IEOR 290G

Professor Dorit S. Hochbaum

Contents

1 Overview of Complexity

1.1 Definitions

Algorithm Complexity: the number of operations ($+, -, \times, /, \geq,$ sometimes $\lfloor\ \rfloor$) required to run the algorithm in the worst case, up to a constant multiple, as a function of properties of the inputs to the algorithm.

Problem Complexity: the complexity of the best possible algorithm to solve the given problem.

The notion of complexity is used to provide a basis of comparison for algorithms. In particular, complexity provides a measure (at least mostly) agnostic to computer speed, which varies with time and technological process. Also, the complexity of an algorithm is able to take into consideration the worst-case performance of an algorithm, whereas empirical results would not necessarily capture this important aspect.

Naturally, determining the complexity of many problems is difficult; proving an upper bound on the complexity of a problem is simply a matter of providing a single algorithm that solves the problem, while proving a lower bound is a matter of establishing that *all* algorithms that solve the problem cannot achieve a certain complexity. For some problems, this analysis is approachable. This is the case for sorting, where we are given a list of n items and are asked to sort them in ascending order. An information theoretical proof shows that the complexity of this problem is $\Omega(n \log n)$. Many sorting algorithms have complexity $O(nlogn)$, so they are asymptotically best possible algorithms.

A couple differing philosophies exist regarding the list of operations to count. Arithmetic models measure complexity by counting $(+, -, *, /, \geq)$ operations. Other operations may also be included, such as $\lfloor\ \rfloor$ [1] and $\sqrt{\ }$; however, it turns out that including $\lfloor\ \rfloor$ has no significant influence on complexity of most algorithms, and $\sqrt{\ }$, though fundamental, is of limited value in complexity analysis.

When we talk about complexity, we usually assume WLOG that inputs are integer-valued and bounded. Numbers are represented in computers with finite accuracy, and we can always transform rationals to integers by multiplying a large enough integer. When we say an algorithm is polynomial, we are also saying that the size of output is polynomial in the size of of input[2].

1.2 Examples

Weaving Problem We seek to find the number of "under" operations necessary to create a weave, like the weave found on a pie. Here, an "under" operation would comprise of lifting a strand of the dough to move a perpendicular strand of dough under it: essentially, creating one weave. For an $n \times m$ weave, a naive algorithm takes $O(\lfloor \frac{nm}{2} \rfloor)$ "under" operations. However, the concrete complexity is attained by an algorithm that runs in $O(\lfloor \frac{nm}{4} \rfloor)$ [**?**].

---

[1] Ceiling function can be calculated with floor function: $\lceil a \rceil = \begin{cases} a & \text{if } a = \lfloor a \rfloor \\ \lfloor a+1 \rfloor & \text{otherwise} \end{cases}$.

[2] It is not obvious whether during the execution of the algorithm the data remain polynomial length, Edmonds proved some result for Gaussian elimination.

Linear Programming

$$\begin{aligned} \max \quad & c^T x \\ s.t. \quad & Ax = b \\ & x \geq 0, \end{aligned}$$

where $A \in \mathbb{Z}^{m \times n}$ has rank m. The basic solutions x_B are giving by some full rank $m \times m$ sub-matrix B of A:

$$x_B = B^{-1} b,$$

whose length are proved to be polynomial in the size of input.

LP can be viewed as a discrete optimization problem, where we optimize over the set of basic feasible solutions, of which there exist at most $\binom{n}{m}$. the Simplex algorithm, in turn, can be viewed as a clever way of searching over the set of basic feasible solutions.

Two known classes of polynomial algorithms that solve LP's are ellipsoid methods and interior point methods. Both make use of continuous techniques: they know a priori that *the optimal solution to LP can be written in a # of digits that is polynomial in* $\log \Delta$*, where* Δ *is the largest subdeterminant of* A[3], and so ensuring a fixed rate of convergence towards the optimal solution per iteration will lead to termination in weakly-polynomial time. In practice however, it may take a large number of iterations for either method to reach the terminating accuracy. However, interior-point methods are used by solvers such as CPLEX as a pre-processor to choose a good initial basis from which to initiate the Simplex algorithm.

2 Nonlinear Optimization

In Nonlinear Optimization problems, both the feasible region and the objective function can be messy, and the optimal solution is no longer guaranteed to an extreme point of the feasible region. If both the objective function and the feasible region are convex, then the problem is called *convex*; this class of problems is much easier and is well studied.

A non-linear function can be represented via an oracle that can return the value of the function at some given input, perhaps through a table look-up:

$$\bar{x} \rightarrow \text{oracle} \rightarrow f(\bar{x}).$$

Whenever we give the oracle a $\bar{x}$ of ϵ-accuracy[4] to our intended input, the oracle tells us the value of $f(\bar{x})$ ϵ-accurate to the intended output.

Bisection algorithms approach one-dimensional non-linear optimization in a way similar to binary search and ellipsoid method. The function is assumed to be unimodal, that is, there exists an $x^* \in S$ (the feasible region) at which f attains a minimum (maximum) and f is nondecreasing (nonincreasing) on the interval $\{x \in S : x \geq x^*\}$ and nonincreasing (nondecreasing) on the interval $\{x \in S : x \leq x^*\}$. The algorithm operates by evaluating the two endpoints and the midpoint of the feasible region; the values are compared, and half of the feasible region is discarded at each iteration.

[3] the number of digits is also polynomial in $\log |b_{max}|$

[4] an ϵ-accuracy number is given with $\leq \lceil \log_2 \frac{1}{\epsilon} \rceil$ bits

2.1 The Allocation Problem

We first consider simple resource allocation problem (SRA):

$$\text{(SRA)} \qquad \begin{array}{rlcll} \max & \sum_{j=1}^n f_j(x_j) & & & \\ \text{s.t.} & \sum_{j=1}^n x_j & = & B & \\ & x_j & \geq & 0 & j = 1, \ldots, n \\ & x_j & \in & \mathbb{Z} & j = 1, \ldots, n, \end{array}$$

where the f_j's are concave functions, making the objective function concave separable. We can replace "=" in the first constraint by "$\leq$" without much changing the complexity of and solution to the problem. Also, in the special case where the f_j's are all linear, that is, $f_j(x) = w_j x,\ j = 1, \ldots, n$, the SRA problem reduces to an equally-weighted instance of the knapsack problem.

SRA can be solved optimally with a greedy algorithm in weakly polynomial time. Essentially, beginning at $\mathbf{x} = 0$, we increment the variables by one unit at a time, in each iteration choosing the variable that provides the largest immediate gain, until the single constraint has been satisfied. So, for given general concave functions $f_j, j = 1, \ldots, n$, we must first calculate the increments

$$\Delta_j^i := f_j(i+1) - f_j(i),\ i = 1, \ldots, B.$$

At each iteration, set $x_{j^*} \leftarrow x_{j^*} + 1$, where $j^* = \arg\max_{j \in \{1,\ldots,n\}} \Delta_j^{x_j}$. Every iteration we pick the largest of n increments, which can be done in $O(\log_2 n)$ time through the method of binary heaps, and we repeat this for B iterations. Thus, the complexity of this algorithm is $O(B \log_2 n)$, which is not strongly polynomial.

It turns out this greedy algorithm extends to quite a large class of more general resource allocation problems.

2.2 Extensions of SRA

Allocation with upper bounds

$$\begin{array}{rlcll} \max & \sum_{j=1}^n f_j(x_j) & & & \\ \text{s.t.} & \sum_{j=1}^n x_j & = & B & \\ & x_j & \leq & u_j & j = 1, \ldots, n \\ & x_j & \geq & 0 & j = 1, \ldots, n, \text{ integer} \end{array}$$

The above greedy algorithm applies when we only increase the components that have not reached their upper bounds.

Allocation with generalized upper bounds Suppose the indexes $\{1, \ldots, n\}$ are partitioned into k disjoint sets $S_1 \cup S_2 \cdots \cup S_k$. Consider

$$\begin{array}{rlcll} \max & \sum_{j=1}^n f_j(x_j) & & & \\ \text{s.t.} & \sum_{j=1}^n x_j & = & B & \\ & \sum_{j \in S_p} x_j & \leq & u_p & p = 1, \ldots, k \\ & x_j & \geq & 0 & j = 1, \ldots, n, \text{ integer} \end{array}$$

Again, the above algorithm solves this problem optimally.

Allocation with nested upper bounds Suppose we have some nested sets of indexes $\{1,\ldots,n\} = S_0 \supset S_1 \supset \cdots \supset S_k$ (note strict containment). Consider

$$\begin{array}{rlcll} \max & \sum_{j=1}^n f_j(x_j) & & & \\ \text{s.t.} & \sum_{j=1}^n x_j & = & B & \\ & \sum_{j\in S_p} x_j & \leq & u_p & p = 0,\ldots,k \\ & x_j & \geq & 0 & j = 1,\ldots,n, \text{ integer} \end{array}$$

Same deal..

General allocation problem (GAP) This is the most general case, which encompasses all previous examples and a more complex nested structure. The formulation is:

$$\text{(GAP)} \qquad \begin{array}{rlcll} \max & \sum_{j=1}^n f_j(x_j) & & & \\ \text{s.t.} & \sum_{j=1}^n x_j & = & B & \\ \text{s.t.} & \sum_{j\in A} x_j & \leq & r(A) & A \subset E \\ & x_j & \geq & 0 & j = 1,\ldots,n, \text{ integer.} \end{array}$$

We call the constraints $\{\sum_{j\in A} x_j \leq r(A), A \subseteq E\}$ *polymatroidal constraints.*, if $r : 2^E \to R$, for $E = \{1,\ldots,n\}$, is a *submodular* rank function, that is, $r(\phi) = 0$ and for all $A, B \subset E$,

$$r(A) + r(B) \geq r(A \cup B) + r(A \cap B).$$

The *polymatroid* defined by the rank function r, is the polytope $\{\mathbf{x} | \sum_{j\in A} x_j \leq r(A), A \subseteq E\}$.

2.2.1 A Lower Bound with the Comparison Model [Hoc94]

First let's look at the 1-d version of the problem, given a concave function f, consider

$$\begin{array}{rlcl} \max & f(x) & & \\ \text{s.t.} & x & \leq & B \\ & x & \geq & 0 \\ & x & \in & \mathbb{Z} \end{array}$$

Finding the optimal x amounts to finding where the sequence $\Delta^1, \Delta^2, \ldots, \Delta^B$ start to become negative. Binary search solves the problem with $O(\lceil \log_2 B \rceil)$ operations[5] .

We can give an information theoretic lower bound for solving this one-dimensional problem. Consider a decision tree with every evaluation of the increments splitting the range of potential solutions offered at that node into two parts. Any algorithm can be viewed as a walk from the root to one of the $B+1$ leaves, where leaf $i, i = 0, 1, ..., B$ represents the outcome at which $f(i)$ attains its maximum. The decision tree hence has depth at least $\lceil \log_2(B+1) \rceil$; the complexity of any algorithm, in turn, must be at least $\Omega(\log_2 B)$. A similar argument applies to sorting, where there are $n!$ leaves in the decision tree for a problem with n input numbers, and hence the depth is no less than $O(\log_2(n!)) = O(n \log_2 n)$.

[5]the operations can be (1) comparisons or (2) arithmetic calculations to find the mid point of the search region or (3) oracle calls to the function.

Now let's extend this argument to a reformulation of the more general model. Specifically, let's call this the Generalized Allocation Comparison Problem:

$$\text{(GACP)} \qquad \begin{array}{lrcll} \max & \sum_{j=1}^{n} f_j(x_j) + cx_{n+1} & & & \\ \text{s.t.} & \sum_{j=1}^{n+1} x_j & = & B & \\ & x_j & \geq & 0 & j = 1, \ldots, n \\ & x_j & \in & \mathbb{Z} & j = 1, \ldots, n. \end{array}$$

Solving the GACP is essentially equivalent to finding, for each $f_j, j = 1, \ldots, n$, the last increment greater than or equal to c in $[0, \frac{B}{n}]$ (otherwise we would opt to increase x_{n+1} instead). Hence $\Omega(n \log_2 \frac{B}{n})$ is the lower bound on the complexity of the algorithm.

2.2.2 The Algebraic-Tree Model Lower Bound [Hoc94]

In the arithmetic model of computation, we rely on Renegar's lower bound proof ([R87]) for finding ε-accurate roots of polynomials of fixed degree ≥ 2. In particular, Renegar shows that the complexity of identifying an ε-accurate real root of such a polynomial in an interval $[O, R]$ is of complexity $\Omega(\log \log \frac{R}{\varepsilon})$, even if the polynomial is monotone in that interval. Now, let $p_1, \ldots, p_n$ be n monotone non-increasing (over $[0, R]$) polynomials that each take the value c exactly once in the interval $[0, \frac{B}{n}]$. Since the choice of these polynomials is arbitrary, the lower bound on finding the n roots of these n polynomials must be $\Omega(n \log \log \frac{B}{n\varepsilon})$, according to Renegar. However if we set $f_j(x_j) = \int_0^{x_j} p_j(x)dx$, then the f_js are polynomials of degree ≥ 3. Thus, the problem,

$$\text{(GACP}_\varepsilon\text{)} \qquad \begin{array}{lrcll} \max & \sum_j f_j(x_j \cdot \varepsilon) + c \cdot x_{n+1} \cdot \varepsilon & & & \\ & \sum_{j=1}^{n+1} x_j & = & \frac{B}{\varepsilon} & \\ & x_j & \geq & 0 & j = 1, \ldots, n+1 \\ & x_j & \in & \mathbb{Z} & j = 1, \ldots, n+1, \end{array}$$

has an optimal solution $\mathbf{x}$ such that $\mathbf{y} = \varepsilon \cdot \mathbf{x}$, which is also an $(n\epsilon)$-accurate vector of roots solving the system,

$$\begin{array}{rcl} p_1(y_1) & = & c \\ p_2(y_2) & = & c \\ & \vdots & \\ p_n(y_n) & = & c. \end{array}$$

Given our design of the functions f_j, the fact that the solution to GACP$_\varepsilon$ is also a approximation of this set of polynomials follows from the KKT conditions of GACP$_\varepsilon$, as well as the proximity theorem to be derived in the following section.

