding; in fact, it is easy to see that the previous rounding is arbitrarily bad for the Generalized Assignment problem. However, we will still look for a matching, but in a slightly different graph.

But before we give the rounding algorithm for the Generalized Assignment problem, we take a small detour into the problem of finding a minimum-cost matching in a bipartite graph. In the Minimum Cost Biparite Matching problem, we are given a bipartite graph $B = (V_1 \cup V_2, E)$ with costs c_e on the edges, and we want to construct a minimum cost matching $\mathcal{M}$ that matches every vertex in V_1, if there is such a matching. For each vertex v, let $\delta(v)$ be the set of all edges incident to v. We can write the following LP for the problem.

BipartiteMatching(B)

$$\begin{aligned}
\text{min} \quad & \sum_{e\in E(B)} c_e y_e & \\
\text{subject to} \quad & \sum_{e\in\delta(v)} y_e = 1 & \forall v \in V_1 \\
& \sum_{e\in\delta(v)} y_e \leq 1 & \forall v \in V_2 \\
& y_e \geq 0 & \forall e \in E(B)
\end{aligned}$$

The following is well-known in combinatorial optimization [137].

Theorem 6.4. *For any bipartite graph B, any vertex solution to **BipartiteMatching**(B) is an integer solution. Moreover, given a feasible fractional solution y, we can find in polynomial time a feasible solution z such that z is integral and*

$$\sum_{e\in E(B)} c_e z_e \leq \sum_{e\in E(B)} c_e y_e.$$

In the rest of the section we give two different proofs that establish our claimed result. One is based on the first work that gave this result [140], and the other is based on iterative rounding [110].

6.2.1 Shmoys-Tardos Rounding

Let x be an optimal vertex solution to **GAP-LP**. As before, we want to construct a graph G that has a matching $\mathcal{M}$ that matches all jobs. The graph G will now have costs on its edges and we want a matching of cost at most C. Recall that for Scheduling on Unrelated Parallel Machines we defined a bipartite graph on the vertex set $J \cup M$ that has an edge ij for every variable x_{ij} that is non-zero. We can construct the same graph for Generalized Assignment, and we can assign a cost c_{ij} to each edge ij. If the solution x was actually a fractional matching — that is, if x was a feasible solution to **BipartiteMatching**(G) — Theorem 6.4 would give us the desired matching. The solution x satisfies the constraints corresponding to vertices $v \in J$, but it does not necessarily satisfy the constraints corresponding vertices $v \in M$, since a machine can be assigned more than one job. To get around this difficulty, we will introduce several nodes representing the same machine, and we will use x to construct a fractional matching for the resulting graph.

The fractional solution x assigns $\sum_{i \in J} x_{ij}$ jobs to machine j; let $k_j = \lceil \sum_{i \in J} x_{ij} \rceil$. We construct a bipartite graph G as follows. For each job i, we have a node i. For each machine j, we have k_j nodes $(j, 1), \cdots, (j, k_j)$. We can think of the nodes $(j, 1), \cdots, (j, k_j)$ as *slots* on machine j. Since now we have multiple slots on each of the machines, we need a fractional assignment y that assigns a job to slots on the machines. More precisely, y has an entry $y_{i,(j,s)}$ for each job i and each slot (j, s) that represents the fraction of job i that is assigned to the slot. We give the algorithm that constructs y from x below. Once we have the solution y, we add an edge between any job i and any machine slot (j, s) such that $y_{i,(j,s)}$ is non-zero. Additionally, we assign a cost $c_{i,(j,s)}$ to each edge $(i, (j, s))$ of G that is equal to c_{ij}.

```
GreedyPacking($x$)
    $y = 0$ ⟨⟨initialize $y$ to 0⟩⟩
    $s = 1$ ⟨⟨$s$ is the current bin⟩⟩
    $R = 1$ ⟨⟨$R$ is the space available on bin $s$⟩⟩
    for $i = 1$ to $h$
    ⟨⟨pack $x_{ij}$ into the bins⟩⟩
    if $x_{ij} \leq R$
        $y_{i,(j,s)} = x_{ij}$
        $R = R - x_{ij}$
        if $R = 0$
            $s = s + 1$
            $R = 1$
    else
        $y_{i,(j,s)} = R$
        $y_{i,(j,s+1)} = x_{ij} - R$ ⟨⟨pack $x_{ij} - R$ in the next bin⟩⟩
        $R = 1 - y_{i,(j,s+1)}$
        $s = s + 1$
    return $y$
```

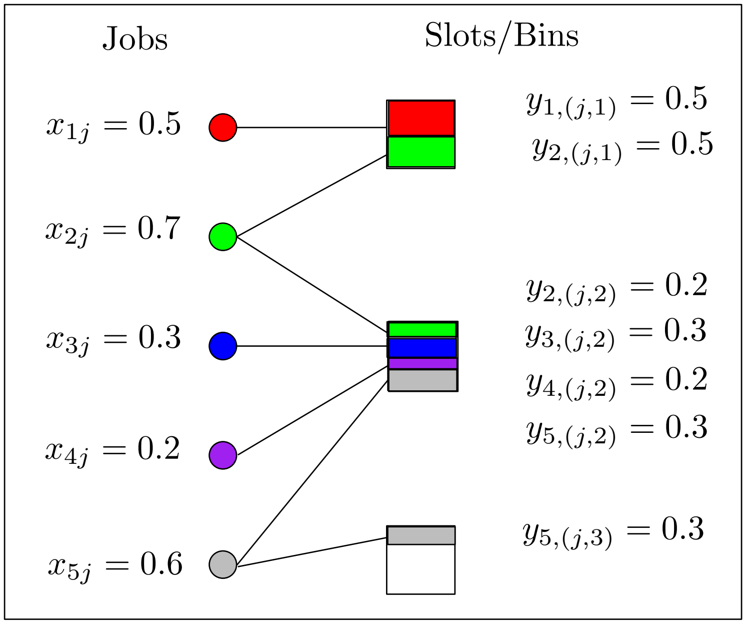


Figure 6.1: Constructing y from x.

When we construct y, we consider each machine in turn. Let j be the current machine. Recall that we want to ensure that y assigns at most one job to each slot; as such, we will think of each slot on machine j as a bin with capacity 1. We "pack" jobs into the bins greedily. We only consider jobs i such that p_{ij} is at most λ; let h denote the number of such jobs. We assume without loss of generality that these are labeled as $1, 2, \cdots, h$, and $p_{1j} \geq p_{2j} \geq \cdots \geq p_{hj}$. Informally, when we construct y, we consider the jobs $1, 2, \cdots, h$ in this order. Additionally, we keep track of the bin that has not been filled and the amount of space s available on that bin. When we consider job i, we try to pack x_{ij} into the current bin: if there is at least x_{ij} space available, i.e., $x_{ij} \leq s$, we pack the entire amount into the current bin; otherwise, we pack as much as we can into the current bin, and we pack the rest into the next bin. (See Figure 1 for an example.)

Lemma 6.5. *The solution y constructed by* GreedyPacking *is a feasible solution to* BipartiteMatching(G). *Moreover,*

$$\sum_{(i,(j,s))\in E(G)} y_{i,(j,s)} c_{i,(j,s)} = \sum_{(i,j)\in \mathcal{S}_\lambda} x_{ij} c_{ij}.$$

Proof. Note that, by construction, $x_{ij} = \sum_{s=1}^{k_j} y_{i,(j,s)}$. Therefore, for any job i, we have

$$\sum_{(i,(j,s))\in \delta(i)} y_{i,(j,s)} = \sum_{j:\,(i,j)\in \mathcal{S}_\lambda} \sum_{s=1}^{k_j} y_{i,(j,s)} = \sum_{j:\,(i,j)\in \mathcal{S}_\lambda} x_{ij} = 1$$

Additionally, since we imposed a capacity of 1 on the bins associated with each slot, it follows that, for any slot (j, s),

$$\sum_{(i,(j,s)\in \delta((j,s)))} y_{i,(j,s)} \leq 1$$

Therefore y is a feasible solution to BipartiteMatching(G). Finally,

$$\sum_{(i,(j,s))\in E(G)} y_{i,(j,s)} c_{i,(j,s)} = \sum_{i=1}^{n} \sum_{j:\,(i,j)\in \mathcal{S}_\lambda} \sum_{s=1}^{k_j} y_{i,(j,s)} c_{ij} = \sum_{(i,j)\in \mathcal{S}_\lambda} x_{ij} c_{ij}$$

■

Theorem 6.4 gives us the following corollary.

Corollary 6.5. *The graph G has a matching $\mathcal{M}$ that matches every job and it has cost at most $\sum_{(i,j)\in \mathcal{S}_\lambda} x_{ij} c_{ij}$. Moreover, we can find such a matching in polynomial time.*

```
GAP-Rounding
  let $x$ be an optimal solution to GAP-LP
  $y$ = GreedyPacking($x$)
  construct the graph $G$
  construct a matching $\mathcal{M}$ in $G$ such that $\mathcal{M}$ matches every job
      and the cost of $\mathcal{M}$ is at most $\sum_{(i,j)\in\mathcal{S}_\lambda} x_{ij}c_{ij}$
  for each edge $(i,(j,s)) \in \mathcal{M}$
      assign job $i$ to machine $j$
```

Theorem 6.6. *Let $C = \sum_{(i,j)\in\mathcal{S}_\lambda} x_{ij}c_{ij}$. The schedule returned by **GAP-Rounding** has cost at most C and makespan at most 2λ.*

Proof. By Corollary 6.5, the cost of the schedule is at most C. Therefore we only need to upper bound the makespan of the schedule.

Consider a machine j. For any slot (j,s) on machine j, let

$$q_{js} = \max_{i: y_{i,(j,s)}>0} p_{ij}$$

That is, q_{js} is the maximum processing time of any pair ij such that job i is assigned (in y) to the slot (j,s). It follows that the total processing time of the jobs that $\mathcal{M}$ assigns to machine j is at most $\sum_{s=1}^{k_j} q_{js}$.

Since **GAP-LP** has a variable x_{ij} only for pairs (i,j) such that p_{ij} is at most λ, it follows that q_{j1} is at most λ. We restrict attention to the case when at least two slots are assigned to j, for otherwise it is easy to see that the load is atmost λ. Therefore we only need to show that $\sum_{s=2}^{k_j} q_{js}$ is at most λ as well. Consider a slot s on machine j such that $s > 1$. Recall that we labeled the jobs that are relevant to machine j — that is, jobs i such that p_{ij} is at most λ — as $1, 2, \cdots, h$ such that $p_{1j} \geq p_{2j} \geq \cdots \geq p_{hj}$. Consider a job ℓ that is assigned to slot s. Since GreedyPacking considers jobs in non-increasing order according to their processing times, the processing time $p_{\ell j}$ of job ℓ is at most the processing time of any job assigned to the slot $s-1$. Therefore $p_{\ell j}$ is upper bounded by any convex combination of the processing times of the jobs that are assigned to the slot $s-1$. Since the slot $s-1$ is full, $\sum_i y_{i,(j,s-1)} = 1$ and thus $p_{\ell j}$ is at most $\sum_i y_{i,(j,s-1)}p_{ij}$. It follows that

$$\sum_{s=2}^{k_j} q_{js} \leq \sum_{s=2}^{k_j} \sum_i y_{i,(j,s-1)}p_{ij} \leq \sum_{s=1}^{k_j} \sum_i y_{i,(j,s)}p_{ij}$$

By construction, $\sum_s y_{i,(j,s)} = x_{ij}$, and therefore

$$\sum_{s=1}^{k_j} \sum_i y_{i,(j,s)}p_{ij} = \sum_i p_{ij} \sum_{s=1}^{s} y_{i,(j,s)} = \sum_i p_{ij}x_{ij}$$

Since x is a feasible solution to the **GAP-LP**,

$$\sum_{s=2}^{k_j} q_{js} \leq \sum_i p_{ij} x_{ij} \leq \lambda$$

which completes the proof. ■

6.2.2 Iterative Rounding

Here we describe an alternative proof/algorithm that illustrates the iterative rounding framework that initially came out of Jain's seminal work [90]. Since then it has become a powerful technique in exact and approximation algorithms. We need some additional formalism to describe the algorithm. We will consider the input instance as being specified by a graph $G = (J \cup M, E)$ where an edge $ij \in E$ implies that i is allowed to be schedule on j and has size p_{ij}. It is also important to consider non-uniform capacities on the machines to allow for a recursive (or iterative) algorithm. Thus we will assume that each machine M_j has a capacity b_j. We will assume that $p_{ij} \leq b_j$ for all $ij \in E$; in fact this assumption will not be needed until the very end when we analyze the approximation ratio. It is easy to generalize the LP relaxation for **GAP-LP** to handle non-uniform capacities and to handle the constraints specified by G. We will use the notation $\delta(i)$ and $\delta(j)$ to denote the edges incident to job i and machine j respectively.

GAP-LP

$$\begin{array}{rlr}
\min & \displaystyle\sum_{(i,j)\in E} c_{ij} x_{ij} & \\
\text{subject to} & \displaystyle\sum_{j:\,(i,j)\in\delta(i)} x_{ij} = 1 & \forall i \in J \\
 & \displaystyle\sum_{i:\,(i,j)\in E} p_{ij} x_{ij} \leq b_j & \forall j \in M \\
 & x_{ij} \geq 0 & \forall (i,j) \in E
\end{array}$$

To explain the underlying intuition for iterated rounding approach in the specific context of **GAP**, consider the situation where each machine j has infinite capacity. In this case it is easy to find the minimum cost assignment. We simply assign each job i to the machine $j = \arg\min_{j'\in\delta(i)} c_{ij'}$ which is the cheapest one that it is allowed to be assigned to. We also observe that if we drop all the capacity constraints from **GAP-LP**, and only leave the assignment constraints ($\sum_j x_{ij} = 1$ for each i), then the optimum solution of this LP is the same as the one obtained by assigning each job to its cheapest allowed machine (one can also argue that the LP is an integer polytope). Now consider another scenario. Suppose each machine j has in-degree at most k in G — that is, there are only k jobs that can ever be assigned to any machine j. Now suppose we assign each job to its cheapest allowed machine. Clearly the cost is at most the optimum cost of any feasible solution. But what about the load? Since each machine had in-degree at most k we will load a machine j to at most kb_j. Thus, if $k = 2$ we will only violate the machine's load by a factor of 2. However, this seems to be very restrictive assumption. Now consider a less restrictive scenario where there is one machine j such that its in-degree is at most 2. Then, in the LP relaxation, we can omit the constraint that limits its load since we are guaranteed that at most 2 jobs can be assigned to it (note that we still have the job assignment constraints which only allow a job to be assigned to machines according to the edges of G). Omitting constraints in an iterative fashion by taking advantage of sparsity in the basic feasible solution is the key idea.

To allow dropping of constraints we need some notation. Given an instance of GAP specified by $G = (J \cup M, E)$ and $M' \subseteq M$, we let $GAPLP(G, M')$ denote the LP relaxation for GAP where we only impose the load constraints for machines in M'. In other words we drop the load constraints for $M \setminus M'$. Note that jobs are still allowed to be assigned to machines in $M \setminus M'$.

The key structural lemma that allows for iterated rounding is the following.

Lemma 6.6. *Let y be a basic feasible solution to $GAPLP(G, M')$. Then one of the following properties holds:*

1. *There is some $ij \in E$ such $y_{ij} = 0$ or $y_{ij} = 1$.*
2. *There is a machine $j \in M'$ such that $d(j) \leq 1$.*
3. *There is a machine $j \in M'$ such that $deg(j) = 2$ and $\sum_{ij} y_{ij} \geq 1$.*

Proof. Let y be a basic feasible solution. If $y_{ij} = 0$ or $y_{ij} = 1$ for some edge $ij \in E$ we are done. Similarly if there is a machine $j \in M'$ with $d(j) \leq 1$ we are done. Thus we restrict our attention to the case when y_{ij} is strictly fractional for *every* edge $ij \in E$ and $d(j) > 1$ for each machine $j \in M'$. Note that $deg(i) \geq 2$ for each job J; otherwise $deg(i) = 1$ and in that case $y_{ij} = 1$ for the lone edge ij incident to i. We will prove that the third property holds.

$GAPLP(G, M')$ has $n + m'$ non-trivial constraints where $n = |J|$ and $m' = |M'|$. Since y is a basic feasible solution, via the rank lemma, this implies that $|E| = n + m'$. But degree of every i and every $j \in M'$ is at least 2. This implies that $deg(i) = 2$ for every $i \in J$ and $deg(j) = 2$ for every $j \in M'$ and $deg(j) = 0$ for every $j \in M \setminus M'$. Thus G consists of a collection of disjoint even cycles. Let S be any such cycle. For every job $i \in S$ we have sum of $\sum_j y_{ij} = 1$; hence

$\sum_{ij \in S} y_{ij} = |S|/2$. Hence there is some machine $j \in S$ such that $\sum_{ij} y_{ij} \geq 1$ and moreover its degree is exacly two as we argued. ■

```
GAP-Iter-Rounding($G$)

1. $F = \emptyset$, $M' = M$
2. While $(|F| < n)$ do
    A. Obtain an optimum basic feasible solution $y$ to $GAPLP(G, M')$
    B. If there is $ij \in E$ such that $y_{ij} = 0$ then $G = G - ij$
    C. Else If there is $ij \in E$ such that $y_{ij} = 1$ then
           $F = F \cup \{(ij)\}$, $G = G - i$, $b_j = b_j - p_{ij}$
    D. Else If there $j \in M'$ such that $d(j) = 1$ or $d(j) = 2$ and $\sum_i y_{ij} \geq 1$ then
           $M' = M' - j$.
3. Output assignment $F$
```

Theorem 6.7. *Given an instance of* **GAP** *that is feasible and has optimum cost C, the algorithm* GAP-Iter-Rounding *outputs an assignment whose cost is at most C and such that each machine j has load at most $2b_j$.*

The proof is by induction on the number of iterations. Alternatively, it is useful to view the algorithm recursively. We will sketch the proof and leave some of the formal details to the reader (who can also consult [110]). We observe that the algorithm makes progress in each iteration via Lemma 6.6. The analysis will consider the four cases that can happen in each iteration: (i) $y_{ij} = 0$ for some $ij \in E$ (ii) $y_{ij} = 1$ for some $ij \in E$ (iii) $d(j) \leq 1$ for some $j \in M'$ and (iv) $d(j) = 2$ and $\sum_{ij} y_{ij} \geq 1$ for some $j \in M'$.

Thus the algorithm terminates in polynomial number of iterations. It is also not hard to see that F corresponds to an assignment of jobs to machines.

Observation 6.8. *The algorithm terminates and outputs an assignment of jobs to machines, and and job i is assigned to j implies $ij \in E$.*

Now we prove that the assignment has good properties in terms of the cost and loads.

Lemma 6.7. *The cost of the LP solution at the start of each iteration is at most $C - \sum_{ij \in F} c_{ij}$. Hence, at the end of the algorithm the cost of the assignment F is at most C.*

Proof. This is true in the first iteration since $F = \emptyset$ and the LP cost is less than that of an optimum integer feasible solution. Now consider an iteration assuming that the precondition holds.

If $y_{ij} = 0$ we remove ij from E and we note that the cost of the LP for the next iteration does not increase since y itself is feasible for the residual instance.

If $y_{ij} = 1$ and we add ij to F, we can charge the cost of ij to what the LP has already paid on the edge ij, and the solution y with ij removed is feasible to the residual instance obtained by removing job i and reducing the capacity of j to $b_j - p_{ij}$.

In the other cases we do not change F but drop constraints so the LP cost can only decrease in the subsequent iteration. ■

Now we upper bound the load on each machine j.

Lemma 6.8. *For each machine j, $\sum_{ij \in F} p_{ij} \leq 2b_j$. In fact, a stronger property holds: for each j, its load at the end of the algorithm is at most b_j or there is a single job assigned to j such that removing it reduces the load of j to at most b_j.*

Proof. The proof is by induction on iterations. We will sketch it. Consider a machine j. If $y_{ij} = 0$ in some iteration we remove ij and the load on any machine does not change. If $y_{ij} = 1$ we add ij to F but for subsequent iterations we reduce b_j by p_{ij} hence we account for the increase in load of j.

Thus, the only reason why the load of j may exceed b_j is because we drop the load constraint for j in some iteration. If we drop it when $d(j) = 1$, then at most one more job can be assigned to j and hence its final load can be at most $b_j + p_{ij}$ for some $ij \in E$. Thus, if $p_{ij} \leq b_j$ for all i the load is at most $2b_j$. We can also drop the constraint for j when $d(j) = 2$. However, in this case we have the property that $y_{i_a,j} + y_{i_b,j} \geq 1$ for some two jobs i_a and i_b which are the only edges incident to j in that iteration. Since y was feasible, we also had the constraint that $p_{i_a j} y_{i_a j} + p_{i_b j} y_{i_b j} \leq b'_j$ where b'_j was the residual capacity of j in that iteration. Assume without loss of generality that $p_{i_a j} \leq p_{i_b j}$; then it follows that $b'_j \geq p_{i_a j}$. Thus the final load of j is at most $b_j - b'_j + p_{i_a j} + p_{i_b j}$ since both i_a and i_b can be assigned to j. But this load is at most $b_j + p_{i_b j} \leq 2b_j$. We leave it to the reader to verify the refined property regarding the load claimed in the lemma. ■

Running time: The algorithm runs in polynomial number of iterations, and in each iteration it requires an optimum basic feasible solution to the $GAPLP(G, M')$. This can be done in polynomial time. We remark that the algorithm is deliberately described in a simple iterative fashion to make the proof easier. One can speed up the algorithm by considering all the cases together in each iteration. Although iterated rounding is a powerful technique, the running time is typically expensive.

Finding faster algorithms that achieve similar guarantees is an open area of research.

6.3 Maximization version of GAP

We consider the maximization version which we refer to **Max-GAP**. We have n items and m bins (instead of jobs and machines) where p_{ij} is the size of item i in bin j. Each bin j has a capacity c_j and assigning item i to bin j yields a profit/weight w_{ij}. The goal is to assign items to bins to maximize the weight while not violating any capacities. When $m = 1$ we obtain the **Knapsack** problem.

Multiple Knapsack Problem (MKP): MKP is a special case of **Max-GAP** in which $w_{ij} = w_i$ for all i, j and $p_{ij} = p_i$ for all i, j. In other words the item characteristics do not depend on where it is assigned to.

Exercise 6.3. Prove that MKP does not admit an FPTAS even for $m = 2$.

MKP admits a PTAS [38] and even an EPTAS [94]. Simply greedy algorithms that pack bins one by one using an algorithm for **Knapsack** as a black box yield a $(1 - 1/e - \epsilon)$ and $1/2 - \epsilon$ approximation for MKP when the bins are indentical and when the bins are arbitrary [38].

In contrast to MKP, **Max-GAP** does not admit a PTAS. There is an absolute constant $c > 1$ such that a $c - \epsilon$ approximation implies $P = NP$ [38]. However, the following is known.

Theorem 6.9. *For every fixed m there is a PTAS for **Max-GAP**.*

The preceding theorem can be shown by generalizing the ideas behind the PTAS for **Knapsack** we discussed in an earlier chapter. An interested reader may try to prove this by considering the case of $m = 2$.

A $\frac{1}{2}$-approximation: There is a simple yet clever way to achieve a $\frac{1}{2}$-approximation for **Max-GAP** via the 2-approximation for the min-cost version that we already saw. Recall that for the min-cost version the algorithm output a solution with cost no more than the optimum cost while violating the capacity of each bin by at most one item. We outline how one can use this to obtain a 2-approximation for **Max-GAP** and leave the details are an exercise to the reader.

- Reduce the exact maximization version to the exact min-cost version in which all items have to be assigned by adding an extra dummy bin.

- Use the result for min-cost version to obtain an assignment with weight at least that of the optimum while violating each bin's capacity by at most one item.

- Use the preceding assignment to find a feasible packing of items that has profit at least OPT/2.

For **Max-GAP** one can use a stronger LP relaxation and obtain a $(1 - 1/e + \delta)$-approximation. We refer the reader to [58] for this result, and also to [28] for connections to submodular function maximization. The latter connection allows one to obtain an extremely simple $1/2 - \epsilon$ greedy approximation algorithm that is not obvious to discover.

6.4 Bibilographic Notes

The 2-approximation for unrelated machine scheduling is by Lenstra, Shmoys and Tardos [115]. The same paper showed that unless $P = NP$ there is no $3/2 - \epsilon$-approximation for unrelated machine scheduling. Bridging this gap has been a major open problem in scheduling. A special case called restricted assignment problem has been extensively studied in this context; in such instances $p_{ij} \in \{p_i, \infty\}$ which means that a job specifies the machines it can be assigned to, but its processing time does not vary among the machines it can be assigned to. The 3/2 hardness from [115] applies to restricted assignement problem as well. Svensson [145] showed that a strengthend form of LP relaxation (called the configuration LP) has an integrality gap better than 2 for restricted assignment problem but so far there is no polynomial time algorithm to actually output an assignment! The best algorithm that beats 2 for this case runs in quasi-polynomial time [95].

As we mentioned Shmoys and Tardos obtained the 2-approximation for **GAP**. The iterated rounding proof is from [110].

The approximability of **Max-GAP** when m is not a fixed constant was studied in [38] although a PTAS for fixed m was known quite early [33]. The current best approximation ratio is via a configuration LP [58]. The precise integrality gap of the configuration LP is an interesting open problem.

Chapter 7

Congestion Minimization in Networks

In Chapter 6 we saw the Scheduling on Unrelated Parallel Machines problem. Here we consider two problems that also consider allocation with the objective of minimizing load/congestion. We will first consider the Congestion Minimization problem in graphs and then the abstract problem of Min-Max-Integer-Programs.

7.1 Congestion Minimization and VLSI Routing

A classical routing problem that was partly inspired by VLSI (very large scale integrated) circuit design is the following. Let $G = (V, E)$ be a directed graph and let $(s_1, t_1), \ldots, (s_k, t_k)$ be k source-sink pairs. We want to connect each pair (s_i, t_i) by a path P_i such that the paths do not share edges (or nodes). Alternatively we would like to minimize the maximum number of paths that use any edge — this is called the Congestion Minimization problem. A special case is EDP which is to decide if there are paths for the pairs that are edge-disjoint (NDP is the version where the paths are required to be node-disjoint). EDP and NDP are classical NP-Complete problems and have many impoartnat connections to multicommodity flows, routing, cuts, and graph theory. Thus Congestion Minimization is an NP-Hard optimization problem. Here we will consider two variants.

Choosing one path from a given collection: We consider a conceptually simpler variant where we are given a finite path collection $\mathcal{P}_i$ for each pair (s_i, t_i) where each path $P \in \mathcal{P}_i$ connects s_i to t_i. The goal is to choose, for each pair (s_i, t_i), one path from $\mathcal{P}_i$ so as to minimize the maximum congestion on any edge. We can develop an integer programming formulation as follows. For each i and each $P \in \mathcal{P}_i$ we have a variable $x_{i,P}$ which indicates whether we choose

P to route pair i. The constraints express that exactly one path is for each pair i. To minimize the maximum number of paths using any edge we introduce a variable λ and minimize it subject it to a natural packing constraint.

$$\begin{array}{lll}
\text{minimize} & \lambda & \\
\text{subject to} & \displaystyle\sum_{P \in \mathcal{P}_i} x_{i,P} = 1 & 1 \le i \le k \\
& \displaystyle\sum_{i=1}^{k} \sum_{P \in \mathcal{P}_i, P \ni e} x_{i,P} \le \lambda & \forall e \in E \\
& x_{i,P} \in \{0,1\} & 1 \le i \le k, P \in \mathcal{P}_i
\end{array}$$

As usual we relax the integer constraints to obtain an LP relaxation where we replace $x_{i,P} \in \{0,1\}$ with $x_{i,P} \in [0,1]$ (in this particular case we can simply use $x_{i,P} \ge 0$ due to the other constraints). Note the similarities with the IP/LP for Scheduling on Unrelated Parallel Machines. The LP relaxation is of size polynomial in the input size since the path collection $\mathcal{P}_i$ is explicitly given for each i as part of the input.

Let λ^* be the optimum LP solution value. There is a technicality that arises just as we saw with Scheduling on Unrelated Parallel Machines. It may happen that $\lambda * < 1$ while we know that the optimum congestion is at least 1. Technically we should find the smallest integer λ^* such that the preceding LP is feasible. We will assume henceforth that λ^* is an integer.

How do we round? A simple stragegy is randomized rounding. In fact the technique of randomized rounding and analysis via Chernoff bounds was developed in the influential paper of Raghavan and Thompson [133] precisely for this problem!

```
Randomized-Rounding

1. Solve LP relaxation and find optimum solution $x^*, \lambda^*$.

2. For $i = 1$ to $k$ do

    A. Pick exactly one path $Q_i \in \mathcal{P}_i$ randomly where the probability of
       picking $P$ is exactly $x^*_{i,P}$.

3. Output $Q_1, Q_2, \ldots, Q_k$.
```

Note that the choices for the pairs done with independent randomness.

The analysis requires the use of Chernoff-Hoeffding bounds. See Chapter B.

Theorem 7.1. *Randomized rounding outputs one path per pair and with probability at least* $(1 - 1/m^2)$ *no edge is contained in more than* $c\frac{\log m}{\log\log m} \cdot \lambda^*$ *paths where* c *is an absolute constant. Here* m *are the number of edges in the graph* G. *One can also show that for any fixed* $\epsilon > 0$ *the congestion is at most* $(1+\epsilon)\lambda^* + c\frac{\log m}{\epsilon^2}$ *with high probability.*

Proof. The proof is a simple application of Chernoff-bounds and the union bound. Fix an edge $e \in E$. Let Y_e be the random variable which is the total number of paths in $Q_1, Q_2, \ldots, Q_k$ that use e. Let $Y_{e,i}$ be the binary random variable that indicates whether $e \in Q_i$. Note that $Y_e = \sum_{i=1}^k Y_{e,i}$.

The first observation is that the variables $Y_{e,1}, Y_{e,2}, \ldots, Y_{e,k}$ are independent since we used independent randomness for the pairs. Second we claim that $\mathbf{E}[Y_{e,i}] = \mathbf{P}[Y_{e,i} = 1] = \sum_{P \in \mathcal{P}_i, e \in P} x^*_{i,P}$. Do you see why? Thus, by linearity of expectation,

$$\mathbf{E}[Y_e] = \sum_i \mathbf{E}[Y_{e,i}] = \sum_i \sum_{P \in \mathcal{P}_i, e \in P} x^*_{i,P} \leq \lambda^*$$

where the last inequality follows from the LP constraint.

Since Y_e is the sum of independent binary valued random variables and $\mathbf{E}[Y_e] \leq \lambda^*$ we can apply Chernoff-bounds to estimate $\mathbf{P}[Y_e \geq c\frac{\log m}{\log\log m}\lambda^*]$. Applying Corollary B.5 we conclude that we can choose c such that this probability is at most $1/m^3$. Now we apply the union bound over all edges and conclude that

$$\mathbf{P}[\exists e \in E, Y_e \geq c\frac{\log m}{\log\log m}\lambda^*] \leq m/m^3 \leq 1/m^2.$$

Thus, with probability $\geq 1 - 1/m^2$ no edge is loaded to more that $c\frac{\log m}{\log\log m}\lambda^*$.

The second bound can be derived in the same way by using the second Chernoff-bound in Corollary B.5. ■

Remark 7.1. We chose the bound $(1-1/m^2)$ for concreteness. A success probability of the form $(1 - 1/\text{poly}(n))$ where n is the input size is typically called "with high probability".

Remark 7.2. The bound $(1+\epsilon)\lambda^* + c\log m/\epsilon^2$ implies that when $\lambda^* \geq c\log m$ we obtain a constant factor approximation.

Remark 7.3. In a graph $m = O(n^2)$ and hence one often sees the bounds expressed in terms of n rather than m. We chose to write in terms of m rather than n to highlight the fact that the bound depends on the number of constraints via the union bound. We will discuss later how column sparsity based bounds can give refined results that avoid the union bound.

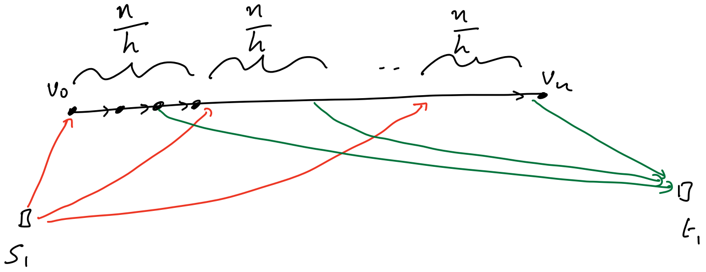
Implicitly given path collections: In the traditional version of Congestion Minimization we are only given G and the pairs, and the goal is to choose one path P for each (s_i, t_i) pair from the set of all paths between s_i and t_i. In other words $\mathcal{P}_i = \{P \mid P \text{ is an } s_i\text{-}t_i \text{ path in } G\}$. $\mathcal{P}_i$ is implicity defined and its size can be exponential in the graph size. It is not obvious that we can solve the LP relaxation that we saw above. However, one can indeed solve it in polynomial-time via the Ellipsoid method. First, we observe that the LP could have an exponential number of variables but only a polynomial number of non-trivial constraints: k for the pairs and m for the edges. Thus, one is guaranteed, by the rank lemma that there is an optimum solution that has only $k + m$ non-zero variables To see that one can indeed find it efficiently, we need to look at the dual and notice that the separation oracle for the dual is the shortest path problem.

Another way to see that the LP can solved is by writing a compact formulation via the well-known multicommodity flow. We want to send one unit of flow from s_i to t_i so that the total flow on any edge is at most λ. We use variables $f(e, i)$ to denote a flow for pair i on edge e.

$$\begin{array}{lll}
\text{minimize} & \lambda & \\
\text{subject to} & \displaystyle\sum_{e\in\delta^+(s_i)} f(e,i) - \sum_{e\in\delta^-(s_i)} f(e,i) = 1 & 1 \le i \le k \\
& \displaystyle\sum_{e\in\delta^+(v)} f(e,i) - \sum_{e\in\delta^-(v)} f(e,i) = 0 & 1 \le i \le k, v \in V - \{s_i, t_i\} \\
& \displaystyle\sum_{i=1}^{k} f(e,i) \le \lambda & \forall e \in E \\
& f(e,i) \ge 0 & 1 \le i \le k, e \in E
\end{array}$$

The preceding multicommodity flow LP has a polynomial number of variables and can be solved in polynomial-time. Given a flow, for each commodity/pair i we can take the one unit of s_i-t_i flow and use standard flow-decomposition to obtain a path-flow with at most m paths in the collection. We can then apply the rounding that we saw above with an given explicit path collection in exactly the same way.

Remark 7.4. The Ellipsoid based algorithm may seem impractical. However, one can approximately solve the implicit path based LP via multiplicative weight update methods efficiently. The implicit formulation and Ellipsoid method is also useful when one may want to restrict $\mathcal{P}_i$ in some fashion. For instance we can set $\mathcal{P}_i$ to be the set of all s_i-t_i paths in G with at most d edges for some given parameter d. This will ensure that we choose only "short" paths for each pair. It



is not hard to see that the separation oracle for the dual is another shortest path type problem that can be solved efficiently (via Bellman-Ford type algorithm). This is not easy to capture/see via the compact flow based formulation.

Derandomization: Is there a deterministic algorithm with the roughly the same approximation guarantee? The algorithm can be derandomized via the notion of pessimistic estimators. Congestion Minimization was one of the first instances with a sophisticated use of this technique [132].

Integrality gap and Hardness of Approximation: There is simple yet clever example demonstrating that the integrality gap of the flow relaxation in directed graphs is $\Omega(\log m / \log \log m)$ [114]. In a remarkable result, [44] showed that $O(\log m / \log \log m)$ is the hardness factor. The complexity of Congestion Minimization is less clear in *undirected* graphs. It is known that the LP integrality gap and hardness of approximation are $\Omega(\log \log n / \log \log \log n)$ [8]. Closing the gap betten the upper and lower bounds is a major open problem.

Here we outline the integrality gap example for directed graphs from [114]. The graph G and the pairs are constructed in a recursive fashion. Let h be a parameter that we will fix later. We start with a directed path $v_0, v_1, \ldots, v_n$. We add a demand pair (s_1, t_1) which connects to the path as follows. We partition the path into n/h paths of equal length: add an arc to s to the start of each sub-path and an arc from the end of each sub-path to t. See figure.

One can see from the figure that the pair (s, t) can splits its flow along h paths. Now we consider each of the h sub-paths and recursively create an instance on the path with length $n/h - 1$ (while keeping parameter h the same). Note that in the second level of the recursion we add h new source-sink pairs, one for each sub-path. We stop the recursion when the size of the sub-path is $\Theta(h)$. Let d be the depth of the recursion.

We claim that there is a fractional routing of all demand pairs where the congestion is at most d/h. This follows by splitting the flow of the pairs h ways. The next claim is that some edge has congestion d in any integral routing. This can be seen inductively. The top level pair (s,t) has to choose one amongst the h sub-paths — all edges in that sub-path will be used by the route for (s,t). Inductively there is some edge in that sub-path with congestion $d-1$ and hence the congestion of that edge will d when we add the path for (s,t).

It now remains to set the parameters. If we choose $h = \log^2 n$ say then $d = \Theta(\log n / \log\log n)$. The fractional congestion is ≤ 1 and integrally congestion is $\Theta(\log n / \log\log n)$.

Short paths and improved congestion via Lovász-Local-Lemma: We consider the congestion minimization problem when the path for each pair is required to be "short". By this we mean that we are required to route on a path with at most d edges where d is some given parameter. One can imagine that in many applications d is small and is a fixed constant, say 10. The question is whether the approximation ratio can be improved. Indeed one can show that the LP integrality gap is $O(\log d / \log\log d)$. Thus, when $d \ll n$ we get a substantial improvement. However, proving this and obtaining a polynomial time algorithm are quite non-trivial. One requires the use of the subtle Lovász-Local-Lemma (LLL), a powerful tool in probabilistic combinatorics. Typically LLL only gives a proof of existence and there was substantial work in making LLL constructive/efficient. Srinivasan obtained an algorithm via derandomization of LLL in this context with a lot of technical work [142]. There was a breakthrough work of Moser and Tardos [123] that gave an extremely simple way to make LLL constructive and this has been refined and developed over the last decade. For the congestion minimization problem we refer the reader to [75] which builds upon [123] and describes an efficient randomized algorithm that outputs a solution with congestion $O(\log d / \log\log d)$. In fact the application is given in the context of a more abstract problem that we discuss in the next section.

Integer flows and Unsplittable flows: We worked with the simple setting where each pair (s_i, t_i) wishes to send one unit of flow. One can imagine a situation where one wants to send d_i units of flow for pair i where d_i is some (integer) demand value. There are two interesting variants. The first one requires integer valued flow for each pair which means that we want to find d_i paths for (s_i, t_i) that each carry one unit of flow (the paths can overlap). This variant can be essentially reduced to the unit demand flow by creating d_i copies of (s_i, t_i) — we leave this as a simple exercise for the reader. The second variant is that we want each pair's flow of d_i units to be sent along a single path — this is called *unsplittable* flow. When discussing unsplittable flow it is also natural to consider capacities on the edges. Thus, each edge has a capacity u_e and one wants to minimize congestion relative to u_e. The techniques we discussed can be generalized relatively easily to this version as well to obtain the same kind of bounds. The unsplittable flow problem is interesting even in the setting where there is a single source/sink or when the graph is a simple ring or a path. Interesting results are known here and we refer the reader to [2, 30, 49, 70, 122, 141] for further pointers.

7.2 Min-max Integer Programs

If one looks at the rounding and analysis for Congestion Minimization we notice that the algorithm uses very little about the structure of the graph. This can be thought about in two ways. One is that perhaps we can do better by exploiting graph structure. Two, we can abstract the problem into a more general class where the same technique applies. As we mentioned, in directed graphs the bound of $O(\log n/\log\log n)$ is tight but the bound may not be tight in undirected graphs which admit more structure.

Here we consider the second point and develop a resource allocation view point while making an analogy to Congestion Minimization so that the abstract problem can be more easily understood. Suppose we have m resources $e_1, e_2, \ldots, e_m$. We have k requests $r_1, r_2, \ldots, r_k$. Each request i can be satisfied in ℓ_i ways — let $\mathcal{P}_i$ denote a collection of ℓ_i vectors $v_{i,1}, v_{i,2}, \ldots, v_{i,\ell_i}$. Each vector $v_{i,j} \in \mathcal{P}_i$ is an m-dimensions: for each $k \in [m]$, $v_{i,j,k}$ is a scalar that represents the load it induces on resource e_k. The goal is to choose, for each i, exactly one $j \in [\ell_i]$ so as to minimize the maximum load on any resource. One can write this conveniently as the following integer program where we have variables $x_{i,j}$ for $1 \le i \le k$ and $1 \le j \le \ell_i$ which indicates whether i chooses j.

$$\begin{array}{lll}
\text{minimize} & \lambda & \\
\text{subject to} & \displaystyle\sum_{1 \le j \le \ell_i} x_{i,j} = 1 & 1 \le i \le k \\
& \displaystyle\sum_{i=1}^{k} \sum_{1 \le j \le \ell_i} v_{i,j,k} x_{i,j} \le \lambda & \forall e_k \\
& x_{i,j} \in \{0,1\} & 1 \le i \le k, 1 \le j \le \ell_i
\end{array}$$

One can view the above integer program compactly as

$$
\begin{array}{lll}
\text{minimize} & \lambda & \\
\text{subject to} & \displaystyle\sum_{1 \le j \le \ell_i} x_{i,j} = 1 & 1 \le i \le k \\
& Ax \le \lambda \mathbb{1} & \\
& x_{i,j} \in \{0,1\} & 1 \le i \le k, 1 \le j \le \ell_i
\end{array}
$$

where A is a non-negative matrix with m rows. As with Scheduling on Unrelated Parallel Machines we need to be careful when relaxing the IP to an LP since the optimum solution to the LP can be a poor lower bound unless we ensure that $x_{i,j} = 0$ if $v_{i,j,k} > \lambda$. We will assume that we have indeed done this.

One can do randomized rounding exactly as we did for Congestion Minimization and obtain an $O(\log m / \log \log m)$ approximation. We say that A is d-column-sparse if the maximum number of non-zeroes in any column of A is at most d. This corresponds to paths in Congestion Minimization being allowed to have only d edges. One can obtain an $O(\log d / \log \log d)$-approximation in this more general setting as well [75].

Chapter 8

Introduction to Local Search

Local search is a powerful and widely used heuristic method (with various extensions). In this lecture we introduce this technique in the context of approximation algorithms. The basic outline of local search is as follows. For an instance I of a given problem let $\mathcal{S}(I)$ denote the set of feasible solutions for I. For a solution S we use the term *(local) neighborhood* of S to be the set of all solutions S' such that S' can be obtained from S via some *local moves*. We let $N(S)$ denote the neighborhood of S.

```
LocalSearch:
Find a “good” initial solution $S_0 \in \mathcal{S}(I)$
$S \leftarrow S_0$
repeat
  If ($\exists S' \in N(S)$ such that val($S'$) is strictly better than val($S$))
      $S \leftarrow S'$
  Else
      $S$ is a local optimum
      return $S$
  EndIf
Until (True)
```

For minimization problems S' is strictly better than S if val(S') $<$ val(S) whereas for maximization problems it is the case if val(S') $>$ val(S).

The running time of the generic local search algorithm depends on several factors. First, we need an algorithm that given a solution S either declares that S is a local optimum or finds a solution $S' \in N(S)$ such that val(S') is strictly better thatn val(S). A standard and easy approach for this is to ensure that the local moves are defined in such a way that $|N(S)|$ is polynomial in the input size $|I|$ and $N(S)$ can be enumerated efficiently; thus one can check each $S' \in N(S)$ to see if any of them is an improvement over S. However, in some more advanced settings, $N(S)$ may be exponential in the input size but one may be able to find a solution in $S' \in N(S)$ that improves on S in polynomial time. Second, the running time of the algorithm depends also on the number of iterations it takes to go from S_0 to a local optimum. In the worst case the number of iterations could be $|\mathrm{OPT} - \mathrm{val}(S_0)|$ which can be exponential in the input size. One can often use a standard scaling trick to overcome this issue; we stop the algorithm unless the improvement obtained over the current S is a significant fraction of $\mathrm{val}(S)$. Finally, the quality of the initial solution S_0 also factors into the running time.