2.2.3 The Floor Operation

[MST91] proved a lower bound on the complexity of finding ϵ-accurate square roots that assumes the floor, $\lfloor\ \rfloor$, operation. In our notation this lower bound is $\Omega(\sqrt{\log \log \frac{B}{\epsilon}})$. Hence, even with this

additional operation the problem cannot be solved in strongly polynomial time.

Floor operation can be important, as in the following example:

$$\begin{aligned} \min \quad & \frac{1}{2}x_1^2 + \frac{1}{2}(a-1)x_2^2 \\ s.t \quad & x_1 + x_2 = b \\ & x_1, x_2 \geq 0 \; integers, \end{aligned}$$

where the parameter $a > 0$. You can compute analytically that the optimal $x_2^* = \lfloor \frac{B}{a} \rfloor$.

2.3 Proximity-Scaling Algorithm for the General Allocation Problem

Consider the general allocation problem:

$$\text{(GAP)} \qquad \begin{array}{lrcll} \max & \sum_{j=1}^n f_j(x_j) & & & \\ \text{s.t.} & \sum_{j=1}^n x_j & = & B & \\ & \sum_{j\in A} x_j & \leq & r(A) & A \subset E \\ & x_j & \geq & \ell_j & j = 1, \ldots, n, \text{ integer.} \end{array}$$

For this section, we use $\mathbf{e}^i$ to denote the ith standard basic vector for $i = 1, \ldots, n$ and $\mathbf{e} = \sum_{i=1}^n \mathbf{e}^i$.

2.3.1 The Greedy Algorithm

It is well know that GAP can be solved optimally by a greedy algorithm:

Algorithm 1 Greedy

0. $\mathbf{x} \leftarrow \boldsymbol{\ell}$, $B \leftarrow B - \sum_j \ell_j$, $E \leftarrow \{1, \ldots, n\}$.
1. Find i such that $\Delta_i(x_i) = \max_{j\in E} \Delta_j(x_j)$ (where $\Delta_j(x_j) = f_j(x_j + 1) - f_j(x_j)$).
2. If $\mathbf{x} + \mathbf{e}^i$ is feasible then $x_i \leftarrow x_i + 1$ and $B \leftarrow B - 1$;
   Else, $E \leftarrow E \setminus \{i\}$.
3. If $B = 0$, output $\mathbf{x}$ and stop.
   Else if $E = \emptyset$, output "no feasible solution" and stop.
   Else, go to step 1.

Complexity: Algorithm 1 will clearly conduct B iterations, each of which requires finding the minimum $\Delta_j(x_j)$ and testing the feasibility of a given solution vector. Using the structure of binary heaps, this will require $O(\log n)$ operations to update the binary heap of $\Delta_j(x_j)$ for $j = 1, \ldots, n$, and $O(F)$ running time for the feasibility check. Thus, the total complexity of this greedy algorithm is $O\left(B(\log n + F)\right)$, which is pseudo-polynomial due to the input B.

2.3.2 The Greedy(s) Algorithm

Consider improving the running time by scaling the largest increment of x_i's to be s units (s integer). This intuition gives us the Greedy(s) algorithm. Note that Greedy(s) does not solve GAP optimally, but it provides a feasible output $\mathbf{x}^{(s)}$, which is later shown to be related to an optimal solution of GAP via the proximity theorem.

Algorithm 2 Greedy(s)

0. $\mathbf{x} \leftarrow \boldsymbol{\ell}$, $B \leftarrow B - \sum_j \ell_j$, $\boldsymbol{\delta} \leftarrow \mathbf{0}$, $E \leftarrow \{1, \ldots, n\}$.
1. Find i such that $\Delta_i(x_i) = \max_{j \in E} \Delta_j(x_j)$ (where $\Delta_j(x_j) = f_j(x_j + 1) - f_j(x_j)$).
2. If $\mathbf{x} + s \cdot \mathbf{e}^i$ is feasible then $x_i \leftarrow x_i + s$, $B \leftarrow B - s$, $\delta_i \leftarrow s$;
   Else, find the largest integer δ such that $\mathbf{x} + \delta \cdot \mathbf{e}^i$ is feasible, and let $x_i \leftarrow x_i + \delta_i$, $B \leftarrow B - \delta_i$, $\delta_i \leftarrow \delta$, $E \leftarrow E \setminus \{i\}$.
3. If $B = 0$, output $\mathbf{x}^s$, $\boldsymbol{\delta}$, and stop.
   Else if $E = \emptyset$, output "no feasible solution" and stop.
   Else, go to step 1.

Complexity: The running time of each iteration in Greedy(s) is the similar to Greedy except for F. In Greedy(s), F is the complexity of determining δ_i, the tightest slack to infeasibility. However, the number of iterations in Greedy(s) is reduced to $O(\frac{B}{s})$. Therefore, the complexity of Greedy(s) is $O\left(\frac{B}{s}(\log n + F)\right)$.

For the output of Greedy(s), $\mathbf{x}^s$, we have the *proximity theorem*:

Theorem 1 (Hoc94)**.** *If there is a feasible solution to GAP then there exists an optimal solution* $\mathbf{x}^*$ *such that* $\mathbf{x}^* > \mathbf{x}^s - \boldsymbol{\delta} \geq \mathbf{x}^s - s \cdot \mathbf{e}$

Based on this proximity theorem we can derive the Proximity-scaling Algorithm solving GAP.

2.3.3 The Proximity-Scaling Algorithm for GAP

Algorithm 3 Proximity-scaling algorithm for GAP

0. $s \leftarrow \lceil B/2n \rceil$.
1. If $s = 1$ call *Greedy*. Output $\mathbf{x}^*$, and stop. Else, continue.
2. Call *Greedy(s)*. Let the output be $\mathbf{x}^s$ and $\boldsymbol{\delta}$.
   $\boldsymbol{\ell} \leftarrow \mathbf{x}^s - \boldsymbol{\delta}$, $s \leftarrow \lceil s/2 \rceil$.
   Go to step 1.

Complexity: There are $\lceil \log_2 \frac{B}{2n} \rceil$ iterations, each calls Greedy(s) once with $s = \Omega(\frac{B}{n})$. So the complexity of the proximity-scaling algorithm is $O\left(\log(\frac{B}{n}) n (\log n + F)\right)$.

This proximity-scaling algorithm leads to the fastest algorithms known for all special cases of the general allocation problem ([Hoc94]). The complexity expressions of the algorithm for the different cases are:

1. For the simple resource allocation problem SRA, $O(n \log \frac{B}{n})$.
2. For the generalized upper bounds resource allocation problem (GUB), $O(n \log \frac{B}{n})$.
3. For the *nested* problem, $O(n \log n \log \frac{B}{n})$.
4. For the *tree constrained* problem, $O(n \log n \log \frac{B}{n})$.

The complexity bounds for SRA and GUB are also shown to be best possible in the comparison model.

2.4 Proximity-scaling for separable convex (concave) optimization over linear constraints

Now, we can generalize away from the single constraint of GAP. The formulation of separable convex minimization programming over general linear constraints is:

$$\begin{array}{lll} & \min & \sum_{j=1}^{n} f_j(x_j) \\ (\text{RP}) & \text{s.t.} & A\mathbf{x} \geq \mathbf{b} \\ & & \ell_j \leq x_j \leq u_j \quad j = 1, \ldots, n. \end{array}$$

Here, $A \in \mathbb{R}^{m \times n}$ and the f_j's are convex functions, making the objective convex separable.

We use $T(n, m, \Delta)$ to denote the running time for solving a linear programming problem with constraints $A\mathbf{x} \geq \mathbf{b}$ such that (n, m, Δ) are the column size, row size and the largest sub-determinant of A.

2.4.1 Convex Piecewise Linear Minimization Over Linear Constraints

Suppose we further assume that the f_j's are convex piecewise linear functions. Let's introduce the following notation for $j = 1, \ldots, n$:

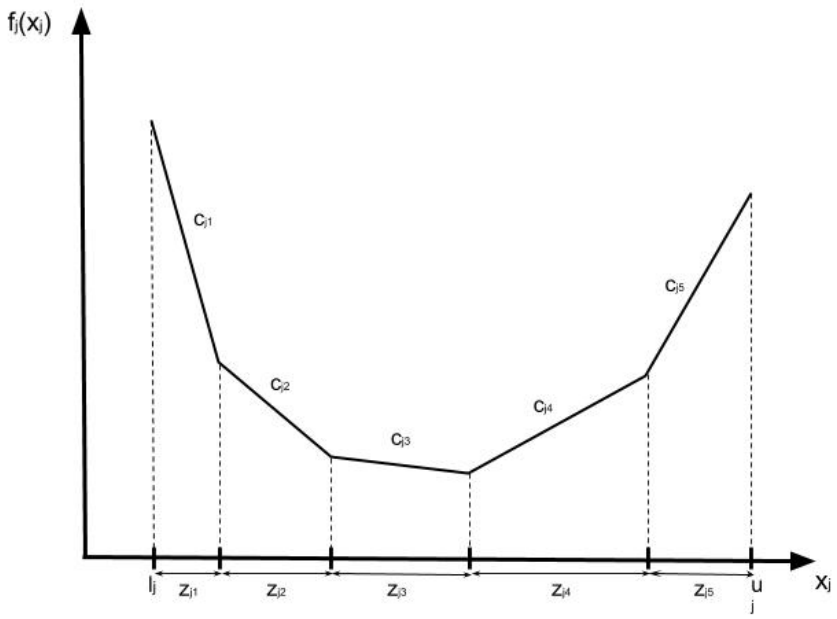
N_j: the number of affine segments that make up f_j in the interval $[\ell_j, u_j]$;

c_{jp}: the slope of the p^{th} segment of f_j;

δ_{jp}: the length of interval for the p^{th} segment of f_j.

Now we can express the variable x_j as $x_j = \sum_{p=0}^{N_j} z_{jp}$, where $z_{j0} = \ell_j$ and $0 \leq z_{jp} \leq \delta_{jp}$, for $j = 1, \ldots, N_j$. Let vector $\mathbf{z} = [z_{10}, z_{11}, \ldots, z_{1N_j}, z_{20}, \ldots, z_{n0}, \ldots, z_{nN_n}]^T$. And let A_j denote the j^{th}

Figure 1: An example of a piecewise linear function



column of matrix A. Then $A\mathbf{x} = \tilde{A}\mathbf{z}$ where $\tilde{A}$ have N_j multiples of column A_j for $j = 1, \ldots, n$. That is, $\tilde{A} = [A_1, A_1, \ldots, A_1, A_2, \ldots, A_2, \ldots, A_n, \ldots, A_n]$. By changing variables $\mathbf{x}$ to $\mathbf{z}$, we arrive at the following linear programming problem:

$$
\text{(LP)} \qquad \begin{array}{lrcll} \min & \sum_{j=1}^{n} \sum_{p=1}^{N_j} c_{jp} z_{jp} & & & \\ \text{s.t.} & \tilde{A}\mathbf{z} & \geq & \mathbf{b} & \\ & 0 \leq z_{jp} & \leq & \delta_{jp} & j = 1, \ldots, n, \quad p = 1, \ldots, N_j \\ & z_{j0} & = & \ell_j & j = 1, \ldots, n. \end{array}
$$

Since the columns of $\tilde{A}$ are merely duplicates of the columns of A, we may make a few observations:

1. The largest sub-determinant of $\tilde{A}$ is the same as that of A.
2. Any sub-matrix with non-zero determinant cannot contain duplicate columns. Therefore, in any basic feasible solution, at most one z_{jp} is not on the boundary for each j. Moreover, as the f_j's are convex, for each j the objective coefficients c_{jp} are increasing in $p = 1, \ldots, N_j$. Hence, the optimal solution for (LP) must have the form $(z_{j0}, \ldots, z_{jN_j}) = (\ell_j, \delta_{j1}, \ldots, \delta_{jp-1}, z_{jp}, 0, \ldots, 0)$ for some $1 \leq p \leq N_j$ and $0 \leq z_{jp} \leq \delta_{jp}$.

Therefore, the basic optimal solution of (LP) is consistent with an optimal solution of (RP) if f_j's are convex piecewise linear, implying that the minimization problem over linear constraints is

solvable as an LP in this case. The complexity here would be $T(Nn, m, \Delta)$, where $N = \max_j N_j$. We may also consider the corresponding integer valued problem:

$$\text{(IP)} \qquad \begin{array}{lrcll} \min & \sum_{j=1}^n f_j(x_j) & & & \\ \text{s.t.} & A\mathbf{x} & \geq & \mathbf{b} & \\ & l_j \leq x_j & \leq & u_j & j = 1, \ldots, n \\ & x_j & \in & \mathbb{Z} & j = 1, \ldots, n, \end{array}$$

Let Δ be the value of the largest subdeterminant of A where $A \in \mathbb{R}^{n \times m}$. We may alternatively define $F(\mathbf{x}) = \sum_{j=1}^n f_j(x_j)$ and represent the objective as $\min F(\mathbf{x})$.