Remark 8.1. There is a distinction made sometimes in the literature between *oblivious* and *non-oblivious* local search. In oblivious local search the algorithm uses f as a black box when comparing S with a candidate solution S'; the analysis and the definition of local moves are typically based on properties of f. In non-oblivious local search one may use an auxiliary function g derived from f in comparing S with S'. Typically the function g is some kind of *potential function* that aids the analysis. This allows one to move to a solution S' even though $f(S')$ may not be an improvement over $f(S)$.

Remark 8.2. There are several heuristics that are (loosely) connected to local search. These include simulated annealing, tabu search, genetic algorithms and others. These fall under the broad terminology of metaheuristics.

8.1 Local Search for Max Cut

We illustrate local search for the well-known Max Cut problem. In Max Cut we are given an undirected graph $G = (V, E)$ and the goal is to partition V into $(S, V \setminus S)$ so as to maximize the number of edges crossing S, that is, $|\delta_G(S)|$; we will use $\delta(S)$ when G is clear. For a vertex v we use $\delta(v)$ instead of $\delta(\{v\})$ to simplify notation. In the weighted version each edge e has a non-negative weight $w(e)$ the goal is to maximize the weight of the edges crossing S, that is, $w(\delta(S))$; here $w(A)$ for a set A denotes the quantity $\sum_{e \in A} w(e)$.

We consider a simple local search algorithm for Max Cut that starts with an arbitrary set $S \subseteq V$ and in each iteration either adds a vertex to S or removes a vertex from S as long as it improves the cut value $\delta(S)$.

```
LocalSearch for Max Cut:
$S \leftarrow \emptyset$
repeat
  If ($\exists v \in V \setminus S$ such that $w(\delta(S+v)) > w(\delta(S))$)
      $S \leftarrow S + v$
  Else If ($\exists v \in S$ such that $w(\delta(S-v)) > w(\delta(S))$)
      $S \leftarrow S - v$
  Else
      $S$ is a local optimum
      return $S$
  EndIf
Until (True)
```

We will first focus on the quality of solution output by the local search algorithm.

Lemma 8.1. *Let S be the output of the local search algorithm. Then for each vertex v, $w(\delta(S) \cap \delta(v)) \geq w(\delta(v))/2$.*

Proof. Let $\alpha_v = w(\delta(S) \cap \delta(v))$ be the weight of edges among those incident to v that cross the cut S. Let $\beta_v = w(\delta(v)) - \alpha_v$.

We claim that $\alpha_v \geq \beta_v$ for each v. If $v \in V \setminus S$ and $\alpha_v < \beta_v$ then moving v to S will strictly increase $w(\delta(S))$ and S cannot be a local optimum. Similarly if $v \in S$ and $\alpha_v < \beta_v$, we would have $w(\delta(S-v)) > w(\delta(S))$ and S would not be a local optimum. ■

Corollary 8.1. *If S is a local optimum then $w(\delta(S)) \geq w(E)/2 \geq \mathrm{OPT}/2$.*

Proof. Since each edge is incident to exactly two vertices we have $w(\delta(S)) = \frac{1}{2}\sum_{v \in V} w(\delta(S) \cap \delta(v))$. Apply the above lemma,

$$\begin{aligned} w(\delta(S)) &= \frac{1}{2}\sum_{v \in V} w(\delta(S) \cap \delta(v)) \\ &\geq \frac{1}{2}\sum_{v \in V} w(\delta(v))/2 \\ &\geq \frac{1}{2} w(E) \\ &\geq \frac{1}{2}\mathrm{OPT}, \end{aligned}$$

since $\mathrm{OPT} \leq w(E)$. ■

The running time of the local search algorithm depends on the number of local improvement iterations; checking whether there is a local move that results in an improvement can be done by trying all possible vertices. If the graph is unweighted then the algorithm terminates in at most $|E|$ iterations. However, in the weighted case, it is known that the algorithm can take an exponential time in $|V|$ when the weights are large. A very interesting problem is whether one can find a local optimum efficiently — note that we do not need to find a local optimum by doing local search! It is believed that finding a local optimum for Max Cut in the weighted case is hard. There is a complexity class called PLS that was defined by Johnson, Papadimitrioiu and Yannakakis for which Max Cut is known to be a complete problem. PLS plays an important role in algorithmic game theory in recent times. We refer the reader to the Wikipedia page on PLS for many pointers.

Many local search algorithms can be modified slightly to terminate with an approximate local optimum such that (i) the running time of the modified algorithm is strongly polynomial in the input size and (ii) the quality of the solution is very similar to that given by the original local search. We illustrate these ideas for Max Cut. Consider the following algorithm where $\epsilon > 0$ is a parameter that can be chosen. Let n be the number of nodes in G.

```
Modified LocalSearch for Max Cut($\epsilon$):
$S \leftarrow \{v^*\}$ where $v^* = \arg\max_{v \in V} w(\delta(v))$
repeat
  If ($\exists v \in V \setminus S$ such that $w(\delta(S+v)) > (1+\frac{\epsilon}{n})w(\delta(S))$)
      $S \leftarrow S + v$
  Else If ($\exists v \in S$ such that $w(\delta(S-v)) > (1+\frac{\epsilon}{n})w(\delta(S))$)
      $S \leftarrow S - v$
  Else
      return $S$
  EndIf
Until (True)
```

The above algorithm terminates unless the improvement is a relative factor of $(1+\frac{\epsilon}{n})$ over the current solution's value. Thus the final output S is an *approximate* local optimum.

Remark 8.3. An alert reader may wonder why the improvement is measured with respect to the global value $w(\delta(S))$ rather than with respect to $w(\delta(v))$. One reason is to illustrate the general idea when one may not have fine grained information about the function like we do here in the specific case of Max Cut. The global analysis will also play a role in the running time analysis as we will see shortly.

Lemma 8.2. *Let S be the output of the modified local search algorithm for* Max Cut*. Then $w(\delta(S)) \geq \frac{1}{2(1+\epsilon/4)} w(E)$.*

Proof. As before let $\alpha_v = w(\delta(S) \cap \delta(v))$ and $\beta_v = w(\delta(v)) - \alpha_v$. Since S is an approximately local optimum we claim that for each v

$$\beta_v - \alpha_v \leq \frac{\epsilon}{n} w(\delta(S)).$$

Otherwise a local move using v would improve S by more than $(1 + \epsilon/n)$ factor. (The formal proof is left as an exercise to the reader).

We have,

$$\begin{aligned} w(\delta(S)) &= \frac{1}{2}\sum_{v \in V} \alpha_v \\ &= \frac{1}{2}\sum_{v \in V}((\alpha_v + \beta_v) - (\beta_v - \alpha_v))/2 \\ &\geq \frac{1}{4}\sum_{v \in V}(w(\delta(v)) - \frac{\epsilon}{n} w(S)) \\ &\geq \frac{1}{2}w(E) - \frac{1}{4}\sum_{v \in V}\frac{\epsilon}{n}w(S) \\ &\geq \frac{1}{2}w(E) - \frac{1}{4}\epsilon \cdot w(S). \end{aligned}$$

Therefore $w(S)(1 + \epsilon/4) \geq w(E)/2$ and the lemma follows. ■

Now we argue about the number of iterations of the algorithm.

Lemma 8.3. *The modified local search algorithm terminates in $O(\frac{1}{\epsilon} n \log n)$ iterations of the improvement step.*

Proof. We observe that $w(S_0) = w(\delta(v^*)) \geq \frac{2}{n} w(E)$ (why?). Each local improvement iteration improves $w(\delta(S))$ by a multiplicative factor of $(1+\epsilon/n)$. Therefore if k is the number of iterations that the algorithm, then $(1+\epsilon/n)^k w(S_0) \leq w(\delta(S)$ where S is the final output. However, $w(\delta(S)) \leq w(E)$. Hence

$$(1 + \epsilon/n)^k 2w(E)/n \leq w(E)$$

which implies that $k = O(\frac{1}{\epsilon} n \log n)$. ■

A tight example for local optimum: Does the local search algorithm do better than 1/2? Here we show that a local optimum is no better than a 1/2-approximation. Consider a complete bipartite graph $K_{2n,2n}$ with $2n$ vertices in each part. If L and R are the parts of a set S where $|S \cap L| = n = |S \cap R|$ is a local optimum with $|\delta(S)| = |E|/2$. The optimum solution for this instance is $|E|$.

Max Directed Cut: A problem related to Max Cut is Max Directed Cut in which we are given a directed edge-weighted graph $G = (V, E)$ and the goal is to find a set $S \subseteq V$ that maximizes $w(\delta^+_G(S))$; that is, the weight of the directed edges leaving S. One can apply a similar local search as the one for Max Cut. However, the following example shows that the output S can be arbitrarily bad. Let $G = (V, E)$ be a directed in-star with center v and arcs connecting each of $v_1, \ldots, v_n$ to v. Then $S = \{v\}$ is a local optimum with $\delta^+(S) = \emptyset$ while OPT $= n$. However, a minor tweak to the algorithm gives a 1/3-approximation! Instead of returning the local optimum S return the better of S and $V \setminus S$. This step is needed because the directed cuts are not symmetric.

8.2 Local Search for Submodular Function Maximization

In this section we consider the utility of local search for maximizing non-negative submodular functions. Let $f : 2^V \to \mathbb{R}_+$ be a *non-negative* submodular set function on a ground set V. Recall that f is submodular if $f(A) + f(B) \geq f(A \cup B) + f(A \cap B)$ for all $A, B \subseteq V$. Equivalently f is submodular if $f(A + v) - f(A) \geq f(B + v) - f(B)$ for all $A \subset B$ and $v \notin B$. f is *monotone* if $f(A) \leq f(B)$ for all $A \subseteq B$. f is *symmetric* if $f(A) = f(V \setminus A)$ for all $A \subseteq V$. Submodular functions arise in a number of settings in combinatorial optimization. Two important examples are the following.

Example 8.1. Coverage in set systems. Let $S_1, S_2, \ldots, S_n$ be subsets of a set $\mathcal{U}$. Let $V = \{1, 2, \ldots, n\}$ and define $f : 2^V \to \mathbb{R}_+$ where $f(A) = |\cup_{i \in A} S_i|$. f is a monotone submodular function. One can also associate weights to elements of $\mathcal{U}$ via a function $w : \mathcal{U} \to \mathbb{R}_+$; the function f defined as $f(A) = w(\cup_{i \in A} S_i)$ is also monotone submodular.

Example 8.2. Cut functions in graphs. Let $G = (V, E)$ be an undirected graph with non-negative edge weights $w : E \to \mathbb{R}_+$. The cut function $f : 2^V \to \mathbb{R}_+$ defined as $f(S) = \sum_{e \in \delta_G(S)} w(e)$ is a symmetric submodular function; it is not monotone unless the graph is trivial. If G is directed and we define f as $f(S) = \sum_{e \in \delta^+_G(S)} w(e)$ then f is submodular but is not necessarily symmetric.

The following problem generalizes Max Cut and Max Directed Cut that we have already seen.

Problem 8.2. Max Submod Func. *Given a non-negative submodular set function f on a ground set V via a* value oracle[1] *find* $\max_{S \subseteq V} f(S)$.

[1] A value oracle for a set function $f : 2^V \to \mathbb{R}$ provides access to the function by giving the value $f(A)$ when presented with the set A.

Note that if f is monotone then the problem is trivial since V is the optimum solution. Therefore, the problem is interesting (and NP-Hard) only when f is not necessarily monotone. We consider a simple local search algorithm for Max Submod Func and show that it gives a 1/3-approximation and a 1/2-approximation when f is symmetric. This was shown in [57].

Remark 8.4. Given a graph $G = (V, E)$ consider the submodular function $f : 2^V \rightarrow \mathbb{R}$ where $f(S) = |\delta(S)| - B$ where B is a fixed number. Is there a polynomial time algorithm to decide whether there is a set S such that $f(S) \geq 0$?

```
LocalSearch for Max Submod Func:
$S \leftarrow \emptyset$
repeat
   If ($\exists v \in V \setminus S$ such that $f(S+v) > f(S)$)
         $S \leftarrow S + v$
   Else If ($\exists v \in S$ such that $f(S-v) > f(S)$)
         $S \leftarrow S - v$
   Else
         $S$ is a local optimum
         return the better of $S$ and $V \setminus S$
   EndIf
Until (True)
```

We start the analysis of the algorithm with a basic lemma on submodularity.

Lemma 8.4. *Let $f : 2^V \rightarrow \mathbb{R}_+$ be a submodular set function on V. Let $A \subset B \subseteq V$. Then*

- *If $f(B) > f(A)$ then there is an element $v \in B \setminus A$ such that $f(A+v) - f(A) > 0$. More generally there is an element $v \in B \setminus A$ such that $f(A+v) - f(A) \geq \frac{1}{|B \setminus A|}(f(B) - f(A))$.*
- *If $f(A) > f(B)$ then there is an element $v \in B \setminus A$ such that $f(B-v) - f(B) > 0$. More generally there is an element $v \in B \setminus A$ such that $f(B-v) - f(B) \geq \frac{1}{|B \setminus A|}(f(A) - f(B))$.*

Exercise 8.1. Prove the preceding lemma.

We obtain the following corollary.

Corollary 8.3. *Let S be a local optimum for the local search algorithm and let S^* be an optimum solution. Then $f(S) \geq f(S \cap S^*)$ and $f(S) \geq f(S \cup S^*)$.*

Theorem 8.4. *The local search algorithm is a 1/3-approximation and is a 1/2-approximation if f is symmetric.*

Proof. Let S be the local optimum and S^* be a global optimum for the given instance. From the previous corollary we have that $f(S) \geq f(S \cap S^*)$ and $f(S) \geq f(S \cup S^*)$. Note that the algorithm outputs the better of S and $V \setminus S$. By submodularity, we have,

$$f(V \setminus S) + f(S \cup S^*) \geq f(S^* \setminus S) + f(V) \geq f(S^* \setminus S)$$

where we used the non-negativity of f in the second inequality. Putting together the inequalities,

$$\begin{aligned} 2f(S) + f(V \setminus S) &= f(S) + f(S) + f(V \setminus S) \\ &\geq f(S \cap S^*) + f(S^* \setminus S) \\ &\geq f(S^*) + f(\emptyset) \\ &\geq f(S^*) = \text{OPT}. \end{aligned}$$

Thus $2f(S) + f(V \setminus S) \geq \text{OPT}$ and hence $\max\{f(S), f(V \setminus S)\} \geq \text{OPT}/3$.

If f is symmetric we argue as follows. Using Lemma 8.4 we claim that $f(S) \geq f(S \cap S^*)$ as before but also that $f(S) \geq f(S \cup \bar{S}^*)$ where $\bar{A}$ is shorthand notation for the the complement $V \setminus A$. Since f is symmetric $f(S \cup \bar{S}^*) = f(V \setminus (S \cup \bar{S}^*)) = f(\bar{S} \cap S^*) = f(S^* \setminus S)$. Thus,

$$\begin{aligned} 2f(S) &\geq f(S \cap S^*) + f(S \cup \bar{S}^*) \\ &\geq f(S \cap S^*) + f(S^* \setminus S) \\ &\geq f(S^*) + f(\emptyset) \\ &\geq f(S^*) = \text{OPT}. \end{aligned}$$

Therefore $f(S) \geq \text{OPT}/2$. ■

The running time of the local search algorithm may not be polynomial but one can modify the algorithm as we did for Max Cut to obtain a strongly polynomial time algorithm that gives a $(1/3 - o(1))$-approximation ($(1/2 - o(1)$ for symmetric). See [57] for more details. There has been much work on submodular function maximization including work on variants with additional constraints. Local search has been a powerful tool for these problems. See [24, 60, 112] for some of the results on local search based method, and [25] for a survey on submodular function maximization.

Chapter 9

Clustering and Facility Location

Clustering and Facility Location are two widely studied topics with a vast literature. Facility location problems have been well-studied in Operations Research and logistics. Clustering is ubiquitious with many applications in data analysis and machine learning. We confine attention to a few central problems and provide some pointers as needed to other topics. These problems have also played an important role in approximation algorithms and their study has led to a variety of interesting techniques. Research on these topics is still quite active.

For both classes of problems a key assumption that we will make is that we are working with points in some underlying metric space. Recall that a space (V, d) where $d : V \times V \to \mathbb{R}_+$ is a metric space if the distance function d satisfies metric properties: (i) $d(u, v) = 0$ iff $u = v$ (reflexivity) (ii) $d(u, v) = d(v, u)$ for all $u, v \in V$ (symmetry) and (iii) $d(u, v) + d(v, w) \geq d(u, w)$ for all $u, v, w \in V$ (triangle inequality). We will abuse the notation and use $d(A, B)$ for two sets $A, B \subseteq V$ to denote the quantity $\min_{p \in A, q \in B} d(p, q)$. Similarly $d(p, A)$ for $p \in V$ and $A \subseteq V$ will denote $\min_{q \in A} d(p, q)$.

Center based clustering: In center based clustering we are given n points $P = \{p_1, p_2, \ldots, p_n\}$ in a metric space (V, d), and an integer k. The goal is to cluster/partition P into k clusters $C_1, C_2, \ldots, C_k$ which are induced by choosing k centers $c_1, c_2, \ldots, c_k$ from V. Each point p_i is assigned to its nearest center from $c_1, c_2, \ldots, c_k$ and this induces a clustering. The nature of the clustering is controlled by an objective function that measures the quality of the clusters. Typically we phrase the problem as choosing $c_1, c_2, \ldots, c_k$ to minimize the clustering objective $\sum_{i=1}^{n} d(p_i, \{c_1, \ldots, c_k\})^q$ for some q. The three most well-studied problems are special cases obtained by choosing an appropriate q.

- k-Center is the problem when $q = \infty$ which can be equivalently phrased as $\min_{c_1, c_2, \ldots, c_k \in V} \max_{i=1}^{n} d(p_i, \{c_1, \ldots, c_k\})$. In other words we want to minimize the maximum distance of the input points to the cluster centers.

- k-Median is problem when $q = 1$. $\min_{c_1,c_2,\ldots,c_k \in V} \sum_{i=1}^{n} d(p_i, \{c_1, \ldots, c_k\})$.
- k-Means is the problem when $q = 2$. $\min_{c_1,c_2,\ldots,c_k \in V} \sum_{i=1}^{n} d(p_i, \{c_1, \ldots, c_k\})^2$.

We will mainly focus on the discrete versions of the problems where $V = \{p_1, p_2, \ldots, p_n\}$ which means that the centers are to be chosen from the input points themselves. However, in many data analysis applications the points lie in $\mathbb{R}^d$ for some d and the centers can be chosen anywhere in the ambient space. In fact this makes the problems more difficult in a certain sense since the center locations now come from an infinite set. One can argue that limiting centers to the input points does not lose more than a constant factor in the approximation and this may be reasonable from a first-cut point of view but perhaps not ideal from a practical point of view. In some settings there may a requirement or advantage in choosing the cluster centers from the input set.

Facility Location: In facility location we typically have two finite sets $\mathcal{F}$ and C where $\mathcal{F}$ represents a set of locations where *facilities* can be located and $\mathcal{D}$ is a set of *client/demand* locations. We will assume that $V = \mathcal{F} \uplus \mathcal{D}$ and that there is a metrid d over V. There are several variants but one of the simplest one is the Uncapacitated Facility Location (UCFL) problem. In UCFL we are given $(\mathcal{F} \uplus \mathcal{D}, d)$ as well auxiliarly information which specifies the cost f_i of opening a facility at location $i \in \mathcal{F}$. The goal is to open a subset of facilities in $\mathcal{F}$ to minimize the sum of the cost of the opened facilities and the total distance traveled by the clients to their nearest open facility. In other words we want to solve $\min_{\mathcal{F}' \subseteq \mathcal{F}} (\sum_{i \in \mathcal{F}'} f_i + \sum_{j \in \mathcal{D}} d(j, \mathcal{F}'))$. The problem has close connections to k-Median problem. The term "uncapacitated" refers to the fact that we do not limit the number of clients that can be assigned to an open facility. In several OR applications that motivate facility location (opening warehouses or distribution facilities), capacity constraints are likely to be important. For this reasons there are several capacitated versions.

9.1 k-Center

Recall that in k-Center we are given n points $p_1, \ldots, p_n$ in a metric space and an integer k and we need to choose k cluster centers $C = \{c_1, c_2, \ldots, c_k\}$ such that we minimize $\max_i d(p_i, C)$. An alternative view is that we wish to find the smallest radius R such that there are k balls of radius R that together cover all the input points. Given a fixed R this can be seen as a Set Cover problem. In fact there is an easy reduction from Dominating Set to k-Center establishing the NP-Hardness. Moreoever, as we saw already in Chapter 1, k-Center has no $2 - \epsilon$-approximation unless $P = NP$ via a reduction from Dominating Set. Here we will see two 2-approximation algorithms that are quite different and have their own advantages. The key lemma for their analysis is common and is stated below.

Lemma 9.1. *Suppose there are $k+1$ points $q_1, q_2, \ldots, q_{k+1} \in P$ such that $d(q_i, q_j) > 2R$ for all $i \neq j$. Then* OPT $> R$.

Proof. Suppose OPT $\leq R$. Then there are k centers $C = \{c_1, c_2, \ldots, c_k\}$ which induces k clusters $C_1, \ldots, C_k$ such that for each cluster C_h and each $p \in C_h$ we have $d(p, c_h) \leq R$. By the pigeon hole principle some q_i, q_j, $i \neq j$ are in the same cluster C_h but this implies that $d(q_i, q_j) \leq d(q_i, c_h) + d(q_j, c_h) \leq 2R$ which contradicts the assumption of the lemma. ∎

Note that the lemma holds even if the centers can be chosen from outside the given point set P.

9.1.1 Gonzalez's algorithm and nets in metric spaces

The algorithm starts with an empty set of centers, and in each iteration picks a new center which is the *farthest* point from the set of centers chosen so far.

```
Gonzalez-$k$-Center$(P, k)$

1. $C \leftarrow \emptyset$
2. For $i = 1$ to $k$ do
    A. Let $c_i = \arg\max_{c \in P} d(c, C)$
    B. $C \leftarrow C \cup \{c_i\}$.
3. Output $C$
```

Note that c_1 is chosen arbitrarily.

Theorem 9.1. *Let $R = \max_{p \in P} d(p, C)$ where C is the set of centers chosen by Gonzalez's algorithm. Then $R \leq 2R^*$ where R^* is the optimum k-Center radius for P.*

Proof. Suppose not. There is a point $p \in P$ such that $d(p, C) > 2R^*$ which implies that $p \notin C$. Since the algorithm chose the farthest point in each iteration and could have chosen p in each of the k iteration but did not, we have the property that $d(c_i, \{c_1, \ldots, c_{i-1}\}) > 2R^*$ for $i = 2$ to k. This implies that the distance between each pair of points in the set $\{c_1, c_2, \ldots, c_k, p\}$ is more than $2R^*$. By Lemma 9.1, the optimum radius must be larger than R^*, a contradiction. ∎

Exercise 9.1. Construct an instance to demonstrate that the algorithm's worst-case performance is 2.

Remark 9.1. Gonzalez's algorithm can be extended in a simple way to create a permutation of the points P; we simply run the algorithm with $k = n$. It is easy to see from the proof above that for any $k \in [n]$, the prefix of the permutation consisting of the first k points provides a 2-approximation for that choice of k. Thus, one can compute the permutation once and reuse it for all k.

9.1.2 Hochbaum-Shmoys bottleneck approach

A second algorithmic approach for k-Center is due to Hochbaum and Shmoys and has played an influential role in variants of this problem.

For a point v and radius r let $B(v, r) = \{u \mid d(u, v) \leq r\}$ denote the ball of radius r around v.

```
HS-$k$-Center$(P, k)$
1. Guess $R^*$ the optimum radius
2. $C \leftarrow \emptyset$, $S \leftarrow P$
3. While ($S \neq \emptyset$) do
    A. Let $c$ be an arbitrary point in $S$
    B. $C \leftarrow C \cup \{c\}$
    C. $S \leftarrow S \setminus B(c, 2R^*)$
4. Output $C$
```

Theorem 9.2. *Let C be the output of the HS algorithm for a guess R. Then for all $p \in P$, $d(p, C) \leq 2R$ and moreover if $R \geq R^*$ then $|C| \leq k$.*

Proof. The first property is easy to see since we only remove a point p from S if we add a center c to C such that $p \in B(c, 2R)$. Let $c_1, c_2, \ldots, c_h$ be the centers chosen by the algorithm. We observe that $d(c_i, \{c_1, \ldots, c_{i-1}\}) > 2R$. Thus, if the algorithm outputs h points then the pairwise distance between any two of them is more than $2R$. By Lemma 9.1, if $h \geq k + 1$ the optimum radius is $> R$. Hence, if the guess $R \geq R^*$ the algorithm outputs at most k centers. ∎

The guessing of R^* can be implemented by binary search in various ways. We omit these routine details.

Exercise 9.2. Describe an example where the algorithm uses exactly k centers even with guess R^*. Describe an example where the algorithm outputs less than k centers with a guess of R^*.

9.1.3 Related Problems and Discussion

The k-Center problem is natural in geometric settings and one can see from the proof that the 2-approximation for the two algorithms holds even when allowing for centers to be chosen outside. A surprising result of Feder and Greene [55] shows that even in two dimensions (the Euclidean plane) one cannot improve the factor of 2 unless $P = NP$.

The k-Supplier problem is closely related to k-Center and is motivated by facility location considerations. Here we are given $\mathcal{F} \cup \mathcal{D}$ which are in a metric space. We need to choose k centers $C \subseteq \mathcal{F}$ to minimize $\max_{p \in \mathcal{D}} d(p, C)$. Note that we don't have to cover the facilities. Hochbaum and Shmoys gave a variant of their algorithm that obtains a 3-approximation for k$-Supplier$ and moreover showed that unless $P = NP$ one cannot improve 3 [84]. Interestingly in Euclidean spaces 3 is not tight [127]. Several generalizations of k-Center which constrain the choice of centers have been considered — see [31] for a general framework that also considers outliers.

One can consider weighted version of k-Center or relabeled as priority version in subsequent work. We refer to the work of Plesnik [131] and a recent one [16] on this variant which has found applications in fair clustering.

We finally mention that k-Center clustering has nice connections to the notion of r-nets in metric spaces. Given a set of points P in a metric space and a radius r, an r-net is a set of centers C such that (i) for all $p \in P$ we have $d(p, C) \leq r$ (that is the points are covered by balls of radius r) and (ii) for any distinct $c, c' \in C$ we have $B(c, r/2)$ and $B(c', r/2)$ are disjoint (packing property or the property that no two centers are too close). r-nets provide a concise summary of distances in a metric space at scale r. One can use r-nets to obtain nearest-neighbor data structures and other applications, especially in low-dimensional settings. We refer the reader to [77, 78].

LP Relaxation: The two k-Center algorithms we described are combinatorial. One can also consider an LP relaxation. Since it is a bottleneck problem, writing an LP relaxation involves issues similar to what we saw with unrelated machine scheduling. Given a guess R we can write an LP to check whether a radius R is feasible and then find the smallest R for which it is feasible. The feasibility LP can be written as follows. Let x_i be an indicator for whether we open a center at point p_i.

$$
\begin{aligned}
\sum_{i=1}^{n} x_i &= k \\
\sum_{p_i \in B(p_j,R)} x_i &\geq 1 \quad \forall p_j \in P \\
x_i &\geq 0 \quad \forall p_i \in P
\end{aligned}
$$

Exercise 9.3. Prove that if R is feasible for the preceding LP then one can obtain a solution with k centers with max radius $2R$.

Exercise 9.4. Generalize the LP for the k-Supplier problem and prove that one can obtain a 3-approximation with respect to lower bound provided via the LP approach.

9.2 Uncapacitated Facility Location

We now discuss UCFL. One can obtain a constant factor for UCFL via several techniques: LP rounding, primal-dual, local search and greedy. The best known approximation bound is 1.488 due to Li [116] while it is known that one cannot obtain a ratio better than 1.463 [71]. We will describe the complicated algorithms and focus on the high-level approaches that yield some constant factor approximation.

It is common to use i to denote a facility in $\mathcal{F}$ and j to denote a demand/client.

9.2.1 LP Rounding

The first constant factor approximation for UCFL was via LP rounding by Aardal, Shmoys and Tardos using a filtering technique of Lin and Vitter. We start with the LP relaxation. We use a variable y_i for $i \in \mathcal{F}$ to indicate whether i is opened or not. We use a variable $x_{i,j}$ to indicate whether j is connected to i (or assigned to i). One set of constraints are natural here: each client has to be assigned/connected to a facility. The other constraint requires that j is assigned to i only if i is open.

$$
\min \sum_{i \in \mathcal{F}} f_i y_i + \sum_{j \in \mathcal{D}} \sum_{i \in \mathcal{F}} d(i,j) x_{i,j}
$$

$$
\begin{aligned}
\sum_{i} x_{i,j} &= 1 \quad \forall j \in \mathcal{D} \\
x_{i,j} &\leq y_i \quad i \in \mathcal{F}, j \in \mathcal{D} \\
x, y &\geq 0
\end{aligned}
$$

Given a feasible solution x, y to the LP the question is how to round. We note that the LP relaxation does not "know" whether d is a metric or not. In fact when d is arbitrary (but non-negative) we obtain the non-metric facility location problem which is as hard as the Set Cover problem but not much harder — one can obtain an $O(\log n)$-approximation. However, we can obtain a constant factor when d is a metric and the rounding will exploit this property.

Given the fractional solution x, y for each j we define α_j to be the distance cost paid for by the LP: therefore $\alpha_j = \sum_{i \in \mathcal{F}} d(i, j) x_{i,j}$. Note that the LP cost is $\sum_i f_i y_i + \sum_j \alpha_j$.

Lemma 9.2. *For each j and each $\delta \in (0, 1)$ there is a total facility value of at least $(1 - \delta)$ in $B(j, \alpha_j/\delta)$. That is, $\sum_{i \in B(j, \alpha_j/\delta)} y_i \geq 1 - \delta$. In particular $\sum_{i \in B(j, 2\alpha_j)} y_i \geq 1/2$.*

Proof. This essentially follows from Markov's inequality or averaging. Note that $\alpha_j = \sum_i d(i, j) x_{i,j}$ and $\sum_i x_{i,j} = 1$. Suppose $\sum_{i \in B(j, \alpha_j/\delta)} y_i < 1 - \delta$. Since $x_{i,j} \leq y_i$ for all i, j, we will have $\alpha_j > \delta \alpha_j / \delta$ which is impossible. ■

We say that two clients j and j' intersect if there is some $i \in \mathcal{F}$ such that $i \in B(j, 2\alpha_j) \cap B(j', 2\alpha_{j'})$. The rounding algorithm is described below.

```
UCFL-primal-rounding

1. Solve LP and let $(x, y)$ be a feasible LP solution
2. For each $j$ compute $\alpha_j = \sum_i d(i, j) x_{i,j}$
3. Renumber clients such that $\alpha_1 \leq \alpha_2 \leq \ldots \leq \alpha_h$ where $h$ is number of clients
4. For $j = 1$ to $h$ do
    A. If $j$ already assigned continue
    B. Open cheapest facility $i$ in $B(j, 2\alpha_j)$ and assign $j$ to $i$
    C. For each remaining client $j' > j$ that intersects with $j$, assigne $j'$ to $i$
5. Output the list of open facilities and the client assignment
```

It is not hard to see that every client is assigned to an open facility. The main issue is to bound the total cost. Let F be the total facility opening cost, and let C be the total connection cost. We will bound these separately.

Lemma 9.3. $F \leq 2 \sum_i f_i y_i$.

Proof. Note that a client j opens a new facility only if it has not been assigned when it is considered by the algorithm. Let $j_1, j_2, \ldots, j_k$ be the clients that open facilities. Let $A_j = \mathcal{F} \cap B(j, 2\alpha_j)$ be the set of facilities in the ball of radius $2\alpha_j$ around j. From the algorithm and the definition of intersection of clients, we see that the sets $A_{j_1}, A_{j_2}, \ldots, A_{j_k}$ are pairwise disjoint. The algorithm opens the cheapest facility in A_{j_ℓ} for $1 \leq \ell \leq k$. Note that $\sum_{i \in A_{j_\ell}} y_i \geq 1/2$ by Lemma 9.2. Hence the cost of the cheapest facility in A_{j_ℓ} is at most $2\sum_{i \in A_{j_\ell}} f_i y_i$ (why?). By the disjointness of the sets $A_{j_1}, \ldots, A_{j_k}$ we see that the total cost of the facilities opened is at most $2\sum_i f_i y_i$. ■

Lemma 9.4. *$C \leq 6\sum_j \alpha_j$.*

Proof. Consider a client j that opens a facility i in $B(j, 2\alpha_j)$. In this case j is assigned to i and $d(i,j) \leq 2\alpha_j$. Now consider a client j that does not open a facility. This implies that there was a client $j' < j$ that opened a facility i' and j' and j intersect, and j was assigned to i'. What is $d(i', j)$? We claim that $d(i', j) \leq 6\alpha_j$. To see this we note that j' and j intersect because there is some facility $i \in B(j, 2\alpha_{j'}) \cap B(j, 2\alpha_j)$. By considering the path $j \rightarrow i \rightarrow j' \rightarrow i'$, via triangle inequality, $d(i', j) \leq 2\alpha_j + 2\alpha_{j'} + 2\alpha_{j'} \leq 6\alpha_j$ since $\alpha_{j'} \leq \alpha_j$. Thus the distance traveled by each client j to its assigned facility is at most $6\alpha_j$. The lemma follows. ■

The two preceding lemmas give us the following which implies that the algorithm yields a 6-approximation.

Theorem 9.3. *$C + F \leq 6\,\mathrm{OPT}_{LP}$.*

It should be clear to the reader that the algorithm and analysis are not optimized for the approximation ratio. The goal here was to simply outline the basic scheme that led to the first constnat factor approximation.

9.2.2 Primal-Dual

Jain and Vazirani developed an elegant and influential primal-dual algorithm for UCFL [91]. It was influential since it allowed new algorithms for k-median and several generalizations of UCFL in a clean way. Moreover the primal-dual algorithm is simple and efficient to implement. On the other hand we should mention that one advantage of having an LP solution is that it gives an explicit lower bound on the optimum value while a primal-dual method yields a lower bound via a feasible dual which may not be optimal. We need some background to describe the primal-dual method in approximation.

Complementary slackness: To understand primal-dual we need some basic background in complementary slackness. Suppose we have a primal LP (P) of the form $\min cx$ s.t $Ax \leq b, x \geq 0$ which we intentionally wrote in this standard form as a covering LP. It's dual (D) is a packing LP $\max b^t y$ s.t $yA^t \geq c, y \geq 0$. We will assume that both primal and dual are feasible and hence the optimum values are finite, and via strong duality we know that the optimum values are the same.

Definition 9.4. *A feasible solution x to (P) and a feasible solution y to (D) satisfy the primal complementary slackness condition with respect to each other if the following is true: for each i, $x_i = 0$ or the corresponding dual constraint is tight, that is $\sum_j A_{i,j} y_j = c_i$. They satisfy the dual complementary slackness condition if the following is true: for each j, $y_j = 0$ or the corresponding primal constraint is tight, that is $\sum_j A_{j,i} x_i = b_j$.*

One of the consequences of the duality theory of LP is the following.

Theorem 9.5. *Suppose (P) and (D) are a primal and dual pair of LPs that both have finite optima. A feasible solution x to (P) and a feasible solution y to (D) satisfy the primal-dual complementary slackness properties with respect to each other* if and only if *they are both optimum solutions to the respective LPs.*

We illustrate the use of complementary slackness in the context of approximation via Vertex Cover. Recall the primal covering LP and we also write the dual.

$$\begin{aligned} \min \sum_{v \in V} w_v x_v & & & \\ x_u + x_v & \geq & 1 & \qquad \forall e = (u,v) \in E \\ x_v & \geq & 0 & \qquad \forall v \in V \end{aligned}$$

$$\begin{aligned} \max \sum_{e \in E} y_e & & & \\ \sum_{e \in \delta(v)} y_e & \leq & w_v & \qquad \forall v \in V \\ y_e & \geq & 0, & \qquad \forall e \in E \end{aligned}$$

Recall that we described a simple rounding algorithm. Given a feasible primal solution x. We let $S = \{v \mid x_v \geq 1/2\}$ and showed that (i) S is a vertex cover for the given graph G and (ii) $w(S) \leq 2 \sum_v w_v x_v$. Now suppose we have an *optimum* solution x^* to the primal rather than an arbitrary feasible solution. We can prove an interesting and stronger statement via complementary slackness.

Lemma 9.5. *Let x^* be an optimum solution to the primal covering LP. Then $S = \{v \mid x^*_v > 0\}$ is a feasible vertex cover for G and moreover $w(S) \leq 2\sum_v w_v x^*_v$.*

Proof. It is easy to see that S is a vertex cover via the same argument that we have seen before. How do we bound the cost now that we may be rounding x^*_v to 1 even though x^*_v may be tiny? Let y^* be any optimum solution to the dual; one exists (why?). Via strong duality we have $\sum_v w_v x^*_v = \sum_e y^*_e$. Via primal-complementary slackness we have the property that if $x^*_v > 0$ then $\sum_{e \in \delta(v)} y^*_e = w_v$. Hence

$$w(S) = \sum_{v:x^*_v>0} w_v = \sum_{v:x^*_v>0} \sum_{e\in\delta(v)} y^*_e.$$

Interchanging the order of summation we obtain that

$$w(S) = \sum_{v:x^*_v>0} \sum_{e\in\delta(v)} y^*_e \leq 2\sum_{e\in E} y^*_e = 2\sum_v w_v x^*_v$$

where we use the fact that an edge e has only two end points in the inequality. ■

Primal-dual for approximating Vertex Cover: We will first study a primal-dual approximation algorithm for the Vertex Cover problem — this algorithm due to Bar-Yehuda and Even [19] can perhaps be considered the first primal-dual algorithm in approximation. Primal-dual is a classical method in optimization and is applicable in both continuous and discrete settings. The basic idea, in the context of LPs (the method applies more generally), is to obtain a solution x to a primal LP and a solution y to the dual LP *together* and certify the optimality of each solution via complementary slackness. It is beyond the scope of this notes to give a proper treatment. One could argue that understanding the setting of approximation is easier than in the classical setting where one seeks exact algorithms, since our goals are more modest.

Typically one starts with one of x, y being feasible and the other infeasible, and evolve them over time. In discrete optimization, this method is successfully applied to LPs that are known to be integer polytopes. Examples include shortest paths, matchings, and others. In approximation the LP relaxations are typically not integral. In such a setting the goal is to produce a primal and dual pair x, y where x is an integral feasible solution, and y is fractional feasible solution. The goal is to approximately bound the value of x via the dual y, and for this purpose we will enforce only the primal complementary slackness condition for the pair (x, y). To make the algorithm and analysis manageable, the primal-dual algorithm is typically done in a simple but clever fashion — there have been several surprisingly strong and powerful approximation results via this approach.

We illustrate this in the context of Vertex Cover first. It is useful to *interpret* the dual as we did already in the context of the dual-fitting technique for Set Cover. We think of the edges $e \in E$ as agents that wish to be covered by the vertices in G at minimum cost. The dual variable y_e can be thought of as the amount that edge e is willing to pay to be covered. The dual packing constraint $\sum_{e \in \delta(v)} y_e \leq w_v$ says that for any vertex v, the total payments of all edges incident to v cannot exceed its weight. This can be understood game theoretically as follows. The set of edges $\delta(v)$ can together pay w_v and get covered, and hence we cannot charge them more. The dual objective is to maximize the total payment that can be extracted from the edges subject to these natural and simple constraints. With this interpretation in mind we wish to produce a feasible dual (payments) and a corresponding feasible integral primal (vertex cover). The basic scheme is to start with an infeasible primal $x = 0$ and a feasible dual $y = 0$ and increase y while maintaining feasibility; during the process we will maintain primal complementary slackness which means that if a dual constraint for a vertex v becomes tight we set $x_v = 1$. Note that we are producing an integer primal solution in this process. How should we increase y values? We will do it in the naive fashion which is to *uniformly* increase y_e for all e that are not already part of a tight constraint (and hence not covered yet).

```
VC-primal-dual($G = (V, E), w : V \to \mathbb{R}_+$)
1. $x \leftarrow 0$, $y \leftarrow 0$            // initialization: primal infeasible, dual feasible
2. $U \leftarrow E$                               // uncovered edges that are active
3. While ($U \neq \emptyset$) do
    A. Increase $y_e$ uniformly for each $e \in U$ until constraint $\sum_{e \in \delta(v)} y_e = w_v$ for
       some vertex $a$
    B. Set $x_a = 1$                              // Maintain primal complementary slackness
    C. $U \leftarrow U \setminus \delta(a)$       // Remove all edges covered by a
4. Output integer solution $x$ and dual certificate $y$
```

Remark 9.2. Note that when checking whether a vertex v is tight we count the payments from $e \in \delta(v)$ even though some of them are no longer active.

Lemma 9.6. *At the end of the algorithm x is a feasible vertex cover for G and $\sum_v w_v x_v \leq 2\,\mathrm{OPT}$.*

Proof. By induction on the iterations one can prove that (i) y remains dual feasible throughout (ii) $ab \in U$ at the start of an iteration iff $e = ab$ is not covered yet (iii) each iteration adds at least one more vertex and hence the algorithm terminates in $\leq n$ iterations and outputs a feasible vertex cover. The main issue is the cost of x.

For this we note that the algorithm maintains primal complementary slackness. That is, $x_v = 0$ or if $x_v = 1$ then $\sum_{e \in \delta(v)} y_e = w_v$. Thus, we have

$$\sum_v w_v x_v = \sum_{v:x_v>0} \sum_{e \in \delta(v)} y_e \leq 2 \sum_e y_e \leq 2\,\mathrm{OPT}_{LP} .$$

We used the fact that e has at most two end points in the first inequality and the fact that y is dual feasible in the second inequality. In terms of payment what this says is that edge uv's payment of y_{uv} can be used to pay for opening u *and* v while the dual pays only once. ■

As the reader can see, the algorithm is very simple to implement.

Exercise 9.5. Describe an example to show that the primal-dual algorithm's worst case performance is 2. Describe an example to show that the dual value constructed by the algorithm is $\simeq \mathrm{OPT}/2$. Are these two parts the same?

Remark 9.3. The algorithm generalizes to give an f-approximation for Set Cover where f is the maximum frequency of any element. There are examples showing that the performance of this algorithm in the worst-case can indeed be a factor of f. We saw earlier that the integrality gap of the LP is at most $1 + \ln d$ where d is the maximum set size. There is no contradiction here since the specific primal-dual algorithm that we developed need not achieve the tight integrality gap.

Primal-Dual for UCFL: Now we consider a primal-dual algorithm for UCFL. Recall the primal LP that we discussed previously. Now we describe the dual LP written below. The dual has two types of variables. For each client j there is a variable α_j corresponding to the primal constraint that each client j needs to be connected to a facility. For each facility i and client j there is a variable $\beta_{i,j}$ corresponding to the constraint that $y_i \geq x_{i,j}$.

$$\begin{aligned} \max \sum_{j \in \mathcal{D}} \alpha_j & & & \\ \sum_i \beta_{i,j} &\leq f_i & & \forall i \in \mathcal{F} \\ \alpha_j - \beta_{i,j} &\leq d(i,j) & & i \in \mathcal{F}, j \in \mathcal{D} \\ \alpha, \beta &\geq 0 & & \end{aligned}$$

It is important to interpret the dual variables. There is a similarity to Set Cover since Non-Metric Facility Location is a generalization, and the LP formulation does not distinguish between metric and non-metric facility location — it is only in the rounding that we can take advantage of metric properties. The variable α_j can be interpreted as the amount of payment client j is willing to make. This comes in two parts — the payment to travel to a facility which it cannot share with any other clients, and the payment it is willing to make to open a facility which it can share with other clients. The variable $\beta_{i,j}$ corresponds to the amount client j is willing to pay to facility i to open it. (In Set Cover there is no need to distinguish between α_j and $\beta_{i,j}$ since there are no distances (or they can be assumed to be 0 or ∞).) The first set of constraints in the dual say that for any facility i, the total payments from all the clients ($\sum_j \beta_{i,j}$) cannot exceed cost f_i. The second set of constraints specify that $\alpha_j - \beta_{i,j}$ is at most $d(i,j)$. One way to understand this is that if $\alpha_j < d(i,j)$ then client j will not even be able to reach i and hence will not contribute to opening i. Via this interpretation it is convenient to assume that $\beta_{i,j} = \max\{0, \alpha_j - d(i,j)\}$.