Our approach to solving this problem involves designing a scaled problem, RP-s, and approximating the functions f_j with analogous piecewise linear functions $f_j^{L:s}$ that agree with f_j at integer multiples of s. Furthermore, we introduce variables $\{z_{j1}, \ldots, z_{jN_j}\}$ for each x_j, effectively splitting each x_j into N_j increments that are $\frac{1}{s}$ times the length of the original range of x_j. With these new variables, we can redefine each x_j as follows:

$$x_j = s \left\lfloor \frac{l_j}{s} \right\rfloor + s \sum_{p=1}^{N_j} z_{jp}$$
$$0 \leq z_{jp} \leq 1$$
$$N_j = \left\lceil \frac{u_j - l_j}{s} \right\rceil$$

Having broken up the variables x_j into discrete parts, we may now linearize the objective functions accordingly, basically affording each z_{jp} with an objective coefficient equal to the "average slope" of f_j over the range of x_j represented by z_{jp}. To this end, define $\Delta_j^{(p)}$ as follows:

$$\Delta_j^{(p)} = f_j\left(s\left(\lfloor \frac{l_j}{s} \rfloor + p\right)\right) - f_j\left(s\left(\lfloor \frac{l_j}{s} \rfloor + p - 1\right)\right), \text{ for } p = 1, \ldots, N_j$$

We can see that $f_j^{L:s}(z_j) = \sum_{j=1}^n f_j\left(s\lfloor \frac{l_j}{s} \rfloor\right) + \sum_{j=1}^n \sum_{p=1}^{N_j} \Delta_j^{(p)} z_{jp}$. Now, we may define a new linear problem that effectively emulates the RP at integer multiples of s:

$$\text{(RP-s)} \qquad \begin{array}{lrcll} \min & \sum_{j=1}^n f_j^{L:s}(z_j) & & & \\ \text{s.t.} & A\mathbf{z} & \geq & \frac{\mathbf{b}}{s} & \\ & \frac{l_j}{s} \leq x_j & \leq & \frac{u_j}{s} & j = 1, \ldots, n \end{array}$$

In the case of IP, we would need to acknowledge that $\frac{\mathbf{b}}{s}$ may not necessarily be integer-valued, and would have to make the modification:

$$\text{(IP-s)} \qquad \begin{array}{lrcll} \min & \sum_{j=1}^n f_j^{L:s}(x_j) & & & \\ \text{s.t.} & A\mathbf{x} & \geq & \lfloor \frac{\mathbf{b}}{s} \rfloor & \\ & \frac{l_j}{s} \leq x_j & \leq & \frac{u_j}{s} & j = 1, \ldots, n \\ & x_j & \in & \mathbb{Z} & j = 1, \ldots, n. \end{array}$$

While an optimal solution to IP-s may not be feasible to IP, we can assert that the distance between an optimal solution to IP-s and IP is bounded. Furthermore, we note that, upon making the transformation from the variables x_j to the variables $z_{j1}, \ldots, z_{jN_j}$, RP-s and IP-s can be solved as linear programming problems. Should the original problem IP represent a network flow problem, then the fact that the node-arc adjacency matrix A is totally unimodular implies that the constraint matrix of this new, linear programming formulation is also totally unimodular, as we are only duplicating its columns.

2.4.2 Proximity Theorems

Now, we present present proximity theorems that allow us to bound the distances of the optimal solutions of IP and RP from those of IP-s and RP-s, respectively. These will then allow us to design a proximity scaling algorithm for this problem analogous to the one used to solve GAP.

Let $F^{L:s}(\mathbf{x}) = \sum_{j=1}^{n} f_j^{L:s}(x_j)$.

Theorem 2 (Proximity Theorem)**.** *Let $\mathbf{x}^{(s)}$ be the optimal solution to $F^{L:S}(\mathbf{x})$ (the scaled problem). Then there exists an optimal solution, $\mathbf{x}^*$, to the original problem such that $||\mathbf{x}^{(s)} - \mathbf{x}^*||_\infty \leq n\Delta s$.*

Theorem 3. *Let $\mathbf{x}^{(1)}$ be the optimal solution to an integer problem. Then there exists a continuous optimal solution $\mathbf{x}^{(\epsilon)}$ such that $||\mathbf{x}^{(1)} - \mathbf{x}^{(\epsilon)}||_\infty \leq n\Delta$.*

2.4.3 The Proximity Scaling Algorithm

The proximity scaling algorithm essentially works by initially splitting each x_j into $4n\Delta$ pieces. It then solves the associated RP-s (or IP-s). If the answer obtained from this is accurate enough, it terminates. Otherwise it shrinks the feasible interval (using the proximity theorem) for each variable, and further splits the functions into smaller pieces.

Algorithm 4 The Proximity Scaling Algorithm

0. Let $U = \max\{u_j - l_j\}$. If the original problem is IP, set $\epsilon = 1$; otherwise we are solving RP to ϵ-accuracy.
1. Set $s = \left\lceil \frac{U}{4n\Delta\epsilon} \right\rceil \epsilon$
2. Solve RP-S or IP-S to get an optimal solution $x^{(s)}$.
   If $s = \epsilon$, STOP and return $x^{(s)}$.
3. Set $l_j = \max\{l_j, x_j^s - n\Delta s\}$.
   Set $u_j = \min\{u_j, x_j^s + n\Delta s\}$.
4. Set $s = \left\lceil \frac{s}{2\epsilon} \right\rceil \epsilon$ and go to step 2.

Each iteration of the algorithm must solve an LP (or IP in the case of IP-S) that has complexity $T(4n^2\Delta, m, \Delta)$ where $4n^2\Delta$ is the number of variables, m is the number of constraints, and Δ is

the value of the largest subdeterminant. The algorithm then iterates $O(\log \frac{U}{4n\Delta\varepsilon})$, times, leading to an overall complexity of $O\left[(log\frac{U}{4n\Delta\epsilon})T(4n^2\Delta, m, \Delta)\right]$.

A corollary of the result of the complexity of this algorithm is that convex separable minimization over totally unimodular constraints is polynomial time solvable (in integers or with ϵ-accuracy).

2.4.4 Application to Convex Cost Network Flows

Let $G = (V, A)$ be a graph with $|V| = n$ and $|A| = m$. Suppose we are trying to find the minimum cost flow on this graph where the cost on an arc is a convex function $f_{ij}(x_{ij})$ of the flow x_{ij} along that arc. We can use the proximity scaling algorithm to find the optimal flow.

The naive approach would be to solve the minimum cost network flow (MCNF) each time we scale the problem. We would be solving an MCNF on a multigraph with $4n$ arcs replacing each arc in the original graph ($\Delta = 1$ since the constraint matrix is always totally unimodular). This would give a total complexity $O(\log \frac{U}{4m} MCNF(\text{multigraph with } 4nm \text{ arcs}))$ for solving the original problem.

However, there have been other implementations that are smarter and give lower complexities. One such implementation has complexity $O(\log \|b\|_\infty m(m + n \log n))$ where $\|b\|_\infty$ is the absolute value of the largest supply/demand in the graph. Note that this is similar in form to the running time for the linear case of MCNF $O(\log nm(m + n \log n))$.

Another algorithm developed by Menoux has complexity $O(\log ||U||_\infty mn^2)$. And one by Kanzanan and McCormick has complexity $O(mn \log n \log(nc))$, where c is related to the slope of the function near the bounds.

2.5 Nonseparable Convex Minimization

While separable convex minimization over linear constraints was shown to be not much harder than linear optimization, nonseparable convex minimization is much more difficult in general. Even quadratic nonseparable minimization is not necessarily solvable in strongly polynomial time.

2.5.1 The Quadratic Case

Although separable convex quadratic problems may be solvable in strongly polynomial time, the question of strong polynomiality of *nonseparable* quadratic continuous optimization problems is open. While it is possible that the nonseparable optimization problem is solvable in strongly polynomial time, it is at least as hard as the question of strong polynomiality of linear programming (this insight is due to I. Adler). To see this, observe that the problem of feasibility of linear programming $\{A\mathbf{x} = \mathbf{b}, \mathbf{x} \geq 0\} \neq \emptyset$? is equivalent to the following quadratic nonseparable convex problem subject only to nonnegativity constraints:

$$\min_{\mathbf{x} \geq 0} (A\mathbf{x} - \mathbf{b})^T (A\mathbf{x} - \mathbf{b})$$

Certain nonseparable convex *continuous* problems, as well as nonseparable quadratic convex continuous problems are solvable in polynomial time as follows. A solution approximating the optimal *objective* value to the convex continuous problem is obtainable in polynomial time, provided that the gradient of the objective functions are available and that the value of the optimal solution is bounded in a certain interval. Such work, based on the Ellipsoid method, is described by A. S. Nemirovsky and D. B. Yudin (1983). In the quadratic case, exact solutions are possible. The best running time reported to date is by Monteiro and Adler, $O(m^3L)$, where L represents the total length of the input coefficients and m the number of variables.

2.5.2 Integer Non-Separable Quadratic Optimization

The case for the *integer* problems that are *nonseparable*, even if convex, is harder. Nonseparable quadratic *integer* problems are NP-hard, To see this consider the following known reduction from the independent set problem. The maximization of the weight of an independent set in a graph is formulated as follows: Given a graph $G = (V, E)$ with nonnegative weights W_v for each $v \in V$, find a subset of vertices $U \subseteq V$ such that for any $i, j \in U, \{i, j\} \notin E$, and such that the total weight $W(U) = \sum_{v \in U} W_v$ is maximum. The weighted independent set problem can be posed as the quadratic maximization problem:

$$\max_{\mathbf{x} \in \mathbb{B}^{|V|}} \sum_{v \in V} W_v x_v - \sum_{\{u,v\} \in E} W(V) \cdot x_u \cdot x_v$$

Let $\mathbf{x}^*$ be the optimal solution. The maximum weight independent set is then $\{v | x_v^* = 1\}$.Note that the reduction also applies for the unweighted case. So even in the absence of the flow balance constraints, the integer problem is NP-hard.

2.5.3 Separation Scheme for Specific Quadratic Minimization

We illustrate one general purpose technique for nonseparable problems that we call a "separating scheme". The technique relies on converting the objective function into a separable function (e.g. by diagonalizing the matrix Q in the quadratic case). This implies a transformation of variables which affects the constraints. If the new constraints are such that they form a totally unimodular matrix then the proximity-scaling algorithm by [HS90] for separable convex optimization over totally unimodular constraints can be employed to obtain an optimal integer solution. This proximity-scaling algorithm solves, at each iteration, the scaled problem in integers using linear programming.

Consider the nonseparable problem:

$$\begin{array}{lrcl} \min & F(\mathbf{x}) & & \\ \text{s.t.} & A\mathbf{x} & = & \mathbf{b} \\ & \mathbf{0} \leq \mathbf{x} & \leq & \mathbf{u} \\ & \mathbf{x} & \in & \mathbb{Z}^n. \end{array}$$

Suppose there exists an invertible $n \times n$ matrix U such that $F(U\mathbf{x})$ is a separable function. Then the newly stated problem is:

$$\begin{array}{rrcl} \min & F(\mathbf{y}) & & \\ \text{s.t.} & AU^{-1}\mathbf{y} & = & \mathbf{b} \\ \mathbf{0} \leq & U^{-1}\mathbf{y} & \leq & \mathbf{u} \\ & U^{-1}\mathbf{y} & \in & \mathbb{Z}^n. \end{array}$$

Now, if the matrix U is totally unimodular, then the integrality requirement is preserved. If the new matrix of constraints $\begin{bmatrix} AU^{-1} \\ U^{-1} \end{bmatrix}$ is totally unimodular, then the nonseparable flow problem is solvable in polynomial time and in integers using the proximity-scaling algorithm.

Consider now the separating scheme for quadratic objective functions. The formulation of a quadratic nonseparable problem on box constraints is as follows:

$$\begin{array}{rrcccl} \max & \sum_{i=1}^n d_i x_i + \sum_{i=1}^n \sum_{j=1}^n q_{ij} \cdot x_i \cdot x_j & & & & \\ \text{s.t.} & \ell_i & \leq & x_i & \leq & u_i \quad i = 1, \ldots, n \\ & & & x_i & \in & \mathbb{Z} \quad i = 1, \ldots, n. \end{array}$$

The idea of making the problem separable so that the resulting constraint matrix is totally unimodular is translated here to finding a totally unimodular matrix U, so that for the matrix $Q = (q_{ij})$, $U^{-1}QU$ is a diagonal matrix. [BW90] used this approach for a problem of electric distribution systems where only box constraints are present.

[Bal95] has further identified several classes of matrices Q where a "diagonalizing" scheme with a totally unimodular matrix exists. The two classes are:

(1) Diagonally dominant matrices, $q_{ii} \geq \sum_{i \neq j} |q_{ij}|$.

(2) Matrices with forest structure: These are matrices with a partial order on the positive coefficients inducing a forest.

For both these classes, with A empty, there are polynomial algorithms. Also if the constraint matrix, $\begin{bmatrix} AU^{-1} \\ U^{-1} \end{bmatrix}$ is totally unimodular then still integer solutions can be obtained in polynomial time. Continuous solutions can be obtained in polynomial time if the largest subdeterminant of the constraint matrix is bounded by a polynomial ([HS90]).

2.6 The Convex Cost Closure Problem

The minimum closure problem defined on a graph $G = (V, A)$ with weights $w_j \ \forall j \in V$ is defined as follows:

$$\begin{aligned} \min \quad & \sum_{j \in V} w_j x_j \\ \text{subject to} \quad & x_i - x_j \leq 0 \quad (i,j) \in A \\ & l \leq x_j \leq u \quad j \in V \\ & x_j \text{ integer} \quad j \in V \end{aligned}$$

Let $V^+ \equiv \{j \in V | w_i > 0\}$, and $V^- \equiv \{i \in V | w_i \leq 0\}$. We construct an s,t-graph G_{st} as follows. Given the graph $G = (V, A)$ we set the capacity of all arcs in A equal to ∞. We add a source s, a sink t, set A_s of arcs from s to all nodes $i \in V^+$ (with capacity $u_{s,i} = w_i$), and set A_t of arcs from all nodes $j \in V^-$ to t (with capacity $u_{j,t} = |w_j| = -w_j$). The graph $G_{st} = \{V \cup \{s,t\}, A \cup A_s \cup A_t\}$ is a *closure graph* (a closure graph is a graph with a source, a sink, and with all finite capacity arcs adjacent only to either the source or the sink.) This construction is illustrated in Figure 2.

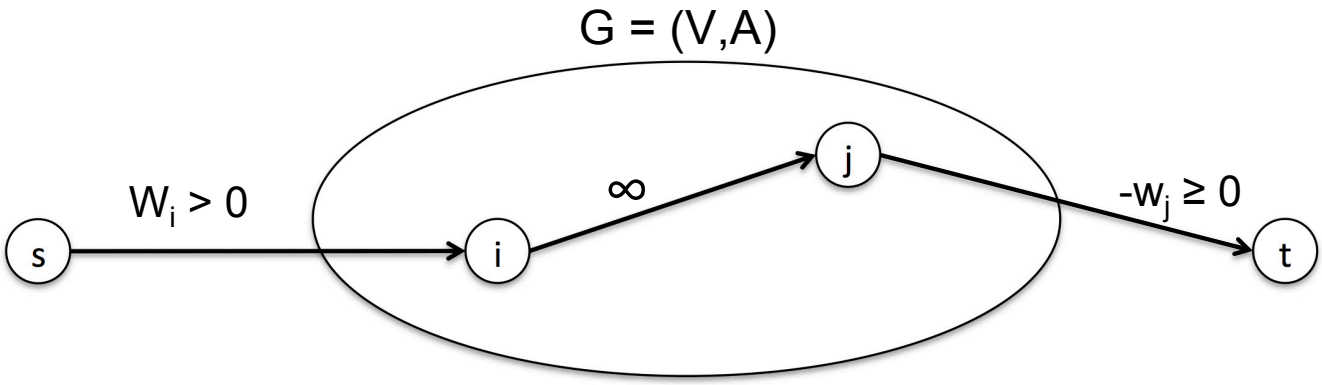


Figure 2: Visual representation of G_{st}.

Proposition 1. *For (S,T) a min (s,t)-cut, T is a minimum weight closure.*

Proof.

$$\begin{aligned} C(S,T) &= \sum_{j \in T \cap V^+} w_j + \sum_{i \in S \cap V^-} && (1) \\ &= \sum_{j \in T \cap V^+} w_j - \left(\sum_{k \in V^- : w_k < 0} w_k - \sum_{p \in T \cap V^+ : w_p < 0} w_p \right) && (2) \\ &= \sum_{j \in T \cap V^+} w_j - \underbrace{\sum_{k \in V^-} w_k}_{W^-} + \sum_{p \in T \cap V^-} && (3) \\ &= \sum_{j \in T} w_j - W^- && (4) \end{aligned}$$

Since W^- is a constant term, we can solve the problem by finding the minimum value of the left side summation:

$$\min_{T \subset V} \sum_{j \in T} w_j \tag{5}$$

If we have a finite cut, the arcs from the cut can go from $s \to t$ or from $t \to s$, but not in between other arcs because they have infinite capacity. S goes to nodes which have positive weight. □

Now, we are interested in the convex cost closure (ccc) problem where the weight for every node $j \in V$ is given by a convex function $f_j(x_j)$. Further, we restrict x_j to be within the range $[l, u]$ and to be integral.

$$\begin{aligned} \min \quad & \sum_{j \in V} f_j(x_j) \\ \text{subject to} \quad & x_i - x_j \leq 0 \quad (i,j) \in A \\ & l \leq x_j \leq u \quad j \in V \\ & x_j \text{ integer} \quad j \in V \end{aligned}$$

Note that integrality is not a real restriction since the constraint matrix is totally unimodular so the solution is guaranteed to be integer. The idea in solving this problem is to reduce it to its binary version.

Proposition 2. *If node weights are linear functions, then in an optimal solution every x_i is either l or u.*

Proof. Observe that the problem can be converted to the minimum closure problem by translating the x variables as follows. For every node $i \in V$, $y_i = \frac{x_i - l}{u - l}$, which yields

$$\begin{aligned} \min \quad & \sum_{i \in V} w_i y_i \\ \text{subject to} \quad & y_i - y_j \leq 0, \quad (i,j) \in E \\ & 0 \leq y_i \leq 1 \quad i \in V \end{aligned}$$

Since the constraint matrix is totally unimodular, the optimal solution is guaranteed to be integer. Hence, every y_i is either 0 or 1. To get the optimal solution to the original problem we solve for x_i and find that

$$x_i = \begin{cases} l & \text{if } y_i = 0 \\ u & \text{if } y_i = 1 \end{cases}$$

□

We therefore conclude that solving (ccc) with a linear objective is no more interesting than solving the minimum closure problem.

Now, for $\alpha \in [\min_{j \in V} l_j, \max_{j \in V} u_j]$, we are going to define the following problem:

$$\begin{aligned} \min \quad & \sum_{j \in V} f_j'(\alpha)\, x_j \\ \text{subject to} \quad & x_i - x_j \leq 0 \quad (i,j) \in A \\ & x_j \in \{0,1\} \end{aligned} \tag{6}$$

We define $f_j'(\alpha)$ as follows:

$$f_j'(\alpha) = f_j(\alpha+1) - f_j(\alpha) = w_j \quad \text{if } \alpha \in [l_j, u_j]$$

$$f_j'(\alpha) = \begin{cases} -W, & \text{if } \alpha < l_j \\ W, & \text{if } \alpha > u_j \end{cases} \quad \text{if } \alpha \notin [l_j, u_j]$$

where $W = \sum_{j \in V : \alpha \in [l_j, u_j]} |w_j|$. This definition follows from the fact that we do not want to include values outside our domain.

Let $x^{(\alpha)}$ be an optimal solution, $S_\alpha = \{x_j^{(\alpha)} = 1\}$. Let S_α be a minimal minimum closet set.

Solving (ccc) with a convex nonlinear objective is not nearly as straight-forward. We introduce the notion of a threshold theorem, which allows us to improve run-time complexity by solving a related problem that prunes the search space.

2.6.1 The Threshold Theorem

Definition 1. *The graph G_α is constructed as follows. Let the weight at node i be the derivative of w_i evaluated at α: $w_i'(\alpha) \approx \lim_{h \to 0} \frac{w_i(\alpha+h) - w_i(\alpha)}{h}$.*

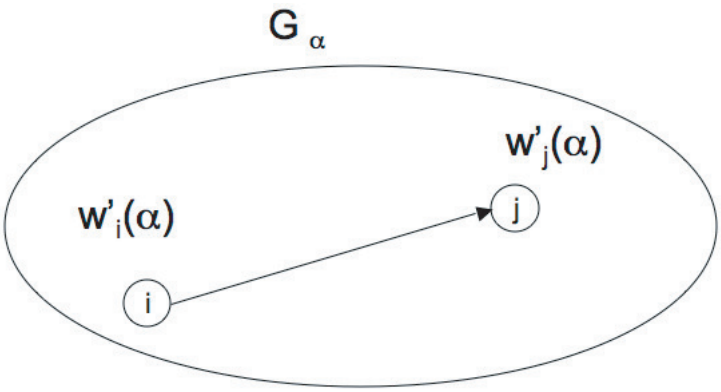


Figure 3: An illustration of G_α

Definition 2. *Let a minimal minimum closed set on a graph be a minimum closed set that does not contain any other minimum closed set.*

We now present the threshold theorem as follows.

Theorem 4. *An optimal solution to (ccc) on G, x^*, satisfies*

$$\begin{aligned} x_i^* &\geq \alpha \quad \text{if } i \in S_\alpha^* \\ x_i^* &< \alpha \quad \text{if } i \in \bar{S}_\alpha^* \end{aligned}$$

where S_α^* is a minimal minimum weight closed set in G_α.

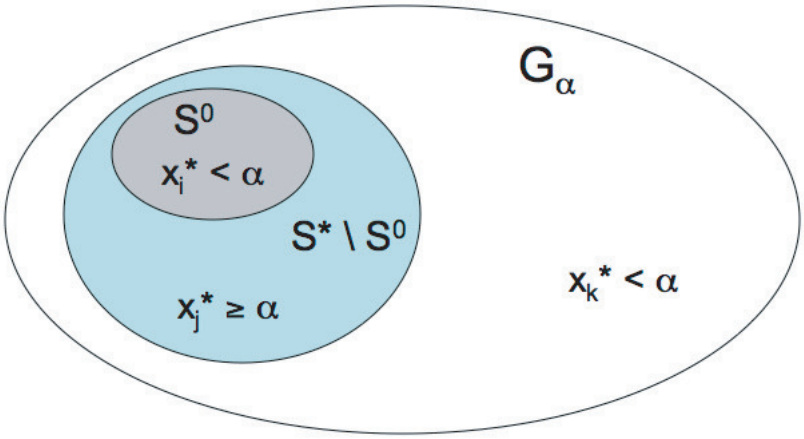


Figure 4: An illustration of the threshold theorem (1)

Proof. For sake of contradiction, let S_α^* be a minimal minimum weight closed set on G_α, and let there be a subset $S_\alpha^0 \subseteq S_\alpha^*$ such that at an optimal solution $x_j^* < \alpha$ for all $j \in S_\alpha^0$. Subsequently, the optimal value for every node $i \in S_\alpha^* \backslash S^0$ has weight $\geq \alpha$. (See Figure 4.)

Recall that the problem requires that $x_i \leq x_j$ for all $(i,j) \in A$. As a consequence, there cannot be a node in S_α^0 that is a successor of a node in $S_\alpha^* \backslash S_\alpha^0$, otherwise the constraint will be violated. Since S_α^* is a closed set and there are no nodes in $S_\alpha^* \backslash S_\alpha^0$ that have successors in S_α^0, $S_\alpha^* \backslash S_\alpha^0$ must be a closed set. But this $S_\alpha^* \backslash S_\alpha^0$ cannot be a minimum closed set, otherwise we would violate the assumption that S_α^* is a minimal minimum closed set (since $S_\alpha^* \backslash S_\alpha^0 \subset S_\alpha^*$). Therefore, it must be the case that:

$$\begin{aligned} \sum_{j \in S_\alpha^* \backslash S_\alpha^0} w_j'(\alpha) &> \sum_{j \in S_\alpha^*} w_j'(\alpha) \\ &= \sum_{j \in S_\alpha^0} w_j'(\alpha) + \sum_{j \in S_\alpha^* \backslash S_\alpha^0} w_j'(\alpha) \end{aligned}$$

which implies that

$$\sum_{j \in S_\alpha^0} w_j'(\alpha) < 0 \,.$$

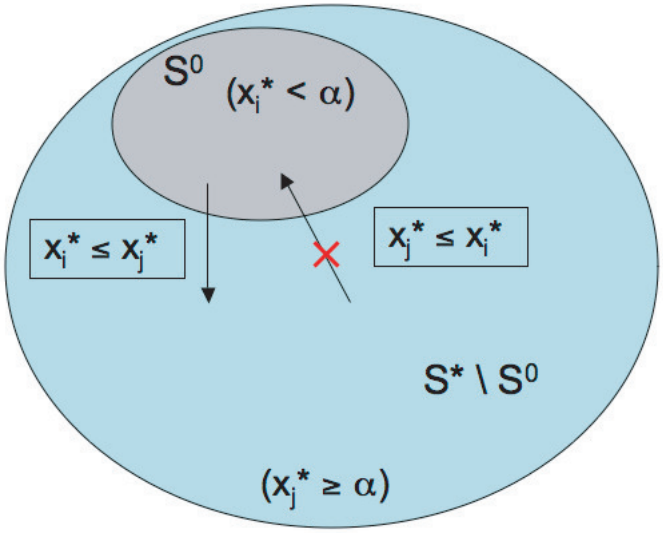


Figure 5: An illustration of the threshold theorem (2)

Next we observe that increasing the values x_i^* to α for all $i \in S^0$ does not violate the constraint that $x_i^* \leq x_j^*$, since by construction $x_i^* \leq \alpha < x_j^*$ for all $j \in S^*$ and $(i, j) \in A$.

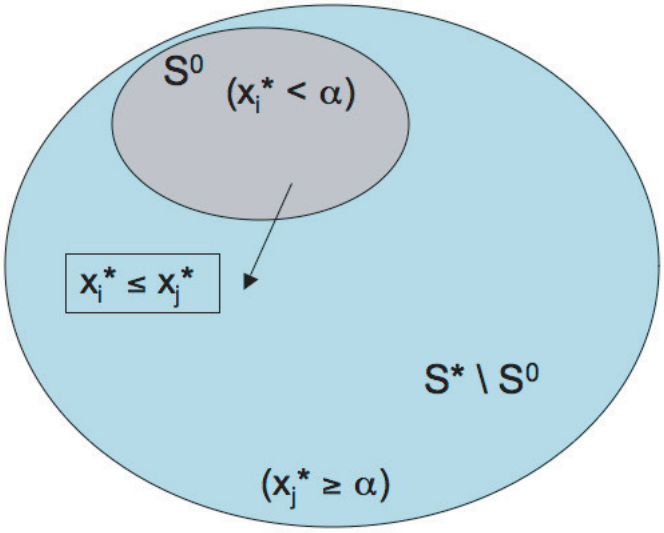


Figure 6: An illustration of the threshold theorem (3)

Since the node weights $w()$ are convex functions, their sum W is also a convex function. Further, we note that the derivative a convex function is monotone nondecreasing, and for any $\epsilon > 0$, if $W'(\alpha) < 0$ then $W(\alpha - \epsilon) > W(\alpha)$. (See Figure 7.) Therefore, if for all $i \in S_\alpha^0$ we increase the value of x_i^* to α, we will strictly reduce the weight of the closure. This is a contradiction on the assumption that x^* is optimal and S_α^* is a minimum closed set. □

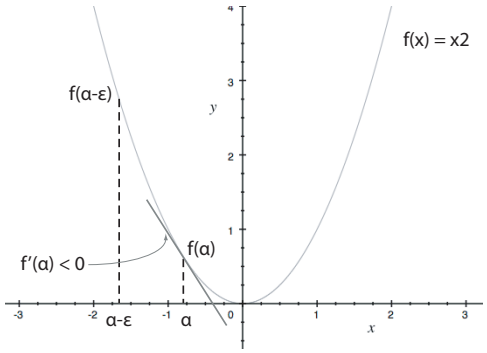


Figure 7: Properties of convex function

2.6.2 A Naive Algorithm for Solving (ccc)

The *Threshold Theorem* can be used to solve the convex cost closure problem by testing all values of $\alpha \in [l, u]$. The optimal value for node $i \in V$ is $x_i^* = \alpha$ such that $i \in S_{\alpha-1}^*$ and $i \in \bar{S}_\alpha^*$. The running time for this algorithm is $(u - l)\ O(\min\ s\text{-}t \text{ cut})$. Since $u = 2^{\log_2 u}$ and $l = 2^{\log_2 l}$, we note that this is an exponential-time algorithm.