The primal-dual algorithm for UCFL will have a growth stage that is similar to what we saw for Vertex Cover. We increase α_j for each “uncovered” client j uniformly. A facility i will receive payment $\beta_{i,j} = \max\{0, \alpha_j - d(i,j)\}$ from j. To maintain dual feasibility, as soon as the constraint for facility i becomes tight we open facility i (in the primal we set $y_i = 1$); note that any client j such that $\beta_{i,j} > 0$ cannot increase its α_j any further and hence will stop growing since it is connected to an open facility. This process is very similar to that in Set Cover. The main issue is that we will get a weak approximation (a factor of $|\mathcal{F}|$) since a client can contribute payments to a large number of facilities. Note that the process so far has not taken advantage of the fact that we have a metric facility location problem. Therefore, in the second phase we will close some facilities which means that a client may need to get connected to a facility that it did not contribute to — however we will use the metric property to show that a client does not need to travel too far to reach a facility that remains open.

With the above discussion in place, we describe the two phase primal-dual algorithm below. The algorithm also creates a bipartite graph G with vertex set $\mathcal{F} \cup \mathcal{D}$ and initially it has no edges.

```
JV-PRIMAL-DUAL($(\mathcal{F} \cup \mathcal{D}, d), f_i, i \in \mathcal{F}$)
1. $\alpha_j = 0$ for all $j \in \mathcal{D}$, $\beta_{i,j} \leftarrow 0$ for all $i, j$    // initialize dual values to 0
2. $A \leftarrow \mathcal{D}$    // active clients that are unconnected
3. $O \leftarrow \emptyset$    // temporarily opened facilities
4. While ($A \neq \emptyset$) do    // Growth phase
    A. Increase $\alpha_j$ uniformly for each $j \in A$ while maintaining the invariant
       $\max\{0, \alpha_j - d(i,j)\} = \beta_{i,j}$ until one of following conditions hold: (i)
       $\alpha_j = d(i,j)$ for some $j \in A$ and $i \in O$ or (ii) $\sum_j \beta_{ij} = f_i$ for some
       $i \in \mathcal{F} \setminus O$
    B. If (i) happens then add edge $(i,j)$ to $G$ and and $A \leftarrow A - \{j\}$    // $j$ is
       connected to already open facility $i \in O$
    C. Else If (ii) happens then
        1. $O \leftarrow O \cup \{i\}$    // temporarily open $i$ that became tight
        2. for each $j \in \mathcal{D}$ such that $\beta_{i,j} > 0$ add edge $(i,j)$to $G$    // note: clients
           not in $A$ may also get edges to $i$
        3. $A \leftarrow A \setminus \{j : \beta(i,j) > 0\}$    // make active clients connected to $i$ inactive
5. Create graph $H$ with vertex set $O$ and edge set $Q$ where $(i,i') \in Q$ iff
   there exists client $j$ such that $\beta(i,j) > 0$ and $\beta(i',j) > 0$
6. $O'$ is any maximal independent set in $H$
7. Close facilities in $O \setminus O'$    // Pruning phase
8. Assign each client $j$ to nearest facility in $O'$
```

Example: The example in Fig 9.2.2 illustrates the need for the pruning phase. There are $2n$ clients and n facilities and the opening costs of the facilities are $n+2$ except for the first one which has an opening cost of $n+1$. The first group of clients shown at the top of the figure have a connection cost of 1 to each facility. The second group of clients have the following property: $d(i_h, j'_h) = 1$ and $d(i_\ell, j'_h) = 3$ when $\ell \neq j$. The rest of the distances are induced by these. One can verify that in the growth phase all the facilities will be opened. However the total dual value after the growth phase is $5n-1$ while the total cost of the opened facilitie is $\Theta(n^2)$ and hence pruning is necessary to obtain a good approximation.

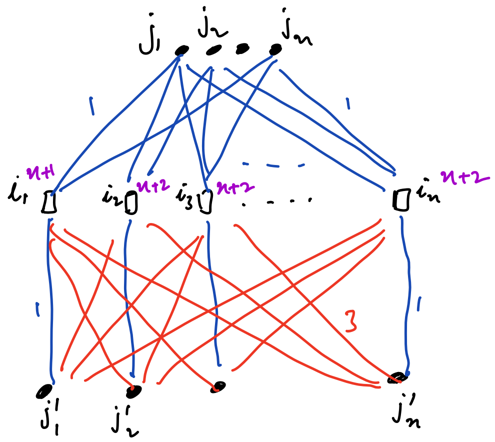


Figure 9.1: Example to illustrate the need for the pruning phase.

We will do a high-level analysis of the algorithm skipping a few fine details. A formal treatment with full details can be found in [152].

The algorithm maintains the property that α_j and $\beta_{i,j}$ variables are dual feasible. Consider the graph G created by the algorithm. We call an edge $(i, j) \in G$ a witness edge for j when j is removed from A (it happens exactly once for each j). We call an edge $(i, j) \in G$ special if $\beta(i, j) > 0$ which means that j paid to temporarily open i. We remark that a client j may have a special edge to i even though (i, j) is not its witness edge — this can happen if i is temporarily opened after j was already removed from A (due to other clients contributing to opening i later). One can associate a notion of time with the progression of the algorithm since dual variables are monotonically increased. This can be used to order events.

A basic property maintained by the algorithm is the following.

Claim 9.2.1. *If (i, j) is an edge in G then $\alpha_j - \beta_{i,j} = d(i, j)$ and hence $\alpha_j \geq d(i, j)$.*

Proof. The algorithm adds an edge (i, j) only if (i, j) is a witness edge for j or if $\beta(i, j) > 0$ (in which case (i, j) is special). The algorithm maintain the invariant $\beta_{i,j} = \max\{0, \alpha_j - d(i, j)\}$ and hence if $\beta_{i,j} > 0$ the claim is clear. If $\beta_{i,j} = 0$ then (i, j) is a witness edge for j and case (i) happens when j is removed from A and in this case $\alpha_j = d(i, j)$. ■

Analysis: We upper bound the cost of opening the facilities in O' and the connection cost of the clients.

We leave the proof of the following lemma as an exercise.

Lemma 9.7. *For each facility $i \in O$ we have $\sum_{(i,j) \text{ is special}} \beta_{i,j} = f_i$.*

Since a client j can pay for multiple facilities which we cannot afford, the pruning phase removes facilities such that no client j is connected two facilities in O' with special edges (otherwise those two facilities will have an edge in H). We say that a client j is directly connected to a facility $i \in O'$ if (i, j) is a special edge. We call all such clients directly connected clients and the rest of the clients are called indirectly connected. We let $\mathcal{D}_1 = \cup_{i \in O'} Z_i$ be the set of all directly connected clients and let $\mathcal{D}_2$ be the set of all indirectly connected clients.

For $i \in O'$ let Z_i be the directly connected clients. By the pruning rule we have the property that a client j is directly connected to at most one facility in O'. We show that each facility in O' can be paid for by its directly connected clients.

Lemma 9.8. *For each $i \in O'$, $\sum_{j \in Z_i} \beta_{i,j} = f_i$.*

Proof. From Lemma 9.7 and the fact that if $i \in O'$ then every client j with special edge (i, j) must be directly connected to i. ■

From Claim 9.2.1 we see that if j is directly connected to i then $\alpha_j - \beta_{i,j} = d(i, j)$, and hence j can pay its connection cost to i and its contribution to opening i. What about indirectly connected clients? The next lemma bounds their connection cost.

Lemma 9.9. *Suppose $j \in \mathcal{D}_2$, that is, it is an indirectly connected client. Let i be its closest facility in O' then $d(i, j) \leq 3\alpha_j$.*

Proof. Let (i, j) be the witness edge for j. Note that $i \in O$. Since j is an indirectly connected client there is no facility $i' \in O'$ such that (i', j) is a special edge. Since $i \notin O'$ it must be because i was closed in the pruning phase and hence there must be a facility $i' \in O'$ such that (i, i') is an edge in H (otherwise O' would not be a maximal independent set). Therefore there is some client $j' \neq j$ such that (i', j') and (i, j') are both special edges. We claim that $\alpha_j \geq \alpha_{j'}$. Assuming this claim we see via triangle inequality and Claim 9.2.1 that,

$$d(i', j) \leq d(i, j) + d(i, j') + d(i', j') \leq \alpha_j + 2\alpha_{j'} \leq 3\alpha_j.$$

Since $i' \in O'$ the nearest client to j is within distance $\leq 3\alpha_j$.

We now prove the claim that $\alpha_j \geq \alpha_{j'}$. Let $t = \alpha_j$ be the time when j connects to facility i as its witness. Consider two cases. In the first case $d(i, j') \leq t$ which means that j' had already reached i at or before t; in this case $\alpha_{j'} \leq t$ since i was open at t. In the second case $d(i, j') > t$; this means that j' reached i strictly after t. Since i was already open at t, j' would not pay to open i which implies that $\beta(i, j') = 0$ but then (i, j') would not be a special edge and hence this case cannot arise. This finishes the proof of the claim. ■

With the preceding two lemmas in place we can bound the total cost of opening facilities in O' and connecting clients to them. We will provide a refined statement that turns out to be useful in some applications.

Theorem 9.6.

$$\sum_{j \in \mathcal{D}} d(O', j) + 3 \sum_{i \in O'} f_i \leq 3 \sum_{j \in \mathcal{D}} \alpha_j \leq 3\,\mathrm{OPT}_{LP}\,.$$

In particular the algorithm yields a 3-approximation.

Proof. Consider directly connected clients $\mathcal{D}_1$. We have $\mathcal{D}_1 = \uplus_{i \in O'} Z_i$ where Z_i are directly connected to i. Via Lemma 9.8 and Claim 9.2.1

$$\begin{aligned}
\sum_{j \in \mathcal{D}_1} \alpha_j &= \sum_{i \in O'} \sum_{j \in Z_i} \alpha_j \\
&= \sum_{i \in O'} \sum_{j \in Z_i} (d(i,j) + \beta_{i,j}) \\
&= \sum_{j \in O'} \left(f_i + \sum_{j \in Z_i} d(i,j) \right) \\
&\geq \sum_{j \in O'} f_i + \sum_{j \in \mathcal{D}_1} d(O', j).
\end{aligned}$$

For indirectly connected clients, via Lemma 9.9, we have $3 \sum_{j \in \mathcal{D}_2} \alpha_j \geq \sum_{j \in \mathcal{D}_2} d(O', j)$. Thus

$$\begin{aligned}
3 \sum_{j \in \mathcal{D}} \alpha_j &= 3 \sum_{j \in \mathcal{D}_1} \alpha_j + 3 \sum_{j \in \mathcal{D}_2} \alpha_j \\
&\geq 3 \sum_{i \in O'} f_i + 3 \sum_{j \in \mathcal{D}_1} d(O', j) + \sum_{j \in \mathcal{D}_2} d(O', j) \\
&\geq 3 \sum_{i \in O'} f_i + \sum_{j \in \mathcal{D}} d(O', j).
\end{aligned}$$

■

The algorithm is easy and efficient to implement. One of the main advantages of the stronger property that we saw in the theorem is that it leads to a nice algorithm for the k-median problem; we refer the reader to Chapter 25 in [152] for a detailed description. In addition the flexibility of the primal-dual algorithm has led to algorithms for several other variants; see [20] for one such example.

9.2.3 Local Search

Local search has been shown to be a very effective heuristic algorithm for facility location and clustering probelms and there is extensive literature on this topic. The first paper that proved constant factor approximation bounds for UCFL is by Korupolu, Plaxtion and Rajaraman [108] and it provided a useful template for many future papers. We refer the reader to Chapter 9 in [155] for local search analysis for UCFL.

9.3 k-Median

k-Median has been extensively studied in approximation algorithms due to its simplicity and connection to UCFL. The first constant factor approximation was obtained in [35] via LP rounding. We will consider a slight generalization of k-Median where the medians are to be selected from the facility set $\mathcal{F}$. We describe the LP which is closely related to that for UCFL which we have already seen. The variables are the same: y_i indicates whether a center is opened at location $i \in \mathcal{F}$ and $x_{i,j}$ indicates whether client j is connected to facility i. The objective and constraints change since the problem requires one to open at most k facilities but there is no cost to opening them.

$$\min \sum_{j \in \mathcal{D}} \sum_{i \in \mathcal{F}} d(i,j) x_{i,j}$$

$$\begin{aligned} \sum_i x_{i,j} &= 1 \quad \forall j \in \mathcal{D} \\ x_{i,j} &\leq y_i \quad i \in \mathcal{F}, j \in \mathcal{D} \\ \sum_i y_i &\leq k \\ x, y &\geq 0 \end{aligned}$$

Rounding of the LP for k-Median is not as simple as it is for UCFL. This is mainly because one needs a global argument. To understanding this consider the following example. Suppose we have $k+1$ points where the distance between each pair is 1 (uniform metric space). Then the optimum integer solution has cost 1 since we can place k medians at k points and the remaining point has to travel a distance of 1. Now consider a fractional solution where $y_i = (1 - 1/(k+1))$ for each point. The cost of this fractional solution is also 1, however, each client now pays a fractional cost of $1/(k+1)$. Any rounding algorithm will make at least one of the clients pay a cost of 1 which is much larger than its fractional cost;

thus the analysis cannot be based on preserving the cost of each client within a constant factor of its fractional cost.

There are several LP rounding algorithms known. An advantage of LP based approach is that it leads to a constant factor approximation for the Matroid Median problem which is a nice and powerful generalization of the k-Median problem; here there is a matroid defined over the facilities and the constraint is that the set of facilities chosen must be independent in the given matroid. One can write a natural LP relaxation for this problem and prove that the LP has a constant integrality gap by appealing to matroid intersection! It showcases the power of classical result in combinatorial optimization. We refer the reader to [109, 147].

9.3.1 Local Search

Local search for center based clustering problems is perhaps one of the most natural heuristics. In particular we will consider the p-swap heuristic. Given the current set of k centers S, the p-swap heuristic will consider swapping out *up to p* centers from S with p new centers. It is easy to see that this local search algorithm can be implemented in $n^{O(p)}$ time for each iteration. When $p = 1$ we simply refer to the algorithm as (basic) local search. We will ignore the convergence time. As we saw for Max Cut, one can use standard tricks to make the algorithm run in polynomial time with a loss of a $(1 + o(1))$-factor in the approximation bound guaranteed by the worst-case local optimum. Thus the main focus will be on the quality of the local optimum. The following is a very nice result.

Theorem 9.7 (Arya et al. [11]). *For any fixed $p \geq 1$ the p-swap local search heuristic has a tight worst-case approximation ratio of $(3 + 2/p)$ for k-Median. In particular the basic local search algorithm yields a 5-approximation.*

Here we give a proof/analysis of the preceding theorem for $p = 1$, following the simplified analysis presented in [74]. See also [155] and the notes from CMU. Given any set of centers S we define $\text{cost}(S) = \sum_{j \in \mathcal{D}} d(j, S)$ to be the sum of the distances of the clients to the centers. Let S be a local optimum and let S^* be some fixed optimum solution to the given k-Median instance. To compare $\text{cost}(S)$ with $\text{cost}(S^*)$ the key idea is to set up a *clever* set of potential swaps between the centers in S and centers in S^*. That is, we consider a swap pair (r, f) where $r \in S$ and $f \in S^*$. Since S is a local optimum it must be the case that $\text{cost}(S - r + f) \leq \text{cost}(S)$. The analysis upper bounds the potential increase in the cost in some interesting fashion and sums up the resulting series of inequalities. This may seem magical, and indeed it is not obvious why the analysis proceeds in this fashion. The short answer is that the analysis ideas required a series of developments with the somewhat easier case of UCFL coming first.

We set up some notation. Let $\phi : \mathcal{D} \rightarrow S$ be the mapping of clients to facilities in S based on nearest distance. Similarly let $\phi^* : \mathcal{D} \rightarrow S^*$ the mapping to facilities in the optimum solution S^*. Thus j connects to facility $\phi(j)$ in the local optimum and to $\phi^*(j)$ in the optimum solution. We also let $N(i)$ denote the set of all clients assigned to a facility $i \in S$ and let $N^*(i)$ denote the set of all clients assigned to a facility $i \in S^*$. Let $A_j = d(j, S)$ and $O_j = d(j, S^*)$ be the cost paid by j in local optimum and optimal solutions respectively. To reinforce the notation we express cost(S) as follows.

$$\text{cost}(S) = \sum_{j \in \mathcal{D}} A_j = \sum_{i \in S} \sum_{j \in N(i)} d(j, i).$$

Similarly

$$\text{cost}(S^*) = \sum_{j \in \mathcal{D}} O_j = \sum_{i \in S^*} \sum_{j \in N^*(i)} d(j, i).$$

Setting up the swap pairs: We create a set of pairs $\mathcal{P}$ as follows. We will assume without loss of generality that $|S| = |S^*| = k$. For technical convenience we also assume $S \cap S^* = \emptyset$; we can always create dummy centers that are co-located to make this assumption. Consider the mapping $\rho : S^* \rightarrow S$ where each $i \in S^*$ is mapped to its *closest* center in S; hence $\rho(i)$ for $i \in S^*$ is the closest center in S to it. Let R_1 be the set of centers in S that have exactly one center in S^* mapped to them. Let R_0 be the set of centers in S with no centers of S^* mapped to them. This means that for each $i \in S \setminus (R_0 \cup R_1)$ there are two or more centers mapped to them. Let $S_1^* \subseteq S^*$ be the centers mapped to R_1. See Figure 9.2. By a simple averaging argument we have the following claim.

Claim 9.3.1. $2|R_0| \geq |S^* \setminus S_1^*|$.

We create a set of pairs $\mathcal{P}$ as follows. There will be exactly k pairs. For each $f^* \in S_1^*$ we add the pair (r, f^*) where $\rho(f^*) = r$. For each $f^* \in S^* \setminus S_1^*$ we add a pair (r, f^*) where r is any arbitrary center from R_0 — however we make sure that a center $r \in R_0$ is in at most two pairs in $\mathcal{P}$; this is possible because of Claim 9.3.1.

The pairs satisfy the following property.

Claim 9.3.2. *If $(r, f^*) \in \mathcal{P}$ then for any facility $\hat{f}^* \neq f^*$, $\rho(\hat{f}^*) \neq r$.*

The intuition for the pairs is as follows. If $\rho(f^*) = \{r\}$ then we are essentially forced to consider the pair (r, f^*) since r could be the only center near f^* with all other centers from S very far away. When considering the swap (r, f^*) we

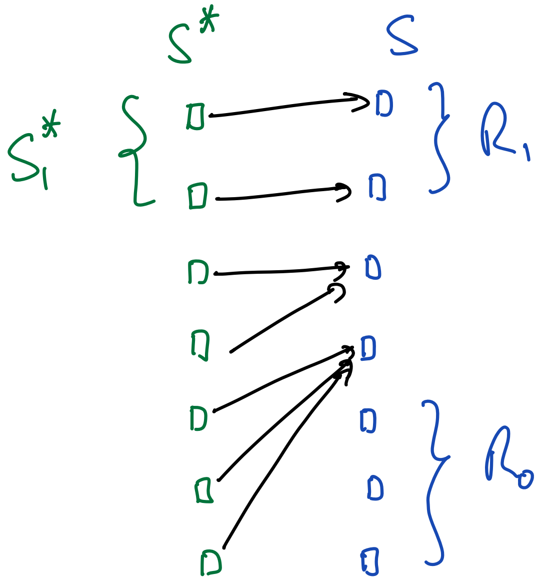


Figure 9.2: Mapping between S^* and S with each $f^* \in S^*$ mapped to its closest center in S.

can move the clients connecting to r to f^*. On the other hand if $|\rho^{-1}(r)| > 1$ then r is close to several centers in S^* and may be serving many clients. Thus we do not consider such an r in the swap pairs.

The main technical claim about the swap pairs is the following.

Lemma 9.10. *Let (r, f^*) be a pair in $\mathcal{P}$. Then*

$$0 \le cost(S + f^* - r) - cost(S) \le \sum_{j \in N^*(f^*)} (O_j - A_j) + \sum_{j \in N(r)} 2O_j.$$

We defer the proof of the lemma for now and use it to show that cost$(S) \le$ 5cost(S^*). We sum over all pairs $(r, f^*) \in \mathcal{P}$ and note that each $f^* \in S$ occurs in exactly one pair and each $r \in S$ occurs in at most two pairs. Note that $O_j - A_j$

can be a negative number while O_j is non-negative number.

$$\begin{aligned}
0 &\leq \sum_{(r,f^*)\in\mathcal{P}} \left(\sum_{j\in N^*(f^*)} (O_j - A_j) + \sum_{j\in N(r)} 2O_j \right) \\
&\leq \sum_{f^*\in S^*} \sum_{j\in N^*(f^*)} (O_j - A_j) + 2\sum_{r\in S} \sum_{j\in N(r)} 2O_j \\
&\leq \text{cost}(S^*) - \text{cost}(S) + 4\text{cost}(S^*).
\end{aligned}$$

This implies the desired inequality.

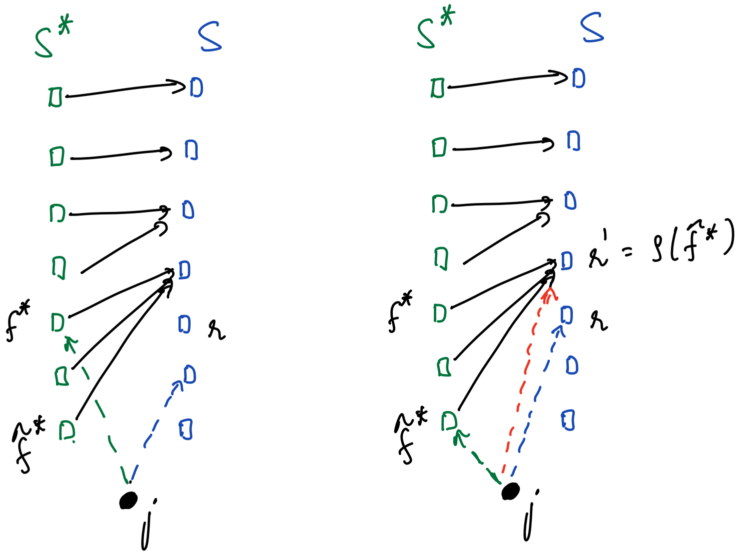


Figure 9.3: Two cases in proof of Lemma 9.10. Consider the swap pair (r, f^*) In the figure on the left the client $j \in N^*(f^*)$ is assigned to f^*. In the figure on the right the client $j \in N(r) \setminus N^*(f^*)$ is is assigned to $r' = \rho(\hat{f}^*)$.

Proof of Lemma 9.10. Since S is a local optimum swapping r with f^* cannot improve the cost and hence we obtain $\text{cost}(S + f^* - r) - \text{cost}(S) \geq 0$. We focus on the more interesting inequality. To bound the increase in cost by removing r and adding f^* to S, we consider a specific assignment of the clients. Any client $j \in N^*(f^*)$ is assigned to f^* (even if it is suboptimal to do so). See Figure 9.3. For such a client j the change in cost is $O_j - A_j$. Now consider any client $j \in N(r) \setminus N^*(f^*)$; since r is no longer available we need to assign it to another facility. Which one? Let $\phi^*(j) = \hat{f}^*$ be the facility that j is assigned to in the optimum solution. Note that $\hat{f}^* \neq f^*$. We assign j to $r' = \rho(\hat{f}^*)$; from Claim 9.3.2, $\rho(\hat{f}^*) \neq r$ and hence $r' \in S - r + f^*$. See Figure 9.3. The change in the cost for such a client j is $d(j, r') - d(j, r)$. We bound it as follows

$$\begin{aligned}
d(j,r') - d(j,r) &\leq d(j,\hat{f}^*) + d(\hat{f}^*, r') - d(j,r) \quad \text{(via triangle inequality)} \\
&\leq d(j,\hat{f}^*) + d(\hat{f}^*, r) - d(j,r) \quad \text{(since r' is closest to $\hat{f}^*$ in S)} \\
&\leq d(j,\hat{f}^*) + d(j,\hat{f}^*) \quad \text{(via triangle inequality)} \\
&= 2O_j.
\end{aligned}$$

Every other client is assigned to its existing center in S. Thus the total increase in the cost is obtained as

$$\sum_{j \in N^*(f^*)} (O_j - A_j) + \sum_{j \in N(r) \setminus N^*(f^*)} 2O_j \leq \sum_{j \in N^*(f^*)} (O_j - A_j) + \sum_{j \in N(r)} 2O_j.$$

■

See [11] for a description of the tight example. The example in the conference version of the paper is buggy.

9.4 *k*-Means

The k-Means problem is very popular in practice for a variety of reasons. In terms of center-based clustering the goal is to choose k centers $C = \{c_1, c_2, \ldots, c_k\}$ to minimize $\sum_p d(p, C)^2$. In the discrete setting one can obtain constant factor approximation algorithms via several techniques that follow the approach of k-Median. We note that the squared distance does *not* satisfy triangle inequality, however, it satisfies a relaxed triangle inequality and this is sufficient to generalize certain LP rounding and local search techniques.

In practice the continuous version is popular for clustering applications. The input points P are in the Euclidean space $\mathbb{R}^d$ where d is typically large. Let X be the set of input points where each $x \in X$ is now a d-dimensional vector. The centers are now allowed to be in ambient space. This is called the Euclidean k-Means. Here the squared distance actually helps in a certain sense. For instance if $k = 1$ then we can see that the optimum center is simply obtained by $\frac{1}{|P|} \sum_{x \in X} x$ — in other words we take the "average". One can see this by considering the problem of finding the center as an optimization problem: $\min_{y \in \mathbb{R}^d} \sum_{x \in X} \|x - y\|_2^2 = \min_{y \in \mathbb{R}^d} \sum_{x \in X} (x_i - y_i)^2$. It can be seen that we can optimize in each dimension separately and that the optimum in dimension

i, can be seen to be $y_i^* = \frac{1}{|X|}x_i$. Surprisingly, hardness results for Euclidean k-Means were only established in the last decade. NP-hardness even when $d = 2$ is established in [118], and APX-hardness for high dimensions was shown in [13].

9.5 Lloyd's algorithm, D^2-sampling and k-Means ++

Llyod's algorithm is a very well-known and widely used heuristic for the Euclidean k-Means problem. It can viewed as a local search algorithm with an alternating optimization flavor. The algorithm starts with a set of centers $c_1, c_2, \ldots, c_k$ which are typically chosen randomly in some fashion. The centers define clusters $C_1, C_2, \ldots, C_k$ based on assigning each point to its nearest center. That is C_i is the set of all points in X that are closest to c_i (ties broken in some fashion). Once we have the clusters, via the observation above, one can find the best center for that cluster by taking the mean of the points in the cluster. That is, for each C_i we find a new center $c'_i = \frac{1}{|C_i|}\sum_{x \in C_i} x$ (if C_i is empty we simply set $c'_i = c_i$). Thus we have a new set of centers and we repeat the process until convergence or some time limit. It is clear that the cost can only improve by recomputing the centers since we know the optimum center for $k = 1$ is obtained by using the average of the points.

```
LLOYDS-k-MEANS($X, k$)

1. Seeding: Pick $k$ centers $c_1, c_2, \ldots, c_k$

2. repeat

    A. Find clusters $C_1, C_2, \ldots, C_k$ where
       $C_i = \{x \in X \mid c_i$ is closest center to $x\}$

    B. cost = $\sum_{i=1}^{k} \sum_{x \in C_i} d(x, c_i)^2$.

    C. For $i = 1$ to $k$ do $c_i = \frac{1}{|C_i|} \sum_{x \in C_i} x$

3. Until cost improvement is too small

4. Output clusters $C_1, C_2, \ldots, C_k$
```

There are two issues with the algorithm. The first issue is that the algorithm can, in the worst-case, run for an exponential number of iterations. This issue is common for many local search algorithms and as we discussed, it can be overcome by stopping when cost improvement is too small in a relative sense. The second issue is the more significant one. The algorithm can get stuck in a local optimum which can be arbitrarily bad when compared to the optimum solution. See figure below for a simple example.

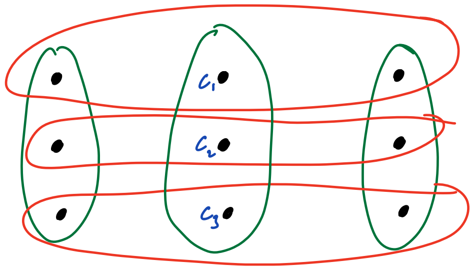


Figure 9.4: Example demonstrating that a local optimum for Lloyd's algorithm can be arbitrarily bad compared to the optimum clustering. The green clusters are the optimum ones and the red ones are the local optimum.

D^2-sampling and k-Means ++: To overcome the bad local optima it is common to run the algorithm with random starting centers. Arthur and Vassilvitskii [151] suggested a specific random sampling scheme to initialize the centers that is closely related to independent work in [130]. This is called D^2 sampling.

```
D²-SAMPLING-k-MEANS ++(X, k)

1. $S = \{c_1\}$ where $c_1$ is chosen uniformly from $X$
2. for $i = 2$ to $k$ do
   A. Choose $c_i$ randomly where $\mathbf{P}[c_i = x] \simeq d(x, S)^2$
   B. $S \leftarrow S \cup \{c_i\}$
3. Output $S$
```

k-Means ++ is Lloyd's algorithm intialized with k centers obtained from D^2 sampling.

Theorem 9.8 ([151]). *Let S be the output of Lloyd's algorithm initialized with D^2 sampling. Then $\mathbf{E}[cost(S)] \le 8(\ln k + 2)\,\mathrm{OPT}$. Moreover there are examples showing that it is no better than $2\ln k$ competitive.*

The analysis establishes that the seeding already creates a good approximation, so in a sense the local search is only refining the initial approximation. [4, 6] show that if one uses $O(k)$ centers, initialized according to D^2 sampling, then the local optimum will yield a constant factor approximation with constant probability; note that this is a bicriteria approximation where the number of centers is a constant factor more than k and the cost is being compared with respect to the optimum cost with k centers. The authors also show that there is a subset of k centers from the output of the algorithm that yields a constant factor approximation. One can then run a discrete optimization algorithm using the centers. Another interesting result based on D^2-sampling ideas yields a PTAS but the running time is of the form $O(nd2^{\tilde{O}(k^2/\epsilon)})$ [92]. See [15] for a scalable version of k-Means ++.

9.6 Bibliographic Notes

Chapter 10

Introduction to Network Design

(Parts of this chapter are based on previous scribed lecture notes by Nitish Korula and Sungjin Im.)

Network Design is a broad topic that deals with finding a subgraph H of a given graph $G = (V, E)$ of minimum cost while satisfying certain requirements. G represents an existing network or a constraint over where one can build. The subgraph H is what we want to select/build. Many natural problems can be viewed this way. For instance the minimum spanning tree (MST) can be viewed as follows: given an undirected graph $G = (V, E)$ with edge costs $c : E \rightarrow \mathbb{R}_+$, find the cheapest connected spanning (spanning means that all vertices are included) subgraph of G. The fact that a minimal solution is a tree is clear, but the point is that the motivation does not explicitly mention the requirement that the output be a tree.

Connectivity problems are a large part of network design. As we already saw MST is the most basic one and can be solved in polynomial time. The Steiner Tree problem is a generalization where we are given a subset S of terminals in an edge-weighted graph $G = (V, E)$ and the goal is to find a cheapest connected subgraph that contains all terminals. This is NP-Hard. Traveling Salesman Problem (TSP) and its variants can also be viewed as network design problems. Network design is heavily motivated by real-world problems in telecommunication networks and those problems combine aspects of connectivity and routing and in this context there are several problems related to buy-at-bulk network design, fixed-charge flow problems etc.

Graph theory plays an important role in most network algorithmic questions. The complexity and nature of the problems vary substantially based on whether the graph is *undirected* or *directed*. To illustrate this consider the Directed Steiner Tree problem. Here $G = (V, E)$ is a directed graph with non-negative edge/arc weights, and we are given a root r and a set of terminals $S \subseteq V$. The goal is to find a cheapest subgraph H of G such that r has a path to each terminal $t \in S$. Note that when $S = V$ the problem is the minimum-cost arborescence problem and is solvable in polynomial-time. One can see that Directed Steiner Tree is a generalization of the (undirected) Steiner Tree problem. While Steiner Tree problem admits a constant factor approximation, it is easy to show that Directed Steiner Tree is at least as hard as Set Cover. This immediately implies that it is hard to approximate to within an $\Omega(\log |S|)$ factor. In fact, substantial technical work has shown that it is in fact harder than Set Cover [76], while we only have a quasi-polynomial time algorithm that gives a poly-logarithmic approximation [34]. In even slightly more general settings, the directed graph problems get much harder when compared to their undirected graph counterparts. This phenomena is generally true in many graph related problems and hence the literature mostly focuses on undirected graphs. We refer the reader to some surveys on network design [72, 107, 129].

This chapter will focus on two basic problems which are extensively studied and describe some simple approximation algorithms for them. Pointers are provided for sophisticated results including some very recent ones.

10.1 The Steiner Tree Problem

In the Steiner Tree problem, the input is a graph $G(V, E)$, together with a set of *terminals* $S \subseteq V$, and a cost $c(e)$ for each edge $e \in E$. The goal is to find a minimum-cost tree that connects all terminals, where the cost of a subgraph is the sum of the costs of its edges.

The Steiner Tree problem is **NP**-Hard, and also **APX**-Hard [22]. The latter means that there is a constant $\delta > 1$ such that it is **NP**-Hard to approximate the solution to within a ratio of less than δ; it is currently known that it is hard to approximate the Steiner Tree problem to within a ratio of 95/94 [42].[1]

Remark 10.1. If $|S| = 2$ (that is, there are only 2 terminals), an optimal Steiner Tree is simply a shortest path between these 2 terminals. If $S = V$ (that is, all vertices are terminals), an optimal solution is simply a minimum spanning tree of the input graph. In both these cases, the problem can be solved exactly in polynomial time.

Remark 10.2. There is $c^k \text{poly}(n)$-time algorithm where k is the number of terminals. Can you figure it out?

[1] Variants of the Steiner Tree problem, named after Jakob Steiner, have been studied by Fermat, Weber, and others for centuries. The front cover of the course textbook contains a reproduction of a letter from Gauss to Schumacher on a Steiner tree question.

Definition 10.1. *Given a connected graph $G(V,E)$ with edge costs, the* metric completion *of G is a complete graph $H(V,E')$ such that for each $u,v \in V$, the cost of edge uv in H is the cost of the shortest path in G from u to v. The graph H with edge costs is a* metric *on V, because the edge costs satisfy the triangle inequality: $\forall u,v,w, \quad cost(uv) \leq cost(uw) + cost(wv)$.*

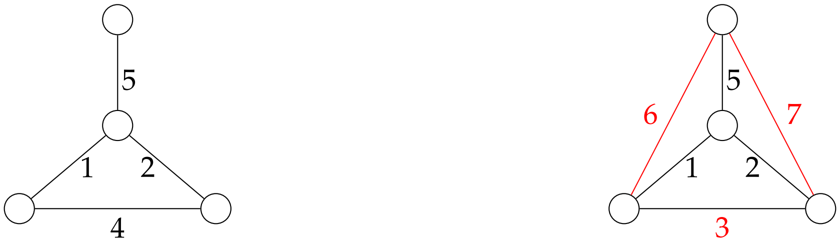


Figure 10.1: On the left, a graph. On the right, its metric completion, with new edges and modified edge costs in red.

Observation 10.2. *To solve the* Steiner Tree *problem on a graph G, it suffices to solve it on the metric completion of G.*

We now look at two approximation algorithms for the Steiner Tree problem.

10.1.1 The MST Algorithm

The following algorithm, with an approximation ratio of $(2 - 2/|S|)$ is due to [148]:

```
SteinerMST($G(V,E), S \subseteq V$):
Let $H(V,E') \leftarrow$ metric completion of $G$.
Let $T \leftarrow$ MST of $H[S]$.
Output $T$.
```

(Here, we use the notation $H[S]$ to denote the subgraph of H induced by the set of terminals S.)

The following lemma is central to the analysis of the algorithm SteinerMST.

Lemma 10.1. *For any instance I of* Steiner Tree, *let H denote the metric completion of the graph, and S the set of terminals. There exists a spanning tree in $H[S]$ (the graph induced by terminals) of cost at most $2(1-\frac{1}{|S|})\,\mathrm{OPT}$, where OPT is the cost of an optimal solution to instance I.*

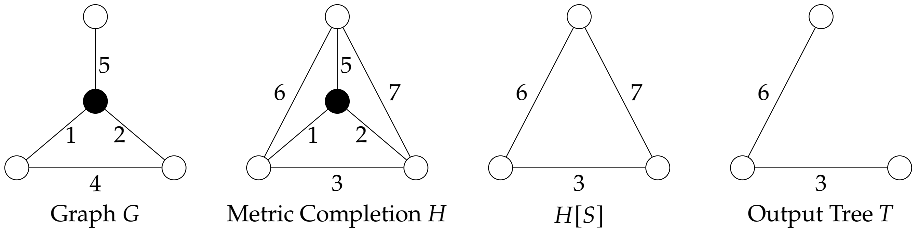


Figure 10.2: Illustrating the MST Heuristic for Steiner Tree

Before we prove the lemma, we note that if there exists *some spanning tree* in $H[S]$ of cost at most $2(1 - \frac{1}{|S|})\,\mathrm{OPT}$, the minimum spanning tree has at most this cost. Therefore, Lemma 10.1 implies that the algorithm SteinerMST is a $2(1 - \frac{1}{|S|})$-approximation for the Steiner Tree problem.

Proof. Proof of Lemma 10.1 Let T^* denote an optimal solution in H to the given instance, with cost $c(T^*)$. Double all the edges of T^* to obtain an Eulerian graph, and fix an Eulerian Tour W of this graph. See Fig 10.3. Now, shortcut edges of W to obtain a tour W' of the vertices in T^* in which each vertex is visited exactly once. Again, shortcut edges of W' to eliminate all non-terminals; this gives a walk W'' that visits each terminal exactly once.

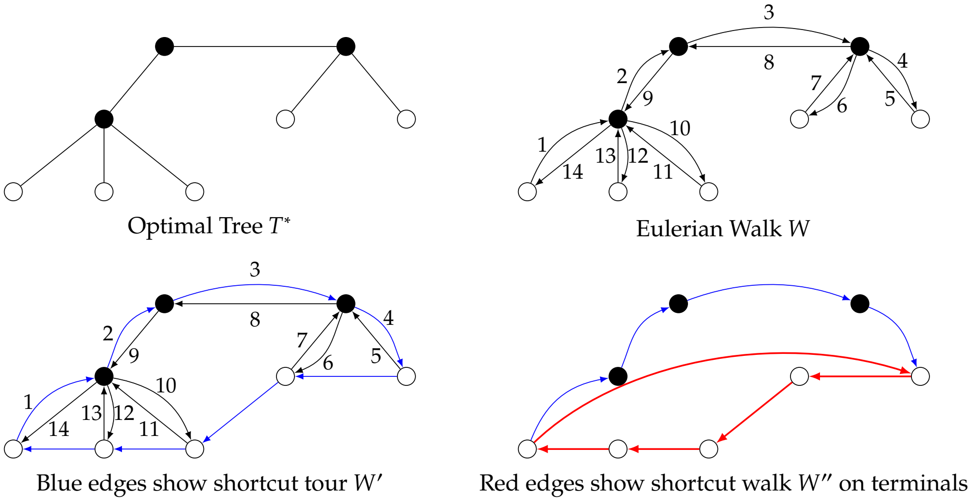
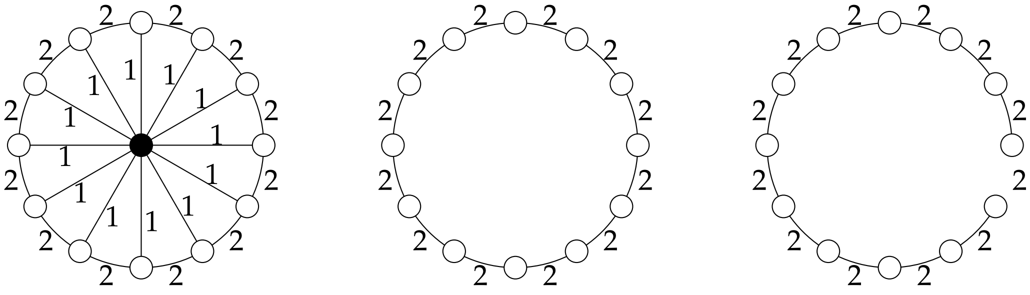

Optimal Tree T^*
3
8
2
9
7
4
6
5
1
10
14
13
12
11
Eulerian Walk W
3
8
2
4
9
7
6
5
10
1
13
12
14
11
Blue edges show shortcut tour W'
Red edges show shortcut walk W'' on terminals

Figure 10.3: Doubling edges of T^* and shortcutting gives a low-cost spanning tree on terminals.

It is easy to see that $c(W'') \leq c(W') \leq c(W) = 2c(T^*)$, where the inequalities follow from the fact that by shortcutting, we can only decrease the length of the walk. (Recall that we are working in the metric completion H.) Now, delete the heaviest edge of W'' to obtain a path through all the terminals in S, of cost at most $(1 - \frac{1}{|S|})c(W'')$. This path is a spanning tree of the terminals, and contains *only* terminals; therefore, there exists a spanning tree in $H[S]$ of cost at most $2(1 - \frac{1}{|S|})c(T^*)$. ■

A tight example: The following example (Fig. 4 below) shows that this analysis is tight; there are instances of Steiner Tree where the SteinerMST algorithm finds a tree of cost $2(1 - \frac{1}{S})$ OPT. Here, each pair of terminals is connected by an edge of cost 2, and each terminal is connected to the central non-terminal by an edge of cost 1. The optimal tree is a star containing the central non-terminal, with edges to all the terminals; it has cost $|S|$. However, the only trees in $H[S]$ are formed by taking $|S| - 1$ edges of cost 2; they have cost $2(|S| - 1)$.



Graph G; not all edges shown $H[S]$; not all edges shown. An MST of $H[S]$.

Figure 10.4: A tight example for the SteinerMST algorithm

10.1.2 The Greedy/Online Algorithm

We now describe another simple algorithm for the Steiner Tree problem, due to [89].

```
GreedySteiner($G(V,E), S \subseteq V$):
Let $\{s_1, s_2, \ldots s_{|S|}\}$ be an arbitrary ordering of the terminals.
Let $T \leftarrow \{s_1\}$
For ($i$ from 2 to $|S|$):
    Let $P_i$ be the shortest path in $G$ from $s_i$ to $T$.
    Add $P_i$ to $T$.
```

GreedySteiner is a $\lceil \log_2 |S| \rceil$-approximation algorithm; here, we prove a slightly weaker result.

Theorem 10.3. *The algorithm* GreedySteiner *has an approximation ratio of* $2H_{|S|} \approx 2\ln|S|$, *where* $H_i = \sum_{j=1}^{i} 1/j$ *denotes the i'th harmonic number.*

Note that this is an *online* algorithm; terminals are considered in an arbitrary order, and when a terminal is considered, it is immediately connected to the existing tree. Thus, even if the algorithm *could not see the entire input at once*, but instead terminals were revealed one at a time and the algorithm had to produce a Steiner tree at each stage, the algorithm GreedySteiner outputs a tree of cost no more than $O(\log|S|)$ times the cost of the optimal tree.

To prove Theorem 10.3, we introduce some notation. Let $c(i)$ denote the cost of the path P_i used in the ith iteration to connect the terminal s_i to the already existing tree. Clearly, the total cost of the tree is $\sum_{i=1}^{|S|} c(i)$. Now, let $\{i_1, i_2, \ldots i_{|S|}\}$ be a permutation of $\{1, 2, \ldots |S|\}$ such that $c(i_1) \geq c(i_2) \geq \ldots \geq c(i_{|S|})$. (That is, relabel the terminals in decreasing order of the cost paid to connect them to the tree that exists when they are considered by the algorithm.)

Claim 10.1.1. *For all* j, *the cost* $c(i_j)$ *is at most* $2\,\mathrm{OPT}/j$, *where* OPT *is the cost of an optimal solution to the given instance.*

Proof. Suppose by way of contradiction this were not true; since s_{i_j} is the terminal with jth highest cost of connection, there must be j terminals that each pay more than $2\,\mathrm{OPT}/j$ to connect to the tree that exists when they are considered. Let $S' = \{s_{i_1}, s_{i_2}, \ldots s_{i_j}\}$ denote this set of terminals.

We argue that no two terminals in $S' \cup \{s_1\}$ are within distance $2\,\mathrm{OPT}/j$ of each other. If some pair x, y were within this distance, one of these terminals (say y) must be considered later by the algorithm than the other. But then the cost of connecting y to the already existing tree (which includes x) must be at most $2\,\mathrm{OPT}/j$, and we have a contradiction.

Therefore, the minimum distance between any two terminals in $S' \cup \{s_1\}$ must be greater than $2\,\mathrm{OPT}/j$. Since there must be j edges in any MST of these terminals, an MST must have cost greater than $2\,\mathrm{OPT}$. But the MST of a subset of terminals cannot have cost more than $2\,\mathrm{OPT}$, exactly as argued in the proof of Lemma 10.1. Therefore, we obtain a contradiction. ∎

Given this claim, it is easy to prove Theorem 10.3.

$$\sum_{i=1}^{|S|} c(i) = \sum_{j=1}^{|S|} c(i_j) \leq \sum_{j=1}^{|S|} \frac{2\,\mathrm{OPT}}{j} = 2\,\mathrm{OPT} \sum_{j=1}^{|S|} \frac{1}{j} = 2H_{|S|} \cdot \mathrm{OPT}\,.$$

Question 10.2. Give an example of a graph and an ordering of terminals such that the output of the Greedy algorithm is $\Omega(\log|S|)\,\mathrm{OPT}$.

Remark 10.3. We emphasize again that the analysis above holds for *every* ordering of the terminals. A natural variant might be to adaptively order the terminals so that in each iteration i , the algorithm picks the terminal s_i to be the one closest to the already existing tree T built in the first i iterations. Do you see that this is equivalent to using the MST Heuristic with Prim's algorithm for MST? This illustrates the need to be careful in the design and analysis of heuristics.

10.1.3 LP Relaxation

A natural LP relaxation for the Steiner Tree problem is the following. For each edge $e \in E$ we have an indicator variable x_e to decide if we choose to include e in our solution. The chosen edges should ensure that no two terminals are separated. We write this via a constraint $\sum_{e\in\delta(A)} x_e \geq 1$ for any set $A \subset V$ such that A contains a terminal and $V \setminus A$ contains a terminal.

$$\begin{aligned}
&\min_{e\in E} c_e x_e \\
&\sum_{e\in\delta(A)} x_e \quad \geq \quad 1 \quad A \cap S \neq \emptyset, (V-A)\cap S \neq \emptyset \\
&\qquad\quad x_e \quad \geq \quad 0
\end{aligned}$$

Note that the preceding LP has an exponential number of constraints. However, there is a polynomial-time separation oracle. Given x it is feasible for the LP iff the s-t cut value is at least 1 between any two terminals $s, t \in S$ with edge capacities given by x. How good is this LP relaxation? We will see later that there is a $2(1 - 1/|S|)$-approximation via this LP. Interestingly the LP has an integrality gap of $2(1 - 1/|S|)$ even if $S = V$ in which case we want to solve the MST problem! Despite the weakness of this cut based LP for these simple cases, we will see later that it generalizes nicely for higher connectivity problems and one can derive a 2-approximation even for those much more difficult problems.

10.1.4 Other Results on Steiner Trees

The 2-approximation algorithm using the MST Heuristic is not the best approximation algorithm for the Steiner Tree problem currently known. Some other results on this problem are listed below.

1. The first algorithm to obtain a ratio of better than 2 was due to due to Alexander Zelikovsky [160]; the approximation ratio of this algorithm was $11/6 \approx 1.83$. This was improved to $1 + \frac{\ln 3}{2} \approx 1.55$ [135] and is based on a local search based improvement starting with the MST heuristic, and follows the original approach of Zelikovsky.

2. Byrka *et al* gave an algorithm with an approximation ratio of $1.39 = \ln 4 + \epsilon$ [26] which is currently the best known for this problem. This was originally based on a combination of techniques and subsequently there is an LP based proof [65] that achieves the same approximation for the so-called Hypergraphic LP relaxation.

3. The *bidirected cut* LP relaxation for the Steiner Tree was proposed by [52]; it has an integrality gap of at most $2(1 - \frac{1}{|S|})$, but it is conjectured that the gap is smaller. No algorithm is currently known that exploits this LP relaxation to obtain an approximation ratio better than that of the SteinerMST algorithm. Though the true integrality gap is not known, there are examples that show it is at least $6/5 = 1.2$ [153].

4. For many applications, the vertices can be modeled as points on the plane, where the distance between them is simply the Euclidean distance. The MST-based algorithm performs fairly well on such instances; it has an approximation ratio of $2/\sqrt{3} \approx 1.15$ [51]. An example which achieves this bound is three points at the corners of an equilateral triangle, say of side-length 1; the MST heuristic outputs a tree of cost 2 (two sides of the triangle) while the optimum solution is to connect the three points to a Steiner vertex which is the circumcenter of the triangle. One can do better still for instances in the plane (or in any Euclidean space of small-dimensions); for any $\epsilon > 0$, there is a $1 + \epsilon$-approximation algorithm that runs in polynomial time [10]. Such an approximation scheme is also known for planar graphs [23] and more generally bounded-genus graphs.

10.2 The Traveling Salesperson Problem (TSP)

10.2.1 TSP in Undirected Graphs

In the Traveling Salesperson Problem (TSP), we are given an undirected graph $G = (V, E)$ and cost $c(e) > 0$ for each edge $e \in E$. Our goal is to find a Hamiltonian cycle with minimum cost. A cycle is said to be Hamiltonian if it visits every vertex in V exactly once.

TSP is known to be NP-Hard. Moreover, we cannot hope to find a good approximation algorithm for it unless $P = NP$. This is because if one can give a good approximation solution to TSP in polynomial time, then we can exactly solve the NP-Complete Hamiltonian cycle problem (HAM) in polynomial time, which is not possible unless $P = NP$. Recall that HAM is a decision problem: given a graph $G = (V, E)$, does G have a Hamiltonian cycle?

Theorem 10.4 ([136])**.** *Let $\alpha : \mathbb{N} \to \mathbb{N}$ be a polynomial-time computable function. Unless $P = NP$ there is no polynomial-time algorithm that on every instance I of TSP outputs a solution of cost at most $\alpha(|I|) \cdot \mathrm{OPT}(I)$.*

Proof. For the sake of contradiction, suppose we have an approximation algorithm $\mathcal{A}$ for TSP with an approximation ratio $\alpha(|I|)$. We show a contradiction by showing that using $\mathcal{A}$, we can exactly solve HAM in polynomial time. Let $G = (V, E)$ be the given instance of HAM. We create a new graph $H = (V, E')$ with cost $c(e)$ for each $e \in E'$ such that $c(e) = 1$ if $e \in E$, otherwise $c(e) = B$, where $B = n\alpha(n) + 2$ and $n = |V|$. Note that this is a polynomial-time reduction since α is a polynomial-time computable function.

We observe that if G has a Hamiltonian cycle, OPT $= n$, otherwise OPT $\geq n - 1 + B \geq n\alpha(n) + 1$. (Here, OPT denotes the cost of an optimal TSP solution in H.) Note that there is a "gap" between when G has a Hamiltonian cycle and when it does not. Thus, if $\mathcal{A}$ has an approximation ratio of $\alpha(n)$, we can tell whether G has a Hamiltonian cycle or not: Simply run $\mathcal{A}$ on the graph H; if $\mathcal{A}$ returns a TSP tour in H of cost at most $\alpha(n)n$ output that G has a Hamilton cycle, otherwise output that G has no Hamilton cycle. We leave it as an exercise to formally verify that this would solve HAM in polynomial time. ■

Since we cannot even approximate the general TSP problem, we consider more tractable variants.

- *Metric-TSP*: In Metric-TSP, the instance is a complete graph $G = (V, E)$ with cost $c(e)$ on $e \in E$, where c satisfies the triangle inequality, i.e. $c(uw) \leq c(uv) + c(vw)$ for any $u, v, w \in V$.
- *TSP-R*: TSP with repetitions of vertices allowed. The input is a graph $G = (V, E)$ with non-negative edge costs as in TSP. Now we seek a minimum-cost *walk* that visits each vertex at least once and returns to the starting vertex.

Exercise 10.1. Show that an α-approximation for Metric-TSP implies an α-approximation for TSP-R and vice-versa.

We focus on Metric-TSP for the rest of this section. We first consider a natural greedy approach, the Nearest Neighbor Heuristic (NNH).

```
NEAREST NEIGHBOR HEURISTIC($G(V,E), c : E \to \mathcal{R}^+$):
Start at an arbitrary vertex $s$,
While (there are unvisited vertices)
  From the current vertex $u$, go to the nearest unvisited vertex $v$.
Return to $s$.
```

Exercise 10.2. 1. Prove that NNH is an $O(\log n)$-approximation algorithm. (**Hint:** Think back to the proof of the $2H_{|S|}$-approximation for the Greedy Steiner Tree Algorithm.)

2. NNH is not an $O(1)$-approximation algorithm; can you find an example to show this? In fact one can show a lower bound of $\Omega(\log n)$ on the approximation-ratio achieved by NNH.

There are constant-factor approximation algorithms for TSP; we now consider an MST-based algorithm. See Fig 10.5.

```
TSP-MST($G(V, E), c : E \rightarrow \mathcal{R}^+$):
Compute an MST $T$ of $G$.
Obtain an Eulerian graph $H = 2T$ by doubling edges of $T$
An Eulerian tour of $2T$ gives a tour in $G$.
Obtain a Hamiltonian cycle by shortcutting the tour.
```

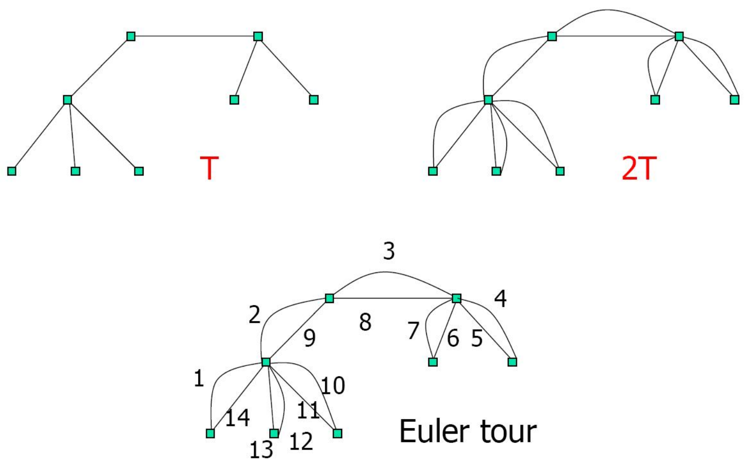


Figure 10.5: MST Based Heuristic

Theorem 10.5. *MST heuristic(TSP-MST) is a 2-approximation algorithm.*

Proof. We have $c(T) = \sum_{e \in E(T)} c(e) \leq \text{OPT}$, since we can get a spanning tree in G by removing any edge from the optimal Hamiltonian cycle, and T is a MST. Thus $c(H) = 2c(T) \leq 2\,\text{OPT}$. Also shortcutting only decreases the cost. ■

We observe that the loss of a factor 2 in the approximation ratio is due to doubling edges; we did this in order to obtain an Eulerian tour. But any graph in which all vertices have even degree is Eulerian, so one can still get an Eulerian tour by adding edges only between odd degree vertices in T. Christofides Heuristic [43] exploits this and improves the approximation ratio from 2 to 3/2. See Fig 10.6 for a snapshot.

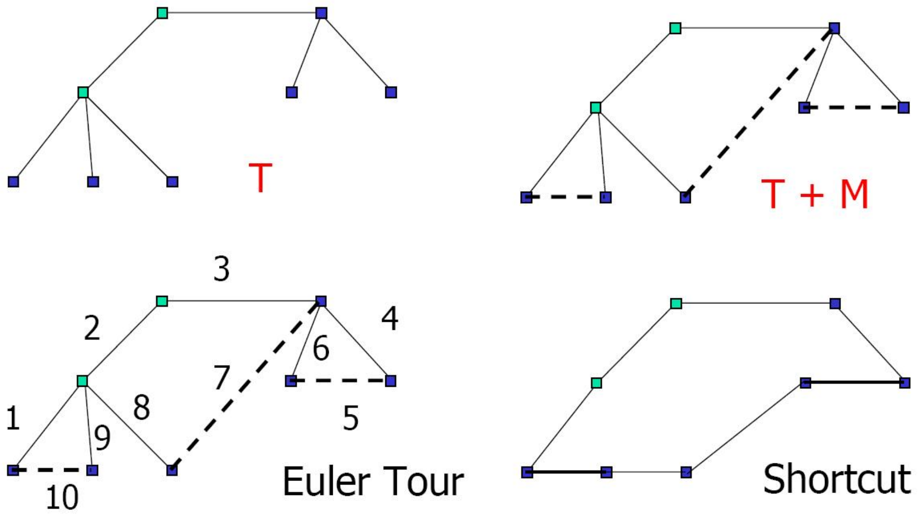


Figure 10.6: Christofides Heuristic

```
CHRISTOFIDES HEURISTIC($G(V, E), c : E \to \mathcal{R}^+$):
Compute an MST $T$ of $G$.
Let $S$ be the vertices of odd degree in $T$. (Note: $|S|$ is even)
Find a minimum cost matching $M$ on $S$ in $G$
Add $M$ to $T$ to obtain an Eulerian graph $H$.
Compute an Eulerian tour of $H$.
Obtain a Hamilton cycle by shortcutting the tour.
```

Theorem 10.6. *Christofides Heuristic is a 1.5-approximation algorithm.*

Proof. The main part of the proof is to show that $c(M) \leq .5\,\text{OPT}$. Suppose that $c(M) \leq .5\,\text{OPT}$. Then, since the solution of Christofides Heuristic is obtained by shortcutting the Eulerian tour on H, its cost is no more than $c(H) = c(T) + c(M) \leq 1.5\,\text{OPT}$. (Refer to the proof of Lemma 10.5 for the fact $c(T) \leq \text{OPT}$.) Therefore we focus on proving that $c(M) \leq .5\,\text{OPT}$.

Let F^* be an optimal tour in G of cost OPT; since we have a metric-instance we can assume without loss of generality that F^* is a Hamiltonian cycle. We obtain a Hamiltonian cycle F^*_S in the graph $G[S]$ by short-cutting the portions of F^*

that touch the vertices $V \setminus S$. By the metric-condition, $c(F^*_S) \leq c(F^*) = \mathrm{OPT}$. Let $S = \{v_1, v_2, \ldots, v_{|S|}\}$. Without loss of generality F^*_S visits the vertices of S in the order $v_1, v_2, \ldots, v_{|S|}$. Recall that $|S|$ is even. Let $M_1 = \{v_1v_2, v_3v_4, ...v_{|S|-1}v_{|S|}\}$ and $M_2 = \{v_2v_3, v_4v_5, ...v_{|S|}v_1\}$. Note that both M_1 and M_2 are matchings, and $c(M_1) + c(M_2) = c(F^*_S) \leq \mathrm{OPT}$. We can assume without loss of generality that $c(M_1) \leq c(M_2)$. Then we have $c(M_1) \leq .5\,\mathrm{OPT}$. Also we know that $c(M) \leq c(M_1)$, since M is a minimum cost matching on S in $G[S]$. Hence we have $c(M) \leq c(M_1) \leq .5\,\mathrm{OPT}$, which completes the proof. ■

10.2.2 LP Relaxation

We describe a well-known LP relaxation for TSP called the Subtour-Elimination LP and sometimes also called the Held-Karp LP relaxation although the formulation was first given by Dantzig, Fulkerson and Johnson [47]. The LP relaxation has a variable x_e for each edge $e \in E$. Note that the TSP solution is a Hamilton Cycle of least cost. A Hamilton cycle can be viewed as a *connected* subgraph of G with degree 2 at each vertex. Thus we write the degree constraints and also the cut constraints.

$$\begin{aligned}
&\min_{e \in E} c_e x_e \\
&\sum_{e \in \delta(v)} x_e = 2 \quad v \in V \\
&\sum_{e \in \delta(S)} x_e \geq 2 \quad \emptyset \subsetneq S \subsetneq V \\
&x_e \in [0,1] \quad e \in E
\end{aligned}$$

The relaxation is not useful for a general graph since we saw that TSP is not approximable. To obtain a relaxation for Metric-TSP we apply the above to the metric completion of the graph G.

Another alternative is to consider the following LP which view the problem as finding a connected Eulerian multi-graph of the underlying graph G. In other words we are allowed to take an integer number of copies of each edge with the constraint that the degree of each vertex is even and the graph is connected. It is not easy to write the even degree condition since we do not have an apriori bound. Instead one can write the following simpler LP, and interestingly one can show that its optimum value is the same as that of the preceding relaxation (when applied to the metric completion).

$$\begin{array}{rcll} \min_{e \in E} c_e x_e & & & \\ \sum_{e \in \delta(S)} x_e & \geq & 2 & \emptyset \subsetneq S \subsetneq V \\ x_e & \in & 0 & e \in E \end{array}$$

Wolsey showed that the 3/2-approximation of Christofides can be analyzed with respect to the LP above. Hence the integrality gap of the LP is at most 3/2 for Metric-TSP. Is it better? There is a well-known example which shows that the gap is at least 4/3. The 4/3 conjecture states that the worst-case integrality gap is at most 4/3. This has been an unsolved problem for many decades and it is very very recently that the 3/2 barrier was broken.

Remarks:

1. In practice, local search heuristics are widely used and they perform extremely well. A popular heuristic *2-Opt* is to swap pairs from xy, zw to xz, yw or xw, yz, if it improves the tour.

2. It was a major open problem to improve the approximation ratio of $\frac{3}{2}$ for Metric-TSP; it is conjectured that the Held-Karp LP relaxation [81] gives a ratio of $\frac{4}{3}$. In a breakthrough Oveis-Gharan, Saberi and Singh [64] obtained a $3/2 - \delta$ approximation for some small but fixed $\delta > 0$ for the important special case where $c(e) = 1$ for each edge e (called Graphic-TSP). Very recently the 3/2 ratio was finally broken for the general case [99].

10.2.3 TSP in Directed Graphs

In this subsection, we consider TSP in directed graphs. As in undirected TSP, we need to relax the problem conditions to get any positive result. Again, allowing each vertex to be visited multiple times is equivalent to imposing the asymmetric triangle inequality $c(u, w) \leq c(u, v) + c(v, w)$ for all u, v, w. This is called the asymmetric TSP (ATSP) problem. We are given a directed graph $G = (V, A)$ with cost $c(a) > 0$ for each arc $a \in A$ and our goal is to find a closed walk visiting all vertices. Note that we are allowed to visit each vertex multiple times, as we are looking for a walk, not a cycle. For an example of a valid Hamiltonian walk, see Fig 10.7.

The MST-based heuristic for the undirected case has no meaningful generalization to the directed setting This is because costs on edges are not symmetric. Hence, we need another approach. The *Cycle Shrinking Algorithm* repeatedly finds a min-cost cycle cover and shrinks cycles, combining the cycle covers

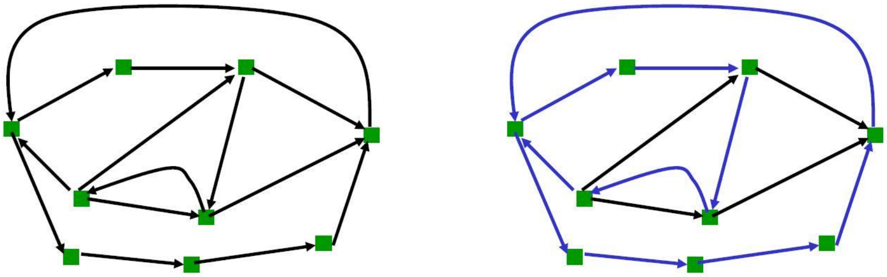


Figure 10.7: A directed graph and a valid Hamiltonian walk

found. Recall that a cycle cover is a collection of disjoint cycles covering all vertices. It is known that finding a minimum-cost cycle cover can be done in polynomial time (see Homework 0). The Cycle Shrinking Algorithm achieves a $\log_2 n$ approximation ratio.

```
CYCLE SHRINKING ALGORITHM($G(V, A), c : A \rightarrow \mathcal{R}^+$):
Transform $G$ s.t. $G$ is complete and satisfies $c(u, v) + c(v, w) \geq c(u, w)$ for $\forall u, v, w$
If $|V| = 1$ output the trivial cycle consisting of the single node
Find a minimum cost cycle cover with cycles $C_1, \ldots, C_k$
From each $C_i$ pick an arbitrary proxy node $v_i$
Recursively solve problem on $G[\{v_1, \ldots, v_k\}]$ to obtain a solution $C$
$C' = C \cup C_1 \cup C_2 \ldots C_k$ is a Eulerian graph.
Shortcut $C'$ to obtain a cycle on $V$ and output $C'$.
```

For a snapshot of the Cycle Shrinking Algorithm, see Fig 10.8.

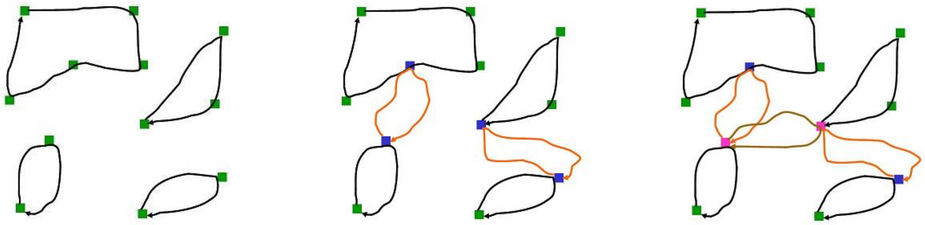


Figure 10.8: A snapshot of Cycle Shrinking Algorithm. To the left, a cycle cover C_1. In the center, blue vertices indicate proxy nodes, and a cycle cover C_2 is found on the proxy nodes. To the right, pink vertices are new proxy nodes, and a cycle cover C_3 is found on the new proxy nodes.

Lemma 10.3. *Let the cost of edges in G satisfy the asymmetric triangle inequality. Then for any $S \subseteq V$, the cost of an optimal TSP tour in $G[S]$ is at most the cost of an optimal TSP tour in G.*

Proof. Since G satisfies the triangle inequality there is an optimal tour TSP tour in G that is a Hamiltonian cycle C. Given any $S \subseteq V$ the cycle C can be short-cut to produce another cycle C' that visits only S and whose cost is at most the cost of C. ■

Lemma 10.4. *The cost of a min-cost cycle-cover is at most the cost of an optimal TSP tour.*

Proof. An optimal TSP tour is a cycle cover. ■

Theorem 10.7. *The Cycle Shrinking Algorithm is a $\log_2 n$-approximation for ATSP.*

Proof. We prove the above by induction on n the number of nodes in G. It is easy to see that the algorithm finds an optimal solution if $n \leq 2$. The main observation is that the number of cycles in the cycle-cover is at most $\lfloor n/2 \rfloor$; this follows from the fact that each cycle in the cover has to have at least 2 nodes and they are disjoint. Thus $k \leq \lfloor n/2 \rfloor$. Let $\mathrm{OPT}(S)$ denote the cost of an optimal solution in $G[S]$. From Lemma 10.3 we have that $\mathrm{OPT}(S) \leq \mathrm{OPT}(V) = \mathrm{OPT}$ for all $S \subseteq V$. The algorithm recurses on the proxy nodes $S = \{v_1, \ldots, v_k\}$. Note that $|S| < n$, and by induction, the cost of the cycle C output by the recursive call is at most $(\log_2 |S|)\,\mathrm{OPT}(S) \leq (\log_2 |S|)\,\mathrm{OPT}$.

The algorithm outputs C' whose cost is at most the cost of C plus the cost of the cycle-cover computed in G. The cost of the cycle cover is at most OPT (Lemma 10.4). Hence the cost of C' is at most $(\log_2 |S|)\,\mathrm{OPT} + \mathrm{OPT} \leq (\log_2 n/2)\,\mathrm{OPT} + \mathrm{OPT} \leq (\log_2 n)\,\mathrm{OPT}$; this finishes the inductive proof. ■

10.2.4 LP Relaxation

The LP relaxation for ATSP is given below. For each arc $e \in E$ we have a variable x_e. We view the problem as finding a connected Eulerian multi-graph in the support of G. That is, we can choose each edge e an integer number of times. We impose Eulerian constraint at each vertex by requiring the in-degree to be equal to the out-degree. We impose connectivity constraint by ensuring that at least one arc leaves each set of vertices S which is not V or $\emptyset$.

$$\begin{array}{rcll} & \min_{e\in E} c_e x_e & & \\ \sum_{e\in\delta^+(v)} x_e - \sum_{e\in\delta^-(v)} x_e & = & 0 & v \in V \\ \sum_{e\in\delta^+(S)} x_e & \geq & 1 & \emptyset \subsetneq S \subsetneq V \\ x_e & \geq & 0 & e \in E \end{array}$$

Remarks:

1. It has remained an open problem for more than 25 years whether there exists a constant factor approximation for ATSP. Asadpour *et al* [12] have obtained an $O(\log n/\log\log n)$-approximation for ATSP using some very novel ideas and a well-known LP relaxation.

2. There is now an $O(1)$-approximation for ATSP with initial breakthrough work by Svensson for a special case [144] and then followed up by Svensson, Tarnawski and Vegh [146] . The current best constant is $22+\epsilon$ due to [150]. The algorithm is highly non-trivial and is based on the LP relaxation.

Chapter 11

Steiner Forest Problem

We discuss a primal-dual based 2-approximation for the Steiner Forest problem[1]. In the Steiner Forest problem there is a graph $G = (V, E)$ where each edge $e \in E$ has a cost $c_e \in \mathbb{R}$. We are given k pairs of vertices $(s_1, t_1), (s_2, t_2), \ldots (s_k, t_k) \in V \times V$. The goal is to find the minimum cost set of edges F such that in the graph (V, F) the vertices s_i and t_i are in the same connected component for $1 \leq i \leq k$. We refer to the set $\{s_1, \ldots, s_k, t_1, \ldots, t_k\}$ as terminals. Notice that the graph (V, F) can contain multiple connected components.

One can see that Steiner Forest generalizes the Steiner Tree problem. It is, however, much less easy to obtain a constant factor approximation for Steiner Forest. A natural greedy heuristic, similar to that for the online Steiner Tree problem is the following. We order the the pairs in some arbitrary fashion, say as 1 to k without loss of generality. We maintain a forest F of edges that we have already bought. When considering pair i we find a shortest path P from s_i to t_i in the graph in which we reduce the cost of each edge in F to 0 (since we already paid for them). We add the new edges in P to F. Although simple to describe, the algorithm is not straight forward to analyze. In fact the best bound we have on the performance of this algorithm is $O(\log^2 k)$ (note that the corresponding bound for Steiner Tree tree is $O(\log k)$) while the lower bound on its peformance is only $\Omega(\log k)$. The analysis of the upper bound requires appealing to certain extremal results [14] and closing the gap between the upper and lower bound on the analysis of this simplest of greedy algorithms is a very interesting open problem.

The first constant factor approximation for Steiner Forest is due to the influential work of Agarwal, Klein and Ravi [5] who gave a primal-dual 2-approximation which is still the best known. Their primal-dual algorithm has since been generalized for a wide variety of network design problems via the

[1] Parts of this chapter are based on past scribed lecture notes by Ben Moseley from 2009.

work of Goemans and Williamson (see their survey [66]) and several others. Steiner Forest has the advantage that one can visualize the algorithm and analysis more easily when compared to the more abstract settings that we will see shortly.

We now describe an integer programming formulation for the problem. In the **IP** we will have a variable x_e for each edge $e \in E$ such that x_e is 1 if and only if e is part of the solution. Let the set $\mathcal{S}$ be the collection of all sets $S \subset V$ such that $|S \cap \{s_i, t_i\}| = 1$ for some $1 \leq i \leq k$. For a set $S \subset V$ let $\delta(S)$ denote the set of edges crossing the cut $(S, V \setminus S)$. The **IP** can be written as the following.

$$\begin{aligned} \text{min} \quad & \sum_{e \in E} c_e x_e \\ \text{such that} \quad & \sum_{e \in \delta(S)} x_e \geq 1 \quad \forall S \in \mathcal{S} \\ & x_e \in \{0, 1\} \quad \forall e \in E \end{aligned}$$

We can obtain an **LP**-relaxation by changing the constraint that $x_e \in \{0, 1\}$ to $x_e \geq 0$. The dual of the **LP**-relaxation can be written as the following.

$$\begin{aligned} \text{max} \quad & \sum_{S \in \mathcal{S}} y_S \\ \text{such that} \quad & \sum_{S : e \in \delta(S)} y_S \leq c_e \quad \forall e \in E \\ & y_S \geq 0 \quad \forall S \in \mathcal{S} \end{aligned}$$

Before we continue, some definitions will be stated which will help to define our algorithm for the problem.

Definition 11.1. *Given a set of edges $X \subseteq E$, a set $S \in \mathcal{S}$ is* violated *with respect to X if $\delta(S) \cap X = \emptyset$. In other words S is violated even with edge set X included.*

Definition 11.2. *Given a set of edges $X \subseteq E$, a set $S \in \mathcal{S}$ is* minimally violated *with respect to X if S is violated with respect to X and there is no $S' \subset S$ that is also violated with respect to X.*

Next we show that any two minimally violated sets are disjoint.

Claim 11.0.1. *$\forall X \subseteq E$ if S and S' are minimally violated sets then $S \cap S' = \emptyset$, i.e. S and S' are disjoint.*

In fact we will prove the following claim which implies the preceding one.

Claim 11.0.2. *Let $X \subseteq E$. The minimially violated sets with respect to X are the connected components of the graph (V, X) that are violated with respect to X.*

Proof. Consider a minimal violated set S. We may assume the set S contains s_i but not t_i for some i. If S is not a connected component of (V, X) then there must be some connected component S' of $G[S]$ that contains s_i. But $\delta_X(S') = \emptyset$, and hence S' is violated; this contradicts the fact that S is minimal. Therefore, if a set S is a minimal violated set then it must be connected in the graph (V, X).

Now suppose that S is a connected component of (V, X); it is easy to see that no proper subset of S can be violated since some edge will cross any such set. Thus, if S is a violated set then it is minimal violated set.

Thus, the minimal violated sets with respect to X are the conected components of the graph (V, X) that are themselves violated sets. It follows that any two distinct minimal violated sets are disjoint. ■

The primal-dual algorithm for Steiner Forest is described below.

```
SteinerForest:
  $F \leftarrow \emptyset$
  while $F$ is not feasible
      Let $C_1, C_2, \ldots, C_h$ be minimally violated sets with respect to $F$
      Raise $y_{C_i}$ for $1 \le i \le h$ uniformly until some edge $e$ becomes tight
      $F \leftarrow F + e$
      $x_e = 1$
  Output $F' = \{e \in F \mid F - e$ is not feasible$\}$
```

The first thing to notice about the algorithm above is that it is closely related to our solution to the Vertex Cover problem, however, there are two main differences. In the Vertex Cover we raised the dual variables for all uncovered edges uniformly, however, in this algorithm we were more careful on which dual variables are raised. In this algorithm, we chose to only raise the variables which correspond to the minimally violated sets. Unlike the case of Steiner Tree, in Steiner Forest, there can be non-trivial connected components that are not violated and hence become inactive. A temporarily inactive component may become part of an active component later if an active component merges with it. The other main difference is that when we finally output the solution, we *prune* F to get F'. This is done for technical reasons, but the intuition is that we should include no edge in the solution which is not needed to obtain a feasible solution. To understand this algorithm, there is a non-trivial example in the textbook [152] that demonstrates the algorithm's finer points.

Lemma 11.1. *At the end of the algorithm, F' and $\mathbf{y}$ are primal and dual feasible solutions, respectively.*

Proof. In each iteration of the while loop, only the dual variables corresponding to connected components were raised. Therefore, no edge that is contained within the same component can become tight, and, therefore, F is acyclic. To see that none of the dual constraints are violated, observe that when a constraint becomes tight (that is, it holds with equality), the corresponding edge e is added to F. Subsequently, since e is contained in some connected component of F, no set S with $e \in \delta(S)$ ever has y_S raised. Therefore, the constraint for e cannot be violated, and so $\mathbf{y}$ is dual feasible.

As long as F is not feasible, the while loop will not terminate, and there are some minimal violated sets that can have their dual variables raised. Therefore, at the end of the algorithm F is feasible. Moreover, since F is acyclic (it is a forest), there is a unique s_i-t_i path in F for each $1 \leq i \leq k$. Thus, each edge on a s_i-t_i path is not redundant and is not deleted when pruning F to get F'. ■

Theorem 11.3. *The primal-dual algorithm for* Steiner Forest *gives a* 2*-approximation.*

Proof. Let F' be the output from our algorithm. To prove this theorem, we want to show that $c(F') \leq 2 \sum_{s \in \mathcal{S}} y_S$ where y_S is the feasible dual constructed by the algorithm. It follows from this that the algorithm is in fact a 2-approximation. First, we know that $c(F') = \sum_{e \in F'} c_e = \sum_{e \in F'} \sum_{S \in \mathcal{S}: e \in \delta(S)} y_s$ because every edge picked is tight. Let $\deg_{F'}(S)$ denote the number of edges of F' that cross the cut $(S, V \setminus S)$. It can be seen that $\sum_{e \in F'} \sum_{S \in \mathcal{S}: e \in \delta(S)} y_s = \sum_{s \in \mathcal{S}} y_S \deg_{F'}(S)$.

Let A_i contain the minimally violated sets in iteration i and let Δ_i denote the amount of dual growth in the ith iteration. Say that our algorithm runs for α iterations. We can then rewrite $\sum_{s \in \mathcal{S}} y_S \deg_{F'}(S)$ as the double summation $\sum_{i=1}^{\alpha} \sum_{S \in A_i} \Delta_i \deg_{F'}(S)$. In the next lemma it will be shown for any iteration i that $\sum_{S \in A_i} \deg_{F'}(S) \leq 2|A_i|$. Knowing this we can prove the theorem:

$$\sum_{S \in \mathcal{S}} y_s \deg_{F'}(S) = \sum_{S \in \mathcal{S}} \sum_{i: S \in A_i} \Delta_i \deg_{F'}(S) \quad = \quad \sum_{i=1}^{\alpha} \sum_{S \in A_i} \Delta_i \deg_{F'}(S) \leq \sum_{i=1}^{\alpha} \Delta_i \cdot 2|A_i| \leq 2 \sum_{S \in \mathcal{S}} y_S.$$

■

Now we show the lemma used in the previous theorem. It is in this lemma that we use the fact that we prune F to get F'.

We need a simple claim.

Lemma 11.2. *Let $T = (V, E)$ be a tree/forest and let $Z \subseteq V$ be a subset of the nodes such that every leaf of T is in Z. Then $\sum_{v \in Z} \deg_T(v) \leq 2|Z| - 2$ where $\deg_T(v)$ is the degree of v in T.*

Proof. We will prove it for a tree. We have $\sum_{v \in V} \deg_T(V) = 2|V| - 2$ since a tree has $|V| - 1$ edges. Every node $u \in V - Z$ has degree at least 2 since it is not a leaf. Thus $\sum_{u \in V-Z} \deg_T(u) \geq 2|V - Z|$. Thus,

$$\sum_{v \in Z} \deg_T(v) \leq 2|V| - 2 - 2|V - Z| \leq 2|Z| - 2.$$

■

Lemma 11.3. *For any iteration i of our algorithm, $\sum_{S \in A_i} \deg_{F'}(S) \leq 2|A_i| - 2$.*

Proof. Consider the graph (V, F'), and fix an iteration i. In this graph, contract each set S active in iteration i to a single node (call such a node an *active node*), and each inactive set to a single node. Let the resulting graph be denoted by H. We know that F is a forest and we have contracted connected subsets of vertices in F; as $F' \subseteq F$, we conclude that H is also a forest.

Claim 11.0.3. *Every leaf of H is an active node.*

Proof. If not, consider leaf node v of H which is an inactive node and let $e \in F'$ be the edge incident to it. We claim that $F' - e$ is feasible which would contradict the minimality of F'. To see this, if x, y are two nodes in H where $v \neq x, v \neq y$ then x and y are connected also in $H - e$ since v is a leaf. Thus the utility of e is to connect v to other nodes in H but if this is the case v would be an active node at the start of the iteration which is not the case. ■

The degree in H of an active node corresponding to violated set S is $\deg_{F'}(S)$. Now we apply Lemma 11.2. ■

Chapter 12

Primal Dual for Constrained Forest Problems

We previously saw a primal-dual based 2-approximation for the Steiner Forest problem. The algorithm can be generalized to a much wider class of problems that involve finding a min-cost forest in an undirected edge-weighted graph that needs to satisfy some constraint. The resulting machinery is a more abstract and requires more advanced tools. We start with some problems all of which are NP-Hard.

Point to point connection problem: Given edge-weighted graph $G = (V, E)$ and two disjoint sets $X = \{s_1, \ldots, s_k\}$ and $Y = \{t_1, \ldots, t_k\}$ of terminals find the min-cost forest in G such that each connected component contains same number of terminals from X and Y.

Lower Capacitated Tree Problem: Given $G = (V, E)$, $c : E \to \mathbb{R}^+$ and a $k \in \mathbb{Z}^+$ find a set $E' \subseteq E$ of minimum cost such that every connected component in (V, E') has at least k edges.

Connectivity Augmentation: Given an undirected k-edge connected graph $G = (V, E)$ and a set of edges $E^{aug} \subseteq V \times V - E$, find a set $E' \subseteq E^{aug}$ of minimum cost such that $G = (V, E \cup E')$ is $(k+1)$-edge connected.

Steiner Connectivity Augmentation: Let $G = (V, E)$ be an edge-weighted graphs and let $(s_1, t_1), \ldots, (s_h, t_h)$ be k pairs such that each pair is k-edge-connected in G. Given a set of edges $E^{aug} \subseteq V \times V - E$, find a set $E' \subseteq E^{aug}$ of minimum cost such that in $G' = (V, E \cup E')$ each pair (s_i, t_i) is $(k+1)$-edge connected.

Each of the preceding problems can be cast as a special case of the following abstract problem. Given an edge-weighted graph $G = (V, E)$ and a function

$f : 2^V \rightarrow \{0, 1\}$, find a min-cost subset of edges E' such that $|\delta_{E'}(S)| \geq f(S)$ for each $S \subseteq V$. We use the notation $\delta_F(S)$ to denote the edges from an edge set F that cross the set/cut S. Alternatively we want a min-cost subset of edges E' such that each set $S \in \mathcal{S}$ is crossed by an edge of E' where $\mathcal{S} = \{S \mid f(S) = 1\}$. It is easy to observe that a minimal solution to this abstract problem is a forest since any cut needs to be covered at most once. This formulation is too general since f may be completely arbitrary. The goal is to find a sufficiently general class that captures interesting problems while still being tractable. The advantage of $\{0, 1\}$ functions is precisely because the minimal solutions are forests. We will later consider integer valued functions.

Definition 12.1. *Given a graph $G = (V, E)$ and an integer valued function $f : 2^V \rightarrow \mathbb{Z}$ we say that a susbet of edges F is feasible for f or covers f iff $|\delta_F(S)| \geq f(S)$ for all $S \subseteq V$.*

Remark 12.1. Even though it may seem natural to restrict attention to requirement functions that only have non-negative entries, we will see later that the flexibility of negative requirements is useful.

Given a network design problem Π in an undirected graph $G = (V, E)$ and an integer valued function $f : 2^V \rightarrow \mathbb{Z}_+$ we say that the *requirement* function of Π is f if covering f is equivalent to satisfing the constraints of Π.

12.1 Classes of Functions and Setup

Here we consider classes of requirement functions $f : 2^V \rightarrow \mathbb{Z}$ and their relationship. Even though we define more generally, for this chapter the focus will be on $\{0, 1\}$ functions.

Definition 12.2. *f is* maximal *if for all disjoint A and B we have $f(A \cup B) \leq \max\{f(A), f(B)\}$.*

Exercise 12.1. Prove that the requirement function of Steiner Forest is maximal.

Definition 12.3. *f is* proper *if it is symmetric, maximal and $f(V) = 0$.*

Exercise 12.2. Prove that the requirement function of Steiner Forest is proper.

Exercise 12.3. Prove that the requirement function Steiner Connectivity Augmentation is proper.

Definition 12.4. *f is* downward monotone *if $f(A) \leq f(B)$ for all $\emptyset \neq B \subset A$.*

Exercise 12.4. Prove that the requirement function of the Lower Capacitated Tree problem as downward monotone.

A very general class is the one given below.

Definition 12.5. *f is* skew-supermodular *(also called* weakly-supermodular*) if for all A and B one of the following conditions hold,*

1. $f(A) + f(B) \leq f(A \cup B) + f(A \cap B)$
2. $f(A) + f(B) \leq f(A - B) + f(B - A)$

A specialization of skew-supermodular for $\{0, 1\}$ functions will be the focus of this chapter.

Definition 12.6. *A $\{0, 1\}$ valued f is* uncrossable *if for all A and B such that $f(A)$ and $f(B) = 1$, one of the following conditions hold,*

1. $f(A \cup B) = 1$ *and* $f(A \cap B) = 1$
2. $f(A - B) = 1$ *and* $f(B - A) = 1$.

Claim 12.1.1. *If f is downward monotone then it is skew-supermodular.*

Proof. Since f is downward monotone, $A - B \subset A$ and $B - A \subset B$ we get:

$$f(A) \leq f(A - B)$$
$$f(B) \leq f(B - A)$$

and hence the second condition of skew-supermodularity always holds. ■

Lemma 12.1. *If f is proper then it is skew-supermodular.*

Proof. Consider two sets A, B. By considering A as disjoint union of $A - B$ and $A \cap B$ we have $(i) f(A) \leq \max\{f(A - B), f(A \cap B)\}$. Similarly (ii) $f(B) \leq \max\{f(B - A), f(A \cap B)\}$.

Now we apply symmetry of f and note that $f(A) = f(V - A)$. Write $V - A$ as disjoint union of $B - A$ and $V - (A \cup B)$. Hence (iii) $f(A) = f(V - A) \leq \max\{f(B - A), f(V - (A \cup B)\} = \max\{f(B - A), f(A \cup B)\}$ where we used symmetry of f in the second equality. Similary, (iv) $f(B) \leq \max\{f(A - B), f(A \cup B\}$.

Summing up the four inequalities and replacing $\max\{x, y\}$ by $(x + y)/2$ we obtain

$$2(f(A) + f(B)) \leq f(A - B) + f(B - A) + f(A \cap B) + f(A \cup B)$$

which implies that $f(A) + f(B) \leq f(A - B) + f(B - A)$ or $f(A) + f(B) \leq f(A \cap B) + f(A \cup B)$. ■

Definition 12.7. *Let $G = (V, E)$ and $f : 2^V \rightarrow \mathbb{Z}$. For each $X \subseteq E$, the residual requirement function $f_X : 2^V \rightarrow \mathbb{Z}$ is defined as:*

$$f_X(A) = f(A) - |\delta_X(A)|.$$

Exercise 12.5. Let $g : 2^V \rightarrow \mathbb{R}$ be a symmetric submodular function. Prove that g satisfies *posi-modularity*:

$$g(A) + g(B) \geq g(A - B) + g(B - A) \quad \forall A, B \subseteq V.$$

Lemma 12.2. *If f is skew-supermodular then f_X is also skew-supermodular for any $X \subseteq E$.*

Proof. The function $|\delta_X(.)|$ is submodular. Hence

$$|\delta_X(A)| + |\delta_X(B)| \geq |\delta_X(A \cap B)| + |\delta_X(B \cup A)| \quad \forall A, B \subseteq V.$$

The function is also symmetric and hence also satisfies posi-modularity. Therefore,

$$|\delta_X(A)| + |\delta_X(B)| \geq |\delta_X(A - B)| + |\delta_X(B - A)| \quad \forall A, B \subseteq V.$$

Now consider the function f_X and let A, B be any two subsets of V. Suppose $f(A) + f(B) \geq f(A \cup B) + f(A \cap B)$. Then

$$\begin{aligned}
f_X(A) + f_X(A) &= f(A) - |\delta_X(A)| + f(B) - |\delta_X(B)| \\
&\geq f(A \cup B) + f(A \cap B) - (|\delta_X(A) + |\delta_X(B)| \\
&\geq f(A \cup B) + f(A \cap B) - (|\delta_X(A \cap B)| + |\delta_X(B \cup A)| \\
&= f_X(A \cup B) + f_X(A \cap B).
\end{aligned}$$

Similarly, if $f(A) + f(B) \geq f(A - B) + f(B - A)$ we can use posi-modularity of $|\delta_X(\cdot)|$ to argue that $f_X(A) + f_X(B) \geq f_X(A - B) + f_X(B - A)$. We note that $|\delta_X|$ is both submodular *and* posi-modular which allows us to use the right inequality. ■

Remark 12.2. Allowing allow negative requirements in the definitoin of skew-supermodularity allows a clean proof when subtracting $|\delta_X(\cdot)|$.