2.6.3 Solving (ccc) in Polynomial Time Using Binary Search

Thanks to the threshold theorem we get some information from α and the solution for every α. Thus, we can apply a more efficient algorithm for solving the problem.

We can improve the running time by using a binary search as follows. Pick a node $v \in V$. Solve for S_α^*, choosing α to be the median of u and l. If $v \in S_\alpha^*$, then by the Threshold Theorem we know that $x_v^* > \alpha$, and we set α to the median of $\frac{u-l}{2}$ and u. Otherwise, let α be the median of l and $\frac{u-l}{2}$. We repeat the process, performing binary search on α until we find the optimal solution. Algorithm 5 (below) describes the procedure in greater detail.

The binary search for every variable takes $O(\log_2 (u - l))$ time to compute, and we must do this a total of n times (once for each node in V). Hence, this method reduces our running time to $n \log_2 (u - l)\ O(\min\ s\text{-}t \text{ cut})$, which is polynomial.

2.6.4 Solving (ccc) Using Parametric Minimum Cut

Hochbaum and Queyranne provide further run-time improvements for optimally solving (ccc) in [HQ03]. This section summarizes their result.

From G we construct a parametric graph G_λ by introducing a source node s and sink node t. We add an arc from s to every node $j \in V$ with capacity $\max\{0,\ f_j'(\lambda)\}$, and an arc from i to j with capacity $-min\{0,\ w_j'(\lambda)\}$. We make the following observations.

1. Every arc in G_λ has nonnegative capacity.

Algorithm 5 Binary search algorithm for (ccc)

Require: $G = (V, A)$ is a closure graph, where node weights $w()$ are convex functions

procedure BINARY SEARCH(G, l_0, u_0)
  for $v = 1, \ldots, n$ **do**
    $l \leftarrow l_0, \quad u \leftarrow u_0$
    while $u - l > 0$ **do**
      $\alpha \leftarrow l + \lceil \frac{u-l}{2} \rceil$ ▷ Set α to the median of u and l
      $S^*_\alpha \leftarrow$ source set of minimum $s - t$ cut on G_α
      if $v \in S^*_\alpha$ **then**
        $l \leftarrow l + \lceil \frac{u-l}{2} \rceil$ ▷ α is the new lower bound on x^*_v
      else
        $u \leftarrow l + \lceil \frac{u-l}{2} \rceil$ ▷ α is the new upper bound on x^*_v
      end if
    end while
    $x^*_v \leftarrow \alpha$
  end for
  return $x^* = [x^*_1, \ldots, x^*_n]$
end procedure

2. In the residual graph of G_λ, every node in V is connected to either the source or the sink (not both).

3. Arcs adjacent to the source have capacities that are monotone nondecreasing in λ, and arcs adjacent to the sink have capacities that are monotone nonincreasing as a function of λ.

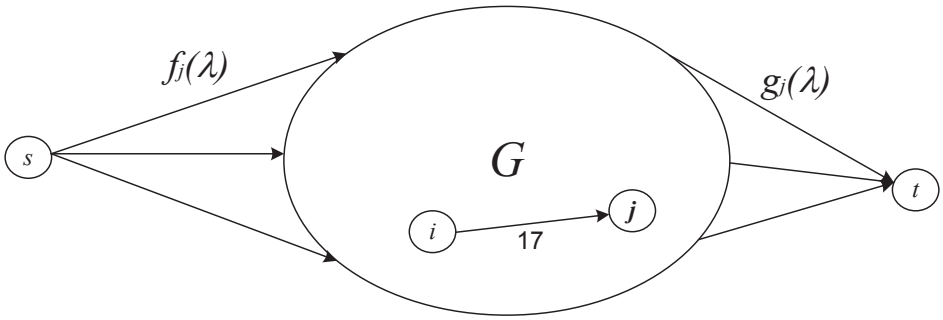


Figure 8: Parametric capacity graph

Let S_λ be the source set of a minimum cut in G_λ; equivalently, S_λ is a minimum closed set on G_λ. As we increase λ in the interval $[l, u]$, we find that the size of the source set S_λ increases, and S_λ is a subset of source sets corresponding to higher values of λ. More formallly,

$$S_\lambda \subseteq S_{\lambda+1} \quad \text{for all } \lambda.$$

Definition 3. *For any node $v \in V$, the node shifting breakpoint is defined as the largest value $\underline{\lambda}$ such that v is in the source set $S_{\underline{\lambda}}$, and the smallest value $\bar{\lambda}$ such that v is not in the minimum closed set ($S_{\bar{\lambda}}$). Note that $\bar{\lambda} = \underline{\lambda} + 1$. Further,*

$$x_v^* = \min_{x \in [\underline{\lambda}, \bar{\lambda}]} w_v(x)$$

If we can find the node shifting breakpoint for each node $v \in V$, then we can construct the optimal solution to (ccc) as shown in Algorithm 6. Hochbaum and Queyranne show that this problem reduces to solving the parametric minimum cut problem on G [HQ03]. Conveniently, Gallo, Grigoriadis, and Tarjan provide a method that solves this problem in the same running time as a single minimum closure (or maximum flow) problem [GGT89].

Algorithm 6 Breakpoint method for solving (ccc)

Require: $G = (V, A)$ is a closure graph, where node weights $w()$ are convex functions

1: **procedure** BREAKPOINT(G, w_v for $v = 1, \ldots, n$)
2: Call (the modified) PARAMETRIC-CUT to find a set of up to n breakpoints $\lambda_1, \ldots, \lambda_n$
3: **for** $v = 1, \ldots, n$ **do**
4: $\min_v = \arg\min_{x \in [l,u]} w_v(x)$
5: Let the optimal value of x_v fall in the interval $(\lambda_{v_i - 1}, \lambda_{v_i}]$
6: Then x_v^* is determined by

$$x_v^* = \begin{cases} \lambda_{v_i - 1} + 1 & \text{if } \min_v \leq \lambda_{v_i - 1} \\ \lambda_{v_i} & \text{if } \min_v \geq \lambda_{v_i} \\ \min_v & \text{if } \lambda_{v_i - 1} \leq \min_v \leq \lambda_{v_i} \end{cases}$$

7: **end for**
8: **return** $x^* = [x_1^*, \ldots, x_n^*]$
9: **end procedure**

The first step of the algorithm requires solving the parametric cut problem on G to find the set of node shifting breakpoints. Following the method outlined in [GGT89], this requires $O\left(mn \log \frac{n^2}{m} + n \log (u - l)\right)$ time to compute, where the second term corresponds to finding the minimum for each node weight function using binary search. Since determining the optimal solution x^* takes time linear in n, the total running time of the algorithm is

$$O\left(mn \log \frac{n^2}{m} + n \log (u - l)\right) .$$

If the node weight functions are quadratic we can find the minimum of each function in constant time, and the complexity of the algorithm reduces to $O\left(mn \log \frac{n^2}{m}\right)$; this is equivalent to solving maximum flow (or minimum $s - t$ cut) on a graph.

2.7 The Convex S-Excess Problem and Hochbaum's Algorithm

The *convex s-excess* problem is a special case of MRF where the separation functions are linear functions. Given a directed graph $G = (V, A)$, the convex s-excess problem is formulated as follows:

$$\text{(convex s-excess)} \qquad \begin{array}{lrcll} \min & \sum_{i\in V} f_i(x_i) + \sum_{(i,j)\in A} u_{ij} z_{ij} & & & \\ \text{s.t.} & x_i - x_j & \leq & z_{ij}, & (i,j) \in A \\ & \ell_i \leq x_i & \leq & u_i, & i \in V, \text{ integer} \\ & z_{ij} & \geq & 0, & (i,j) \in A. \end{array}$$

Here, the f_i are convex. The convex s-excess problem is a relaxation of the convex cost closure (CCC) problem, where the partial order constraint $x_i \leq x_j$ is relaxed and the amount of violation to the partial order constraint, reflected in the z_{ij} variable, is penalized in the objective function.

One application of convex s-excess is in *image segmentation*, as shown by Hochbaum in [Hoc01]. Note that although the graph employed in the image segmentation problem is a grid, convex s-excess permits arbitrary directed graphs.

In [Hoc01], Hochbaum devises an efficient algorithm to solve the convex s-excess problem that is provably the fastest possible. This algorithm relies on solving another subproblem, the s-excess problem:

$$\text{(s-excess)} \qquad \begin{array}{lrcll} \min & \sum_{j\in V} f'_j(\alpha) x_j + \sum_{(i,j)\in A} u_{ij} z_{ij} & & & \\ \text{s.t.} & x_i - x_j & \leq & z_{ij} & \forall (i,j) \in A \\ & x_j & \in & \mathbb{B} & \forall j \in V \\ & z_{ij} & \in & \mathbb{B} & \forall (i,j) \in A. \end{array}$$

This problem is effectively a specification of the convex s-excess. This problem can by solved by creating a parametric graph $G^{st}(\alpha) = (V_{st}, A_{st})$ for some scalar integer value of $\alpha \in [\min_{i\in V} \ell_i, \max_{i\in V} u_i]$, from the original graph G. To this end, $V_{st} = V \cup \{s, t\}$ and $A_{st} = A \cup A_s \cup A_t$. The appended node s is called the *source node* and t is called the *sink node*. The respective sets of *source adjacent arcs* and *sink adjacent arcs* are defined as $A_s = \{(s, i) : i \in V\}$ and $A_t = \{(i, t) : i \in V\}$. Each arc $(i, j) \in A_{st}$ has an associated nonnegative capacity c_{ij}. For arcs $(i, j) \in A$, $c_{ij} = u_{ij}$. Each arc in $A_s = \{(s, i)\}_{i\in V}$ has capacity $c_{si} = -\min\{0, f'_i(\alpha)\}$ and each arc in $A_t = \{(i, t)\}_{i\in V}$ has capacity $c_{it} = \max\{0, f'_i(\alpha)\}$, where $f'_i(\alpha) = f_i(\alpha + 1) - f_i(\alpha)$, the right sub-gradient of $f_i(x_i)$ at argument α on the integer grid. Note that, for any given value of α, either $c_{si} = 0$ or $c_{it} = 0$. Then, the s-excess problem can be solved by finding a minimum cut $(S^*, \bar{S}^*)$ on this graph G^{st} and setting $x_i = 0 \forall i \in S^*$ and $x_i = 1 \forall i \in \bar{S}^*$.

A key idea in Hochbaum's algorithm is the following *threshold theorem* (analogous to the similarly-named theorem for (ccc)), which links the optimal solution of the convex s-excess problem with the minimum cut partitions in $G^{st}(\alpha)$:

Theorem 5. (Threshold Theorem) *For any given α, let S^* be the maximal source set of the minimum cut in graph $G^{st}(\alpha)$. Then there is an optimal solution $\mathbf{x}^*$ to the convex s-excess problem satisfying:*

$$x_j^* \begin{cases} > \alpha & \forall j \in S_\alpha \\ \leq \alpha & \forall j \notin S_\alpha. \end{cases}$$

An important property of $G^{st}(\alpha)$ is that the capacities of source adjacent arcs are non-increasing functions of α (due to the convexity of functions $f_i(x_i)$), the capacities of sink adjacent arcs are non-decreasing functions of α (due to the convexity of functions $f_i(x_i)$), and the capacities of all the other arcs are constants. This implies the following *nested cut property*:

Lemma 1. (Nested Cut Property [GGT89, Hoc01]) *For any two parameter values $\alpha_1 \leq \alpha_2$, let $S^*_{\alpha_1}$ and $S^*_{\alpha_2}$ be the respective maximal source set of the minimum cuts of $G^{st}(\alpha_1)$ and $G^{st}(\alpha_2)$, then $S^*_{\alpha_1} \supseteq S^*_{\alpha_2}$.*

According to the nested cut property, the dynamic of the minimum cuts in $G^{st}(\alpha)$ as α increases from $\min_{i \in V} \ell_i$ to $\max_{i \in V} u_i$ works as follows: initially, all nodes in V are in the source set, as α increases, some nodes jump from the source set to the sink set, and those nodes in the sink set will never jump back to the source set as α continues to grow. Ultimately, all nodes will join the sink set. As a result, in order to solve the minimum cuts in $G^{st}(\alpha)$ for all values of α, it is sufficient to find the "key" values of α at which at least one node jumps from source set to sink set. We call these values as *breakpoints*:

Definition 4. *A value of α is a breakpoint if $S^*_\alpha \supset S^*_{\alpha+1}$.*

Let $\alpha_1 < \alpha_2 < \ldots < \alpha_q$ be the list of breakpoints. Thus we have $V = S^*_{\alpha_1} \supset S^*_{\alpha_2} \supset \ldots \supset S^*_{\alpha_q} \supset S^*_{\alpha_{q+1}} = \emptyset$. According the nested cut property, we have $q = O(n)$. For each node $i \in V$, if $i \in S_{\alpha_j} \setminus S_{\alpha_{j+1}}$, we have $x_i^* = \alpha_j$.

In [Hoc01], Hochbaum shows that it is possible to solve for all breakpoints on the parametric graph $G^{st}(\alpha)$ in time $O(nm \log \frac{n^2}{m} + n \log U)$, where $U = \max_i\{u_i - \ell_i\}$. Note that $O(nm \log \frac{n^2}{m})$ is the complexity of a *single* minimum s,t-cut. Hochbaum's algorithm is fastest possible for convex s-excess since the convex s-excess problem generalizes the minimum cut problem on a graph, of complexity $O(nm \log \frac{n^2}{m})$, and the problem of finding the minimum of n convex functions, of complexity $O(n \log U)$.

2.8 Properties of Cut Functions of a Parametric Graph

A *parametric graph* $G^{st}(\alpha)$ is a capacitated s,t-graph where the capacities of source and sink adjacent arcs are functions of a parameter α, and satisfies the following properties:

1. The capacities of arcs that are not adjacent to source or sink are constants.
2. The capacities of source adjacent arcs are non-increasing functions of α.
3. The capacities of sink adjacent arcs are non-decreasing functions of α.

Note that it can also be the case that the capacities of source adjacent arcs are non-decreasing functions of α and the capacities of sink adjacent arcs are non-increasing functions of α.