In the case of $\{0, 1\}$ functions we make the following claim and leave the proof as an exercise which is similar to the the one above.

Lemma 12.3. *Let $f : 2^V \rightarrow \{0, 1\}$ be an uncrossable function and let $X \subseteq E$. Let $h : 2^V \rightarrow \{0, 1\}$ be the residular function where $h(S) = 1$ iff $|\delta_X(S)| = 0$. Then h is uncrossable.*

Definition 12.8. *Let f be a $\{0, 1\}$ requirement function over V and let $X \subseteq E$. A set S is* violated *with respect to X if $f_X(S) = 1$ (in other words S is not yet covered by the edge set X). A set S is a* minimal *violated set if there is no $S' \subsetneq S$ such that S' is violated.*

Lemma 12.4. *Let f be an uncrossable function and $X \subseteq E$, then the minimial violated sets of f with respect to X are disjoint. That is, if A, B are minimal violated sets then $A = B$ or $A \cap B = \emptyset$.*

Proof. Since f_X is also uncrossable it sufficies to consider minimal violated sets A, B with respect to f (hence the empty set of edges). Suppose the property does not hold. Then we can assume that A, B propertly cross, that is, $A - B, A \cap B, B - A$ are all non-empty. We have $f(A) = f(B) = 1$ since both are violated. Since f is uncrossable, we have $f(A - B) = f(B - A) = 1$ or $f(A \cap B) = f(A \cup B) = 1$. In both cases we see that we violate minimality of A, B. ■

12.2 A Primal-Dual Algorithm for Covering Uncrossable Functions

In this section we consider the following problem. Given a graph $G = (V, E)$ with non-negative edge weights $c : E \rightarrow \mathbb{R}_+$ and a uncrossable function $f : 2^V \rightarrow \{0, 1\}$ find a min-cost set of edges $F \subseteq E$ such that $|\delta_F(S)| \geq f(S)$ for all S. An important computational issue is how f is specified. A natural model is to consider the value oracle one where we have access to $f(S)$ via an oracle. Given $S \subseteq V$, the oracle returns $f(S)$. However this is not sufficient in the general context of uncrossable functions as we see from the following example. Fix some set A. Define the function f_A where $f_A(S) = 1$ iff $S = A$. It is easy to see that there is a feasible solution iff $\delta_E(A) \neq \emptyset$. How do we even verify that there is a feasible solution via a value oracle? In general it may take an exponential number of queries to find A. Hence we need more. We will assume that there is an oracle that given G and $X \subseteq E$ outputs the set of all minimal violated sets of f_X. This will typically be easy to ensure for specific functions of interest. We note that for some special class of functions such as proper functions, a value oracle suffices to find the minimal violated sets.

We write the primal and dual LPs for covering f via edges of a given graph $G = (V, E)$.

$$\begin{aligned} \min \quad & \sum_{e \in E} c_e x_e \\ \text{such that} \quad & \sum_{e \in \delta(S)} x_e \geq f(S) \quad \forall S \subseteq V \\ & x_e \geq 0 \quad \forall e \in E \end{aligned}$$

$$\begin{aligned} \max \quad & \sum_{S \in \mathcal{S}} f(S) y_S \\ \text{such that} \quad & \sum_{S : e \in \delta(S)} y_S \leq c_e \quad \forall e \in E \\ & y_S \geq 0 \quad \forall S \subseteq V \end{aligned}$$

The primal-dual algorithm is similar to the one for Steiner Forest with a growth phase and reverse-delete phase.

```
CoverUncrossableFunc($G = (V, E), f$):
  $F \leftarrow \emptyset$
  while $F$ is not feasible
      Let $C_1, C_2, \ldots, C_\ell$ be minimally violated sets of $f$ with respect to $F$
      Raise $y_{C_i}$ for $1 \leq i \leq \ell$ uniformly until some edge $e$ becomes tight
      $F \leftarrow F + e$
      $x_e = 1$
  Let $F' = \{e_1, e_2, \ldots, e_t\}$ where $e_i$ is edge added to $F$ in iteration $i$
  $F' = F$
  for $i = t$ down to 1 do
      if $(F' - e_i)$ is feasible then $F' = F' - e_i$
  Output $F'$
```

Analysis: We can prove the following by induction on the iterations and omit the routine details.

Lemma 12.5. *The output of the algorithm, F', is a feasible solution that covers f. Assuming oracle access to finding the minimal violated sets in each iteration, the algorithm can be implemented in polynomial time.*

The preceding lemma shows that the algorithm correctly outputs a feasible solution. The main part is to analyze the cost of the solution.

Theorem 12.9. *Let F' be the output of the algorithm. Then $c(F') \leq 2 \sum_S f(S) y_S$ and hence $c(F') \leq 2\,\mathrm{OPT}$.*

The cost analysis is based on the following key structural lemma.

Lemma 12.6. *Let $G = (V, E)$ be a graph and let $f : 2^V \to \{0, 1\}$ be an uncrossable function and let $\mathcal{C}$ be the set of minimal violated sets of f with respect with respect to $\emptyset$. Let $F \subseteq E$ be* any minimal *feasible solution that covers f. Then $\sum_{C \in \mathcal{C}} \deg_F(C) \leq 2|\mathcal{C}|$ where $\deg_F(C) = |\delta(C) \cap F|$ is the number of edges of F crossing C.*

We defer the proof of the preceding lemma to the following subsection and finish the analysis of the 2-approximation. This is similar to the one for Steiner Forest. Let t be the number of iterations of the algorithm. Let $\mathcal{C}_i$ be the minimal violated sets in iteration i of the algorithm and let Δ_i be the amount by which the duals grow in iteration i. We call any $C \in \mathcal{C}_i$ *active* in iteration i. We have $\sum_{e \in F'} c_e = \sum_{e \in F'} \sum_{S: e \in \delta(S)} y_S$ since we add edges to F only when the dual constraint is tight and $F' \subseteq F$. We also observe that the duals are grown only for sets $S \subseteq V$ where $f(S) = 1$ and hence $\sum_S f(S) y_S = \sum_S y_S$ for the dual solution created by the algorithm.

$$c(F') = \sum_{e \in F'} \sum_{S: e \in \delta(S)} y_S = \sum_S f(S) y_S \deg_{F'}(S) = \sum_S \deg_{F'}(S) \sum_{S \in \mathcal{C}_i} \Delta_i = \sum_{i=1}^{t} \Delta_i \sum_{S \in \mathcal{C}_i} \deg_{F'}(S).$$

To complete the analysis we need the following lemma which can be seen as a corollary of Lemma 12.6.

Lemma 12.7. *Consider any iteration i of the algorithm. Then $\sum_{C \in \mathcal{C}_i} \deg_{F'}(C) \leq 2|\mathcal{C}_i|$.*

Proof. Consider iteration i. Let $X = \{e_1, e_2, \ldots, e_{i-1}\}$ which are the set of edges added by algorithm in the growth phase prior to iteration i. Thus $\mathcal{C}_i$ is the minimal violated sets with respect to X. Consider the graph $G' = (V, E \setminus X)$ and the function f_X. We observe that $F'' = F' \setminus X$ is a minimal feasible solution to covering f_X in G' — this is because of the reverse delete process. Moreover we claim that $\deg_{F'}(C) = \deg_{F''}(C)$ for any $C \in \mathcal{C}_i$ (why?). We can now apply Lemma 12.6 to the function f_X (which is uncrossable) and G' and F'' so obtain:

$$\sum_{C \in \mathcal{C}_i} \deg_{F'}(C) = \sum_{C \in \mathcal{C}_i} \deg_{F''}(C) \leq 2|\mathcal{C}_i|.$$

■

With the preceding lemma in place we have,

$$c(F') = \sum_{i=1}^{t} \Delta_i \sum_{S \in \mathcal{C}_i} \deg_{F'}(S) \leq \sum_{i=1}^{t} \Delta_i \cdot 2|\mathcal{C}_i| = 2 \sum_S f(S) y_S.$$

12.2.1 Proof of Lemma 12.6

Since F is a *minimal* feasible solution to cover f, for every $e \in F$ there must be a set S_e such that $f(S_e) = 1$ and $\delta_F(S_e) = \{e\}$; this is the reason we cannot remove e from F and maintain feasibility. We call such a set S_e a *witness set* for e. Clearly a witness set S_e is a violated set.

Claim 12.2.1. *Let S_e be a witness set for $e \in F$. Then for any minimal violated set C we have $C \subseteq S_e$ or $C \cap S_e = \emptyset$.*

Proof. If C crosses S_e then either $C - S_e$ or $C \cap S_e$ would also be violated (since f is uncrossable) contradicting minimality of C. ■

Note that there can be many witness sets for the same edge, however, the same set cannot be a witness set for two different edges.

Definition 12.10. *Given a minimal feasible solution F we call a family of sets $\mathcal{S}$ a witness family for F if there is a bijection $h : F \longrightarrow \mathcal{S}$ such that $h(e)$ is a witness set for e.*

Given F we can construct a witness family by considering each edge $e \in F$ and picking an arbitrary witness set for e and additing it to the collection.

Definition 12.11. *A family $\mathcal{S}$ of finite sets is* laminar *if no two sets $A, B \in \mathcal{S}$ cross.*

We have seen that there is a witness family for F since it is minimal. We can obtain a special witness family starting with an arbitrary witness family.

Lemma 12.8. *There is a witness family for F which is laminar.*

Proof. The process is based on *uncrossing*. Suppose we start with an arbitrary witness family $\mathcal{S}$ and it is not laminar. Let S_{e_1}, S_{e_1} be the witness sets for $e_1, e_2 \in F$ such that S_{e_1}, S_{e_2} cross.

Claim 12.2.2. *One of the following holds: (i) $(\mathcal{S} \setminus \{S_{e_1}, S_{e_2}\}) \cup \{S_{e_1} - S_{e_2}, S_{e_2} - S_{e_1}\}$ is a witness family for F (ii) $(\mathcal{S} \setminus \{S_{e_1}, S_{e_2}\}) \cup \{S_{e_1} \cap S_{e_2}, S_{e_2} \cup S_{e_2}\}$ is a witness family for F.*

The new sets don't cross each other but what about other sets in the family? One can argue that the number of crossing can only decrease. More formally, suppose we have three sets A, B, D and say D crosses p sets from A, B (p is one of $\{0, 1, 2\}$). If we uncross A, B (that is replace A, B by $A - B, B - A$ or $A \cap B, A \cup B$) then the number of sets that D crosses after uncrossing cannot increase. We leave this claim as an exercise. This means that repeated uncrossing of two crossing sets will eventually lead to a laminar family.

Now we prove the claim. S_{e_1}. S_{e_1} and S_{e_2} are violated sets which means that $f(S_{e_1}) = 1$ and $f(S_{e_2}) = 1$. Since f is uncrossable, say $f(S_{e_1} - S_{e_2}) = 1$ and

$f(S_{e_2} - S_{e_1}) = 1$. The only edges from F that can cross $S_{e_1} - S_{e_2}$ and $S_{e_2} - S_{e_1}$ are e_1 and e_2 — if there is another edge e that crosses one of them then it would also cross one of S_{e_1} and S_{e_2} and hence they would not be witness sets. Since F' is a feasible solution, $S_{e_1} - S_{e_2}$ and $S_{e_2} - S_{e_1}$ are both crossed by some edge from $Y = \{e_1, e_2\}$. We claim that each of them is crossed by exactly one of them. If this is the case then they are valid witness sets for the two edges e_1, e_2. To see why exaclty one of them crosses each set we use submodularity/posi-modularity:

$$2 \geq |\delta_Y(S_{e_1} - S_{e_2})| + |\delta_Y(S_{e_2} - S_{e_1})| \leq |\delta_Y(S_{e_1})| + |\delta_Y(S_{e_2})| = 2.$$

The first inequality is since each is covered and the second inequality is because of posi-modularity of $|\delta_Y(\cdot)|$.

The other case when $f(S_{e_1} \cap S_{e_2}) = 1$ and $f(S_{e_1} \cup S_{e_2}) = 1$ can also be handled with a similar argument. ■

Let $\mathcal{S}$ be a laminar witness family for F. We create a rooted tree P from $\mathcal{S}$ as follows. To do this we first add V to the family. For each set $S \in \mathcal{S}$ we add a vertex v_S. For any S, T such that T is the smallest set in $\mathcal{S}$ containing S we add the edge (v_S, v_T) which makes v_T the parent of v_S. Since we added V to the family we obtain a tree since every maximal set in the original family becomes a child of v_V. Note that P has $|F|$ edges and $|F| + 1$ nodes and each $e \in F$ corresponds to the edge (v_{S_e}, v_T) of P where v_T is the parent of v_{S_e} in P. We keep this bijection between F and edges of P in mind for later.

See figure.

Consider any minimal violated set $C \in \mathcal{C}$. We observe that C cannot cross any set in $\mathcal{S}$ since it is a witness family. For each C we associate the minimal set $S \in \mathcal{S}$ such that $C \subseteq S$. We call v_S an active node of T if there is a $C \in \mathcal{S}$ associated with it. Note that (i) not all nodes of P may be active (ii) every C is associated with exactly one active node of P (iii) multiple sets from $\mathcal{C}$ can be associated with the same active node of P.

Lemma 12.9. *Let P_a be the active nodes of P. Then $|P_a| \leq |\mathcal{C}|$ and every leaf of P other than potentially the root is an active node.*

Proof. Since each C is associated with an active node we obtain $|P_a| \leq |\mathcal{C}|$. A leaf of P corresponds to a minimal set from $\mathcal{S}$ or the root. If S_e is a minimal set from $\mathcal{S}$ then S_e is a violated set and hence must contain a minimal violated set C. But then S_e is active because of C. Consider v_V. If it is a leaf then it has only one child which is the unique maximal witness set S. The root can be inactive since the function f is not necessarily symmetric. (However, if f is symmetric then $V - S$ would also be a violated set and hence contain a minimal violated set from $\mathcal{C}$.) ■

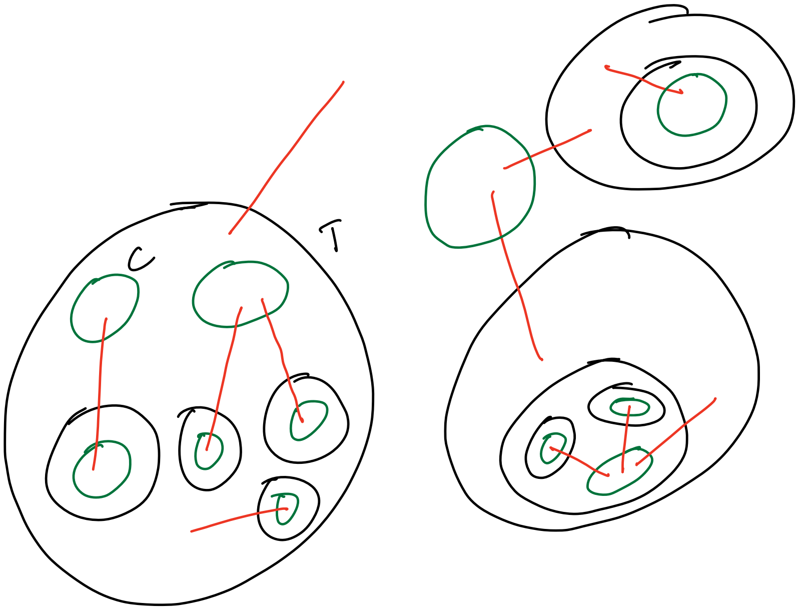


Figure 12.1: Laminar witness family for a minimal solution F shown as red edges. Sets in green are the minimal violated sets and the sets in black are the witness sets. The root set V is not drawn.

Lemma 12.10. *Let v_T be an active node in P. Let $\mathcal{C}' \subseteq \mathcal{C}$ be set of all minimal violated sets associated with v_T. Then $\sum_{C \in \mathcal{C}'} \deg_F(C) \leq \deg_P(v_T)$.*

Proof. Let $Y = \cup_{C \in \mathcal{C}'} \delta_F(C)$ be the set of edges incident to the sets in $\mathcal{C}'$. Consider any $C \in \mathcal{C}'$. $C \subset T$ and C is also disjoint from the children of T. Consider an edge $e \in \delta_F(C)$. If e crosses T then T is the witness set for e (since only one edge from F can cross T for which it is the witness) and we charge e to the parent edge of T. If e does not cross T then the witness set S_e must be a child of T. Since only one edge can cross each child of T we can charge e to one child of T. Note that no edge $e \in Y$ can be incident to two set $C_1, C_2 \in \mathcal{C}'$ since both one end point of e must be contained in a child of T (assuming e does not cross T) and both C_1, C_2 are contained in T and disjoint from the children of T. See figure. Therefore, we can charge $\sum_{C \in \mathcal{C}'} \deg_{F'}(C)$ to the number of children of T plus the parent edge of T, which is $\deg_P(v_T)$. ■

Now we are ready to finish the proof. From Lemma 12.10 and the fact that each $C \in \mathcal{C}$ is associated to some active node, we have

$$\sum_{C \in \mathcal{C}} \deg_F(C) \le \sum_{v \in P_a} \deg_P(v).$$

To bound $\sum_{v \in P_a} \deg_P(v)$ we had observed that in the tree P every leaf except perhaps the root node is an active node. Suppose root is an active node or it is not a leaf. Then from Lemma 11.2, $\sum_{v \in P_a} \deg_P(v) \le 2|P_a| - 2$. Suppose root is a leaf and inactive. Again from Lemma 11.2 where we consider $Z = P_a \cup \{v_V\}$, we have $\sum_{v \in P_a} \deg_P(v) + 1 \le 2|P_a + 1| - 2 = 2|P_a|$. Hence $\sum_{v \in P_a} \deg_P(v) \le 2|P_a| - 1$. Thus, in both cases we see that $\sum_{v \in P_a} \deg_P(v) \le 2|P_a|$. Finally we note that $|P_a| \le |\mathcal{C}|$ and hence putting together we have

$$\sum_{C \in \mathcal{C}} \deg_F(C) \le 2|P_a| \le 2|\mathcal{C}|$$

as desired.

Bibliographic Notes: The primal-dual algorithm for uncrossable function is from the paper of Williamson, Goemans, Mihail and Vazirani [157]. The proof in the paper assumes that h is symmetric without explicitly stating it; for symmetric functions one obtains a slight advantage over 2. See Williamson's thesis [156] where he gives the proof for both symmetric and general uncrossable functions. The survey by Goemans and Williamson [67] describes the many applications of the primal-dual method in network design.

Chapter 13

Survivable Network Design Problem

In this chapter we consider the Survivable Network Design Problem problem. The input is an undirected graph $G = (V, E)$ with edge-weights $c : E \longrightarrow \mathbb{R}_+$ and integer requirements $r(uv)$ for each pair of vertices uv. We wrote uv instead of (u, v) to indicate that the requirement function is for unordered pairs (alternatively, $r(u, v) = r(v, u)$ for all u, v). The goal is to find a min-cost subgraph $H = (V, F)$ of G such that each the connectivity betweeen u and v in H is at least $r(uv)$. We obtain two versions of the problem: EC-SNDP if the connectivity requirement is edge-connectivity and VC-SNDP for vertex connectivity. It turns out that EC-SNDP is much more tractable than VC-SNDP and we will focus on EC-SNDP.

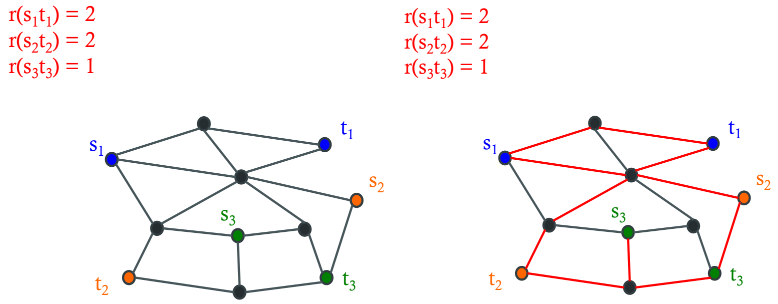


Figure 13.1: Example of EC-SNDP. Requirement only for three pairs. A feasible solution shown in the second figure as red edges. In this example the paths for each pair are also vertex disjoint even though the requirement is only for edge-disjointness.

For EC-SNDP there is a seminal work of Jain based on iterated rounding that yields a 2-approximation as a special case of a more general problem. Prior to his work there was an augmentation based approach that yields $2k$ and $2H_k$ approximations where $k = \max_{uv} r(uv)$ is the maximum connectivity requirement. Despite being superceded by Jain's result in terms of the ratio, the augmentation approach is important for various reasons and we will discuss both.

We first consider the LP relaxation for the EC-SNDP. We do this by setting up the requirement function $f : 2^V \rightarrow \mathbb{Z}$ where we let $f(S) = \max_{u \in S, v \in V-S} r(uv)$. The goal is to find a min-cost subgraph H of G such that $\delta_H(S) \geq f(S)$.

Claim 13.0.1. *The requirement function f that captures* EC-SNDP *is proper and hence skew-supermodular.*

Proof. It is easy to see f is symmetric. Consider disjoint sets A, B. Suppose $f(A \cup B) = k$ which means that there is some $s \in A \cup B$ and $t \in V - (A \cup B)$ such that $r(st) = k$. If $s \in A$ then $f(A) \geq k$ and if $s \in B$ then $f(B) \geq k$. Therefore $\max\{f(A), f(B)\} \geq k = f(A \cup B)$ since $s \in A$ or $s \in B$. ∎

13.1 Augmentation approach

The augmentation approach for EC-SNDP is based on iteratively increasing the connectivity of pairs from 1 to k where $k = \max_{uv} r(uv)$. In fact this works for any proper function $f : 2^V \rightarrow \mathbb{Z}$ and we will work in this generality rather than focus only on EC-SNDP.

Claim 13.1.1. *Let f be a proper function and let p be an integer. Then the truncated function $g_p : 2^V \rightarrow \mathbb{Z}$ defined as $g_p(S) = \min\{p, f_p(S)\}$ is proper.*

Proof. Exercise. ∎

Lemma 13.1. *Let $G = (V, E)$ be graph and let $f : 2^V \rightarrow \mathbb{Z}$ be a proper function and let $p \geq 0$ be a non-negative integer. Let $X \subseteq E$ be a set of edges such that $|\delta_X(S)| \geq g_p(S)$ Consider the function $h_{p+1} : 2^V \rightarrow \{0, 1\}$ where $h_{p+1}(S) = 1$ iff $f(S) \geq p + 1$ and $|\delta_X(S)| = p$. Then h_{p+1} is uncrossable and symmetric.*

Proof. Consider the function g_{p+1} which is proper and hence also skew-supermodular. For notational convenience we use h for h_{p+1}. Suppose $h(A) = h(B) = 1$. This implies $g_{p+1}(A) \geq p+1$ and $g_{p+1}(B) \geq p+1$ and $|\delta_X(A)| = \delta_X(B) = p$. g_{p+1} is skew-supermodular. First case is when $g_{p+1}(A) + g_{p+1}(B) \leq g_{p+1}(A \cup B) + g_{p+1}(A \cap B)$. This implies that $g_{p+1}(A \cup B) = g_{p+1}(B) = p + 1$. By submodularity of $|\delta_X|$ we have $|\delta_X(A)| + |\delta_X(B)| \geq |\delta_X(A \cap B)| + |\delta_X(A \cup B)|$ and by feasibility of X for g_p we have $|\delta_X(A \cap B)| = |\delta_X(A \cap B)| = p$. This implies that $h(A \cap B) = h(A \cup B) = 1$.

Similarly, if $g_{p+1}(A) + g_{p+1}(B) \leq g_{p+1}(A - B) + g_{p+1}(B - A)$ we can argue that $h(A - B) = h(B - A) = 1$ via posi-modularity of $|\delta_X|$. ∎

Exercise 13.1. Let $G = (V, E)$ be a graph and let $f : 2^V \longrightarrow \mathbb{Z}$ be a proper function. Suppose F be a feasible cover for g_p. Let h_{p+1} be the residulal uncrossable function that arises from g_p as in the preceding lemma. Let $F' \subseteq E \setminus F$ be a feasible cover for h_{p+1} in the graph $G' = (V, E \setminus F)$. Then $F \cup F'$ is a feasible cover for g_{p+1}.

Lemma 13.2. *Let f be the requirement function of an instance of* EC-SNDP *in $G = (V, E)$ and let p be an integer and let $X \subseteq E$ be a set of edges. There is a polynomial time algorithm to find the minimal violated sets of g_p with respect to X.*

Proof. For each pair or nodes (s, t) find a source-minimal s-t mincut S in the graph $H = (V, X)$ and a sink-minimal mincut T via maxflow[1]. Let S be the cut. If $|\delta_X(S)| < p$ then p is a violated set. We compute all such minimal cuts over all pairs of vertices and take the minimal sets in this collection. We leave it as an exercise to check that the minimal violated sets of g_p are the minimal sets in this collection and will be disjoint. ■

Corollary 13.1. *Let f be the requirement function of an instance of* EC-SNDP *in $G = (V, E)$ and let p be an integer. Let X be set of edges such that X is feasible to cover g_p. In the graph $G' = (V, E \setminus X)$ and for any $F \subseteq (E \setminus X)$ the minimal violated sets of h_{p+1} with respect to F can computed in polynomial time.*

Proof. The minimial violated sets of h_{p+1} with respect to F are the same as the minimal violated sets of g_{p+1} with respect to $X \cup F$. ■

[1] The source minimal s-t mincut in a directed/undirected graph is unique via submodularity and can be found by computing s-t maxflow and finding the reachable set from s in the residual graph. Similarly sink minimal set.

```
Augmentation-Algorithm($G = (V, E), f$)
1. If $E$ does not cover $f$ output “infeasible”
2. $k = \max_S f(S)$ is the maximum requirement
3. $A \leftarrow \emptyset$
4. for ($p = 1$ to $k$) do
    A. $G' = (V, E \setminus A)$
    B. Let $g_p$ be the function defined as $g_p(S) = \min\{f(S), p\}$
    C. Let $h_p$ be the uncrossable function where $h_p(S) = 1$ iff $g_p(S) > |\delta_A(S)|$
    D. Find $A' \subseteq E \setminus A$ that covers $h_p$ in $G'$
    E. $A \leftarrow A \cup A'$
5. Output $A$
```

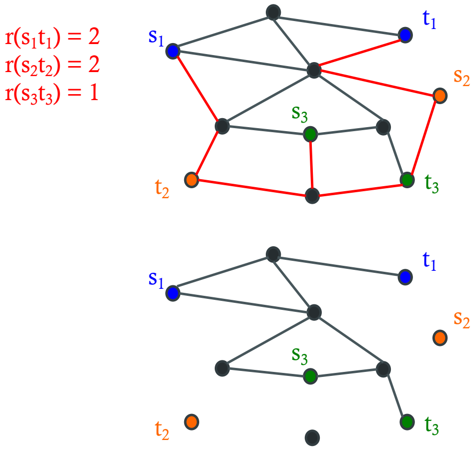


Figure 13.2: Example to illustrate the augmentation approach. Top picture shows a set of edges that connect all pairs with connectivity requirement at least 1. Second picture shows the residual graph in which one needs to solve the augmentation problem. Note that s_2 and t_2 are isolated vertices in the residual graph, however, the cuts induced by them are already satisfied by the edges chosen in the first iteration.

Theorem 13.2. *The augmentation algorithm yields a $2k$-approximation for* EC-SNDP *where k is the maximum connectivity requirement.*

Proof. We sketch the proof. The algorithm has k iterations and in each iteration it uses a black box algorithm to cover an uncrossable function. We saw a primal-dual 2-approximation for this problem. We observe that if F^* is an optimum solution to the given instance then in each iteration $F^* \setminus A$ is a feasible solution to the covering problem in that iteration. Thus the cost paid by the algorithm in each iteration can be bound by $2c(F^*)$ and hence the total cost is at most $2k\,\mathrm{OPT}$. The preceding lemmas argue that the primal-dual algorithm can be implemented in polynomial time. ■

Remark 13.1. A different algorithm that is based on augmentation in *reverse* yields a $2H_k$ approximation where H_k is the k'th harmonic number. We refer the reader to [66].

13.2 Iterated rounding based 2-approximation

In the section we describe the seminal result of Jain [90] who obtained a 2-approximation for EC-SNDP via iterated rounding. He proved a more general polyhedral result. Consider the problem of covering a skew-supermodular function $f : 2^V \to \mathbb{Z}$ by the edges of a graph $G = (V, E)$. The natural cut covering LP relaxation for the problem is given below.

$$\begin{aligned} \min \sum_{e \in E} c(e) x_e & & & \\ \sum_{e \in \delta(S)} x_e & \geq & f(S) & \quad S \subset V \\ x_e & \in & [0,1] & \quad e \in E \end{aligned}$$

Note that upper bound constraints $x_e \leq 1$ are necessary in the general setting when f is integer valued since we can only take one copy of an edge. The key structural theorem of Jain is the following.

Theorem 13.3. *Let x be a basic feasible solution to the LP relaxation. Then there is some edge $e \in E$ such that $x_e = 0$ or $x_e \geq 1/2$.*

With the above in place, and the observation that the residual function of a skew-supermodular function is again a skew-supermodular function, one obtains an interative rounding algorithm.

```
Cover-Skew-Supermodular($G, f$)

1. If $E$ does not cover $f$ output “infeasible”
2. $A \leftarrow \emptyset$, $g = f$
3. While $A$ is not a feasible solution do
    A. Find an optimum basic feasible solution $x$ to cover $g$ in $G' = (V, E \setminus A)$.
    B. If there is some $e$ such that $x_e = 0$ then $E \leftarrow E - \{e\}$
    C. Else If there is some $e$ such that $x_e \geq 1/2$ then
        1. $A = A \cup \{e\}$
        2. $g = f_A$ (recall $f_A(S) = f(S) - |\delta_A(S)|$)
4. Output $A$
```

Corollary 13.4. *The integrality gap of the cut LP is at most* 2 *for any skew-supermodular function f.*

Proof. We consider the iterative rounding algorithm and prove the result via induction on m the number of edges in G. The base case of $m = 0$ is trivial since the function has to be 0.

Let x^* be an optimum basic feasible solution to the LP relaxation. We have $\sum_{e \in E} c_e x^*_e \leq \text{OPT}$. We can assume without loss of generality that f is not trivial in the sense that $f(S) \geq 1$ for at least some set S, otherwise $x = 0$ is optimal and there is nothing to prove. By Theorem 13.3, there is an edge $\tilde{e} \in E$ such that $x^*_{\tilde{e}} = 0$ or $x^*_{\tilde{e}} \geq 1/2$. Let $E' = E \setminus \tilde{e}$ and $G' = (V, E')$. In the former case we can discard $\tilde{e}$ and the current LP solution restricted to E' is a feasible fractional solution and we obtain the desired result via induction since we have one less edge.

The more interesting case is when $x^*_{\tilde{e}} \geq 1/2$. The algorithm includes $\tilde{e}$ and recurses on G' and the residual function $g : 2^V \rightarrow \mathbb{Z}$ where $g(S) = f(S) - |\delta_{\tilde{e}}(S)|$. Note that g is skew-supermodular. We observe that $A' \subseteq E'$ is a feasible solution to cover g in G' iff $A' \cup \{\tilde{e}\}$ is a feasible solution to cover f in G. Furthermore, we also observe that the fractional solution x' obtained by restricting x to E' is a feasible fractional solution to the LP relaxation to cover g in G'. Thus, by induction, there is a solution $A' \subseteq E'$ such that $c(A') \leq 2 \sum_{e \in E'} c(e) x^*_e$. The algorithm outputs $A = A' \cup \{\tilde{e}\}$ which is feasible to cover f in G. We have

$$c(A) = c(A') + c(\tilde{e}) \leq c(A') + 2c(\tilde{e})x^*_{\tilde{e}} \leq 2 \sum_{e \in E'} c(e)x^*_e + 2c(\tilde{e})x^*_{\tilde{e}} = 2 \sum_{e \in E} c(e)x^*_e.$$

We used the fact that $x^*_{\tilde{e}} \geq 1/2$ to upper bound $c(\tilde{e})$ by $2c(\tilde{e})x^*_{\tilde{e}}$. ■

2-approximation for EC-SNDP : We had already seen that the requirement function for EC-SNDP is skew-supermodular. To applyTheorem 13.3 and obtain a 2-approximation for EC-SNDP we need to argue that the LP relaxation can be solved efficiently. We observe that the LP relaxation at the top level can be solved efficiently via maxflow. We need to check that in the graph G with edge capacities given by the fractional solution x the min-cut between every pair of vertices (s, t) is at least $r(s, t)$. Note that the algorithm is iterative. As we proceed the function $g = f_A$ where f is the original requirement function and the A is the set of edges already chosen.

Exercise 13.2. Prove that there is an efficient separation oracle for each step of the iterative rounding algorithm when f is the requirement function for a given EC-SNDP instance.

We now prove Theorem 13.3. The proof consists of two steps. The first step is a characterization of basic feasible solutions via laminar tight sets. The second step is a counting argument.

13.2.1 Basic feasible solutions and laminar family of tight sets

Let x be a basic feasible solution to the LP relaxation. We are done if there is any edge e such that $x_e = 0$ or $x_e = 1$. Hence the interesting case is when x is fully fractional, that is, $x_e \in (0, 1)$ for every $e \in E$.

Definition 13.5. *A set $S \subseteq V$ is* tight *with respect to x if $x(\delta(S)) = f(S)$.*

The LP relaxation is of the form $Ax \geq b, x \in [0, 1]^m$. We number the edges as $e_1, e_2, \ldots, e_m$ arbitarily. Note that each row of A corresponds to a set S and the non-zero entries in the row corresponding to S are precisely for edges in $\delta(S)$. For notational convenience we use χ_S to denote the m-dimensional row vector where $\chi_S(i) = 0$ if $e_i \notin \delta(S)$ and $\chi_S(i) = 1$ if $e \in \delta(S)$. By the rank lemma, if x is a basic feasible solution that is fully fractional, then there are m tight sets $S_1, S_2, \ldots, S_m$ such that the vectors $\chi_{S_1}, \chi_{S_2}, \ldots, \chi_{S_m}$ are linearly independent in $\mathbb{R}^m$. In other words x is the unique solution to the system $\chi^T_{S_i} x = f(S_i), 1 \leq i \leq m$. Note that for a given basic feasible solution x there can be many such bases. A key technical lemma is that one choose a nice one.

Lemma 13.3. *Let x be a basic feasible solution to the cut covering LP relaxation of a skew-supermodular function f where $x_e \in (0, 1)$ for all e. Then there is a* laminar *family $\mathcal{L}$ of tight sets $S_1, S_2, \ldots, S_m$ such that x is the unique solution to the system $\chi^T_{S_i} x = f(S_i)$.*

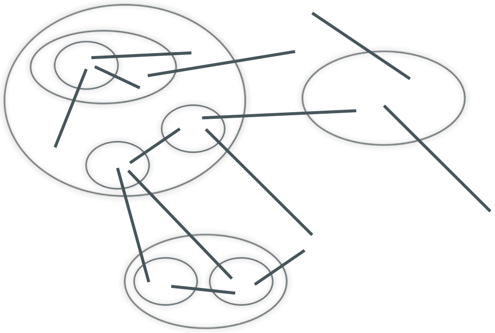


Figure 13.3: Laminar family of tight sets.

We need an auxiliary uncrossing lemma.

Lemma 13.4. *Suppose A and B are two tight sets with respect to x such that A, B cross. Then one of the following holds:*

- *$A \cap B, A \cup B$ are tight and $\chi_A + \chi_B = \chi_{A\cup B} + \chi_{A\cap B}$.*
- *$A - B, B - A$ are tight and $\chi_A + \chi_B = \chi_{A-B} + \chi_{B-A}$.*

Proof. Since f is skew-supermodular $f(A) + f(B) \le f(A \cap B) + f(A \cup B)$ or $f(A) + f(B) \le f(A - B) + f(B - A)$. We will consider the first case.

A, B are tight, hence $x(\delta(A)) = f(A)$ and $x(\delta(B)) = f(B)$. Moreover the function $h(S) = x(\delta(S))$ is submodular (recall that the cut capacity function in an undirected graph is symmetric submodular). Thus $x(\delta(A)) + x(\delta(B)) \ge x(\delta(A\cup B))+x(\delta(A\cap B))$. We also have by feasibility of x that $x(\delta(A\cup B)) \ge f(A\cup B)$ nad $x(\delta(A \cap B)) \ge f(A \cap B)$. Putting together we have

$$x(\delta(A))+x(\delta(B)) = f(A)+f(B) \le f(A\cap B)+f(A\cup B) \le x(\delta(A\cup B))+x(\delta(A\cap B)) \le x(\delta(A))+x(\delta(B)).$$

This implies that $x(\delta(A \cup B)) = f(A \cup B)$ and $x(\delta(A \cap B)) = f(A \cap B)$. Thus both $A \cap B$ and $A \cup B$ are tight. Moreover we observe that $x(\delta(A)) + x(\delta(B)) = x(\delta(A \cup B)) + x(\delta(A \cap B)) + 2x(E(A - B, B - A))$ where $E(A - B, B - A)$ is the set of edges between $A - B$ and $B - A$. From the above tightness we see that $x(\delta(A)) + x(\delta(B)) = x(\delta(A \cup B)) + x(\delta(A \cap B))$, and since x is fully fractional it means that $E(A - B, B - A) = \emptyset$. This implies that $\chi_A + \chi_B = \chi_{A\cup B} + \chi_{A\cap B}$ (why?).

The second case is similar where we use posimodularity of the cut function. ■

Proof of Lemma 13.3. One natural way to proceed is as follows. We start with tight sets $\mathcal{S} = \{S_1, S_2, \ldots, S_m\}$ such that x is characterized as the unique solution of the equations implied by these sets. If the family $\{S_1, S_2, \ldots, S_m\}$ is laminar we are done. Otherwise we pick some two crossing sets, say S_1, S_2 without loss of generality and uncross them using Lemma **??**. We get a new family $\mathcal{S}'$ with m tight sets and the number of crossings in the new family goes down by at least one (as we saw in Lemma **??** previously). Suppose we replace S_1, S_2 by $S_1 \cap S_2, S_1 \cup S_2$. The technical issue is to argue linear independence of the vectors in the new family. This is where we need the property $\chi_{S_1} + \chi_{S_2} = \chi_{S_1 \cap S_2} + \chi_{S_1 \cup S_2}$. Although natural, the linear algebraic argument turns out to be a bit tedious.

Instead we use a slick argument. Let $\mathcal{L}$ be a *maxmial* laminar family of x-tight sets such that the vectors $\chi_S, S \in \mathcal{L}$ are linearly independent. If $\mathcal{L} = m$ then we are done because we have m linearly independent vectors that together span R^m. Suppose $|\mathcal{L}| < m$. Then there must be a tight set S such that χ_S is not spanned by the vectors in $\mathcal{L}$. Choose a tight set S that is not spanned and crosses the fewest number of sets from $\mathcal{L}$. Since $\mathcal{L}$ is maximal, there must be some set $T \in \mathcal{L}$ such that S, T cross (otherwise we can add S to $\mathcal{L}$). Here we use Lemma 13.4 and consider two cases. Suppose $S \cap T, S \cup T$ are tight. Note that $S \cap T, S \cup T$ cross fewer sets in $\mathcal{L}$ than S does. By the choice of S, it must be the case that both $S \cap T$ and $S \cup T$ are spanned by $\mathcal{L}$. However, we have $\chi_S + \chi_T = \chi_{S \cap T} + \chi_{S \cup T}$ which implies that χ_S is also spanned, a contradiction. The proof for the other case when $S - T$ and $T - S$ are tight is similar. Thus we have $\mathcal{L} = m$ and this is the desired family. ■

13.2.2 Counting argument

The second key ingredient in the proof is a counting argument that exploits Lemma 13.3. An easier counting argument shows that there is an edge with $x_e \geq 1/3$ in any basic feasible solution. The tight bound of $1/2$ is more delicate and Jain's original proof is perhaps a bit hard to understand (see [152]). The argument has been subsequently refined and a "fractional token" based analysis [126] was developed and this is the proof in [155]. The token based analysis is flexible and powerful in iterated rounding based algorithms. In an attempt to make the proof even more transparent, the author of this notes developed yet another proof in [41]. We describe that proof below.

The proof is via contradiction where we assume that $0 < x_e < \frac{1}{2}$ for each $e \in E$. We call the two nodes incident to an edge as the endpoints of the edges. We say that an endpoint u *belongs* to a set $S \in \mathcal{L}$ if u is the minimal set from $\mathcal{L}$ that contains u.

We consider the simplest setting where $\mathcal{L}$ is a collection of disjoint sets, in other words, all sets are maximal. In this case the counting argument is easy. Let $m = |E| = |\mathcal{L}|$. For each $S \in \mathcal{L}$, $f(S) \geq 1$ and $x(\delta(S)) = f(S)$. If we assume that $x_e < \frac{1}{2}$ for each e, we have $|\delta(S)| \geq 3$ which implies that each S contains at least

3 distinct endpoints. Thus, the m disjoint sets require a total of $3m$ endpoints. However the total number of endpoints is at most $2m$ since there are m edges, leading to a contradiction.

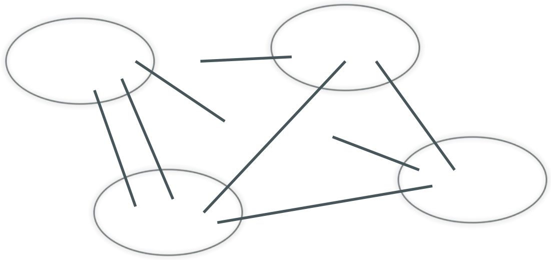


Figure 13.4: Easy case of counting argument.

Now we consider a second setting where the forest associated with $\mathcal{L}$ has k leaves and h internal nodes but each internal node has at least two children. In this case, following Jain, we can easily prove a weaker statement that $x_e \geq 1/3$ for some edge e. If not, then each leaf set S must have four edges leaving it and hence the total number of endpoints must be at least $4k$. However, if each internal node has at least two children, we have $h < k$ and since $h + k = m$ we have $k > m/2$. This implies that there must be at least $4k > 2m$ endpoints since the leaf sets are disjoint. But m edges can have at most $2m$ endpoints. Our assumption on each internal node having at least two children is obviously a restriction. So far we have not used the fact that the vectors $\chi_S, S \in \mathcal{L}$ are linearly independent. We can handle the general case to prove $x_e \geq 1/3$ by using the following lemma.

Lemma 13.5. *Suppose C is a unique child of S. Then there must be at least two endpoints in S that belong to S.*

Proof. If there is no endpoint that belongs to S then $\delta(S) = \delta(C)$ but then χ_S and χ_C are linearly dependent. Suppose there is exactly one endpoint that belongs to S and let it be the endpoint of edge e. But then $x(\delta(S)) = x(\delta(C)) + x_e$ or $x(\delta(S)) = x(\delta(C)) - x_e$. Both cases are not possible because $x(\delta(S)) = f(S)$ and $x(\delta(C)) = f(C)$ where $f(S)$ and $f(C)$ are positive integers while $x_e \in (0, 1)$. Thus there are at least two end points that belong to S. ■

Using the preceding lemma we prove that $x_e \geq 1/3$ for some edge e. Let k be the number of leaves in $\mathcal{L}$ and h be the number of internal nodes with at least two children and let ℓ be the number of internal nodes with exactly one child. We again have $h < k$ and we also have $k + h + \ell = m$. Each leaf has at least four endpoints. Each internal node with exactly one child has at least two end points which means the total number of endpoints is at least $4k + 2\ell$. But $4k + 2\ell = 2k + 2k + 2\ell > 2k + 2h + 2\ell > 2m$ and there are only $2m$ endpoints for m edges. In other words, we can ignore the internal nodes with exactly one child since there are two endpoints in such a node/set and we can effectively charge one edge to such a node.

We now come to the more delicate argument to prove the tight bound that $x_e \geq \frac{1}{2}$ for some edge e. We describe invariant that effectively reduces the argument to the case where we can assume that $\mathcal{L}$ is a collection of leaves. This is encapsulated in the lemma below which requires some notation. Let $\alpha(S)$ be the number of sets of $\mathcal{L}$ contained in S including S itself. Let $\beta(S)$ be the number of edges whose *both* endpoints lie inside S. Recall that $f(S)$ is the requirement of S.

Lemma 13.6. *For all $S \in \mathcal{L}$, $f(S) \geq \alpha(S) - \beta(S)$.*

Assuming that the lemma is true we can do an easy counting argument. Let $R_1, R_2, \ldots, R_h$ be the maximal sets in $\mathcal{L}$ (the roots of the forest). Note that $\sum_{i=1}^{h} \alpha(R_i) = |\mathcal{L}| = m$. Applying the claim to each R_i and summing up,

$$\sum_{i=1}^{h} f(R_i) \geq \sum_{i=1}^{h} \alpha(R_i) - \sum_{i=1}^{h} \beta(R_i) \geq m - \sum_{i=1}^{h} \beta(R_i).$$

Note that $\sum_{i=1}^{h} f(R_i)$ is the total requirement of the maximal sets. And $m - \sum_{i=1}^{h} \beta(R_i)$ is the total number of edges that cross the sets $R_1, \ldots, R_h$. Let E' be the set of edges crossing these maximal sets. Now we are back to the setting with h disjoint sets and E' edges with $\sum_{i=1}^{h} f(R_i) \geq |E'|$. This easily leads to a contradiction as before if we assume that $x_e < \frac{1}{2}$ for all $e \in E'$. Formally, each set R_i requires $> 2f(R_i)$ edges crossing it from E' and therefore R_i contains at least $2f(R_i) + 1$ endpoints of edges from E'. Since $R_1, \ldots, R_h$ are disjoint the total number of endpoints is at least $2\sum_i f(R_i) + h$ which is strictly more than $2|E'|$.

Proof of Lemma 13.6. Thus, it remains to prove the claim which we do by inductively starting at the leaves of the forest for $\mathcal{L}$.

Case 1: S is a leaf node. We have $f(S) \geq 1$ while $\alpha(S) = 1$ and $\beta(S) = 0$ which verifies the claim.

Case 2: S is an internal nodes with k children $C_1, C_2, \ldots, C_k$. See Fig 13.5 for the different types of edges that are relevant. E_{cc} is the set of edges with end points in two different children of S. E_{cp} be the set of edges that cross exactly one child but do not cross S. E_{po} be the set of edges that cross S but do not cross

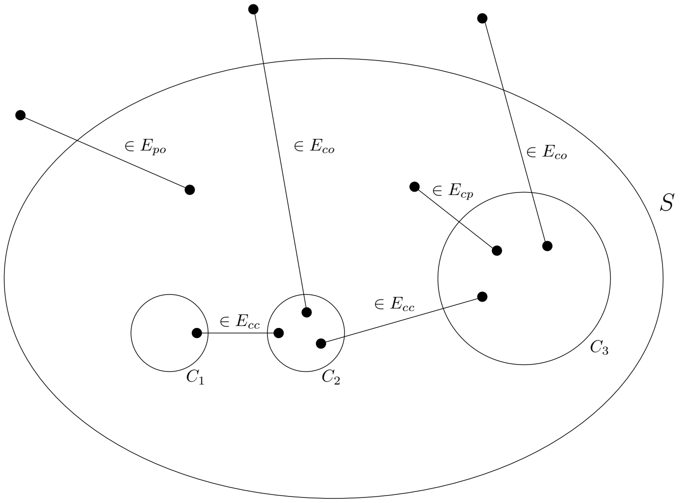


Figure 13.5: S is an internal node with several children. Different types of edges that play a role. p refers to parent set S, c refer to a child set and o refers to outside.

any of the children. E_{co} is the set of edges that cross both a child and S. This notation is borrowed from [155].

Let $\gamma(S)$ be the number of edges whose both endpoints belong to S but not to any child of S. Note that $\gamma(S) = |E_{cc}| + |E_{cp}|$.

Then,

$$\begin{aligned}\beta(S) &= \gamma(S) + \sum_{i=1}^{k} \beta(C_i) \\ &\geq \gamma(S) + \sum_{i=1}^{k} \alpha(C_i) - \sum_{i=1}^{k} f(C_i) \qquad (13.1) \\ &= \gamma(S) + \alpha(S) - 1 - \sum_{i=1}^{k} f(C_i)\end{aligned}$$

(13.1) follows by applying the inductive hypothesis to each child. From the preceding inequality, to prove that $\beta(S) \geq \alpha(S) - f(S)$ (the claim for S), it suffices to show the following inequality.

$$\gamma(S) \geq \sum_{i=1}^{k} f(C_i) - f(S) + 1. \qquad (13.2)$$

The right hand side of the above inequality can be written as:

$$\sum_{i=1}^{k} f(C_i) - f(S) + 1 = \sum_{e \in E_{cc}} 2x_e + \sum_{e \in E_{cp}} x_e - \sum_{e \in E_{po}} x_e + 1. \tag{13.3}$$

We consider two subcases.

Case 2.1: $\gamma(S) = 0$. This implies that E_{cc} and E_{cp} are empty. Since $\chi(\delta(S))$ is linearly independent from $\chi(\delta(C_1)), \ldots, \chi(\delta(C_k))$, we must have that E_{po} is not empty and hence $\sum_{e \in E_{po}} x_e > 0$. Therefore, in this case,

$$\sum_{i=1}^{k} f(C_i) - f(S) + 1 = \sum_{e \in E_{cc}} 2x_e + \sum_{e \in E_{cp}} x_e - \sum_{e \in E_{po}} x_e + 1 = -\sum_{e \in E_{po}} x_e + 1 < 1.$$

Since the left hand side is an integer, it follows that $\sum_{i=1}^{k} f(C_i) - f(S) + 1 \le 0 = \gamma(S)$.

Case 2.2: $\gamma(S) \ge 1$. Recall that $\gamma(S) = |E_{cc}| + |E_{cp}|$.

$$\sum_{i=1}^{k} f(C_i) - f(S) + 1 = \sum_{e \in E_{cc}} 2x_e + \sum_{e \in E_{cp}} x_e - \sum_{e \in E_{po}} x_e + 1 \le \sum_{e \in E_{cc}} 2x_e + \sum_{e \in E_{cp}} x_e + 1$$

By our assumption that $x_e < \frac{1}{2}$ for each e, we have $\sum_{e \in E_{cc}} 2x_e < |E_{cc}|$ if $|E_{cc}| > 0$, and similarly $\sum_{e \in E_{cp}} x_e < |E_{cp}|/2$ if $|E_{cp}| > 0$. Since $\gamma(S) = |E_{cc}| + |E_{cp}| \ge 1$ we conclude that

$$\sum_{e \in E_{cc}} 2x_e + \sum_{e \in E_{cp}} x_e < \gamma(S).$$

Putting together we have

$$\sum_{i=1}^{k} f(C_i) - f(S) + 1 \le \sum_{e \in E_{cc}} 2x_e + \sum_{e \in E_{cp}} x_e + 1 < \gamma(S) + 1 \le \gamma(S)$$

as desired. ■

Tightness of the analysis: The LP relaxation has an integrality gap of 2 even for the MST problem. Let G be the cycle on n vertices with all edge costs equal to 1. Then setting $x_e = 1/2$ on each edge is feasible and the cost is $n/2$ while the MST cost is $n - 1$. Note that the optimum fractional solution here is 1/2-integral. However, there are more involved examples (see Jain's paper or [152]) based on the Petersen graph where the optimum basic feasible solution is not half-integral while there are one or more edges with fractional value at least 1/2. Jain's iterated rounding algorithm is an unusual algorithm in that the output of the algorithm may not have any discernible structure until it is completely done.

Running time: The strength of the iterated rounding approach is the remarkable approximation guarantees it delivers for various problems. The weakness is the high running time which is due to two reasons. First, one needs a basic feasible solution for the LP — this is typically much more expensive than finding an approximately good feasible solution. Second, the algorithm requires computing an LP solution many times. Finding faster algorithms with comparable approximation guarantees is an open research area.

Chapter 14

Introduction to Cut and Partitioning Problems

Graph cut and partitoning problems such as the well-known s-t minimum cut problem play a fundamental role in combinatorial optimization. Many natural cut problems that go beyond the s-t cut problem are NP-Hard and there has been extensive work on approximation algorithms and heuristics since they arise in many applications. In addition to algorithms, the structural results that capture approximate relationships between flows and cuts (called flow-cut gaps), and the connections to the theory of metric embeddings as well as graph theory have led to many beautiful and important results.

14.1 s-t mincut via LP Rounding and Maxflow-Mincut

Let $G = (V, E)$ be a directed graph with edge costs $c : E \rightarrow \mathbb{R}_+$. Let $s, t \in V$ be distinct vertices. The s-t mincut problem is to find the cheapest set of edges $E' \subseteq E$ such that there is no s-t path in $G - E'$. An s-t cut is often also defined as $\delta^+(S)$ for some $S \subset V$ where $s \in S, t \in V - S$. Suppose E' is an s-t cut. Let S be the set of nodes reachable from s in $G - E'$, then $\delta(S) \subseteq E'$ and moreover $\delta(S)$ is an s-t cut. Thus, it suffices to focus on such limited type of cuts, however in some more general settings it is useful to keep these notions separate.

It is well-known that s-t mincut can be computed efficiently via s-t maximumflow which also establishes the maxflow-mincut theorem. This is a fundamental theorem in combinatorial optimization with many direct and indirect applications.

Theorem 14.1. *Let $G = (V, E)$ be a directed graph with rational edge capacities $c : E \rightarrow \mathbb{Q}_+$ and let $s, t \in V$ be distinct vertices. Then the s-t maximum flow value in G*

is equal to the s-t minimum cut value and both can be computed in strongly polynomial time. Further, if c is integer valued then there exists an integer-valued maximum flow.

The proof of the preceding theorem is typically established via the augmenting path algorithm for computing a maximum flow. Here we take a different approach to finding an s-t cut via an LP relaxation whose dual can be seen as the the maxflow LP.

Suppose we want to find an s-t mincut. We can write it as an integer program as follows. For each edge $e \in E$ we have a boolean variable $x_e \in \{0, 1\}$ to indicate whether we cut e. The constraint is that for any path $P \in \mathcal{P}_{s,t}$ (here $\mathcal{P}_{s,t}$ is the set of all s-t paths) we must choose at least on edge from P. This leads to the following IP.

$$
\begin{aligned}
\min \sum_{e \in E} c(e) x_e & & & \\
\sum_{e \in P} x_e & \geq & 1 & \quad P \in \mathcal{P}_{s,t} \\
x_e & \in & \{0, 1\} & \quad e \in E.
\end{aligned}
$$

The LP relaxation is obtained by changing $x_e \in \{0, 1\}$ to $x_e \geq 0$ since we can omit the constraints $x_e \leq 1$. We note that the LP has an exponential number of constraints, however, we have an efficient separation oracle since it corresponds to computing the shortest s-t path. The LP can be viewed as assigning lengths to the edges such that the shortest path between s and t according to the lengths is at least 1. This is a fractional relaxation of the cut.

Rounding the LP: We will prove that the LP relaxation can be rounded without any loss! The rounding algorithm is described below.

```
Theta-Rounding($G, s, t$)

1. Solve LP to obtain fractional solution $y$

2. For each $v \in V$ let $d_y(s, v)$ be the shortest path distance from $s$ to $v$
   according to edge lengths $y_e$.

3. Pick $\theta$ uniformly at random from $(0, 1)$

4. Output $E' = \delta^+(B(s, \theta))$ where $B(s, \theta) = \{v \mid d_y(s, v) \leq \theta\}$ is the ball of
   radius $\theta$ around $s$
```

It is easy to see that the algorithm outputs a valid s-t since $d_y(s, t) \geq 1$ by feasibility of the LP solution y and hence $t \notin B(s, \theta)$ for any $\theta < 1$.

Lemma 14.1. *Let $e = (u, v)$ be an edge.* $\mathbf{P}$[*e is cut by algorithm*] $\leq y(u, v)$.

Proof. An edge $e = (u, v)$ is cut iff $d_y(s, u) \leq \theta < d_y(s, v)$. Hence the edge is not cut if $d_y(s, v) \leq d_y(s, u)$. If $d_y(s, v) > d_y(s, u)$ we have $d_y(s, v) - d_y(s, u) \leq y(u, v)$. Since θ is chosen uniformly at random from $(0, 1)$ the probabilty that θ lies in the interval $[d_y(s, u), d_y(s, v)]$ is at most $y(u, v)$. ∎

Corollary 14.2. *The expected cost of the cut output by the algorithm is at most* $\sum_e c(e)y_e$.

The preceding corollary shows that there is an integral cut whose cost is at most that of the LP relaxation which implies that the LP relaxation yields an optimum solution. The algorithm can be easily derandomized by trying "all possible value of θ". What does this mean? Once we have y we compute the shortest path distances from s to each vertex v. We can think of these distances as producing a *line embedding* where we place s at 0 and each vertex v at $d_y(s, v)$. The only interesting choices for θ are given by the n values of $d_y(s, v)$ and one can try each of them and the corresponding cut and find the cheapest one. It is guaranteed to be at most $\sum_e c(e)y_e$.

What is the dual LP? We write it down below and you can verify that it is the path version of the maxflow!

$$\begin{aligned} \max_{P \in \mathcal{P}_{s,t}} z_P & & & \\ \sum_{P : e \in P} z_P &\leq& c(e) & \quad e \in E \\ z_P &\geq& 0 & \quad P \in \mathcal{P}_{s,t}. \end{aligned}$$

Thus, we have seen a proof of the maxflow-mincut theorem via LP rounding of a relaxation for the s-t cut problem.

A compact LP via distance variables: The path based LP relaxation for the s-t mincut problem is natural and easy to formulate. We can also express shortest path constraints via distance variables. We first write a bigger LP than necessary via variables $d(u, v)$ for all ordered pairs of vertices (hence there are n^2 variables). We need triangle inequality constraints to enforce that $d(u, v)$ values respect shortest path distances.

$$\begin{aligned} \min_{(u,v) \in E} c(u, v)d(u, v) & & & \\ d(u, v) + d(v, w) - d(u, w) &\geq& 0 & \quad u, v, w \in V \\ d(s, t) &\geq& 1 & \\ d(u, v) &\geq& 0 & \quad (u, v) \in V \times V \end{aligned}$$

Although the preceding LP is wasteful in some ways it is quite generic and can be used for many cut problems where we are interested in distances between multiple pairs of vertices.

Now we consider a more compact LP formulation. We have two types of variables, $x(u,v)$ for each edge $(u,v) \in E$ and d_v variables for each $v \in V$ to indicate distances from s.

$$
\begin{aligned}
\min_{(u,v)\in E} c(u,v)x(u,v) & & & \\
d_v &\leq d_u + x(u,v) & & (u,v) \in E \\
d_t &\geq 1 & & \\
d_v &\geq 0 & & v \in V \\
x(u,v) &\geq 0 & & (u,v) \in E
\end{aligned}
$$

Exercise 14.1. Write the dual of the above LP and see it as the standard edge-based flow formulation for s-t maximum flow.

14.2 A Catalog of Cut and Partitioning Problems

Here we list a few prominent problems.

Multiway Cut: Input is an undirected graph $G = (V, E)$ and edge weights $w : E \rightarrow \mathbb{R}_+$ and a set of k terminals $\{s_1, s_2, \ldots, s_k\} \subseteq V$. The goal is to remove a minimum weight subset of edges such that there is no path from s_i to s_j for any $i \neq j$. This problem is NP-Hard even for $k = 3$. **Node-weighted Multiway Cut** is the generalization to the setting when G has node-weights instead of edge-weights. **Directed Multiway Cut** is the version where G is a directed graph and the goal is to remove a minimum weight set of edges so that there is no s_i-s_j path for any $i \neq j$. Note that the directed version generalizes the node-weighted undirected problem as well.

***k*-Cut:** In k-Cut the input is a graph G and integer k. The goal is to remove a minimum weight set of edges such that the resulting graph has at least k non-trivial components. The problem is NP-Hard when k is part of the input but admits a polynomial time algorithm for any fixed k. A common generalization of k-Cut and Multiway Cut is the so-called **Steiner *k*-Cut**: here the input is a graph G, a set of terminals $S \subseteq V$ and an integer k where $k \leq |S|$. The goal is to remove a minimum-weight subset of edges such that there are at least k components that each contain a terminal.

Multicut: The input is an edge-weighted graph $G = (V, E)$ and k source-sink pairs $(s_1, t_1), (s_2, t_2), \ldots, (s_k, t_k)$ and the goal is to remove a minimum-weight set of edges so that there is no path from s_i to t_i for all $i \in [k]$. Note that Multiway Cut is a special case. Multicut also naturally generalizes to the node-weighted setting and to directed graphs.

Sparsest Cut: We discuss the more general version first called the *Non-uniform* Sparsest Cut. The input is an edge-capacitated graph $G = (V, E)$ and k source-sink pairs $(s_1, t_1), \ldots, (s_k, t_k)$ and associated non-negative scalar demand values $D_1, D_2, \ldots, D_k$. The goal is to find a minimum-weight subset of edges $E' \subseteq E$ such the ratio $\frac{c(E')}{D(E')}$ is minimized where $c(E')$ is the total capacity of edges in E' and $D(E')$ is the total demand of pairs separated by removing E'. In undirected graphs one can see that this ratio is also minimized by a connected component and hence one can alternatively phrase the problem as $\min_{S \subseteq V} \frac{c(\delta(S))}{D(\delta(S))}$. In the *Uniform* Sparsest Cut problem we associate $D(u, v) = 1$ for every pair of vertices and hence one wants to find $\min_{S \in V} \frac{c(\delta(S))}{|S||V-S|}$. This is closely related (to within a factor of 2) to the problem of finding the *expansion* of a graph where the objective is $\min_{S \in V, |S| \leq |V|/2} \frac{c(\delta(S))}{|S|}$. Other closely related variants are to find the *conductance* and sparsest cut for *product* multicommodity flow instances where the demands are induced by vertex weights (that is, $D(u, v) = \pi(u)\pi(v)$ where $\pi : V \rightarrow mathbbR_+$).

Sparsest Cut can be generalized to node-weighted settings and to directed graphs. In directed graphs there is a substantial difference when considering the non-uniform versus the uniform settings because the latter can be thought of a symmetric version. We will not detail the issues here.

Minimum Bisection and Balanced Partitioning: The input is an undirected edge-weighted graph G. In Minimum Bisection the goal is to partition V into V_1, V_2 where $\lfloor |V|/2 \rfloor \leq |V_1|, |V_2| \leq \lceil |V|/2 \rceil$ so as to minimize the weight of the edges crossing the partition. In Balanced Partition the sizes of the two parts can be approximately equal — there is a balance parameter $\alpha \in (0, 1/2])$ and the goal is to partition V into V_1, V_2 such that $\alpha|V| \leq |V_1|, |V_2| \leq (1-\alpha)|V|$. These problems are partially motivated by parallel and distributed computation where a graph representing some computation is recursively decomposed into several pieces while minimizing the communication required between the pieces (captured by the edge weights). In this context partitioning into k given pieces to minimize the number of edges while each piece has size roughly $|V|/k$ is also considered as Balanced k-Partition.

Hypergraphs and Submodular functions: One can generalize several edge-weighted problems to node-weighted problems in a natural fashion but in some cases it is useful to consider other ways to model. In this context hypergraphs come in handy and have their own intrinsic appeal. Recall that a hypergraph $H = (V, E)$ consists of a vertex set V and a set of hyperedges E where each $e \in E$ is a subset of vertices; thus $e \subseteq V$. The rank of a hypergraph is the maximum cardinality of a hyperedge, typically denote by r. Graphs are rank 2 hypergraphs. One can typically reduce a hypergraph cut problem to a node-weighted cut problem and vice-versa with some distinctions based on the specific problem at hand. Finally some of the problems can be naturally lifted to the setting where we consider an abstract submodular set function defined over the vertex set V; that is we are given $f : 2^V \rightarrow \mathbb{R}_+$ rather than a graph or a hypergraph and the goal is to partition V where the cost of the partition is now measured with respect to f.

Chapter 15

Multiway Cut

Multiway Cut problem is the following[1]. Given a graph $G = (V, E)$ with edge weights $w : E \rightarrow \mathbb{R}_+$, and k terminal vertices $\{s_1, s_2, \ldots, s_k\}$, remove a minimum weight set of edges such that there is no s_i-s_j path left for any $i \neq j$. We phrase it in this long-winded way since the definition then naturally generalizes to other settings. In the case of undirected graphs it is also useful to view the problem as a *partition* problem. Consider a feasible solution to the given instance; it consists of a set of edges whose removal leaves at least k components and no two terminals are in the same connected component. One can verify that in a minimal solution, if the original graph is connected, then we can assume that there will be exactly k components with each terminal contained in a separate one. Thus, an alternative view is to find a partition of V into k sets $V_1, V_2, \ldots, V_k$ such that $s_i \in V_i$ for $1 \leq i \leq k$ and the goal is to minimize $\sum_i w(\delta(V_i))/2$ where the factor of 2 is because any edge crossing the partition is counted exactly twice in $\sum_i w(\delta(V_i))/2$. When $k = 2$ we can solve the problem in polynomial time since it is the same problem as finding an s_1-s_2 minimum cut in G. The problem is known to be NP-Hard and APX-Hard even when $k = 3$ [46]. In this chapter we first consider a simple combinatorial heuristic that yields a $2(1 - 1/k)$-approximation, followed by a distance based LP relaxation that also yields a $2(1 - 1/k)$-approximation with a matching integrality gap. We then describe a geometric relaxation that yields the current best approximation ratio of 1.2965 [139] (the best known lower bound on the integrality gap for this relaxation is 1.20016 [21]). We will also discuss node-weighted and directed versions in the last section.

[1] In the literature the problem is also referred to as the *Multiterminal Cut* problem.

15.1 Isolating Cut Heuristic

Let $S = \{s_1, s_2, \ldots, s_k\}$ be the given set of terminals. A feasible solution to a given instance of Multiway Cut separates each terminal s_i from the terminals $S - \{s_i\}$. One can find the cheapest cut separating s_i from $S - \{s_i\}$ via a minimum-cut computation (we shrink $S - \{s_i\}$ to a single vertex t and compute an s-t minimum cut). Taking the union of such cuts for each $i \in [k]$ clearly yields a feasible solution. It turns that this simple heuristic is not too bad.

```
IsolatingCut($G = (V, E), S = \{s_1, \ldots, s_k\}$)
1. for each $i \in [k]$ do
    A. Let $E_i$ be a minimum weight cut separating $s_i$ from $S - \{s_i\}$ in $G$
2. Without loss of generality $w(E_1) \leq w(E_2) \leq \ldots \leq w(E_k)$ (otherwise
   reindex)
3. Ouput $\cup_{i=1}^{k-1} E_i$
```

We leave the following as an easy exercise.

Lemma 15.1. *The algorithm outputs a feasible solution.*

Theorem 15.1. *The Isolating Cut heuristic yields a* $2(1 - 1/k)$*-approximation for* Multiway Cut.

Proof. Recall the partition view of the Multiway Cut problem. Let E^* be an optimum and minimal solution and let $V_1, V_2, \ldots, V_k$ be the components of $G - E^*$ such that $s_i \in V_i$ for $i \in [k]$. Each edge $e \in E^*$ crosses the partition and since it has exactly two end points, we see that $w(E^*) = 2\sum_{i=1}^{k} w(\delta(V_i))$.

We claim that for each $i \in [k]$ $w(E_i) \leq w(\delta(V_i))$. This follows from the fact that $\delta(V_i)$ is a cut that separates s_i from $S - \{s_i\}$. Therefore $\sum_{i=1}^{k} w(E_i) \leq \sum_{i=1}^{k} w(\delta(V_i)) = 2w(E^*)$. Since $w(E_k) \geq w(E_i)$ for all $i \in [k]$, $\sum_{i=1}^{k-1} w(E_i) \leq (1 - 1/k)\sum_{i=1}^{k} w(E_i)$, and hence we have $\sum_{i=1}^{k-1} w(E_i) \leq 2(1 - 1/k)w(E^*) = 2(1 - 1/k)\,\mathrm{OPT}$. ■

Exercise 15.1. The analysis of the algorithm is tight. Find an example to demonstrate this.

15.2 Distance based LP Relaxation

We describe an LP relaxation based on distance/length variables that generalizes the relaxation for s-t cut that we saw previously. For each edge e there is a length variable x_e indicating whether e is cut or not. We require that the length of any path p connecting s_i and s_j, $i \neq j$, should be at least 1.

$$\begin{array}{lll} \min & \sum_{e \in E} c_e x_e & \\ \text{s.t.} & & \\ & \sum_{e \in p} x_e \geq 1 & p \in \mathcal{P}_{s_i, s_j}, i \neq j \\ & x_e \geq 0 & e \in E \end{array}$$

The preceding LP can be solved in polynomial time via the Ellipsoid method since the separation oracle is the shortest path problem. Alternatively, one can write a compact LP. We focus on rounding the LP and establishing its integrality gap.

```
BallCut($G = (V, E)$, $S \subseteq V$)

1. Solve distance based LP relaxation to obtain solution $x$

2. Let $d_x$ be the metric induced on $V$ by edge lengths $x$

3. Pick $\theta$ uniformaly at random from $(0, 1/2)$

4. Output $\cup_{i=1}^{k} \delta(B_d(s_i, \theta))$ where $B(s_i, \theta)$ is the ball of radius $\theta$ around $s_i$
```

Lemma 15.2. *The algorithm outputs a feasible solution.*

Proof. Consider a terminal s_i. $B_d(s_i, \theta)$ does not contain any other terminal s_j, $j \neq i$ since $\theta < 1/2$ and $d(s_i, s_j) \geq 1$. Thus removing the edges $\delta(B_d(s_i, \theta))$ disconnects s_i from every other terminal. Since this holds for each s_i taking the union of the edges $\cup_i \delta(B_d(s_i, \theta))$ disconnects each terminal from all other terminals. ■

Consdier the open balls of radius $1/2$ around the terminals, that is, $B_d(s_i, 1/2)$ for $i \in [k]$. We observe that they are disjoint for if $v \in B(s_i, 1/2) \cap B(s_j, 1/2)$ with $i \neq j$ then $d(s_i, s_j) < 1$ which would violate the LP constraint. Thus the algorithm is essentially running an s-t cut type algorithm in the disjoint balls but since we only have half the radius we lose a factor of 2. In fact, as we will see in a remark later, the analysis holds even if each s_i chose its own θ_i independently — the correlated choice is not exploited here — but a correlated choice is quite useful when consider other cut problems including Directed Multiway Cut later in the chapter.

Lemma 15.3. *Let $e = (u, v)$.* $\mathbf{P}[e \text{ is cut}] \leq 2x_e$.

Proof. There are several cases to consider but the main one is the one where both $u, v \in B_d(s_i, 1/2]$ for some s_i. Suppose this is the case. Then only s_i can cut the edge (u, v). It is easy to see that $\mathbf{P}[e \text{ is cut}] = 2|d(s_i, u) - d(s_i, v)|$ since we pick θ uniformly at random from $(0, 1/2)$. But $|d(s_i, u) - d(s_i, v)| \leq x_e$ by triangle inequality and hence we obtain the desired claim.

Now we consider the case when $u \in B(s_i, 1/2)$ and $v \in B(s_j, 1/2)$ where $i \neq j$. Let $\alpha = 1/2 - d(s_i, u)$ and let $\beta = 1/2 - d(s_j, v)$. We observe that $\alpha + \beta \leq x_e$ for otherwise $d(s_i, u) + x_e + d(s_j, v) < 1$. We see that e is cut iff θ lies in the interval $[d(s_i, u), 1/2)$ or it lies in the interval $[d(s_j, v), 1/2)$. Thus e is cut with probability $2\max(\alpha, \beta) \leq 2(\alpha + \beta) \leq 2x_e$.

There are two other cases. One is when both u, v are outside every half-ball. In this case the edge e is not cut. The other is when $u \in B_d(s_i, 1/2)$ for some i and v is not inside any ball. The analysis here is similar to the second case and we leave it as an exercise. ■

Thus the expected cost of the cut is at most $2\sum_e c_e x_e \leq 2\,\mathrm{OPT}_{LP} \leq 2\,\mathrm{OPT}$. One can improve and obtain a $2(1 - 1/k)$-approximation by saving on one terminal as we did in the preceding section.

Exercise 15.2. Modify the algorithm to obtain a $2(1 - 1/k)$-approximation with respect to the LP relaxation.

Exercise 15.3. Consider a variant of the algorithm where each s_i picks an independent $\theta_i \in (0, 1/2)$; we output the cut $\cup_i \delta(B_d(s_i, \theta_i))$. Prove that this also yields a 2-approximation (can be improved to $2(1 - 1/k)$-approximation).

Integrality gap: The analysis is tight as shown by the following integrality gap example. Consider a star with center r and k leaves $s_1, s_2, \ldots, s_k$ which are the terminals. All edges have cost 1. Then it is easy to see that a feasible integral solution consists of removing $k - 1$ edges from the star. Hence $\mathrm{OPT} = k - 1$. On the other hand, setting $x_e = 1/2$ for each edge is a feasible fractional solution of cost $k/2$. Hence the integrality gap is $2(1 - 1/k)$.

15.3 A Partitioning View and Geometric Relaxation

Section to be filled in later. For now see notes form 2018 `https://courses.engr.illinois.edu/cs583/sp2018/Notes/multiwaycut-ckr.pdf`.

15.4 Node-weighted and Directed Multiway Cut

Section to be filled in later. For now see paper which has very short and elementary proofs of 2-approximations for both problems which are tight.

http://chekuri.cs.illinois.edu/papers/dir-multiway-cut-soda.pdf.

Chapter 16

Multicut

In the Multicut problem, we are given a graph $G = (V, E)$, a capacity function that assigns a capacity c_e to each edge e, and a set of pairs $(s_1, t_1), \ldots, (s_k, t_k)$. The Multicut problem asks for a minimum capacity set of edges $F \subseteq E$ such that removing the edges in F disconnects s_i and t_i, for all i. Note that the Multicut problem generalizes the Multiway Cut problem that we saw previously. We describe an $O(\log k)$ approximation algorithm for Multicut.

We start by describing an LP formulation for the problem. For each edge e, we have a variable d_e. We interpret each variable d_e as a distance label for the edge. Let $\mathcal{P}_{s_i,t_i}$ denote the set of all paths between s_i and t_i. We have the following LP for the problem:

$$\begin{array}{lrl} \min & \sum_{e \in E} c_e d_e & \\ \text{s.t.} & & \\ & \sum_{e \in p} d_e \geq 1 & p \in \mathcal{P}_{s_i,t_i}, 1 \leq i \leq k \\ & d_e \geq 0 & e \in E \end{array}$$

The LP assigns distance labels to edges so that, on each path p between s_i and t_i, the distance labels of the edges on p sum up to at least one. Note that, even though the LP can have exponentially many constraints, we can solve the LP in polynomial time using the ellipsoid method and the following separation oracle. Given distance labels d_e, we set the length of each edge to d_e and, for each pair (s_i, t_i), we compute the length of the shortest path between s_i and t_i and check whether it is at least one. If the shortest path between s_i and t_i has length smaller than one, we have a violated constraint. Conversely, if all shortest paths have length at least one, the distance labels define a feasible solution.

We also consider the dual of the previous LP. For each path p between any pair (s_i, t_i) we have a dual variable f_p. We interpret each variable f_p as the amount of flow between s_i and t_i that is routed along the path p. We have the following dual LP:

$$\begin{aligned} \max \quad & \sum_{i=1}^{k} \sum_{p \in \mathcal{P}_{s_i,t_i}} f_p \\ \text{s.t.} \quad & \\ & \sum_{p:\, e \in p} f_p \leq c_e & e \in E(G) \\ & f_p \geq 0 & p \in \mathcal{P}_{s_1,t_1} \cup \ldots \cup \mathcal{P}_{s_k,t_k} \end{aligned}$$

The dual is an LP formulation for the Maximum Throughput Multicommodity Flow problem. In the Maximum Throughput Multicommodity Flow problem, we have k different commodities. For each i, we want to route commodity i from the source s_i to the destination t_i. Each commodity must satisfy flow conservation at each vertex other than its source and its destination. Additionally, the total flow routed on each edge must not exceed the capacity of the edge. The goal is to maximize the sum of the commodities routed.

The dual LP tries to assign an amount of flow f_p to each path p so that the total flow on each edge is at most the capacity of the edge (the flow conservation constraints are automatically satisfied). Note that the endpoints of the path p determine which kind of commodity is routed along the path.

Exercise 16.1. Write the Multicut LP and its dual in a compact form with polynomially many constraints.

16.1 Upper Bound on the Integrality Gap

In this section, we will show that the integrality gap of the LP is $O(\log k)$ using a randomized rounding algorithm due to Calinescu, Karloff, and Rabani [27]. The first algorithm that achieved an $O(\log k)$-approximation for Multicut is due to Garg, Vazirani, and Yannakakis [63] (see [152] and [155]), and it is based on the region growing technique introduced by Leighton and Rao [113]. The reason that we choose to present the randomized rounding algorithm is due to its future application for metric embeddings.

Let $B_d(v, r)$ denote the ball of radius r centered at the vertex v in the metric induced by the distance labels d_e.

```
CKR-RandomPartition:
  Solve the LP to get the distance labels $d_e$
  Pick $\theta$ uniformly at random from $[0, 1/2)$
  Pick a random permutation $\sigma$ on $\{1, 2, ..., k\}$
  for $i = 1$ to $k$
      $V_{\sigma(i)} = B_d(s_{\sigma(i)}, \theta) \setminus \bigcup_{j<i} V_{\sigma(j)}$
  Output $\bigcup_{i=1}^{k} \delta(V_i)$
```

Lemma 16.1. *CKR-RandomPartition correctly outputs a feasible multicut for the given instance.*

Proof. Let F be the set of edges output by the algorithm. Suppose F is not a feasible multicut. Then there exists a pair of vertices (s_i, t_i) such that there is a path between s_i and t_i in $G - F$. Therefore there exists a j such that V_j contains s_i and t_i. Since $V_j \subseteq B_d(s_j, \theta)$, both s_i and t_i are contained in the ball of radius θ centered at s_j. Consequently, the distance between s_j and s_i is at most θ and the distance between s_j and t_i is at most θ. By the triangle inequality, the distance between s_i and t_i is at most 2θ. Since θ is smaller than $1/2$, it follows that the distance between s_i and t_i is smaller than one. This contradicts the fact that the distance labels d_e are a feasible solution for the LP. Therefore F is a multicut, as desired. ■

Lemma 16.2. *The probability that an edge e is cut is at most $2H_k d_e$, where H_k is the k-th harmonic number and d_e is the distance label of the edge e.*

Proof. Fix an edge $e = (u, v)$. Let:

$$L_i = \min\{d(s_i, u), d(s_i, v)\}$$

$$R_i = \max\{d(s_i, u), d(s_i, v)\}$$

We may assume without loss of generality that $L_1 \leq L_2 \leq ... \leq L_k$ (be reindexing the pairs as needed). See Fig 16.1.

Let A_i be the event that the edge e is cut *first* by s_i. More precisely, A_i is the event that $|V_i \cap \{u, v\}| = 1$ and $|V_j \cap \{u, v\}| = 0$ for all j such that $\sigma(j) < \sigma(i)$. Note that $|V_i \cap \{u, v\}| = 1$ simply says that s_i cuts the edge e. If s_i is the first to cut the edge e, for all j that come before i in the permutation, neither u nor v can be in V_j (if only one of u and v is in V_j, s_j cuts the edge e; if both u and v are in V_j, s_i cannot cut the edge e).

Note that the event that the edge e is cut is the union of the disjoint events $A_1, ..., A_k$. Therefore we have:

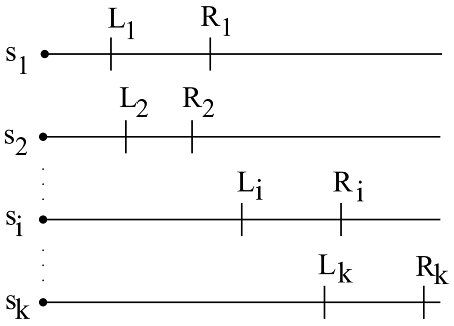
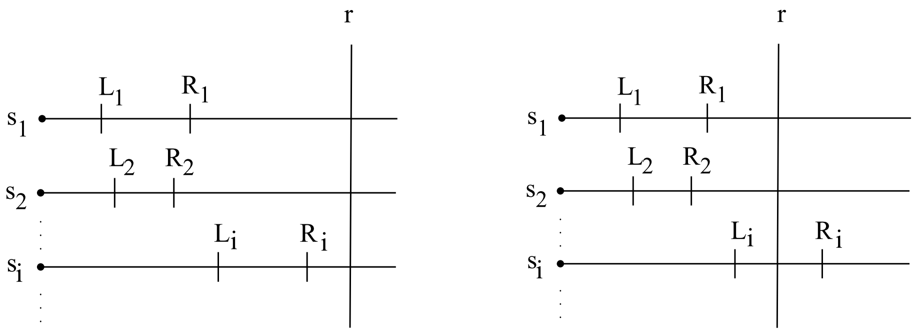


Figure 16.1: For a fixed edge $e = (u, v)$ we renumber the pairs such that $L_1 \leq L_2 \leq \ldots \leq L_k$.

$$\mathbf{P}[e \text{ is cut}] = \sum_i \mathbf{P}[A_i].$$

Let us fix $r \in [0, 1/2)$ and consider $\mathbf{P}[A_i \mid \theta = r]$. Note that s_i cuts the edge e only if one of u, v is inside the ball of radius r centered at s_i and the other is outside the ball. Differently said, s_i cuts the edge only if $r \in [L_i, R_i)$:

$$\mathbf{P}[A_i \mid \theta = r] = 0 \quad \text{if } r \notin [L_i, R_i)$$



Now suppose that $r \in [L_i, R_i)$. Let us fix $j < i$ and suppose j comes before i in the permutation (that is, $\sigma(j) < \sigma(i)$). Recall that, since $j < i$, we have $L_j \leq L_i \leq r$. Therefore at least one of u, v is inside the ball of radius r centered at s_j. Consequently, s_i cannot be the first to cut the edge e. Therefore s_i is the first to cut the edge e only if $\sigma(i) < \sigma(j)$ for all $j < i$. See Fig 16.2. Since σ is a random permutation, i appears before j for all $j < i$ with probability $1/i$. Therefore we have:

$$\mathbf{P}[A_i \mid \theta = r] \le \frac{1}{i} \quad \text{if } r \in [L_i, R_i)$$

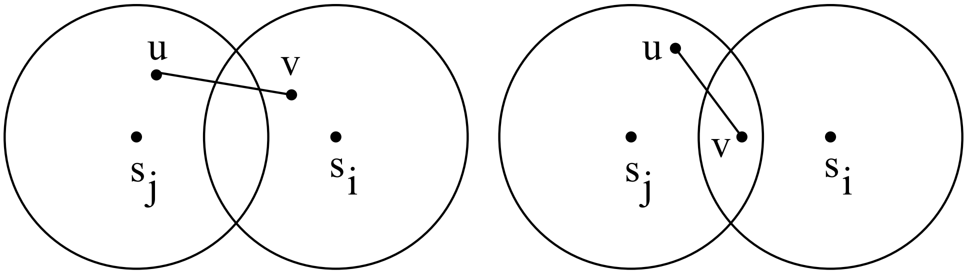


Figure 16.2: If $\sigma(j) < \sigma(i)$, s_i cannot be the first to cut the edge $e = (u, v)$. On the left s_j also cuts the edge. On the right s_j captures both end points and therefore s_i cannot cut it.

Since θ was selected uniformly at random from the interval $[0, 1/2)$, and independently from σ, we have:

$$\mathbf{P}[A_i] \le \frac{1}{i}\ \mathbf{P}[\theta \in [L_i, R_i)] = \frac{2}{i}\ (R_i - L_i)$$

By the triangle inequality, $R_i \le L_i + d_e$. Therefore:

$$\mathbf{P}[A_i] \le \frac{2d_e}{i}$$

Consequently,

$$\mathbf{P}[e \text{ is cut}] = \sum_i \mathbf{P}[A_i] \le 2H_k d_e.$$

■

Corollary 16.1. *The integrality gap of the* Multicut *LP is* $O(\log k)$*.*

Proof. Let F be the set of edges outputted by the Randomized Rounding algorithm. For each edge e, let χ_e be an indicator random variable equal to 1 if and only if the edge e is in F. As we have already seen,

$$\mathbf{E}\,\chi_e = \mathbf{P}[\chi_e = 1] \le 2H_k d_e$$

Let $c(F)$ be a random variable equal to the total capacity of the edges in F. We have:

$$\mathbf{E}\, c(F) = \mathbf{E} \sum_e c_e \chi_e = \sum_e c_e\, \mathbf{P}[\chi_e] \leq 2H_k \sum_e c_e d_e = 2H_k \text{ OPT}_{\text{LP}}$$

Consequently, there exists a set of edges F such that the total capacity of the edges in F is at most $2H_k$ OPT_{LP}. Therefore OPT $\leq 2H_k$ OPT_{LP}, as desired. ■

Corollary 16.2. *The algorithm achieves an $O(\log k)$-approximation (in expectation) for the* Multicut *problem.*

Proof. As we have already seen,

$$\mathbf{E}\, c(F) \leq 2H_k \text{ OPT}_{\text{LP}}$$

where F is the set of edges output by the algorithm and $c(F)$ is the total capacity of the edges in F. Since $\text{OPT}_{LP} \leq \text{OPT}$,

$$\mathbf{E}\, c(F) \leq 2H_k \text{ OPT} = O(\log k)\text{OPT}$$

■

Remark 16.1. The expected cost analysis can be used to obtain an algorithm, via repetition, a randomized algorithm that ouputs an $O(\log k)$-approximation with high probability. The algorithm can also be derandomized but it is not straight forward. As we remarked there is an alternative deterministic $O(\log k)$-approximation algorithm via region growing.

Flow-Cut Gap: Recall that when $k = 1$ we have the well-known maxflow-mincut theorem. The integrality gap of the standard LP for MulitCut is the same as the relative gap between flow and cut when k is arbitrary. The upper bound on the integrality gap gives an upper bound on the gap.

Corollary 16.3. *We have:*

$$\max_{\textit{m.c. flow } f} |f| \leq \min_{\textit{multicut } C} |C| \leq O(\log k) \left(\max_{\textit{m.c. flow } f} |f| \right)$$

where $|f|$ represents the value of the multicommodity flow f, and $|C|$ represents the capacity of the multicut C.

Proof. Let OPT_{LP} denote the total capacity of an optimal (fractional) solution for the Multicut LP. Let OPT_{dual} denote the flow value of an optimal solution for the dual LP. Since OPT_{LP} is a lower bound on the capacity of the minimum (integral) multicut, we have:

$$\max_{\text{m.c. flow } f} |f| = \text{OPT}_{dual} = \text{OPT}_{\text{LP}} \leq \min_{\text{multicut } C} |C|$$

As we have already seen, we have:

$$\min_{\text{multicut } C} |C| \leq 2H_k \ \mathrm{OPT}_{\mathrm{LP}} = 2H_k \ \mathrm{OPT}_{dual} = 2H_k \left(\max_{\text{m.c. flow } f} |f| \right)$$

■

16.2 Lower Bound on the Integrality Gap

In this section, we will show that the integrality gap of the LP is $\Omega(\log k)$. That is, we will give a Multicut instance for which the LP gap is $\Omega(\log k)$. Let's start by looking at expander graphs and their properties.

16.2.1 Expander Graphs

Definition 16.4. *A graph $G = (V, E)$ is an α-edge-expander if, for any subset S of at most $|V|/2$ vertices, the number of edges crossing the cut $(S, V \backslash S)$ is at least $\alpha|S|$.*

Note that the complete graph K_n is a $(|V|/2)$-edge-expander. However, the more interesting expander graphs are also sparse. Cycles and grids are examples of graphs that are very poor expanders.

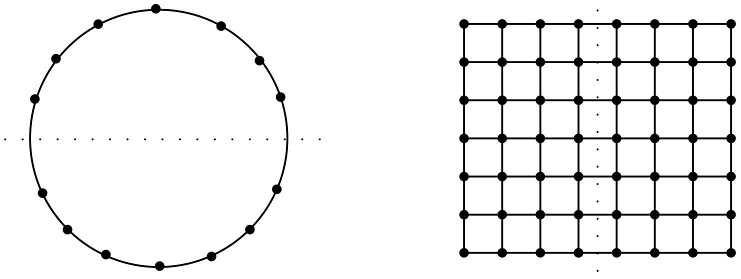


Figure 16.3: The top half of the cycle has $|V|/2$ vertices and only two edges crossing the cut. The left half of the grid has roughly $|V|/2$ vertices and only $\sqrt{|V|}$ edges crossing the cut.