We are interested in the minimum cut partitions in $G^{st}(\alpha)$ for different values of α. Besides the nested cut property introduced above, the minimum cut partitions in $G^{st}(\alpha)$ have some additional interesting properties. We define $C_{\hat{\alpha}}(\alpha)$ as the cut capacity function of α with respect to the minimum cut partition generated at the value of $\hat{\alpha}$. The set of cut capacity functions have the following properties:

1. Let $\alpha_1 < \alpha_2 < \ldots \alpha_q$ be the list of breakpoints. $C_{\alpha_j}(\alpha) - C_{\alpha_{j-1}}(\alpha)$ is monotone non-increasing. This property is called "breakpoint-concavity".
2. If there are two or more breakpoints in $[\alpha_\ell, \alpha_u]$, then α^*, the intersection of $C_{\alpha_\ell}(\alpha)$ and $C_{\alpha_u}(\alpha)$, "separates" the breakpoints in the interval $[\alpha_\ell, \alpha_u]$, in that $[\alpha_\ell, \alpha^*]$ and $[\alpha^*, \alpha_u]$ each contains at least one breakpoint.
3. The complexity of finding the intersection α^*, is the same as the complexity of finding the value for which the sum of the derivatives of convex functions is equal to k (Same as finding the minimum of a sum of convex functions (for the capacities of the source and sink adjacent arcs) plus a linear function (for the constant capacities of the non-source and non-sink adjacent arcs).

2.9 Applications of Markov Random Fields

Consider a graph $G = (V, A)$ and a set of possible values $\mathcal{X}$. In the following examples, we consider problems of the form:

$$\text{(MRF)} \qquad \min_{x \in \mathcal{X}^n} \sum_{i \in V}^{n} G_i(x_i) + \sum_{(i,j) \in A} F_{ij}(x_i - x_j), \tag{7}$$

where the G_i's and F_{ij}'s can take a number of forms. Note that this is a generalization of the convex s-excess problem. Figure 9 provides the complexity of solving the MRF problem conditioned on a few example forms for these functions.

Deviation function $G_i()$	**Separation function $F_{ij}()$**	**Complexity**
Convex	Convex	$O(mn \log \frac{n^2}{m} \log nU)$
Convex	Bilinear	$O(mn \log \frac{n^2}{m} + n \log U)$
Quadratic	Bilinear	$O(mn \log \frac{n^2}{m})$
General (nonlinear)	convex	$O(mnU^2 \log \frac{n^2 U}{m})$
Linear	Nonlinear	NP-hard

Figure 9: Complexity of the MRF problem given various forms for the G_i and F_{ij} functions [Hoc13].

2.9.1 Multiway Cut

Given a pre-determined set of "terminal" nodes $\{s_1, \ldots, s_k\}$, the multiway cut problem is that of that of choosing the minimum weight subset of arcs in A such that the removal of this subset partitions the graph into k subgraphs, each of which contains exactly one of the terminals $s_1, \ldots, s_k$. In the case where $k = 2$, this is clearly the minimum cut problem, and so can be solved in polynomial time. However, this problem is NP-hard for general k.

We may represent this problem as an MRF by setting $G_i(x_i) = 0 \forall i$ and $F_{ij}(x_i - x_j)$ to be the dirac function, taking value u_{ij} (the weight of arc $(i, j) \in A$) when $x_i = x_j$, and zero otherwise. Additionally, we would fix $x_{s_1} = s_1, \ldots, x_{s_k} = s_k$ to ensure that there will be k cuts. Since this formulation falls into the final category of MRF problems denoted in Figure 9, we can see that solving MRF problems with linear G_i and generic nonlinear F_{ij} must be at least as hard as solving the multiway cut problem, that is to say, it is NP-hard as well.

2.9.2 Isotonic Regression [Hoc01]

Another, more hopeful example of the use of MRF is in the case of isotonic regression, also referred to as statistical inference. In this problem, the goal is to estimate a generic monotonically nondecreasing function $f : \mathbb{R} \to \mathbb{R}$, given a series of noisy samples $\{(\hat{x}_i, \hat{y}_i) \in \mathbb{R} \times \mathbb{R}\}_{k \in K}$.

Since f is nonparametric and nothing is assumed about it other than its monotonicity, we can only estimate the function at the points $\{\hat{x}_k\}_{k \in K}$. We denote our estimates $\{y_k\}_{k \in K}$. Now, we may define deviation penalties $G_k(y_k - \hat{y}_k)$ for each point estimated in order to force our estimates to reflect the samples as closely as possible. For example, we may set G_k to be the absolute value function. Then, for any $\hat{x}_i \geq \hat{x}_j$, we could set the bilinear cross function:

$$F_{ij}(y_i - y_j) = \begin{cases} M(y_i - y_j), & y_i - y_j < 0 \\ 0, & \text{otherwise,} \end{cases} \tag{8}$$

where M is some large coefficient to keep the model from selecting $y_i < y_j$ for $\hat{x}_i \geq \hat{x}_j$. If the functions G_k are each Lipschitz with parameter L_k, then we may define $M = \sum_{k \in K} L_k$. Alternatively, we may include the constraints $y_i \geq y_j$ explicitly and instead use extant algorithms for the convex closure problem. We can see that this problem is effectively the MRF on a graph that is a path; that is, the graph is composed of a node for every data point, and arcs tracing a simple path passing through every node.

2.9.3 Image Segmentation [Hoc13]

In the image segmentation (IS) problem, we are given a picture composed of a grid of pixels, some of which have been corrupted. The goal, assuming that the original picture was not merely random noise and was of some object, is to reconstruct the picture by selecting the proper colors for the pixels, using information from surrounding pixels as well as the possibly incorrect color of the pixel as we are given it. Effectively, we make the assumption that adjacent pixels are likely to be similarly

colored (since the pixels are in a grid, we define adjacency to be directly above, below, or to the right or left of another pixel), and so may use that to decide when to deviate from the picture that we are originally given.

To that extent, we may define the MRF problem over a graph $G = (V, A)$, where the set of nodes V has a node x_i for each pixel i and there are two arcs $(i, j), (j, i)$ for every two adjacent pixels i, j (as the pixels are arranged in a grid, we define two pixels as adjacent if one is directly above the other or to the left/right of the other). We let each pixel take values in some set X denoting the possible colors and refer to the *a priori* color of each pixel i as $r_i \in X$. Then, we may define the separation penalty functions F_{ij} similarly as above:

$$F_{ij}(x_i - x_j) = \begin{cases} a(x_i - x_j), & x_i - x_j < 0 \\ 0, & \text{otherwise,} \end{cases} \tag{9}$$

and the deviation penalty functions $G_i(x_i - r_i)$ as some convex function, possibly a quadratic or weighted absolute value function $b|\cdot|$. Clearly, our choice of a, b signals the incentive to deviate from the given colors r_i. Figure 10 shows the effects of increasing the ratio $S = \frac{a}{b}$.

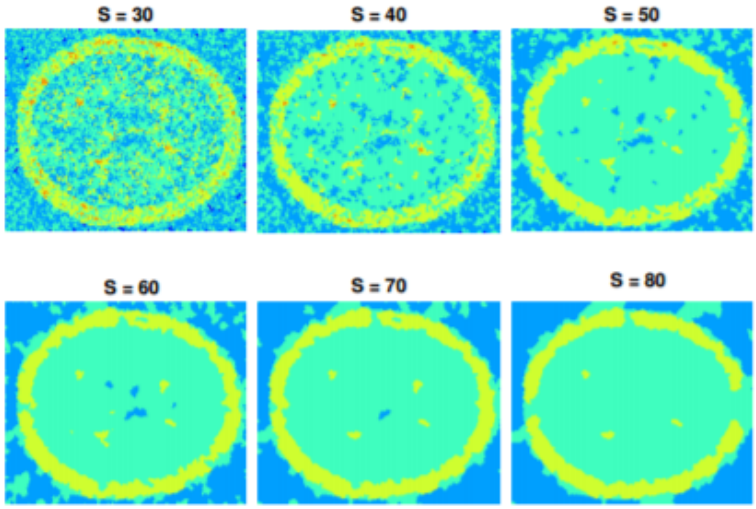


Figure 10: Impact of increasing the penalty for separation vs. that for deviation [Hoc13].

Modifying the set X also has an impact. Figure 11 shows an example of the effects of increasing $k = |X|$.

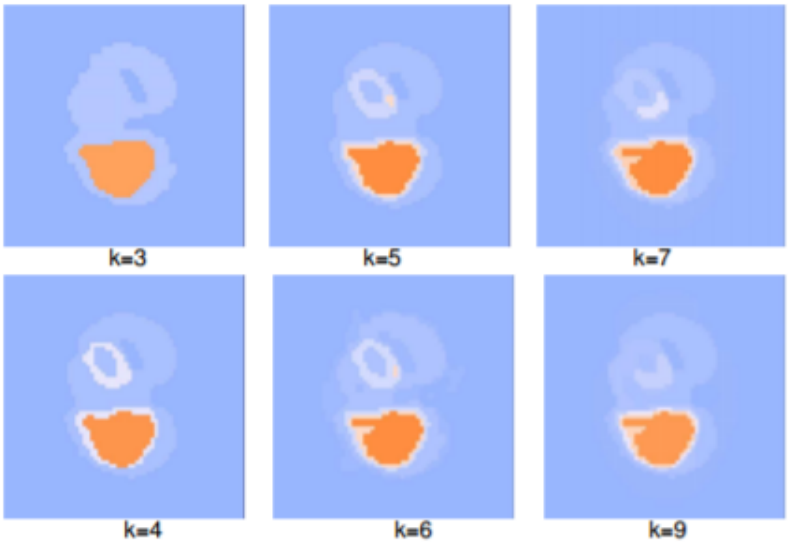


Figure 11: The impact of a larger set of possible colors [Hoc13].

2.9.4 Ising Model [KS80]

The concept of the MRF may seem somewhat detached from its name, as there does not seem to be any mention of randomness, the Markovian property, or even fields of any sort. Interestingly, the origins of MRF came from German scientist Ernst Ising in the 1920's, who was looking to develop a mathematic model of ferromagneticism in substances, i.e. the propensity of certain substances to turn into magnets at certain temperatures. To this end, he modeled a substance as a set of particles I arranged in a grid (not necessarily 2-dimensional), not unlike the one we defined in the previous section for the pixels in an image, each with a certain spin $\omega_i \in \{+, -\}$. Let $\sigma(\omega_i)$ be an indicator that takes the value 1 if $w_i = +$ and -1 otherwise. Adjacent particles had a certain level of impact on each other, and there was assumed to be an underlying magnetic field that impacted the spin of particles as well. To this end, for some configuration of spins $\omega = (\omega_1, \ldots, \omega_{|I|}) \in \Omega$ (Ω denotes the set of all possible configurations), Ising defined the energy measure:

$$U(\omega) = -\left(H \sum_{i \in I} \sigma(\omega) + J \sum_{i \in I} \sum_{j \in N(i)} \sigma(\omega_i), \sigma(\omega_j) \right) \tag{10}$$

Here, the constant J is a property of the material that determines the tendency for neighboring spins to align, the constant H is the intensity of the underlying magnetic field, and $N(i)$ represents the set of particles neighboring particle i. Ising then assigned a likelihood measure on these states:

$$e^{-\frac{1}{kT}U(\omega)},$$

as well as the Gibbs probability measure:

$$P(\omega) = \frac{e^{-\frac{1}{kT}U(\omega)}}{Z}, \quad Z = \sum_{\omega \in \Omega} e^{-\frac{1}{kT}U(\omega)}, \tag{11}$$

where T is the temparature and k is some universal constant.

Although this probability measure may seem a bit arbitrary, it has a number of benefits in the context of the original problem; namely, it has explicit ties to the entropy of the distribution of configuration and gives the model a Markovian property, from which the MRF derives its name. However, these are beyond the scope of this class. Ising himself was ultimately unable to make any meaningful computations in anything beyond the one-dimensional version of this model before being forced out of Germany in 1936, but contemporary tools in MRF make the problem much more tractable in higher dimensions (one can immediately see the similarity of this model to the IS problem from the previous section). The model itself faded into obscurity in the physics literature with the introduction of the more complicated Heisenberg model in the 1930's, but clearly remains of mathematical interest and practical use up to the present day.

3 Approximation Algorithms

In this section, we will cover approximation algorithms for several intractable graph problems, such as the set cover, as well as analogous problems on planar graphs. We also provide approximation algorithms for metric covering and packing problems and the Knapsack problem. First, we state some definitions and a result regarding approximation schemes:

Definition 5. *A Polynomial-Time Approximation Scheme (PTAS) delivers a* $(1+\epsilon)$*-approximation for every fixed* ϵ*.*

Definition 6. *A Fully-Polynomial-Time Approximation Scheme (FPTAS) delivers a* $(1+\epsilon)$*-approximation in time polynomial in both* n *and* $\frac{1}{\epsilon}$*.*

Theorem 6 (GJ 1979)**.** *Unless* $P = NP$*, there is no FPTAS for the strongly NP-complete problems.*

3.1 The Set Cover Problem

Given an universal set I of m elements to be covered and a collection of sets $S_j \subseteq I$, $j \in J = \{1, \ldots, n\}$ with each set having a weight w_j associated with it, the *set cover problem* is to identify the smallest weight collection of sets so that all elements of I are included in their union (or "covered"). We define:

$$x_j = \begin{cases} 1 & \text{if } S_j \text{ is selected} \\ 0 & \text{otherwise} \end{cases}$$

$$a_{ij} = \begin{cases} 1 & \text{if item } i \text{ belongs to set } S_j \\ 0 & \text{otherwise} \end{cases}$$

$$\text{(SC)} \qquad \begin{array}{rlcll} \text{Min} & \sum_{j=1}^{n} w_j x_j & & & \\ \text{s.t.} & \sum_{j=1}^{n} a_{ij} x_j & \geq & 1 & i = 1, \ldots, m \\ & x_j & \in & \mathbb{B} & j = 1, \ldots, n. \end{array}$$

3.1.1 The Greedy Algorithm for the Set Cover Problem

A greedy algorithm is the most natural heuristic for set cover. It works by selecting one set at a time that covers the most elements among the uncovered ones. Johnson and Lovász ([Joh74], [Lov75]) were the first to demonstrate that the greedy algorithm is a $\mathcal{H}(d)$- approximation algorithm for the unweighted set cover problem, where $\mathcal{H}(d) = \sum_{i=1}^{d} \frac{1}{i}$ and d is the size of the largest set. $\mathcal{H}(d)$ is bounded by $1 + \log d$.