Definition 16.5. *A graph G is d-regular if every vertex in G has degree d.*

Note that 2-regular graphs consist of a collection of edge disjoint cycles and therefore they have poor expansion. However, for any $d \geq 3$, there exist d-regular graphs that are very good expanders.

Theorem 16.6. *For every $d \geq 3$ there exists an infinite family of d-regular 1-edge-expanders.*

We will only need the following special case of the previous theorem.

Theorem 16.7. *There exists a universal constant $\alpha > 0$ and an integer n_0 such that, for all even integers $n \geq n_0$, there exists an n-vertex, 3-regular α-edge-expander.*

Proof Idea. The easiest way to prove this theorem is using the probabilistic method. The proof itself is beyond the scope of this lecture[1]. The proof idea is the following.

Let's fix an even integer n. We will generate a 3-regular random graph G by selecting three random perfect matchings on the vertex set $\{1, 2, ..., n\}$ (recall that a perfect matching is a set of edges such that every vertex is incident to exactly one of these edges). We select a random perfect matching as follows. We maintain a list of vertices that have not been matched so far. While there is at least one vertex that is not matched, we select a pair of distinct vertices u, v uniformly at random from all possible pairs of unmatched vertices. We add the edge (u, v) to our matching and we remove u and v from the list. We repeat this process three times (independently) to get three random matchings. The graph G will consist of the edges in these three matchings. Note that G is actually a 3-regular multigraph since it might have parallel edges (if the same edge is in at least two of the matchings). There are two properties of interest: (1) G is a simple graph and (2) G is an α-edge-expander for some constant $\alpha > 0$. If we can show that G has both properties with positive probability, it follows that there exists a 3-regular α-edge-expander (if no graph is a 3-regular α-edge-expander, the probability that our graph G has both properties is equal to 0).

It is not very hard to show that the probability that G does not have property (1) is small. To show that the probability that G does not have property (2) is small, for each set S with at most $n/2$ vertices, we estimate the expected number of edges that cross the cut $(S, V \backslash S)$ (e.g., we can easily show that $|\delta(S)| \geq |S|/2$). Using tail inequalities (e.g., Chernoff bounds), we can show that the probability that $|\delta(S)|$ differs significantly from its expectation is extremely small (i.e., small enough so that the sum – taken over all sets S – of these probabilities is also small) and we can use the union bound to get the desired result. ■

Note that explicit constructions of d-regular expanders are also known. Margulis [119] gave an infinite family of 8-regular expanders. There are many explicit construction by now and it is a very important topic of study — we refer the reader to the survey on expanders by Hoory, Linial and Wigderson

[1] A more accurate statement is that the calculations are a bit involved and not terribly interesting for us.

[86]. The vertex set of a graph G_n in Margulis' construction is $\mathbb{Z}_n \times \mathbb{Z}_n$, where $\mathbb{Z}_n$ is the set of all integers mod n. The neighbors of a vertex (x, y) in G_n are $(x+y, y), (x-y, y), (x, y+x), (x, y-x), (x+y+1, y), (x-y+1, y), (x, y+x+1)$, and $(x, y-x+1)$ (all operations are mod n). Another example is the following infinite family of 3-regular expanders. For each prime p, we have a 3-regular graph G_p. The vertex set of G_p is $\mathbb{Z}_p$. The neighbors of a vertex x in G_p are $x+1$, $x-1$, and x^{-1} (as before, all operations are mod p; x^{-1} is the inverse of x mod p, and we define the inverse of 0 to be 0)[2].

We conclude this section with the following observations (they will be very useful in showing the $\Omega(k)$ lower bound on the integrality gap of the LP).

Claim 16.2.1. *Let G be an n-vertex d-regular α-edge-expander, for some constants $d \geq 3$ and $\alpha > 0$. Then the diameter of G is $\Theta(\log n)$.*

Proof. For any two vertices u and v, let $dist(u, v)$ denote the length of a shortest path between u and v (the length of a path is the number of edges on the path). Let's fix a vertex s. Let L_i be the set of all vertices v such that $dist(s, v)$ is at most i. Now let's show that $(1 + \alpha/d)|L_{i-1}| \leq |L_i| \leq d|L_{i-1}|$. Clearly, $|L_1| = d$ (since s has degree d). Therefore we may assume that $i > 1$. Every vertex in L_i is in L_{i-1} or it has a neighbor in L_{i-1}. Therefore it suffices to bound $|L_i \backslash L_{i-1}|$.

Note that any vertex in L_{i-1} has at least one neighbor in L_{i-1}. Therefore the vertices in L_{i-1} have at most $(d-1)|L_{i-1}|$ neighbors outside of L_{i-1}. Consequently, $|L_i| \leq d|L_{i-1}|$.

Now one of L_{i-1}, $V \backslash L_{i-1}$ has at most $|V|/2$ vertices. Let's assume without loss of generality that L_{i-1} has at most $|V|/2$ vertices (the other case is symmetric). Let $A = L_{i-1}$ and let B be the set of all vertices in $V \backslash L_{i-1}$ that have a neighbor in L_{i-1} (note that $|L_i| = |A| + |B|$). Let F be the set of all edges that cross the cut $(L_{i-1}, V \backslash L_{i-1})$. Now let's look at the bipartite graph $H = (A, B, F)$. Since G is an α-edge-expander, we have $|F| \geq \alpha|A|$. Moreover, $|F| = \sum_{v \in B} d_H(v)$, where $d_H(v)$ is the degree of v in H. Since $d_H(v)$ is at most d, we have $\alpha|A| \leq |F| \leq d|B|$. Therefore we have:

$$L_i = |A| + |B| \geq (1 + \alpha/d)|A| = (1 + \alpha/d)|L_{i-1}|$$

It follows by induction that $d(1 + \alpha/d)^{i-1} \leq |L_i| \leq d^i$. Therefore $dist(s, v)$ is $O(\log n)$ for all v and there exists a vertex v such that $dist(s, v)$ is $\Omega(\log n)$. Since this is true for any s, it follows that the diameter of G is $\Theta(\log n)$. ■

Claim 16.2.2. *Let G be an n-vertex 3-regular α-edge-expander and let $B(v, i)$ be the set of all vertices u such that there is a path between u and v with at most i edges. For any vertex v, $|B(v, \log_3 n/2)| \leq \sqrt{n}$.*

[2] Note that, unlike Margulis' construction, this construction is not very explicit since we don't know how to generate large primes deterministically.

Proof. Note that $B(v, \log_3 n/2)$ is the set of all vertices w such that $dist(v,w)$ is at most $\log_3 n/2$. As we have seen in the proof of the previous claim, we have $|B(v, \log_3 n/2)| \leq 3^{\log_3 n/2} = \sqrt{n}$. ■

16.2.2 The Multicut Instance

Let n_0, α be as in Theorem 7. Let $n \geq n_0$ and let G be an n-vertex 3-regular α-edge-expander. For each edge e in G, we set the capacity c_e to 1. Now let $X = \{(u,v) | u \notin B(v, \log_3 n/2)\}$. The pairs in X will be the pairs (s_i, t_i) that we want to disconnect. Let (G, X) be the resulting Multicut instance.

Claim 16.2.3. *There exists a feasible fractional solution for (G, X) of capacity $O(n/\log n)$.*

Proof. Let $d_e = 2/\log_3 n$, for all e. Note that, since G is 3-regular, G has $3n/2$ edges. Therefore the total capacity of the fractional solution is

$$\sum_e d_e = \frac{3n}{2} \cdot \frac{2}{\log_3 n} = \frac{3n}{\log_3 n}$$

Therefore we only need to show that the solution is feasible. Let (u,v) be a pair in X. Let's consider a path p between u and v. Since u is not in $B(v, \log_3 n/2)$, the path p has more than $\log_3 n/2$ edges (recall that $B(v,i)$ is the set of all vertices u such that there is a path between u and v with at most i edges). Consequently,

$$\sum_{e \in p} d_e > \frac{\log_3 n}{2} \cdot \frac{2}{\log_3 n} = 1$$

■

Claim 16.2.4. *Any integral solution for (G, X) has capacity $\Omega(n)$.*

Proof. Let F be an integral solution for (G, X). Let $V_1, ..., V_h$ be the connected components of $G - F$. Fix an i and let v be an arbitrary vertex in the connected component V_i. Note that, for any u in V_i, there is a path between v and u with at most $\log_3 n/2$ edges (if not, (u,v) is a pair in X which contradicts the fact that removing the edges in F disconnects every pair in X). Therefore V_i is contained in $B(v, \log_3 n/2)$. It follows from Claim 16.2.2 that $|V_i| \leq \sqrt{n}$. Since G is an α-edge-expander and $|V_i| \leq |V|/2$, we have $|\delta(V_i)| \geq \alpha |V_i|$, for all i. Consequently,

$$|F| = \frac{1}{2} \sum_{i=1}^{h} |\delta(V_i)| \geq \frac{\alpha}{2} \sum_{i=1}^{h} |V_i| = \frac{\alpha n}{2}$$

Therefore F has total capacity $\Omega(n)$ (recall that every edge has unit capacity). ■

Theorem 16.8. *The integrality gap of the* Multicut *LP is* $\Omega(\log k)$.

Proof. Note that $k = |X| = O(n^2)$. It follows from claims 10 and 11 that the LP has integrality gap $\Omega(\log n) = \Omega(\log k)$, as desired. ∎

Bibliographic Notes

Multicut is closely related to the Sparsest Cut problem. Initial algorithm for Multicut were based on algorithms for Sparsest Cut. Garg, Vazirani and Yannakakis [63] then used Leighton and Rao's region growing argument (as well as their integrality gap example on expanders for the uniform sparsest cut problem) [113] to obtain a tight $O(\log k)$ bound on the integrality gap for Multicut. The randomized proof that we described is from the work of Calinescu, Karloff and Rabani [27] on the 0-extension problem; their algorithm and analysis eventually led to an optimal bound for approximating an arbitrary metric via random trees [54]. For planar graphs (and more generally any proper minor closed family of graphs) the integrality gap is $O(1)$, as shown by Klein, Plotkin and Rao [104] — the constant depends on the family. There have been several subsequent refinements of the precise dependence of the constant in the integrality gap — see [1]. The $O(\log k)$ bound extends to node-weighted case and the $O(1)$ approximation for planar graphs also extends to the node-weighted case. Multicut is APX-Hard even on trees and in general graphs. Assuming UGC, the problem is known to be hard to approximate to a super-constant factor [36]. For some special case of Multicut based on the structure of the demand graph one can obtain improved approximation ratios [39].

The directed graph version of Multicut turns out be much more difficult. The flow-cut gap is known to be $\tilde{\Omega}(n^{1/7})$ and the problem is also known to be hard to approximate to almost polynomial factors; these negative results are due to Chuzhoy and Khanna [45]. The best known approximation ratio is $\min\{k, \tilde{O}(n^{11/23})\}$ [3]. Very recently Kawarabayashi and Sidiropoulos obtained a poly-logarithmic approximation for Directed Multicut if G is a planar directed graph [101]. There is a notion of symmetric demands in directed graphs and for that version of Multicut one can get a poly-logarithmic flow-cut gap and approximation; see [37, 105]. This is closely connected to the Feedback Arc Set problem in directed graphs [53, 138].

Chapter 17

Sparsest Cut

Sparsest Cut is a fundamental problem in graph algorithms with many applications and connections. There are several variants that are considered in the literature and they are closely related but it is useful to have proper terminology and understand the similarities and differences.

Non-Uniform Sparsest Cut: We consider the general one first. The input is a graph $G = (V, E)$ with non-negative edge capacities $c : E \to \mathbb{R}_+$ and a set of pairs $(s_1, t_1), ..., (s_k, t_k)$ along with non-negative demand values $D_1, D_2, \ldots, D_k$. When considering undirected graphs the demand pairs unordered — by this we mean that we do not distinguish (s_1, t_1) from (t_1, s_1). One can also think of the demand values as "weights" but the demand terminology makes more sense when considering the dual flow problem. Given a set/cut $S \subseteq V$ the *sparsity* of the cut S is defined as the ratio $\frac{c(\delta(S))}{(\sum_{i:S\cap\{s_i,t_i\}=1} D_i)}$. The numerator is the capacity of the cut and the denominator is the total demand of the pairs separated by S. The goal is to find the cut S with minimum sparsity. In other words we are trying to find the best "bang per buck" cut: how much capacity do we need to remove per demand separated? It is sometime convenient to consider G as the supply graph and the demands as coming from a demand graph $H = (V, F)$ where F represents the pairs and we associate $D : F \to \mathbb{R}_+$ to represent the demand value (alternatively we can also consider multigraphs). With this representation of the pairs the sparsity of cut S is simply $\frac{c(\delta_G(S))}{D(\delta_H(S))}$ note that $\delta_G(S)$ represents the supply edges crossing S and $\delta_H(S)$ represents the demand edges crossing the cut S.

Remark 17.1. One can define a cut as removing a set of edges. In the case of sparsest cut in undirected graphs it suffices to restrict attention to cuts of the form $\delta(S)$ for some $S \subseteq V$. It is a useful exercise to see why there is always a sparsest cut of that form for any given instance. This is not necessarily true for directed graphs.

Uniform Sparsest Cut: Very often when people say Sparsest Cut they mean the uniform version. This is the version in which $D(u,v) = 1$ for each unordered pair of vertices (u,v). For these demands the a cut S is $\frac{c(\delta_G(S))}{|S||V \setminus S|}$. Alternatively the demand graph H is a complete graph with unit demand values on each edge. A slightly generalization of Uniform Sparsest Cut is obtained by considering demands induced by weights on vertices (the dual flow instances are called Prodcut Multicommodity Flow instances). There is a weight function $\pi : V \to \mathbb{R}_+$ on the vertices and and demand $D(u,v)$ for pair (u,v) is set to be $\pi(u)\pi(v)$. Note that if $\pi(u) = 1$ for all u then we obtain Uniform Sparsest Cut. If $\pi(u) \in \{0,1\}$ for all u then we are focusing our attention on sparsity with respect to the set $V' = \{v \mid \pi(v) = 1\}$ since the the vertices with $\pi(u) = 0$ play no role. This may seem unnatural at first but it is closely connected to expansion and conductance as we will see below.

Expansion: The expansion of a multi-graph $G = (V,E)$ is defined as $\min_{S:|S| \le |V|/2} \frac{|\delta(S)|}{|S|}$. Recall that G is an α-expander if the expansion of G is at least α. A random 3-regular graph is α-expander with $\alpha = \Omega(1)$ with high probability. Thus, to find an α-expander one can obtain an efficient randomized algorithm by picking a random graph and then verifying its expansion. However, checking expansion is coNP-Hard. Expansion is closely related to Uniform Sparsest Cut. Note that when $|S| \le |V|/2$ we have

$$\frac{1}{|V|}\frac{|\delta(S)|}{|S|} \le \frac{|\delta(S)|}{|S||V \setminus S|} \le \frac{2}{|V|}\frac{|\delta(S)|}{|S|}.$$

Thus Expansion and Uniform Sparsest Cut are within a factor of 2 of each other. Sometimes it is useful to consider expansion with vertex weights $w : V \to \mathbb{R}_+$. Here the expansion is defined as $\min_{S:w(S) \le w(V)/2} \frac{|\delta(S)|}{w(S)}$. This corresponds to product multicommodity flow instances where $\pi(v) = w(v)$. The term Conductance is sometimes used to denote the quantity $\frac{|\delta(S)|}{\mathrm{vol}(S)}$ where $\mathrm{vol}(S) = \sum_{v \in S} \deg(v)$ (here vol is short for volume). When a graph is regular the definition of expansion and conductance are the same but not in the general setting. Note that we can capture conductance by setting weights on vertices where $w(v) = \deg(v)$.

Some key applications: Uniform Sparsest Cut is fundamentally interesting because it helps us directly and indirectly solve the Balanced Separator problem. In the latter problem we want to partition $G = (V,E)$ into two pieces $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$ where $|V_1|$ and $|V_2$ are roughly the same size so that we minimize the number of edges between V_1 and V_2. One can repeatedly use a sparse cut routine to get a balanced separator. The other key application is to certify expansion of a graph. Expander graphs and relatives arise in many applications and knowing whether a graph is expanding or not is very useful.

17.0.1 LP Relaxation and Maximum Concurrent Flow

How do we write an LP relaxtion for Sparsest Cut? This is less obvious than it is for Multicut and other cut problems where we have explicit terminal pairs that we wish to separate. We consider the Non-Uniform Sparsest Cut since it is the most general version. First we will try to develop an integer program. We will have two sets of variables. For each pair (s_i, t_i) we will have a variable y_i to indicate whether we want to separate the pair. For each edge we will have a variable x_e to indicate whether e is cut. If we decide to separate pair i then for every path between s_i and t_i we should cut at least one edge on the path — this is similar to relaxations we have seen before. The following captures the problem:

$$\min \frac{\sum_{e \in E} c_e x_e}{\sum_{i=1}^{k} D_i y_i}$$

$$\begin{array}{rcll} \sum_{e \in p} x_e & \geq & y_i & p \in \mathcal{P}_{s_i,t_i}, i \in [k] \\ x_e & \in & \{0,1\} & e \in E \\ y_i & \in & \{0,1\} & i \in [k] \end{array}$$

Note, however, that the objective is a ratio and not linear. It is a standard trick to obtain an LP relaxation wherein we normalize the denominator in the ratio to 1 and relaxt the variables to be real-valued. Thus we obtain the following LP relaxation.

$$\min \sum_{e \in E} c_e x_e$$

$$\begin{array}{rcll} \sum_{i=1}^{k} D_i y_i & = & 1 & \\ \sum_{e \in p} x_e & \geq & y_i & p \in \mathcal{P}_{s_i,t_i}, i \in [k] \\ x_e & \geq & 0 & e \in E \\ y_i & \geq & 0 & i \in [k] \end{array}$$

Exercise 17.1. Show that the LP is indeed a relaxation for the Sparsest Cut problem. Formally, given an integer feasible solution with sparsity λ find a feasible solution to the relaxation such that its value is no more than λ.

Now we consider the dual LP.

$$\begin{aligned} & \max \lambda \\ \sum_{p \in \mathcal{P}_{s_i,t_i}} y_p &\geq \lambda D_i \quad i \in [k] \\ \sum_{i=1}^{k} \sum_{p \in \mathcal{P}_{s_i,t_i}, e \in p} y_p &\leq c_e \quad e \in E \\ y_p &\geq 0 \quad p \in \mathcal{P}_{s_i,t_i}, i \in [k] \end{aligned}$$

The dual LP is a multicommodity flow. It solves the Maximum Concurrent Multicommodity Flow problem for the given instance. In other words it finds the largest value of λ such that there is a feasibly multicommodity flow for the given pairs in which the flow routed for pair (s_i, t_i) is at least λD_i. It is called "concurrent flow" since we need to route all demand pairs to the same factor which is in constrast to the dual of Multicut which corresponds to the maximum throughput multicommodity flow (in which some pairs may have zero flow while others have a lot of flow).

Exercise 17.2. Suppose we have a cut S with sparsity $c(\delta(S))/(\sum_{i:S \cap \{s_i,t_i\}=1} D_i)$. Why is the maximum concurrent flow at most the sparsity of S?

Note that the LP can be solved via the Ellipsoid method. One can also write a compact LP via distance variables which will help us later to focus on constraining the metric in other ways.

$$\begin{aligned} \min \sum_{uv \in E} c(uv) d(uv) & \\ \sum_{i=1}^{k} D_i d(s_i t_i) &= 1 \\ d \text{ is a metric on } V & \end{aligned}$$

Flow-cut gap: The flow-cut gap in this context is the following equivalent way of thinking about the problem. Consider a multicommodity flow instance on G with demand pairs $(s_1, t_1), \ldots, (s_k, t_k)$ and demand values $D_1, \ldots, D_k$. Suppose G satisfies the *cut-condition*, that is, for every $S \subseteq V$ the capacity $c(\delta(S))$ is at least the demand separated by S. Can we route all the demand pairs? This is true when $k = 1$ but is not true in general even for $k = 3$ in undirected graphs. The question is the maximum value of λ such that we can route λD_i for every pair i? The worst-case integrality gap of the preceding LP relaxation for Sparsest Cut is precisely the flow-cut gap. One can ask about the flow-cut gap for all graphs, a specific class of graphs, for a specific class of demand graphs, a specific class of supply and demand graphs, and so on.

In these notes we will establish that the flow-cut gap in general undirected graphs is at most $O(\log k)$. And there are instances where the gap is $\Omega(\log k)$. It is conjectured that the gap is $O(1)$ for planar graphs but the best upper bound we have is $O(\sqrt{\log n})$. Resolving the flow-cut gap in planar graphs is a major open problem.

Remark 17.2. Approximating the Sparsest Cut problem is not the same as establishing flow-cut gaps. One can obtain improved approximations for Sparsest Cut via stronger relaxations than the natural LP. Indeed the best approximation ratio for Sparsest Cut is $O(\sqrt{\log n})$ via an SDP relaxation.

17.1 Rounding LP via Connection to Multicut

There are close connections between Sparsest Cut and Multicut. By repeatedly using Sparsest Cut routine and Set Cover style analysis prove the following.

Exercise 17.3. Suppose there is an $\alpha(k, n)$-approximation for Non-Uniform Sparsest Cut. Prove that this implies an $O(\alpha(k, n) \ln k)$-approximation for Multicut.

Can we prove some form a converse? That is, can we use an approximation algorithm for Multicut to obtain an approximation algorithm for Sparsest Cut? Note that if someone told us the pairs to separate in an optimum solution to the Sparsest Cut instance then we can use an (approximation) algorithm for Multicut to separate those pairs. Here we show that one can use the LP relaxation and obtain an algorithm via the integrality gap that we have already establised for Multicut LP. We sketch the argument and focus our attention on the simpler case when $D_i = 1$ for all $i \in [k]$. We give this argument even though it does not lead to the optimum ratio, for historical interest, as well as to illustrate a useful high-level technique that has found applications in other settings.

Identifying the pairs to separate from LP solution: Suppose we solve the LP and obtain a feasible solution (x, y). y_i indicates the extent to which pair i is separated. Suppose we have an ideal situation where $y_i \in \{0, p\}$ for every i. Let $A = \{i \mid y_i = p\}$. We have $|A| = 1/p$ since $\sum_i y_i = 1$. Then it is intutively clear that the LP is separating the pairs in A. We can then solve the Multicut problem for the pairs in A and consider the ratio of the cost of the cut to $|A|$. How do we argue about this algorithm? We do the following. Consider a fractional assignment $x' : E \to \mathbb{R}_+$ where $x'_e = \min\{1, x_e/p\}$; in other words we scale each x_e by $1/p$. Note that $y_i = d_x(s_i, t_i)$. Since we scaled up each x_e by $1/p$ it is not hard to see that $d_{x'}(s_i, t_i) \geq 1$; in other words x' is a feasible solution to the Multicut instance on G for the pairs in A. The fractional cost of x' is $\sum_e c_e x'_e \leq \sum_e c_e x_e/p$. Thus, by the algorithm for Multicut in the previous chapter, we can find a feasible Multicut $E' \subseteq E$ that separates all pairs in A and $c(E') = O(\log k) \sum_e c_e x_e/p$. What is the sparsity of this cut? It is $c(E')/|A|$ which is $O(\log k) \sum_e x_e$. Thus the sparsity of the cut is $O(\log k)\lambda$ where λ is the value of the LP relaxation.

Now we consider the general setting. Recall that $\sum_i y_i = 1$. We partition the pairs into groups that have similar y_i values. For $j \geq 0$, let $A_j = \{i \mid y_i \in (1/2^{j+1}, 1/2^j]$. Thus all pairs in A_j have a y_i value that are within a factor of 2 of each other.

Claim 17.1.1. *There exists a $j \leq \log_2 k$ such that $\sum_{i \in A_j} y_i \geq \frac{1}{2(1+\log_2 k)} \geq \frac{1}{4 \log k}$.*

Proof. Consider any i such that $i \in A_j$ where $j > \log_2 k$. By definition we have $y_i \leq 1/2k$. Since there are only k pairs, $\sum_{j > \log_2 k} \sum_{i \in A_j} y_i \leq k/2k \leq 1/2$. Thus $\sum_{j \leq \log_2 k} \sum_{i \in A_j} y_i \geq 1/2$ and therefore, there must be a $j \leq \log_2 k$ such that $\sum_{i \in A_j} y_i \geq \frac{1}{2(1+\log_2 k)}$ (there are only so many groups). ■

Consider the A_j with $\sum_{i \in A_j} y_i \geq \frac{1}{4 \log_2 k}$. For each $i \in A_j$ we have $1/2^{j+1} \leq y_i \leq 1/2^j$. Therefore $|A|_j| \geq \min\{1, 2^j/4 \log_2 k\}$. The algorithm now separates the pairs in A_j via an algorithm for Multicut.

Claim 17.1.2. *Consider the fractional solution $x' : E \to [0,1]$ where $x'_e = \min\{1, 2^{j+1} x_e\}$. Then $d_{x'}(s_i, t_i) \geq 1$ for all $i \in A_j$. Thus x' is a feasible fractional solution to the Multicut LP for separating the pairs in A_j.*

Via the rounding algorithm in the preceding chapter we have there is a set $E' \subseteq E$ such that E' is a feasible multicut for the pairs in A_j and $c(E') = O(\log k) 2^{j+1} \sum_e c_e x_e$. The sparsity of this cut is $c(E')/|A_j| = O(\log^2 k) \sum_e c_e x_e$. Thus we obtained an $O(\log^2 k)$-approximation for Sparsest Cut when $D_i = 1$ for each pair.

Remark 17.3. When demands are not 1 (or identical) the preceding argument yields an $O(\log k \log D)$ approximation where $D = \sum_i D_i$ with the normalization that $D_i \geq 1$ for all i.

17.2 Rounding via ℓ_1 embeddings

The optimal rounding of the LP relaxation turns out to go via metric embedding theory and this connection was discovered by Linial, London and Rabinovich [117] and Aumann and Rabani [**AR98**]. We need some basics in metric embeddings to point out the connection and rounding.

17.2.1 A digression through trees

It is instructive to consider the simple setting when G is a tree $T = (V, E)$. In this case it is easy to find the sparsest cut. For each edge $e \in T$ we can associate a cut S_e which is one side of the two components in $T - e$. The capacity of the cut $\delta(S_e)$, by defintion, is c_e. Let $D(e) = \sum_{i: S_e \cup \{s_i, t_i\} = 1} D_i$ be the demand separated by e. The sparsity of the cut S_e is simply c_e / D_e. Finding sparsest cut in a tree is easy from the following exercise.

Exercise 17.4. The sparsest cut in a tree is given by $\arg\min_e c_e / D_e$.

A more interesting exercise is to prove that the LP relaxation gives an optimum solution on a tree.

Lemma 17.1. *Let (x, y) be a feasible solution to the LP with objectie value λ. If G is a tree T then there is an edge $e \in T$ such that $c_e / D_e \leq \lambda$.*

Proof. We have $\lambda = \frac{\sum_e c_e x_e}{\sum_i D_i d_x(s_i, t_i)}$ where $d_x(s_i, t_i)$ is the shortest path distances between s_i and t_i. There is a unique path P_{s_i, t_i} from s_i to t_i in a tree so $d_x(s_i, t_i) = \sum_{e \in P_{s_i, t_i}} x_e$. Thus,

$$\begin{aligned} \lambda &= \frac{\sum_e c_e x_e}{\sum_i D_i d_x(s_i, t_i)} \\ &= \frac{\sum_e c_e x_e}{\sum_i D_i \sum_{e \in P_{s_i, t_i}} x_e} \\ &= \frac{\sum_e c_e x_e}{\sum_e D_e} \\ &\geq \min_e \frac{c_e}{D_e}. \end{aligned}$$

In the last inequality we are using the simple fact that $\frac{a_1 + a_2 + \ldots + a_n}{b_1 + b_2 + \ldots + b_n} \geq \min_i \frac{a_i}{b_i}$ for positive a's and b's. ∎

What made the proof work for trees? Is there a more general phenomenon than the fact that trees are pretty simple structures? It turns out that the key fact is that shortest path distances induced by a tree are ℓ_1 metrics or equivalently cut metrics.

17.2.2 Cut metrics, line metrics, and ℓ_1 metrics

Let (V, d) be a finite metric space. We will be interested in two special types of metrics.

Definition 17.1. *Let V be a finite set and let $S \subseteq V$. The metric d_S associated with the cut S is the following: $d_S(u, v) = 1$ if $|S \cap \{u, v\}| = 1$ and $d_S(u, v) = 0$ otherwise.*

Definition 17.2. *Let (V, d) be a finite metric space. The metric d is a* cut metric *if there is a set $S \subset V$ such that $d = d_S$. d is in the cut cone (or in the cone of cut metric) if there exist non-negative scalars $y_S, S \subset V$ such that $d(u, v) = \sum_{S \subset V} y_S d_S(u, v)$ for all $u, v \in V$.*

Definition 17.3. *Let (V, d) be a finite metric space. The metrid d is a* line metric *if there is a mappting $f : V \longrightarrow \mathbb{R}$ (the real line) such that $d(u, v) = |f(u) - f(v)|$ for all $u, v \in V$.*

Definition 17.4. *Let (V, d) be a finite metric space. The metric d is an ℓ_1 metric if there is some integer d and a mapping $f : V \longrightarrow \mathbb{R}^d$ (Euclidean space in d dimensions) such that $d(u, v) = |f(u) - f(v)|_1$ (the ℓ_1 distance) for all $u, v \in V$.*

Claim 17.2.1. *A metric (V, d) is an ℓ_1 metric iff it is a non-negative combination of line metrics (in the cone of line metrics).*

Proof Sketch. If d is an ℓ_1 metric then each dimension corresponds to a line metric and since the ℓ_1 metric is separable over the dimensions it is a non-negative combination of line metric. Conversely, any non-negative combination of line metrics can be made into an ℓ_1 metric where each line metric becomes a separate dimension (scalar multiplication of a line metric is also a line metric). ■

Lemma 17.2. *d is an ℓ_1 metric iff d is in the cut cone.*

Proof. Consider the metric d_S. It is easy to that it is a simple line metric. Map all vertices in S to 0 and all vertices in $V - S$ to 1. If d is in the cut cone then it is a non-negative combination of the cut metrics, and hence it is a non-negative combination of line metrics, and hence an ℓ_1 metric.

To prove the converse, it suffices to argue that any line metric is in the cut cone. Let $V = \{v_1, v_2, \ldots, v_n\}$ and let d be a line metric on V. Without loss of generality assume that the coordinates of the points corresponding to the line metric d are $x_1 \leq x_2 \leq\leq x_n$ on the real line. For $1 \leq i < n$ let $S_i = \{v_1, v_2, \ldots, v_i\}$. It is not hard to verify that $\sum_i |x_{i+1} - x_i| d_{S_i} = d$. ■

17.2.3 Brief introducton to metric embeddings

Let (V, d) me a finite metric space. Note that any finite metric space can be viewed as one that is derived from the shortest path metric induced on a graph with some non-negative edge lengths. Thus if $G = (V, E)$ is a simple graph and $\ell : E \rightarrow \mathbb{R}_+$ are some edge-lengths then the metric induced on V depends both on the "topology" of G as well as the lengths. Finite metrics can encode graph structure and hence can be diverse. When trying to round we may want to work with simpler metric spaces. One way to do this is to *embed* a given metric space (V, d) into a simpler *host* metric space (V', d'). An embedding is simply a mapping of V to V'. Note that that even though we may be interested in a finite metric space V the host metric space can be continuous/infinite such as the Euclidean space in some dimenstion d. Embedding typically *distorts* the distances and thus one wants to find embeddings with small distortion. Distortion can be measured as additive or in a relative sense and for our purposes we will mainly focus on relative notion of distortion.

Definition 17.5. *An embedding of a finite metric space (V, d) to a host metric space (V', d') is a mapping $f : V \rightarrow V'$. The embedding is an* isometric embedding *if $d(u, v) = d'(f(u), f(v))$ for all $u, v \in V$. An embedding is a contraction if $d'(f(u), f(v)) \leq d(u, v)$ for all $u, v \in V$. An embedding is non-contracting if $d'(f(u), f(v)) \geq d(u, v)$ for all $u, v \in V$.*

Definition 17.6. *Let (V, d) and (V', d') be two metric spaces and let $f : V \rightarrow V'$ be an embedding. The* distortion *of f is $\max_{u,v \in V, u \neq v} \max\{\frac{d(u,v)}{d'(f(u),f(v))}, \frac{d'(f(u),f(v))}{d(u,v)}\}$.*

Of particular importance are embeddings of finite metric spaces into Euclidean space $\mathbb{R}^d$ where the distance is measured under various norms such as the ℓ_p norm for various values of p. Of particular interest are $\ell_1, \ell_2, \ell_\infty$. An embedding of a finite metric space (V, d) into $\mathbb{R}^d$ means that we map each v to a point $(x_1, x_2, \ldots, x_d)$ and the distance between say x, y is measured as $\|x - y\|$ for some norm of interest.

The dimension d is also important in various applications but in some settings like with Sparsest Cut the dimension is not important.

Theorem 17.7 (Bourgain). *Any n-point finite metric space can be embedded into ℓ_2 (and hence also ℓ_1) with distortion $O(\log n)$. Moreover the embedding is a contraction and can be constructed in randomized polynomial time and embeds points into $\mathbb{R}^d$ where $d = O(\log^2 n)$.*

In fact one can obtain a refined theorem that is useful for Sparsest Cut.

Theorem 17.8 (Bourgain). *Let (V, d) be n-point finite metric and let $S \subseteq V$ with $|S| = k$. Then there is a randomized polynomial time algorithm to compute an* *embedding $f : V \longrightarrow \mathbb{R}^{O(\log^2 n)}$ such that (i) the embedding is a contraction (that is, $\|f(u) - f(v)\|_1 \leq d(u, v)$ for all $u, v \in V$ and (ii) for every $u, v \in S$, $\|f(u) - f(v)\|_1 \geq \frac{c}{\log k} d(u, v)$ for some universal constant c.*

17.2.4 Utilizing the ℓ_1 embedding

We saw that the integrality gap of the LP is 1 on trees since the shortest path metric on trees is in the cut cone (equivalently ℓ_1-embeddable). More generally one can prove that if the shortest path metric on a graph G embeds into ℓ_1 with distortion α then the integrality gap of the LP for Sparsest Cut is at most α. This will imply an $O(\log n)$-integrality gap via Bourgain's theorem since any n point finite metric embeds in to ℓ_1 with distortion $O(\log n)$.

Theorem 17.9. *Let $G = (V, E)$ be a graph. Suppose any finite metric induced by edge lengths on E can be embedded into ℓ_1 with distortion α. Then the integrality gap of the LP for* Sparsest Cut *is at most α for any instance on G.*

Proof. Let (x, y) be a feasible fractional solution and let d be the metric induced by edge lengths given by x. Let λ be the value of the solution and recall that $\lambda = \frac{\sum_{uv \in E} c(uv) d(uv)}{\sum_{i=1}^{k} D_i d(s_i, t_i)}$.

Since d can be embedded into ℓ_1 with distortion at most α and any ℓ_1 metric is in the cut-cone, it implies that there are scalaras $z_S, S \subset V$ such that for all u, v

$$\frac{1}{\alpha} \sum_{S \subset V} y_S d_S(u, v) \leq d(u, v) \leq \sum_{S \subset V} y_S d_S(u, v).$$

Here we assumed without loss of generality that the embedding is a contraction. For a set $S \subset V$ we use $\mathrm{Dem}(\delta(S)) = \sum_{i:|S \cap \{s_i, t_i\}|=1} D_i$ to denote the total demand crossing the cut S.

$$\begin{aligned} \lambda &= \frac{\sum_{uv \in E} c(uv) d(uv)}{\sum_{i=1}^{k} D_i d(s_i, t_i)} \\ &\geq \frac{1}{\alpha} \frac{\sum_{uv \in E} c(uv) \sum_{S \subset V} z_S d_S(uv)}{\sum_{i=1}^{k} D_i \sum_{S \subset V} d_S(s_i, t_i)} \\ &= \frac{1}{\alpha} \frac{\sum_{S \subset V} z_S c(\delta(S))}{\sum_{S \subset V} z_S \mathrm{Dem}(\delta(S))} \\ &\geq \frac{1}{\alpha} \min_{S \subset V} \frac{c(\delta(S))}{\mathrm{Dem}(\delta(S))}. \end{aligned}$$

Thus there is a cut whose sparsity is at most $\alpha \cdot \lambda$. ∎

Polynomial-time algorithm: How do we fine a sparse cut? The preceding proof used the embedding into a metric in the cut-cone. The proof shows that one of the cuts with $z_S > 0$ has sparsity at most $\alpha \cdot \lambda$. Recall the proof that a metric is in the cut-cone iff it is ℓ_1-embeddable. That argument shows the following. Suppose we have an ℓ_1 embedding into d-dimensions. Each dimension corresponds to a line-embedding. Each line embedding is in the cut-cone with only $n-1$ cuts used to express it. Thus, given an ℓ_1 embedding into d dimensions with distortion α we only need to try $d(n-1)$ cuts and one of them will be guaranteed to have sparsity at most $\alpha \cdot \lambda$.

Via Theorem 17.5 we can obain an $O(\log k)$ randomized approximation and the algorithm is described below.

```
SparseCutviaEmbedding

1. Solve LP relaxation to obtain $(x, y)$ and metric $d_x$ on $V$

2. Use Theorem 17.5 obtain map $f : V \to \mathbb{R}^d$

3. For $i =$ to $d$ do

   A. Let $v_{j_1}, v_{j_2}, \ldots, v_{j_n}$ be the sorting of $V$ according to dimension $i$
   B. For $h = 1$ to $n - 1$ let $S_{i,h} = \{v_{j_1}, v_{j_2}, \ldots, v_{j_h}\}$

4. Output among all cuts $S_{i,h}$ with $1 \le i \le d$ and $1 \le h \le n - 1$ output the one
   with the smallest sparsity.
```

Exercise 17.5. Use the refined guarantee in and the proof outline in Theorem 17.9 to show that the described algorithm is a randomized $O(\log k)$-approximation algorithm for Sparsest Cut.

17.3 SDP and Spectral Relaxations

To be filled.

Bibliographic Notes

The highly influential paper of Leighton and Rao [113] obtained an $O(\log n)$-approximation and flow-cut gap for Uniform Sparsest Cut and introduced the region growing argument as well as the lower bound via expanders (an important influence is the paper of Sharokhi and Matula [**SharokhiM99**]). [113] demonstrated many applications of the divide and conquer approach. There is a large literature on Sparsest Cut and related problems and we only touched upon a small part. An outstanding open problem is whether the flow-cut gap for Non-Uniform Sparsest Cut in planar graphs is $O(1)$ (this called the GNRS conjecture [73] in the more general context of minor-free graphs); Rao, building on ideas from [104], showed that the gap is $O(\sqrt{\log n})$ [134]. No super-constant lower bound is known for planar graphs. The theory of metric embeddings has been a fruitful bridge between TCS and mathematics and there are several surveys and connections from both perspectives.

Bibliography

[1] Ittai Abraham, Cyril Gavoille, Anupam Gupta, Ofer Neiman, and Kunal Talwar. "Cops, robbers, and threatening skeletons: Padded decomposition for minor-free graphs". In: *SIAM Journal on Computing* 48.3 (2019), pp. 1120–1145.

[2] Anna Adamaszek, Parinya Chalermsook, Alina Ene, and Andreas Wiese. "Submodular unsplittable flow on trees". In: *International Conference on Integer Programming and Combinatorial Optimization*. Springer. 2016, pp. 337–349.

[3] Amit Agarwal, Noga Alon, and Moses S Charikar. "Improved approximation for directed cut problems". In: *Proceedings of the thirty-ninth annual ACM symposium on Theory of computing*. 2007, pp. 671–680.

[4] Ankit Aggarwal, Amit Deshpande, and Ravi Kannan. "Adaptive sampling for k-means clustering". In: *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques*. Springer, 2009, pp. 15–28.

[5] Ajit Agrawal, Philip Klein, and Ramamoorthi Ravi. "When trees collide: An approximation algorithm for the generalized Steiner problem on networks". In: *SIAM journal on Computing* 24.3 (1995), pp. 440–456.

[6] Nir Ailon, Ragesh Jaiswal, and Claire Monteleoni. "Streaming k-means approximation." In: *NIPS*. Vol. 22. 2009, pp. 10–18.

[7] Karhan Akcoglu, James Aspnes, Bhaskar DasGupta, and Ming-Yang Kao. "Opportunity cost algorithms for combinatorial auctions". In: *Computational Methods in Decision-Making, Economics and Finance*. Springer, 2002, pp. 455–479.

[8] Matthew Andrews, Julia Chuzhoy, Venkatesan Guruswami, Sanjeev Khanna, Kunal Talwar, and Lisa Zhang. "Inapproximability of edge-disjoint paths and low congestion routing on undirected graphs". In: *Combinatorica* 30.5 (2010), pp. 485–520.

[9] Kenneth Appel, Wolfgang Haken, et al. "Every planar map is four colorable. Part I: Discharging". In: *Illinois Journal of Mathematics* 21.3 (1977), pp. 429–490.

[10] Sanjeev Arora. "Polynomial time approximation schemes for Euclidean traveling salesman and other geometric problems". In: *Journal of the ACM (JACM)* 45.5 (1998), pp. 753–782.

[11] Vijay Arya, Naveen Garg, Rohit Khandekar, Adam Meyerson, Kamesh Munagala, and Vinayaka Pandit. "Local search heuristics for k-median and facility location problems". In: *SIAM Journal on computing* 33.3 (2004), pp. 544–562.

[12] Arash Asadpour, Michel X Goemans, Aleksander Mądry, Shayan Oveis Gharan, and Amin Saberi. "An O (log n/log log n)-approximation algorithm for the asymmetric traveling salesman problem". In: *Operations Research* 65.4 (2017). Preliminary version in Proc. of ACM-SIAM SODA, 2010., pp. 1043–1061.

[13] Pranjal Awasthi, Moses Charikar, Ravishankar Krishnaswamy, and Ali Kemal Sinop. "The Hardness of Approximation of Euclidean k-Means". In: *31st International Symposium on Computational Geometry (SoCG 2015)*. Ed. by Lars Arge and János Pach. Vol. 34. Leibniz International Proceedings in Informatics (LIPIcs). Dagstuhl, Germany: Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 2015, pp. 754–767. ISBN: 978-3-939897-83-5. DOI: 10.4230/LIPIcs.SOCG.2015.754. URL: http://drops.dagstuhl.de/opus/volltexte/2015/5117.

[14] Baruch Awerbuch, Yossi Azar, and Yair Bartal. "On-line generalized Steiner problem". In: *Theoretical Computer Science* 324.2-3 (2004), pp. 313–324.

[15] Bahman Bahmani, Benjamin Moseley, Andrea Vattani, Ravi Kumar, and Sergei Vassilvitskii. "Scalable K-Means++". In: *Proc. VLDB Endow.* 5.7 (Mar. 2012), pp. 622–633. ISSN: 2150-8097. DOI: 10.14778/2180912.2180915. URL: https://doi.org/10.14778/2180912.2180915.

[16] Tanvi Bajpai, Deeparnab Chakrabarty, Chandra Chekuri, and Maryam Negahbani. "Revisiting Priority k-Center: Fairness and Outliers". In: *arXiv preprint arXiv:2103.03337* (2021).

[17] Brenda S Baker. "Approximation algorithms for NP-complete problems on planar graphs". In: *Journal of the ACM (JACM)* 41.1 (1994), pp. 153–180.

[18] Nikhil Bansal, Nitish Korula, Viswanath Nagarajan, and Aravind Srinivasan. "Solving packing integer programs via randomized rounding with alterations". In: *Theory of Computing* 8.1 (2012), pp. 533–565.

[19] Reuven Bar-Yehuda and Shimon Even. "A linear-time approximation algorithm for the weighted vertex cover problem". In: *Journal of Algorithms* 2.2 (1981), pp. 198–203.

[20] Yair Bartal, Moses Charikar, and Danny Raz. "Approximating Min-Sum <i>k</i>-Clustering in Metric Spaces". In: *Proceedings of the Thirty-Third Annual ACM Symposium on Theory of Computing*. STOC '01. Hersonissos, Greece: Association for Computing Machinery, 2001, pp. 11–20. ISBN: 1581133499. DOI: 10.1145/380752.380754. URL: https://doi.org/10.1145/380752.380754.

[21] Kristóf Bérczi, Karthekeyan Chandrasekaran, Tamás Király, and Vivek Madan. "Improving the integrality gap for multiway cut". In: *Mathematical Programming* 183.1 (2020), pp. 171–193.

[22] Marshall Bern and Paul Plassmann. "The Steiner problem with edge lengths 1 and 2". In: *Information Processing Letters* 32.4 (1989), pp. 171–176.

[23] Glencora Borradaile, Philip Klein, and Claire Mathieu. "An $O(n \log n)$ approximation scheme for Steiner tree in planar graphs". In: *ACM Transactions on Algorithms (TALG)* 5.3 (2009), pp. 1–31.

[24] Simon Bruggmann and Rico Zenklusen. "Submodular maximization through the lens of linear programming". In: *Mathematics of Operations Research* 44.4 (2019), pp. 1221–1244.

[25] Niv Buchbinder and Moran Feldman. "Submodular functions maximization problems". In: *Handbook of Approximation Algorithms and Metaheuristics, Second Edition*. Chapman and Hall/CRC, 2018, pp. 753–788.

[26] Jaroslaw Byrka, Fabrizio Grandoni, Thomas Rothvoß, and Laura Sanita. "An improved LP-based approximation for Steiner tree". In: *Proceedings of the forty-second ACM symposium on Theory of computing*. 2010, pp. 583–592.

[27] G. Calinescu, H. Karloff, and Y. Rabani. "Approximation algorithms for the 0-extension problem". In: *Proceedings of the twelfth annual ACM-SIAM symposium on Discrete algorithms*. Society for Industrial and Applied Mathematics Philadelphia, PA, USA. 2001, pp. 8–16.

[28] Gruia Calinescu, Chandra Chekuri, Martin Pal, and Jan Vondrák. "Maximizing a monotone submodular function subject to a matroid constraint". In: *SIAM Journal on Computing* 40.6 (2011), pp. 1740–1766.

[29] Robert Carr and Santosh Vempala. "Randomized metarounding". In: *Proceedings of the thirty-second annual ACM symposium on Theory of computing*. 2000, pp. 58–62.

[30] Amit Chakrabarti, Chandra Chekuri, Anupam Gupta, and Amit Kumar. "Approximation algorithms for the unsplittable flow problem". In: *Algorithmica* 47.1 (2007), pp. 53–78.

[31] Deeparnab Chakrabarty and Maryam Negahbani. "Generalized center problems with outliers". In: *ACM Transactions on Algorithms (TALG)* 15.3 (2019), pp. 1–14.

[32] Timothy M Chan. "Approximation schemes for 0-1 knapsack". In: *1st Symposium on Simplicity in Algorithms (SOSA 2018)*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik. 2018.

[33] Ashok K Chandra, Daniel S. Hirschberg, and Chak-Kuen Wong. "Approximate algorithms for some generalized knapsack problems". In: *Theoretical Computer Science* 3.3 (1976), pp. 293–304.

[34] Moses Charikar, Chandra Chekuri, To-Yat Cheung, Zuo Dai, Ashish Goel, Sudipto Guha, and Ming Li. "Approximation algorithms for directed Steiner problems". In: *Journal of Algorithms* 33.1 (1999), pp. 73–91.

[35] Moses Charikar, Sudipto Guha, Éva Tardos, and David B Shmoys. "A constant-factor approximation algorithm for the k-median problem". In: *Journal of Computer and System Sciences* 65.1 (2002), pp. 129–149.

[36] Shuchi Chawla, Robert Krauthgamer, Ravi Kumar, Yuval Rabani, and D Sivakumar. "On the hardness of approximating multicut and sparsest-cut". In: *computational complexity* 15.2 (2006), pp. 94–114.

[37] Chandra Chekuri, Sreeram Kannan, Adnan Raja, and Pramod Viswanath. "Multicommodity flows and cuts in polymatroidal networks". In: *SIAM Journal on Computing* 44.4 (2015), pp. 912–943.

[38] Chandra Chekuri and Sanjeev Khanna. "A polynomial time approximation scheme for the multiple knapsack problem". In: *SIAM Journal on Computing* 35.3 (2005), pp. 713–728.

[39] Chandra Chekuri and Vivek Madan. "Approximating multicut and the demand graph". In: *Proceedings of the Twenty-Eighth Annual ACM-SIAM Symposium on Discrete Algorithms*. SIAM. 2017, pp. 855–874.

[40] Chandra Chekuri and Kent Quanrud. "On approximating (sparse) covering integer programs". In: *Proceedings of the Thirtieth Annual ACM-SIAM Symposium on Discrete Algorithms*. SIAM. 2019, pp. 1596–1615.

[41] Chandra Chekuri and Thapanapong Rukkanchanunt. "A note on iterated rounding for the Survivable Network Design Problem". In: *1st Symposium on Simplicity in Algorithms (SOSA 2018)*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik. 2018.

[42] Miroslav Chlebık and Janka Chlebıková. "Approximation hardness of the Steiner tree problem on graphs". In: *Scandinavian Workshop on Algorithm Theory*. Springer. 2002, pp. 170–179.

[43] Nicos Christofides. *Worst-case analysis of a new heuristic for the travelling salesman problem*. Tech. rep. Carnegie-Mellon Univ Pittsburgh Pa Management Sciences Research Group, 1976.

[44] Julia Chuzhoy, Venkatesan Guruswami, Sanjeev Khanna, and Kunal Talwar. "Hardness of Routing with Congestion in Directed Graphs". In: *Proceedings of the Thirty-Ninth Annual ACM Symposium on Theory of Computing*. STOC '07. San Diego, California, USA: Association for Computing Machinery, 2007, pp. 165–178. ISBN: 9781595936318. DOI: `10.1145/1250790.1250816`. URL: `https://doi.org/10.1145/1250790.1250816`.

[45] Julia Chuzhoy and Sanjeev Khanna. "Polynomial flow-cut gaps and hardness of directed cut problems". In: *Journal of the ACM (JACM)* 56.2 (2009), pp. 1–28.

[46] Elias Dahlhaus, David S. Johnson, Christos H. Papadimitriou, Paul D. Seymour, and Mihalis Yannakakis. "The complexity of multiterminal cuts". In: *SIAM Journal on Computing* 23.4 (1994), pp. 864–894.

[47] George Dantzig, Ray Fulkerson, and Selmer Johnson. "Solution of a large-scale traveling-salesman problem". In: *Journal of the operations research society of America* 2.4 (1954), pp. 393–410.

[48] W Fernandez De La Vega and George S. Lueker. "Bin packing can be solved within 1+ ε in linear time". In: *Combinatorica* 1.4 (1981), pp. 349–355.

[49] Yefim Dinitz, Naveen Garg, and Michel X Goemans. "On the single-source unsplittable flow problem". In: *Combinatorica* 19.1 (1999), pp. 17–41.

[50] Irit Dinur and Samuel Safra. "On the hardness of approximating minimum vertex cover". In: *Annals of mathematics* (2005), pp. 439–485.

[51] D-Z Du and Frank K. Hwang. "A proof of the Gilbert-Pollak conjecture on the Steiner ratio". In: *Algorithmica* 7.1 (1992), pp. 121–135.

[52] Jack Edmonds. "Optimum branchings". In: *Journal of Research of the National Bureau of Standards, B* 71 (1967), pp. 233–240.

[53] Guy Even, Joseph Seffi Naor, Satish Rao, and Baruch Schieber. "Divide-and-conquer approximation algorithms via spreading metrics". In: *Journal of the ACM (JACM)* 47.4 (2000), pp. 585–616.

[54] Jittat Fakcharoenphol, Satish Rao, and Kunal Talwar. "A tight bound on approximating arbitrary metrics by tree metrics". In: *Journal of Computer and System Sciences* 69.3 (2004), pp. 485–497.

[55] Tomás Feder and Daniel Greene. "Optimal algorithms for approximate clustering". In: *Proceedings of the twentieth annual ACM symposium on Theory of computing*. 1988, pp. 434–444.

[56] Uriel Feige. "A threshold of ln n for approximating set cover". In: *Journal of the ACM (JACM)* 45.4 (1998), pp. 634–652.

[57] Uriel Feige, Vahab S Mirrokni, and Jan Vondrak. "Maximizing non-monotone submodular functions". In: *SIAM Journal on Computing* 40.4 (2011), pp. 1133–1153.

[58] Uriel Feige and Jan Vondrak. "Approximation algorithms for allocation problems: Improving the factor of 1-1/e". In: *2006 47th Annual IEEE Symposium on Foundations of Computer Science (FOCS'06)*. IEEE. 2006, pp. 667–676.

[59] Moran Feldman, Joseph Seffi Naor, Roy Schwartz, and Justin Ward. "Improved approximations for k-exchange systems". In: *European Symposium on Algorithms*. Springer. 2011, pp. 784–798.

[60] Moran Feldman, Joseph Seffi Naor, Roy Schwartz, and Justin Ward. "Improved approximations for k-exchange systems". In: *European Symposium on Algorithms*. Springer. 2011, pp. 784–798.

[61] Marshall L Fisher, George L Nemhauser, and Laurence A Wolsey. "An analysis of approximations for maximizing submodular set functions II". In: *Polyhedral combinatorics* (1978), pp. 73–87.

[62] Michael R Garey and David S Johnson. *Computers and intractability*. Vol. 174. Freeman, San Francisco, 1979.

[63] N. Garg, V.V. Vazirani, and M. Yannakakis. "Approximate max-flow min-(multi) cut theorems and their applications". In: *Proceedings of the twenty-fifth annual ACM symposium on Theory of computing*. ACM New York, NY, USA. 1993, pp. 698–707.

[64] Shayan Oveis Gharan, Amin Saberi, and Mohit Singh. "A randomized rounding approach to the traveling salesman problem". In: *2011 IEEE 52nd Annual Symposium on Foundations of Computer Science*. IEEE. 2011, pp. 550–559.

[65] Michel X Goemans, Neil Olver, Thomas Rothvoß, and Rico Zenklusen. "Matroids and integrality gaps for hypergraphic steiner tree relaxations". In: *Proceedings of the forty-fourth annual ACM symposium on Theory of computing*. 2012, pp. 1161–1176.

[66] Michel X Goemans and David P Williamson. “The primal-dual method for approximation algorithms and its application to network design problems”. In: *Approximation algorithms for NP-hard problems* (1997), pp. 144–191.

[67] Michel X Goemans and David P Williamson. “The primal-dual method for approximation algorithms and its application to network design problems”. In: *Approximation algorithms for NP-hard problems* (1997), pp. 144–191.

[68] Ronald L Graham. “Bounds for certain multiprocessing anomalies”. In: *Bell system technical journal* 45.9 (1966), pp. 1563–1581.

[69] Ronald L. Graham. “Bounds on multiprocessing timing anomalies”. In: *SIAM journal on Applied Mathematics* 17.2 (1969), pp. 416–429.

[70] Fabrizio Grandoni, Tobias Mömke, Andreas Wiese, and Hang Zhou. “A (5/3+ ε)-approximation for unsplittable flow on a path: placing small tasks into boxes”. In: *Proceedings of the 50th Annual ACM SIGACT Symposium on Theory of Computing*. 2018, pp. 607–619.

[71] Sudipto Guha and Samir Khuller. “Greedy strikes back: Improved facility location algorithms”. In: *Journal of algorithms* 31.1 (1999), pp. 228–248.

[72] Anupam Gupta and Jochen Könemann. “Approximation algorithms for network design: A survey”. In: *Surveys in Operations Research and Management Science* 16.1 (2011), pp. 3–20.

[73] Anupam Gupta, Ilan Newman, Yuri Rabinovich, and Alistair Sinclair. “Cuts, trees and 1-embeddings of graphs”. In: *Combinatorica* 24.2 (2004), pp. 233–269.

[74] Anupam Gupta and Kanat Tangwongsan. *Simpler Analyses of Local Search Algorithms for Facility Location*. 2008. arXiv: `0809.2554 [cs.DS]`.

[75] Bernhard Haeupler, Barna Saha, and Aravind Srinivasan. “New constructive aspects of the Lovász local lemma”. In: *Journal of the ACM (JACM)* 58.6 (2011), pp. 1–28.

[76] Eran Halperin and Robert Krauthgamer. “Polylogarithmic inapproximability”. In: *Proceedings of the thirty-fifth annual ACM symposium on Theory of computing*. 2003, pp. 585–594.

[77] Sariel Har-Peled and Manor Mendel. “Fast construction of nets in low-dimensional metrics and their applications”. In: *SIAM Journal on Computing* 35.5 (2006), pp. 1148–1184.

[78] Sariel Har-Peled and Benjamin Raichel. “Net and prune: A linear time algorithm for euclidean distance problems”. In: *Journal of the ACM (JACM)* 62.6 (2015), pp. 1–35.

[79] Sariel HarPeled. *Concentration of Random Variables —- Chernoff's Inequality*. Avaialble at `https://sarielhp.org/teach/13/b_574_rand_alg/lec/07_chernoff.pdf`.

[80] Johan Hastad. "Clique is hard to approximate within n/sup 1-/spl epsiv". In: *Proceedings of 37th Conference on Foundations of Computer Science*. IEEE. 1996, pp. 627–636.

[81] Michael Held and Richard M Karp. "The traveling-salesman problem and minimum spanning trees". In: *Operations Research* 18.6 (1970), pp. 1138–1162.

[82] Rebecca Hoberg and Thomas Rothvoss. "A logarithmic additive integrality gap for bin packing". In: *Proceedings of the Twenty-Eighth Annual ACM-SIAM Symposium on Discrete Algorithms*. SIAM. 2017, pp. 2616–2625.

[83] Dorit S Hochbaum and David B Shmoys. "A polynomial approximation scheme for scheduling on uniform processors: Using the dual approximation approach". In: *SIAM journal on computing* 17.3 (1988), pp. 539–551.

[84] Dorit S Hochbaum and David B Shmoys. "A unified approach to approximation algorithms for bottleneck problems". In: *Journal of the ACM (JACM)* 33.3 (1986), pp. 533–550.

[85] Ian Holyer. "The NP-completeness of edge-coloring". In: *SIAM Journal on computing* 10.4 (1981), pp. 718–720.

[86] Shlomo Hoory, Nathan Linial, and Avi Wigderson. "Expander graphs and their applications". In: *Bulletin of the American Mathematical Society* 43.4 (2006), pp. 439–561.

[87] Ellis Horowitz and Sartaj Sahni. "Exact and approximate algorithms for scheduling nonidentical processors". In: *Journal of the ACM (JACM)* 23.2 (1976), pp. 317–327.

[88] Wen-Lian Hsu and George L Nemhauser. "Easy and hard bottleneck location problems". In: *Discrete Applied Mathematics* 1.3 (1979), pp. 209–215.

[89] Makoto Imase and Bernard M Waxman. "Dynamic Steiner tree problem". In: *SIAM Journal on Discrete Mathematics* 4.3 (1991), pp. 369–384.

[90] Kamal Jain. "A factor 2 approximation algorithm for the generalized Steiner network problem". In: *Combinatorica* 21.1 (2001), pp. 39–60.

[91] Kamal Jain and Vijay V Vazirani. "Approximation algorithms for metric facility location and k-median problems using the primal-dual schema and Lagrangian relaxation". In: *Journal of the ACM (JACM)* 48.2 (2001), pp. 274–296.

[92] Ragesh Jaiswal, Amit Kumar, and Sandeep Sen. "A simple D 2-sampling based PTAS for k-means and other clustering problems". In: *Algorithmica* 70.1 (2014), pp. 22–46.

[93] Klaus Jansen. "An EPTAS for scheduling jobs on uniform processors: using an MILP relaxation with a constant number of integral variables". In: *SIAM Journal on Discrete Mathematics* 24.2 (2010), pp. 457–485.

[94] Klaus Jansen. "Parameterized approximation scheme for the multiple knapsack problem". In: *SIAM Journal on Computing* 39.4 (2010), pp. 1392–1412.

[95] Klaus Jansen and Lars Rohwedder. "A quasi-polynomial approximation for the restricted assignment problem". In: *International Conference on Integer Programming and Combinatorial Optimization*. Springer. 2017, pp. 305–316.

[96] David S Johnson. "Approximation algorithms for combinatorial problems". In: *Journal of computer and system sciences* 9.3 (1974), pp. 256–278.

[97] EG Co man Jr, MR Garey, and DS Johnson. "Approximation algorithms for bin packing: A survey". In: *Approximation algorithms for NP-hard problems* (1996), pp. 46–93.

[98] George Karakostas. "A better approximation ratio for the vertex cover problem". In: *ACM Transactions on Algorithms (TALG)* 5.4 (2009), p. 41.

[99] Anna R Karlin, Nathan Klein, and Shayan Oveis Gharan. "A (slightly) improved approximation algorithm for metric TSP". In: *Proceedings of the 53rd Annual ACM SIGACT Symposium on Theory of Computing*. 2021, pp. 32–45.

[100] Narendra Karmarkar and Richard M Karp. "An efficient approximation scheme for the one-dimensional bin-packing problem". In: *23rd Annual Symposium on Foundations of Computer Science (sfcs 1982)*. IEEE. 1982, pp. 312–320.

[101] Ken-ichi Kawarabayashi and Anastasios Sidiropoulos. "Embeddings of Planar Quasimetrics into Directed ℓ_1 and Polylogarithmic Approximation for Directed Sparsest-Cut". In: *Proceedigns of IEEE FOCS (2021)*. To appear. 2021.

[102] H. Kellerer, H.K.U.P.D. Pisinger, U. Pferschy, and D. Pisinger. *Knapsack Problems*. Springer Nature Book Archives Millennium. Springer, 2004. ISBN: 9783540402862. URL: `https://books.google.com/books?id=u5DB7gck08YC`.

[103] Subhash Khot and Oded Regev. "Vertex cover might be hard to approximate to within 2- ε". In: *Journal of Computer and System Sciences* 74.3 (2008), pp. 335–349.

[104] Philip Klein, Serge A Plotkin, and Satish Rao. “Excluded minors, network decomposition, and multicommodity flow”. In: *Proceedings of the twenty-fifth annual ACM symposium on Theory of computing*. 1993, pp. 682–690.

[105] Philip N Klein, Serge A Plotkin, Satish Rao, and Eva Tardos. “Approximation algorithms for Steiner and directed multicuts”. In: *Journal of Algorithms* 22.2 (1997), pp. 241–269.

[106] Stavros G Kolliopoulos and Neal E Young. “Approximation algorithms for covering/packing integer programs”. In: *Journal of Computer and System Sciences* 71.4 (2005), pp. 495–505.

[107] Guy Kortsarz and Zeev Nutov. “Approximating minimum cost connectivity problems”. In: *Parameterized complexity and approximation algorithms*. Ed. by Erik D. Demaine, MohammadTaghi Hajiaghayi, and Dániel Marx. Dagstuhl Seminar Proceedings 09511. Dagstuhl, Germany: Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, Germany, 2010. URL: `http://drops.dagstuhl.de/opus/volltexte/2010/2497`.

[108] Madhukar R Korupolu, C Greg Plaxton, and Rajmohan Rajaraman. “Analysis of a local search heuristic for facility location problems”. In: *Journal of algorithms* 37.1 (2000), pp. 146–188.

[109] Ravishankar Krishnaswamy, Amit Kumar, Viswanath Nagarajan, Yogish Sabharwal, and Barna Saha. “The matroid median problem”. In: *Proceedings of the twenty-second annual ACM-SIAM symposium on Discrete Algorithms*. SIAM. 2011, pp. 1117–1130.

[110] Lap Chi Lau, Ramamoorthi Ravi, and Mohit Singh. *Iterative methods in combinatorial optimization*. Vol. 46. Cambridge University Press, 2011.

[111] Eugene L Lawler. “Fast approximation algorithms for knapsack problems”. In: *Mathematics of Operations Research* 4.4 (1979), pp. 339–356.

[112] Jon Lee, Maxim Sviridenko, and Jan Vondrák. “Matroid matching: the power of local search”. In: *SIAM Journal on Computing* 42.1 (2013), pp. 357–379.

[113] T. Leighton and S. Rao. “Multicommodity max-flow min-cut theorems and their use in designing approximation algorithms”. In: *Journal of the ACM (JACM)* 46.6 (1999). Conference version is from 1988., pp. 787–832.

[114] Tom Leighton, Satish Rao, and Aravind Srinivasan. “Multicommodity flow and circuit switching”. In: *Proceedings of the Thirty-First Hawaii International Conference on System Sciences*. Vol. 7. IEEE. 1998, pp. 459–465.

[115] Jan Karel Lenstra, David B Shmoys, and Éva Tardos. “Approximation algorithms for scheduling unrelated parallel machines”. In: *Mathematical programming* 46.1 (1990), pp. 259–271.

[116] Shi Li. “A 1.488 approximation algorithm for the uncapacitated facility location problem”. In: *Information and Computation* 222 (2013), pp. 45–58.

[117] Nathan Linial, Eran London, and Yuri Rabinovich. “The geometry of graphs and some of its algorithmic applications”. In: *Combinatorica* 15.2 (1995), pp. 215–245.

[118] Meena Mahajan, Prajakta Nimbhorkar, and Kasturi Varadarajan. “The planar k-means problem is NP-hard”. In: *Theoretical Computer Science* 442 (2012), pp. 13–21.

[119] G.A. Margulis. “Explicit constructions of expanders”. In: *Problemy Peredaci Informacii* 9.4 (1973), pp. 71–80.

[120] Julián Mestre. “Greedy in approximation algorithms”. In: *European Symposium on Algorithms*. Springer. 2006, pp. 528–539.

[121] Michael Mitzenmacher and Eli Upfal. *Probability and computing: Randomization and probabilistic techniques in algorithms and data analysis*. Cambridge university press, 2017.

[122] Sarah Morell and Martin Skutella. “Single source unsplittable flows with arc-wise lower and upper bounds”. In: *Mathematical Programming* (2021), pp. 1–20.

[123] Robin A Moser and Gábor Tardos. “A constructive proof of the general Lovász local lemma”. In: *Journal of the ACM (JACM)* 57.2 (2010), pp. 1–15.

[124] Dana Moshkovitz. “The Projection Games Conjecture and the NP-Hardness of $\ln n$-Approximating Set-Cover”. In: *Theory of Computing* 11.1 (2015), pp. 221–235.

[125] Rajeev Motwani and Prabhakar Raghavan. *Randomized algorithms*. Cambridge university press, 1995.

[126] Viswanath Nagarajan, R Ravi, and Mohit Singh. “Simpler analysis of LP extreme points for traveling salesman and survivable network design problems”. In: *Operations Research Letters* 38.3 (2010), pp. 156–160.

[127] Viswanath Nagarajan, Baruch Schieber, and Hadas Shachnai. “The Euclidean k-supplier problem”. In: *Mathematics of Operations Research* 45.1 (2020), pp. 1–14.

[128] George L Nemhauser, Laurence A Wolsey, and Marshall L Fisher. “An analysis of approximations for maximizing submodular set functions—I”. In: *Mathematical Programming* 14.1 (1978), pp. 265–294.

[129] Zeev Nutov. “Node-connectivity survivable network problems”. In: *Handbook of Approximation Algorithms and Metaheuristics*. Chapman and Hall/CRC, 2018, pp. 233–253.

[130] Rafail Ostrovsky, Yuval Rabani, Leonard J Schulman, and Chaitanya Swamy. “The effectiveness of Lloyd-type methods for the k-means problem”. In: *Journal of the ACM (JACM)* 59.6 (2013), pp. 1–22.

[131] Ján Plesnık. “A heuristic for the p-center problems in graphs”. In: *Discrete Applied Mathematics* 17.3 (1987), pp. 263–268.

[132] Prabhakar Raghavan. “Probabilistic construction of deterministic algorithms: approximating packing integer programs”. In: *Journal of Computer and System Sciences* 37.2 (1988), pp. 130–143.

[133] Prabhakar Raghavan and Clark D Tompson. “Randomized rounding: a technique for provably good algorithms and algorithmic proofs”. In: *Combinatorica* 7.4 (1987), pp. 365–374.

[134] Satish Rao. “Small distortion and volume preserving embeddings for planar and Euclidean metrics”. In: *Proceedings of the fifteenth annual symposium on Computational geometry*. 1999, pp. 300–306.

[135] Gabriel Robins and Alexander Zelikovsky. “Tighter bounds for graph Steiner tree approximation”. In: *SIAM Journal on Discrete Mathematics* 19.1 (2005), pp. 122–134.

[136] Sartaj Sahni and Teofilo Gonzalez. “P-complete approximation problems”. In: *Journal of the ACM (JACM)* 23.3 (1976), pp. 555–565.

[137] Alexander Schrijver. *Combinatorial optimization: polyhedra and efficiency*. Vol. 24. Springer Science & Business Media, 2003.

[138] Paul D. Seymour. “Packing directed circuits fractionally”. In: *Combinatorica* 15.2 (1995), pp. 281–288.

[139] Ankit Sharma and Jan Vondrák. “Multiway cut, pairwise realizable distributions, and descending thresholds”. In: *Proceedings of the forty-sixth annual ACM symposium on theory of computing*. 2014, pp. 724–733.

[140] David B Shmoys and Éva Tardos. “An approximation algorithm for the generalized assignment problem”. In: *Mathematical programming* 62.1 (1993), pp. 461–474.

[141] Martin Skutella. “A note on the ring loading problem”. In: *SIAM Journal on Discrete Mathematics* 30.1 (2016), pp. 327–342.

[142] Aravind Srinivasan. “An extension of the Lovász Local Lemma, and its applications to integer programming”. In: *SIAM Journal on Computing* 36.3 (2006), pp. 609–634.

[143] Larry Stockmeyer. “Planar 3-colorability is Polynomial Complete”. In: *SIGACT News* 5.3 (July 1973), pp. 19–25. ISSN: 0163-5700. DOI: 10.1145/1008293.1008294. URL: http://doi.acm.org/10.1145/1008293.1008294.

[144] Ola Svensson. “Approximating ATSP by relaxing connectivity”. In: *2015 IEEE 56th Annual Symposium on Foundations of Computer Science*. IEEE. 2015, pp. 1–19.

[145] Ola Svensson. “Santa claus schedules jobs on unrelated machines”. In: *SIAM Journal on Computing* 41.5 (2012), pp. 1318–1341.

[146] Ola Svensson, Jakub Tarnawski, and László A Végh. “A constant-factor approximation algorithm for the asymmetric traveling salesman problem”. In: *Journal of the ACM (JACM)* 67.6 (2020), pp. 1–53.

[147] Chaitanya Swamy. “Improved approximation algorithms for matroid and knapsack median problems and applications”. In: *ACM Transactions on Algorithms (TALG)* 12.4 (2016), pp. 1–22.

[148] Hiromitsu Takahashi and A Matsuyama. “An approximate solution for the Steiner problem in graphs”. In: *Math. Jap.* 24.6 (1980), pp. 573–577.

[149] Robin Thomas. “An update on the four-color theorem”. In: *Notices of the AMS* 45.7 (1998), pp. 848–859.

[150] Vera Traub and Jens Vygen. “An improved approximation algorithm for ATSP”. In: *Proceedings of the 52nd Annual ACM SIGACT Symposium on Theory of Computing*. 2020, pp. 1–13.

[151] Sergei Vassilvitskii and David Arthur. “k-means++: The advantages of careful seeding”. In: *Proceedings of the eighteenth annual ACM-SIAM symposium on Discrete algorithms*. 2006, pp. 1027–1035.

[152] Vijay V Vazirani. *Approximation algorithms*. Springer Science & Business Media, 2013.

[153] Robert Vicari. “Simplex based Steiner tree instances yield large integrality gaps for the bidirected cut relaxation”. In: *arXiv preprint arXiv:2002.07912* (2020).

[154] Douglas Brent West et al. *Introduction to graph theory*. Vol. 2. Prentice hall Upper Saddle River, 2001.

[155] David P Williamson and David B Shmoys. *The design of approximation algorithms*. Cambridge university press, 2011.

[156] David P. Williamson. “On the design of approximation algorithms for a class of graphs problems”. PhD thesis. Cambridge, MA: MIT, 1993.

[157] David P. Williamson, Michel X. Goemans, Milena Mihail, and Vijay V. Vazirani. “A Primal-Dual Approximation Algorithm for Generalized Steiner Network Problems”. In: 15 (1995), pp. 435–454. DOI: `http://dx.doi.org/10.1007/BF01299747`.

[158] Laurence A Wolsey. “An analysis of the greedy algorithm for the submodular set covering problem”. In: *Combinatorica* 2.4 (1982), pp. 385–393.

[159] Yuli Ye and Allan Borodin. “Elimination graphs”. In: *ACM Transactions on Algorithms (TALG)* 8.2 (2012), pp. 1–23.

[160] Alexander Z Zelikovsky. “An 11/6-approximation algorithm for the network Steiner problem”. In: *Algorithmica* 9.5 (1993), pp. 463–470.

Appendix A

Basic Feasible Solutions to LPs and the Rank Lemma

We discuss the rank lemma about vertex solutions for linear programs. Recall that a *polyhedron* in $\mathbb{R}^n$ is defined as the intersection of finite collection of half spaces. Without loss of generality we can assume that it is defined by a system of inequalities of the form $Ax \leq b$ where A is a $m \times n$ matrix and b is a $m \times 1$ vector. A polyhedron P is bounded if P is contained in finite radius ball around the origin. A *polytope* in $\mathbb{R}^n$ is defined as the convex hull of a finite collection of points. A fundamental theorem about linear programming states that any bounded polyhedron is a polytope. If the polyhedron is not bounded then it can be expressed as the Minkowski sum of a polytope and a cone.

A bounded polyhedron P in $\mathbb{R}^n$ defined by a system $Ax \leq b$ must necessarily have $m \geq n$. A point $p \in P$ is a basic feasible solution or a vertex solution of the system if it is the unique solution to a system $A'y = b'$ where A' is a sub-matrix of A with n inequalities and the rank of A' is equal to n. The inequalities in A' are said to be *tight* for y. Note that there may be many other inequalities in $Ax \leq b$ that are tight at y and in general there many be many different rank n sub-matrices that give rise to the same basic feasible solution y.

Lemma A.1. *Suppose y is a basic feasible solution of a system $Ax \leq b, \ell \leq x \leq u$ where A is a $m \times n$ matrix and ℓ and u are vectors defining lower and upper bounds on the variables $x \in \mathbb{R}^n$. Let $S = \{i : \ell_i < y_i < u_i\}$ be the set of indices of "fractional" variables in y. Then $|S| \leq rank(A) \leq m$. In particular the number of fractional variables in y is at most the number of "non-trivial" constraints (those that are defined by A).*

An extension of the previous lemma is often useful when the system defining the polyhedron has equality constraints.

Corollary A.1. *Suppose y is a basic feasible solution of a system $Ax \leq b, Cx = d, \ell \leq x \leq u$ where A is a $m \times n$ matrix, C is a $m' \times n$ matrix, and ℓ and u are vectors defining lower and upper bounds on the variables $x \in \mathbb{R}^n$. Let $S = \{i : \ell_i < y_i < u_i\}$ be the set of indices of "fractional" variables in y. Then $|S| \leq rank(A, C) \leq m + m'$.*

A special case of the preceding corollary is called the rank lemma in [110].

The lemmas are a simple consequence of the definition of basic feasible solution. We will focus on the proof of Lemma A.1. It is interesting only when $rank(A)$ or m is smaller than n, otherwise the claim is trivial. Before we prove it formally we observe some simple corollaries. Suppose we have a system $Ax \leq b, x \geq 0$ where $m < n$. Then the number of non-zero variables in a basic feasible solution is at most m. Similarly if the system is $Ax \leq b, x \in [0, 1]^n$ then the number of non-integer variables in y is at most m. For example in the knapsack LP we have $m = 1$ and hence in any basic feasible solution there can only be one fractional variable.

Now for the proof. We consider the system $Ax \leq b, -x \leq -\ell, x \leq u$ as a single system $Cx \leq d$ which has $m + 2n$ inequalities. Since y is a basic feasible solution to this system, from the definition, it is the unique solution of sub-system $C'x = d'$ where C' is a $n \times n$ full-rank matrix. How many rows of C' can come from A? At most $rank(A) \leq m$ rows. It means that the rest of the rows of C' are of the from the other set of inequalities $-x \leq \ell$ or $x \leq u$. There are at least $n - rank(A)$ such rows which are tight at y. Thus $n - rank(A)$ variables in y are tight at lower or upper bounds and hence there can only be $rank(A)$ fractional variables in y.

See [110] for iterated rounding based methodology for exact and approximation algorithms. The whole methodology relies on properties of basic feasible solutions to LP relaxations of combinatorial optimization problems.

A.0.1 Some Examples

We give some examples to illustrate the utility of the rank lemma in the context of LP relaxations that arise in approximation algorithms.

Knapsack: The natural LP relaxation for this of the form max $\sum_{i=1}^{n} w_i x_i$ subject to $x \in [0, 1]^n, \sum_{i=1}^{n} s_i x_i \leq 1$ consisting of a single non-trivial constraint. A basic feasible solution has at most 1 fractional variable. See Chapter 3.

Packing Integer Programs (PIPs): The LP relaxation is of the form max wx subject to $Ax \leq b, x \in [0, 1]^n$ where A is a $m \times n$ non-negative matrix. See Chapter 4. When $m = 1$ we have the Knapsack problem and hence the general problem is some-times referred to as the m-dimensional Knapsack problem, especially when m is a fixed constant. A basic feasible solution for the LP has at most m fractional variables. When m is a fixed constant one can exploit this after guessing the big items to obtain a PTAS.

Generalized Assignment: See Chapter 6.

A.0.2 Connection to Caratheodary's Theorem

Suppose we have n points $P = \{p_1, p_2, \ldots, p_n\}$ in d-dimensional Euclidean space $\mathbb{R}^d$. A point $p \in \mathbb{R}^d$ is in the convex hull of P iff p is a convex combination of points in P. Formally, this means that exist scalars $\lambda_1, \ldots, \lambda_n \geq 0$ such that $\sum_i \lambda_i = 1$ and $p = \sum_i \lambda_i p_i$ (note that this is a vector sum). Caratheordary's theorem states that if p is in the convex hull of P then there is subset $P' \subseteq P$ such that p is in the convex hull of P' and $|P'| \leq d + 1$. One can prove Caratheodary's theorem directly but it is helpful to see it also as a consequence of the rank lemma. Consider the system of inequalities $\lambda_i \geq 0, 1 \leq i \leq n, \sum_i \lambda_i = 1, \sum_i \lambda_i p_i = p$ where the system $\sum_i \lambda_i p_i = p$ consists of d equalities. This system of inequalities in the variables $\lambda_1, \ldots, \lambda_n$ is feasible by assumption (since p is in the convex hull of P). If we take any basic feasible solution of this system of inequalities we see that at most $d + 1$ of them are non-zero by the rank lemma.

One implication of Caratheordary's theorem in the context of combinatorial optimization is the following. Suppose we have a polytope P which is an LP relaxation of some combinatorial problem and let $x \in P$ be any feasible point. Then x can be written as a convex combination of at most $n+1$ vertices of P where n is number of variables. Moreover, via the Ellipsoid method, one can find such a convex combination efficiently as long as one can optimize over P efficiently. As an example suppose $G = (V, E)$ is a graph and P is the spanning tree polytope of G (the vertices are the characteristic vectors of spanning trees) which is $\mathbb{R}^m$ ($m = |E|$). Then any fractional spanning tree $x \in P$ can be decomposed into at most $m + 1$ spanning trees.

In the context of approximation P is typically a relaxation of some hard combinatorial optimization problem. In such a case the vertices of P do not correspond to structures we are interested in. For example we can consider the minimum Steiner tree problem in a graph $G = (V, E)$ with terminal set $S \subseteq V$. There are several LP relaxations but perhaps the simplest one is the cut relaxation which has an integrality gap of 2. In such a case a feasible point $x \in P$ cannot be decomposed into convex combination of Steiner trees. However it can be shown that $2x$ dominates a convex combination of Steiner trees and such a convex combination can be found efficiently. It requires more technical work to precisely formalize this and we refer the reader to the work of Carr and Vempala [29] — you can also find a few applications of such decompositions in the same paper and it is a simple yet powerful tool to keep in mind.

Appendix B

Probabilistic Inequalities

The course will rely heavily on proababilistic methods. We will mostly rely on discrete probability spaces. We will keep the discussion high-level where possible and use certain results in a black-box fashion.

Let Ω be a finite set. A probability measure p assings a non-negative number $p(\omega)$ for each $\omega \in \Omega$ such that $\sum_{\omega \in \Omega} p(\omega) = 1$. The tuple (Ω, p) defines a discrete probability space; an event in this space is any subset $A \subseteq \Omega$ and the probability of an event is simply $p(A) = \sum_{\omega \in A} p(\omega)$. When Ω is a continuous space such as the interval $[0, 1]$ things get trickier and we need to talk about a measure spaces σ-algebras over Ω; we can only assign probability to certain subsets of Ω. We will not go into details since we will not need any formal machinery for what we do in this course.

An important definition is that of a *random variable*. We will focus only on real-valued random variables in this course. A random variable X in a probability space is a function $X : \Omega \to \mathbb{R}$. In the discrete setting the *expectation* of X, denoted by $\mathbf{E}[X]$, is defined as $\sum_{\omega \in \Omega} p(w)X(\omega)$. For continuous spaces $\mathbf{E}[X] = \int X(\omega)dp(\omega)$ with appropriate definition of the integral. The variance of X, denoted by $\mathrm{Var}[X]$ or as σ_X^2, is defined as $\mathbf{E}\left[(X - \mathbf{E}[X])^2\right]$. The standard deviation is σ_X, the square root of the variance.

Theorem B.1 (Markov's Inequality). *Let X be a non-negative random variable such that $\mathbf{E}_X$ is finite. Then for any $t > 0$, $\mathbf{P}[X \geq t] \leq \mathbf{E}[X]/t$.*

Proof. The proof is in some sense obvious, especially in the discrete case. Here is a sketch. Define a new random variable Y where $Y(\omega) = X(\omega)$ if $X(\omega) < t$ and $Y(\omega) = t$ if $X(\omega) \geq t$. Y is non-negative and $Y \leq X$ point-wise and hence

$\mathbf{E}[Y] \leq \mathbf{E}[X]$. We also see that:

$$\begin{aligned}\mathbf{E}[X] \geq \mathbf{E}[Y] &= \sum_{\omega : X(\omega) < t} X(\omega)p(\omega) + \sum_{\omega : X(\omega) \geq t} tp(\omega) \\ &\geq t \sum_{\omega : X(\omega) \geq t} p(\omega) \qquad \text{(since } X \text{ is non-negative)} \\ &\geq t\,\mathbf{P}[X \geq t].\end{aligned}$$

The continuous case follows by replacing sums by integrals. ■

Markov's inequality is tight under the assumption. It is useful to construct an example. The more information we have about a random variable the better we can bound its deviation from the expectation.

Theorem B.2 (Chebyshev's Inequality). *Let X be a random variable with $\mathbf{E}[X]$ and $\mathrm{Var}[X]$ are finite. Then $\mathbf{P}[|X| \geq t] \leq \mathbf{E}\left[X^2\right]/t^2$ and $\mathbf{P}[|X - \mathbf{E}[X]| \geq t\sigma_X] \leq 1/t^2$.*

Proof. Consider the non-negative random variable $Y = X^2$. $\mathbf{P}[|X| \geq t] = \mathbf{P}[Y \geq t^2]$ and we apply Markov's inequality to the latter. The second inequality is similar by considering $Y = (X - \mathbf{E}[X])^2$. ■

Chernoff-Hoeffding Bounds: We will use several times various forms of the Chernoff-Hoeffding bounds that apply to a random variable that is a a finite sum of bounded and *independent* random variables. There are several versions of these bounds. First we state a general bound that is applicable to non-negative random variables and is dimension-free in that it depends only the expectation rather than the number of variables.

Theorem B.3 (Chernoff-Hoeffding). *Let $X_1, X_2, \ldots, X_n$ be independent binary random variables and let $a_1, a_2, \ldots, a_n$ be coefficients in $[0, 1]$. Let $X = \sum_i a_i X_i$. Then*

- *For any $\mu \geq \mathbf{E}[X]$ and any $\delta > 0$, $\mathbf{P}[X > (1 + \delta)\mu] \leq \left(\frac{e^\delta}{(1+\delta)^{(1+\delta)}}\right)^\mu$.*
- *For any $\mu \leq \mathbf{E}[X]$ and any $\delta > 0$, $\mathbf{P}[X < (1 - \delta)\mu] \leq e^{-\mu\delta^2/2}$.*

The following corollary bounds the deviation from the mean in both directions.

Corollary B.4. *Under the conditions of Theorem B.3, the following hold:*

- *If $\delta > 2e - 1$, $\mathbf{P}[X \geq (1 + \delta)\mu] \leq 2^{-(1+\delta)\mu}$.*
- *For any U there is a constant $c(U)$ such that for $0 < \delta < U$, $\mathbf{P}[X \geq (1 + \delta)\mu] \leq e^{-c(U)\delta^2\mu}$. In particular, combining with the lower tail bound,*

$$\mathbf{P}[|X - \mu| \geq \delta\mu] \leq 2e^{-c(U)t^2\mu}.$$

We refer the reader to the standard books on randomized algorithms [125] and [121] for the derivation of the above bounds.

If we are interested only in the upper tail we also have the following bounds which show the dependence of μ on n to obtain an inverse polynomial probability.

Corollary B.5. *Under the conditions of Theorem B.3, there is a universal constant α such that for any $\mu \geq \max\{1, \mathbf{E}[X]\}$, and sufficiently large n and for $c \geq 1$, $\mathbf{P}[X > \frac{\alpha c \ln n}{\ln \ln n} \cdot \mu] \leq 1/n^c$. Similarly, there is a constant α such that for any $\epsilon > 0$, $\mathbf{P}[X \geq (1+\epsilon)\mu + \alpha c \log n/\epsilon] \leq 1/n^c$.*

Remark B.1. If the X_i are in the range $[0, b]$ for some b not equal to 1 one can scale them appropriately and then use the standard bounds.

Some times we need to deal with random variables that are in the range $[-1, 1]$. Consider the setting where $X = \sum_i X_i$ where for each i, $X_i \in [-1, 1]$ and $\mathbf{E}_{X_i} = 0$, and the X_i are independent. In this case $\mathbf{E}[X] = 0$ and we can no longer expect a dimension-free bound. Suppose each X_i is 1 with probability 1/2 and −1 with probability 1/2. Then $X = \sum_i X_i$ corresponds to a 1-dimensional random walk and even though the expected value is 0 the standard deviation of X is $\Theta(\sqrt{n})$. One can show that $\mathbf{P}[|X| \geq t\sqrt{n}] \leq 2e^{-t^2/2}$. For these settings we can use the following bounds.

Theorem B.6. *Let $X_1, X_2, \ldots, X_n$ be independent random variables such that for each i, $X_i \in [a_i, b_i]$. Let $X = \sum_i a_i X_i$ and let $\mu = \mathbf{E}[X]$. Then*

$$\mathbf{P}[|X - \mu| \geq t] \leq 2e^{-\frac{2t^2}{\sum_{i=1}^n (b_i - a_i)^2}}.$$

In particular if $b_i - a_i \leq 1$ for all i then

$$\mathbf{P}[|X - \mu| \geq t] \leq 2e^{-\frac{2t^2}{n}}.$$

Note that $\text{Var}[X] = \sum_i \text{Var}[X_i]$. One can show a bound based of the following form

$$\mathbf{P}[|X - \mu| \geq t] \leq 2e^{-\frac{t^2}{2(\sigma_X^2 + Mt/3)}}$$

where $|X_i| \leq M$ for all i.

Remark B.2. Compare the Chebyshev bound to the Chernoff-Hoeffding bounds for the same variance.

Sariel Har-Peled maintains a cheat sheet of Chernoff bounds and also has an interesting derivation. See his notes [79].

Statistical Estimators, Reducing Variance and Boosting: Randomized algorithms compute a function f of the input. In many cases they producing an unbiased estimator, via a random variable X, for the the function value. That is, the algorithm will have the property that the $\mathbf{E}[X]$ is the desired value. Note that the randomness is internal to the algorithm and not part of the input (we can also consider randomness in the input). Having an estimator is not often useful. We will also typically try to evaluate $\text{Var}[X]$ and then we can use Chebyshev's inequality. One way to reduce the variance of the estimate is to run the algorithm in parallel (with separate random bits) and get estimators $X_1, X_2, \ldots, X_h$ and use $X = \frac{1}{h}\sum_i X_i$ as the final estimator. Note that $\text{Var}[X]) = \frac{1}{h}\sum_i \text{Var}[X_i]$ since the X_i are independent. Thus the variance has been reduced by a factor of h. A different approach is to use the *median* value of $X_1, X_2, \ldots, X_h$ as the final estimator. We can then use Chernoff-Hoeffding bounds to get a much better dependence on h. In fact both approaches can be combined.