Chvátal [Chv79] extended the applicability of the greedy algorithm to the weighted set cover. This version of greedy selects a set with the minimum ratio of weight to remaining coverage. Chvátal proved that this greedy algorithm is still a $\mathcal{H}(d)$-approximation algorithm. Although the algorithm is easily stated, its analysis is far from trivial. That analysis is particularly instructive as it introduces the use of linear programming duality in approximations which we will present next. The formal statement of the greedy is,

Algorithm 7 The Greedy Algorithm [Chvátal]

1. Set $C^G = \emptyset$; $S_j^1 = S_j, j \in J$; $I = \{1, \ldots, m\}$; $k = 0$.
2. $k \leftarrow k + 1$. Select a set S_{j_k}, such that $\frac{w_{j_k}}{|S_{j_k}^k|} = \min_{j \in J} \frac{w_j}{|S_j^k|}$.
3. Set $C^G \leftarrow C^G \cup \{j_k\}$, $J \leftarrow J \setminus \{j_k\}$ and $S_j^{k+1} = S_j^k \setminus S_{j_k}^k, j \in J$, $I \leftarrow I \setminus S_{j_k}^k$.
4. If $I = \emptyset$, stop and output cover C^G. Else, go to Step 2.

Consider the linear programming relaxation of (SC), with the upper bound $x_j \leq 1$ constraints omitted (an optimal solution will satisfy those constraints automatically). The dual problem is

$$\text{(SC-dual)} \qquad \begin{array}{rlcll} \text{Max} & \sum_{i=1}^{m} y_i & & & \\ \text{s.t.} & \sum_{i=1}^{m} a_{ij} y_i & \leq & w_j & j = 1, \ldots, n \\ & y_i & \geq & 0 & i = 1, \ldots, m. \end{array}$$

The analysis of Greedy relies on allocating the weights of the set selected by the greedy heuristic to the elements covered and interpreting those as a form of dual, not quite feasible, solution.

Theorem 7. *The greedy heuristic is a $\mathcal{H}(d)$-approximation algorithm.*

Proof. To prove the desired result, it suffices to show that for *any* cover C, indicated by the characteristic vector $\{x_j\}$, and a cover delivered by the greedy C^G,

$$\sum_{j\in C}\mathcal{H}(d)w_j = \sum_{j=1}^{n}\mathcal{H}(d)w_j x_j \geq \sum_{j\in C^G} w_j \ . \tag{12}$$

Applying this inequality to C^*, the optimal cover, yields that the value of the solution delivered by the greedy is at most $\mathcal{H}(d)$ times the value of the optimal solution. To prove (12), it is sufficient to find an "almost feasible" dual solution $\mathbf{y}$ such that,

$$\sum_{i=1}^{m} a_{ij}y_i \leq \mathcal{H}(|S_j|)w_j \quad j = 1,\ldots,n \tag{13}$$

and so that the weight of the sets selected is accounted for by $\mathbf{y}$,

$$\sum_{i=1}^{m} y_i = \sum_{j\in C^G} w_j. \tag{14}$$

Such $\mathbf{y}$ satisfying these inequalities is feasible within a factor of $\mathcal{H}(d)$, and it satisfies (12) since,

$$\sum_{j=1}^{n}\mathcal{H}(d)w_j x_j \overset{(13)}{\geq} \sum_{j=1}^{n}(\sum_{i=1}^{m} a_{ij}y_i)x_j = \sum_{i=1}^{m}(\sum_{j=1}^{n} a_{ij}x_j)y_i \geq \sum_{i=1}^{m} y_i \overset{(14)}{=} \sum_{j\in C^G} w_j. \tag{15}$$

Let S_j^k be the set S_j with the remaining elements at the beginning of iteration k, and its size, $|S_j^k| = s_j^k$. The dual vector $\mathbf{y}$ that will satisfy (13) and (14) has y_i for the average price paid by the greedy to cover an element i. Whenever a set is selected, its weight is divided evenly among the elements it has newly covered, For each $i \in I$ let $k = k(i)$ be the iteration index in which i is covered for the first time by S_{jk}. Assign $y_i = \frac{w_{jk}}{s_{jk}^k}$.

Let the sets greedy selects in the first k iterations be $\{1, 2, \ldots, k\}$. Since k is the index for which the ratio is minimum

$$\frac{w_{jk}}{s_{jk}^k} \leq \frac{w_j}{s_j^k} \qquad \forall j. \tag{16}$$

Assume that there are t iterations altogether. Then $\sum_{j\in C^G} w_j = \sum_{j=1}^{t} w_j$. Each element $i \in I$ belongs to one set $S_k^k, k = 1,\ldots,t$, so for $i \in S_k^k$, $y_i = \frac{w_{jk}}{s_{jk}^k}$. (14) now follows as,

$$\sum_{i=1}^{m} y_i = \sum_{k=1}^{t}\sum_{i\in S_k^k} y_i = \sum_{k=1}^{t} s_k^k(\frac{w_k}{s_k^k}) = \sum_{k=1}^{t} w_k.$$

To prove (13), observe that $S_j \cap S_k^k = S_j^k \setminus S_j^{k+1}$ and $I = \bigcup_{k=1}^{t} S_k^k$. Hence,

$$\sum_{i=1}^{m} a_{ij}y_i = \sum_{k=1}^{t}\sum_{i\in S_j\cap S_k^k} y_i = \sum_{k=1}^{t}\sum_{i\in S_j^k\setminus S_j^{k+1}} y_i = \sum_{k=1}^{t}\left(s_j^k - s_j^{k+1}\right)\frac{w_k}{s_k^k}.$$

For a given set S_j, let $p + p(j)$ be the largest index such that $s_j^p > 0$, then

$$\sum_{i=1}^{m} a_{ij} y_i = \sum_{k=1}^{p} (s_j^k - s_j^{k+1}) \frac{w_k}{s_k^k} \overset{(16)}{\leq} w_j \sum_{k=1}^{p} \frac{s_j^k - s_j^{k+1}}{s_j^k}.$$

We now use the inequality $\frac{s_j^k - s_j^{k+1}}{s_j^k} \leq \mathcal{H}(s_j^k) - \mathcal{H}(s_j^{k+1})$ to establish,

$$\sum_{i=1}^{m} a_{ij} y_i \leq w_j \sum_{k=1}^{p} \left(\mathcal{H}(s_j^k) - \mathcal{H}(s_j^{k+1}) \right) \leq w_j \mathcal{H}(s_j^1).$$

□

The greedy algorithm is thus an $O(\log n)$-approximation algorithm for any set cover. This matches the recently proved lower bound for approximating the set cover [F95]. Still considerably better results are possible for special cases. Consider for instance the performance of the greedy algorithm on *high coverage* instances of unweighted set cover.

3.1.2 The LP Algorithm for Set Cover

A different approximation algorithm - one which is duality based - was devised for the set cover problem by Hochbaum [Hoc82]. This was motivated by an approximation to the *unweighted* vertex cover problem by Gavril (reported as private communication in [GJ79]). Gavril's algorithm is based on the idea of solving for a maximal (not necessarily maximum) matching, and taking both endpoints of the edges in the matching. The number of edges in the maximal matching $|M|$, is a lower bound on the optimum, $|VC^*|$. This is because a single vertex cannot cover two edges in M. On the other hand, if we pick both endpoints of each matched edge we get a feasible cover, VC^M, as otherwise there would be an edge with both endpoints unmatched, and therefore this edge could be added to the matching M - contradicting its maximality. These two statements lead to the inequalities,

$$|VC^M| = 2|M| \leq 2|VC^*|. \tag{17}$$

Hence, this cover is at most twice the optimal cover.

The extension of this idea to the weighted case and to the set cover problem was inspired by an alternative way of viewing the maximal matching algorithm as a feasible dual solution. To see that, consider the dual of the linear programming relaxation of the unweighted vertex cover problem.

$$\text{(VC-LP)} \qquad \begin{array}{lrcll} \text{Min} & \sum_{j \in V} x_j & & & \\ \text{s.t.} & x_i + x_j & \geq & 1 & \forall\ (i,j) \in E \\ & 0 \leq x_j & \leq & 1 & \forall\ j \in V. \end{array}$$

$$\begin{array}{lrlll} & \text{Max} & \sum_{(i,j)\in E} y_{ij} & & \\ \text{(VC-dual)} & \text{s.t.} & \sum_{(i,j)\in E} y_{ij} & \leq & 1 \quad \forall\, j \in V \\ & & y_{ij} & \geq & 0 \quad \forall\, (i,j) \in E. \end{array}$$

An integer feasible solution to $(VC - dual)$ is a matching in the graph. Consider a feasible solution to the dual, $\bar{\mathbf{y}}$, and let $\bar{x}_j = 1$ whenever the dual constraint is binding, $\sum_{(i,j)\in E} \bar{y}_{ij} = 1$. We show that the solution $\bar{\mathbf{x}}$ is a feasible vertex cover. To that end, we introduce the concept of maximality: a feasible solution to (VC-dual), $\bar{\mathbf{y}}$, is said to be *maximal* if there is no feasible solution $\mathbf{y}$ such that $y_{ij} \geq \bar{y}_{ij}$ and $\sum_{(i,j)\in E} y_{ij} > \sum_{(i,j)\in E} \bar{y}_{ij}$.

Lemma 2. *Let $\bar{\mathbf{y}}$ be a maximal feasible solution to $(VC-dual)$. Then the set $VC = \{i | \sum_{(i,j)\in E} \bar{y}_{ij} = 1\}$ is a feasible solution to (VC).*

Proof. Suppose (i^*, j^*) is not covered. Then this edge can be added to the matching, violating maximality assumption. □

For the weighted vertex cover problem we have the dual LP

$$\begin{array}{lrlll} & \text{Max} & \sum_{(i,j)\in E} y_{ij} & & \\ \text{(VC-dual)} & \text{s.t.} & \sum_{(i,j)\in E} y_{ij} & \leq & w_j \quad \forall\, j \in V \\ & & y_{ij} & \geq & 0 \quad \forall\, (i,j) \in E. \end{array}$$

Lemma 3. *Let $\bar{\mathbf{y}}$ be a maximal feasible solution to $(VC-dual)$. Then the set $VC = \{i | \sum_{(i,j)\in E} \bar{y}_{ij} = w_i\}$ is a feasible solution to (VC).*

Proof. Suppose that VC is not a feasible cover. Then there is an edge $[u, v]$ which is uncovered, i.e., $\sum_{(u,j)\in E} y_{uj} < w_u$ and $\sum_{(v,j)\in E} y_{vj} < w_v$.

Let $\delta = \text{ Min } \{w_u - \sum_{(u,j)\in E} y_{uj}, w_v - \sum_{(v,j)\in E} y_{vj}\}$. Then the vector, $\mathbf{y} = \bar{\mathbf{y}} + \delta \cdot \mathbf{e}_{uv}$ (for $\mathbf{e}_{uv}$ denoting the vector of all zeros except for a 1 in the uv entry), is a feasible solution satisfying, $y_{ij} \geq \bar{y}_{ij}$ and $\sum_{(i,j)\in E} y_{ij} > \sum_{(i,j)\in E} \bar{y}_{ij}$. This contradicts the maximality of $\bar{\mathbf{y}}$. Hence every edge must be covered by VC and VC is therefore a feasible cover. □

Consider now the generalization of this approach for the set cover problem.

Definition 7. *A feasible solution to $(SC - dual)$ $\bar{\mathbf{y}}$ is said to be* maximal *if there is no feasible solution $\mathbf{y}$ such that $y_i \geq \bar{y}_i$ and $\sum_{i=1}^n y_i > \sum_{i=1}^n \bar{y}_i$.*

Algorithm 8 The LP-algorithm [Hochbaum]

---

1. Find a maximal dual feasible solution for (SC), $\bar{\mathbf{y}}$.
2. Output the cover $C^H = \{j | \sum_{i=1}^n a_{ij}\bar{y}_i = w_j\}$.

Lemma 4. *C^H is a feasible solution to (SC).*

Proof. The proof is an obvious extension of Lemma 2: Consider an element q that is not covered. Let $\delta = \text{Min}_{j|q\in S_j}\{w_j - \sum_{j=1}^{n} a_{ij}\bar{y}_i\} > 0$. Then the vector, $\mathbf{y} = \bar{\mathbf{y}} + \delta \cdot \mathbf{e}_q$, is a feasible solution contradicting the maximality of $\bar{\mathbf{y}}$. □

Let C^* be the optimal cover. Define for any set cover C, $w(C) = \sum_{j\in C} w_j$. The following lemma shows that the the LP- algorithm is a $\max_i\{\sum_j a_{ij}\}$- approximation algorithm due to the dual constraint being binding for every $j \in C^H$.

Lemma 5. $w(C^H) \leq \max_i\{\sum_j a_{ij}\}w(C^*)$.

Proof. First, $w(C^H) = \sum_{j\in C^H} w_j = \sum_{j\in C^H}(\sum_{i=1}^{m} a_{ij}y_i)$. Using the weak duality theorem we get that for any solution to the linear programming relaxation, and in particular for the optimal solution $\mathbf{x}^*$,

$$\begin{aligned}\sum_{j\in C^H}(\sum_{i=1}^{m} a_{ij}y_i) &\leq \max_i\{\sum_{j\in C^H} a_{ij}\}\sum_{i=1}^{m} y_i \leq \max_i\{\sum_{j\in C^H} a_{ij}\}\sum_{j=1}^{n} w_j x_j^* \\ &\leq \max_i\{\sum_{j\in C^H} a_{ij}\}w(C^*).\end{aligned}$$

Now the value of the optimal solution to the linear programming relaxation is a lower bound to the optimal integer solution. It follows that,
$w(C^H) \leq \max_i\{\sum_{j\in C^H} a_{ij}\}w(C^*)$. □

This proves a slightly stronger approximation factor as we can consider the row sums restricted only to those sets in the cover (alternatively the row sums are calculated in the submatrix of the columns in the cover). For instance, if the cover C^H has only one set covering each element, then it is optimal. Let the maximum number of sets covering an element, $\max_i\{\sum_j a_{ij}\}$ be denoted by f.

An immediate corollary of the f-approximation is that the LP-algorithm is a 2-approximation algorithm for the (weighted) vertex cover problem.

Corollary 1. *The LP-algorithm is a 2-approximation algorithm for the vertex cover problem.*

Proof. (VC) is a special case of (SC) with each element - an edge - belonging to precisely two "sets" representing its endpoints. Hence, $\sum_j a_{ej} = 2$ for all edges $e \in E$. □

The implementation of the LP-algorithm proposed in [Hoc82] used the optimal dual solution as a maximal feasible solution. Still, the role of the dual solution is only to aid the analysis of the algorithm. It need not be generated explicitly by the algorithm:

Algorithm 9 The rounding algorithm II [Hochbaum]

1. Solve optimally the linear programming relaxation of (SC). Let an optimal solution be $\{x_j^*\}$
2. Output the cover $C^H = \{j|x_j^* > 0\}$. Equivalently, set $x_j^H = \left\lceil x_j^* \right\rceil$.

The rounding algorithm is indeed a special case of the LP-algorithm as $x_j^* > 0$ implies that the corresponding dual constraint is binding by complementary slackness optimality conditions, $\sum_{i=1}^{m} a_{ij} y_i = w_j$.

A minor variation of the rounding algorithm is still a f-approximation algorithm. We replace Step 2 by:

Step 2' : Output the cover $C^H = \{j | x_j^* \geq \frac{1}{f}\}$.

The feasibility of this cover is obvious, as in any fractional solution $\mathbf{x}$ corresponding to a cover C, $\sum_{j=1}^{n} x_j \geq 1$, and there are at most f positive entries per such inequality. So at least one must be at least as large as the average $\frac{1}{f}$. This rounding algorithm (rounding II) will always produce a cover no larger than the rounding algorithm. It does not offer any advantage though in terms of worst case analysis. Moreover, for any cover produced by an approximation algorithm, it is easy to prune it of unnecessary extra sets, and leave a "prime" cover, which is a *minimal* cover. Although a prime cover can only be a better solution, this approach has not provided *guaranteed* tighter approximation factors.

3.1.3 The Feasible Dual Approach

Solving the linear programming relaxation of set cover can be done in polynomial time. Yet, much more efficient algorithms are possible that find a feasible and maximal dual solution. The advantage of such algorithms is in the improved complexity. The approximation ratios derived are the same as for the dual optimal solution.

Bar-Yehuda and Even [BYE81] devised an efficient algorithm for identifying a maximal feasible dual solution to be used in the LP-algorithm. The idea is to identify a feasible primal constraint and then to increase its dual variable till at least one of the dual constraints becomes binding.

As before, the derivation of the dual solution is implicit and exists only in order to analyze the approximation factor. This dual information is placed in square brackets in the description of the algorithm to stress that it is not an integral part of the procedure.

Algorithm 10 The Dual-feasible Algorithm [Bar-Yehuda and Even]

1. Set $C = \emptyset$; $I = \{1, \ldots, m\}$; $[\mathbf{y} = 0]$.
2. Let $i \in I$. Let $w_{j(i)} = \min_{a_{ij}=1} w_j$. $[y_i = w_{j(i)}]$; $C \leftarrow C \cup \{j(i)\}$.
3. {update} For all j such that $a_{ij} = 1$, $w_j \leftarrow w_j - w_{j(i)}$, $I \leftarrow I \setminus S_j(i)$.
4. If $I = \emptyset$, stop and output cover C. Else go to Step 1.

Throughout this procedure the updated w_js remain nonnegative. The weights w_j are in fact the reduced costs corresponding to the solution $\mathbf{y}$, and at each iteration they quantify the amount of slack in the dual constraint. In this sense, the algorithm is dual-feasible - throughout the procedure

it maintains the feasibility of the dual vector $\mathbf{y}$. For any set added to the cover, the updated value of the reduced cost is 0, i.e. the corresponding dual constraint is binding. The resulting dual vector is maximal since each element belongs to some set in the cover, and therefore to some set with a binding dual constraint. Hence there is no vector that is larger or equal to $\mathbf{y}$ in all its components which is feasible, unless it is equal to $\mathbf{y}$.

Lemma 5 applies to the cover delivered by the dual-feasible I algorithm, namely, dual-feasible is a f-approximation algorithm to the set cover problem. The complexity of the dual-feasible algorithm is $O(mn)$, which is linear in the size of the input matrix, and hence a considerable improvement to the complexity of the respective linear program.

As in corollary 1, this algorithm is a 2-approximation to the vertex cover problem.

3.2 Approximating the Multicover Problem

In this section we present variations of the LP-algorithm, the rounding algorithm and the dual-feasible algorithm that also work for the multicover problem (MC). The description of the dual-feasible algorithm and its analysis are from [HH86]. Consider the linear programming relaxation of the multicover problem and its dual:

$$\text{(MCR)} \qquad \begin{array}{rlll} \text{MC}^* = & \text{Min} & \sum_{j=1}^n w_j x_j & \\ & \text{s.t.} & \sum_{j=1}^n a_{ij} x_j \geq b_i & i = 1, \ldots, m \\ & & 0 \leq x_j \leq 1 & j = 1, \ldots, n \end{array}$$

MC* obviously bounds from below the optimal value to the multicover problem. The dual to the linear programming relaxation above reads:

$$\text{MC-dual} \qquad \begin{array}{rlll} \text{Max} & \sum_{i=1}^m b_i y_i - \sum_{j=1}^n v_j & & \\ \text{s.t.} & \sum_{i=1}^m a_{ij} y_i - v_j \leq w_j & & j = 1, \ldots, n \\ & y_i, v_j \geq 0 & & i = 1, 2, \ldots, m,\ j = 1, \ldots, n \end{array}$$

Here, a feasible dual solution $(\bar{\mathbf{y}}, \bar{\mathbf{v}})$ is called *maximal* if it satisfies:
(i) There is no other dual feasible solution (y, v) such that $y_i \geq \bar{y}_i, v_i \geq \bar{v}_i$ and $\sum_{i=1}^m b_i y_i - \sum_{j=1}^n v_j > \sum_{i=1}^m b_i \bar{y}_i - \sum_{j=1}^n \bar{v}_j$.
(ii) $\bar{v}_j = 0$ whenever $\sum_{i=1}^m a_{ij} y_i < w_j$.
(iii) $\sum_{i=1}^m \bar{y}_i \leq \sum_{i=1}^m b_i \bar{y}_i - \sum_{j=1}^n \bar{v}_j$

With this definition of maximality, the following LP algorithm works as a f-approximation algorithm, where $f = \max_{\text{i}}\{\sum_{\text{j}} \text{a}_{\text{ij}}\}$.

Lemma 6. *The LP-algorithm is a f-approximation algorithm for (MC).*

Proof. First, we establish that C^H is a feasible multicover. Consider an uncovered element (row) q. Notice that for the problem to be feasible, a necessary condition is that every row i has at least b_i sets covering it. Let $\delta = \text{ Min }_{j|q \in S_j}\{\delta_j = w_j - \sum_{j=1}^n a_{ij}\bar{y}_i \text{ and } \delta_{\text{j}} > 0\}$. Since there must be

Algorithm 11 The LP-algorithm for multicover

1. Find a maximal dual feasible solution for (MC), $\bar{\mathbf{y}}, \bar{\mathbf{v}}$.
2. Output the cover $C^H = \{j | \sum_{i=1}^{n} a_{ij}\bar{y}_i - \bar{v}_j = w_j\}$.

at least b_q sets that q belongs to, and at most $b_q - 1$ of them are in C^H, it follows that δ is well defined. Now set, $\mathbf{y} = \bar{\mathbf{y}} + \delta \cdot \mathbf{e}_q$ and $\mathbf{v} = \bar{\mathbf{v}} + \sum_{\mathrm{j}\in\mathrm{c^H}|\mathrm{q}\in\mathrm{S_j}} \delta \mathbf{e}_\mathrm{j}$.

It is easy to verify that the vector $(\mathbf{y}, \mathbf{v})$ is a feasible solution, thus contradicting property (i) of the maximality of $(\bar{\mathbf{y}}, \bar{\mathbf{v}})$. This is because the first term has increased by $b_q\delta$ at least whereas the second term has increased by $(b_q - 1)\delta$ at most, thus contributing to a net increase of the objective function by at least δ.

Now $(\bar{\mathbf{y}}, \bar{\mathbf{v}})$ is a feasible dual solution, hence the weak duality theorem applies:

$$\sum_{i\in I} b_i\bar{y}_i \leq \min\left(\sum_{j\in J} w_j x_j\right) + \sum_{j\in J}\bar{v_j} \leq \mathrm{MC}^* + \sum_{j\in J}\bar{v_j}.$$

From that and property (iii),

$$\sum_{i\in I} y_i \leq \mathrm{MC}^*. \tag{18}$$

Using the construction of C^H:

$$\begin{aligned}\sum_{j\in \mathrm{C^H}} w_j + \sum_{j\in \mathrm{C^H}} v_j &= \sum_{j\in \mathrm{C^H}}\sum_{i\in I} a_{ij}\bar{y}_i = \sum_{i\in I}\left(\sum_{j\in \mathrm{C^H}} a_{ij}\right)\bar{y}_i \\ &\leq \left(\max_{i\in I}\sum_{j\in \mathrm{C^H}} a_{ij}\right)\cdot\sum_{i\in I}\bar{y_i} \leq f\cdot \mathrm{MC}^*.\end{aligned}$$

The last inequality follows from 18 and the definition of f. Recalling that the $\bar{v_j}$'s are nonnegative and that $\mathrm{MC}^* \leq w(C^*)$ where $w(C^*)$ is the value of the optimal integer solution, we derive the stated result. □

A rounding algorithm is also a f-approximation algorithm. It offers the advantage of a smaller weight multicover.

Algorithm 12 The Rounding Algorithm [Hall and Hochbaum]

1. Solve the linear programming relaxation of (MC) optimally. Let an optimal solution be $\{x_j^*\}$
2. Output the cover $C^H = \{j | x_j^* \geq \frac{1}{f}\}$.

The feasibility of this cover is obvious, as in any fractional solution $\mathbf{x}$ corresponding to a cover C, $\sum_{j=1}^{n} a_{ij}x_j \geq b_i$. So at least b_i entries must be at least as large as the average $\frac{1}{f}$.

Next we present a dual-feasible algorithm that delivers a f-approximate solution to the multicovering problem. This algorithm has a better complexity than the one requiring to solve the relaxation optimally.

The input to the algorithm is the matrix A and the vectors $\mathbf{b}$ and $\mathbf{w}$. The output is COVER – the indices of the sets selected and a vector $(y_i, i = 1, \ldots, m; v_j, j = 1, \ldots, n)$ that will later be proved to constitute a feasible dual solution.

Algorithm 13 The Dual-feasible MC Algorithm

1. (initialize) $v_j = 0, j \in J$. $y_j = 0, i \in J$. COVER $= \emptyset$.
2. Let $i \in I$. Let $w_k = \min\{w_j | j \in J\text{-COVER and } a_{ij} = 1\}$. ($k$ is the minimum cost column covering row i.) If no such minimum exists, stop - the problem is infeasible.
3. Set $y_i \leftarrow y_i + w_k$. COVER $\leftarrow$ COVER$\cup\{k\}$. For all $j \in J$ such that $a_{ij} = 1$ set $w_j \leftarrow w_j - w_k$. If $w_j < 0$ then $v_j \leftarrow v_j - w_j$ and $w_j \leftarrow 0$.
4. Set $b_i \leftarrow b_i - 1, i = 1, \ldots, m$. For all i' such that $b_{i'} = 0, I \leftarrow I - \{i'\}$. If $I = \emptyset$ stop, else go to Step 1.

The algorithm repeats Step 1 at most n times, since if following n iterations the set I is not yet empty, then there is no feasible solution. This could occur for instance if the amount of required coverage exceeds the number of covering sets, n. At each iteration there are at most $(\max\{n, m\})$ operations resulting in a total complexity of $O(\max\{n, m\} \cdot n)$.

The output of the algorithm is the set COVER of indices of the selected sets that multicover all elements, or a statement that the problem is infeasible. We shall now prove that dual vector derived is maximal and satisfies the three properties.

In the proofs of the facts that follow we shall use the notation $S_j = \{i | a_{ij} = 1\}$, i.e., S_j denotes the j^{th} set.

- For each $j \in$ COVER, $w_j = \sum_{i \in S_j} y_i - v_j$.

  Proof. By construction $w_j + v_j = \sum_{i \in S_j} y_i$. □

- $\sum_{i \in S_j} y_i \leq w_j + v_j \quad \forall j \in J$.

  Proof. This follows from Fact 1 and from Step 1 since the minimum cost column is always selected. □

- The output of the algorithm $(\mathbf{y}, \mathbf{v}) = \left(\{y_i\}_{i=1}^{m}, \{v_j\}_{j=1}^{n}\right)$ is a feasible solution to the dual problem.

Proof. First y_i, v_j are always nonnegative. This follows since y_i is equal to a cost w_j during one of the iterations and the $w_j's$ are always maintained as nonnegative numbers. Each v_j is a sum of positive numbers and hence nonnegative as well. Finally, Fact 2 establishes the feasibility with respect to the constraints. □

- $v_j = 0$ for all $j \in J$-COVER. This fact follows from the selection made at Step 1 of the algorithm.

The following lemma is useful in the proof of property (iii).

Lemma 7. $\sum_{j\in\text{COVER}} v_j \leq \sum_{i\in I}(b_i - 1)y_i$ (note that $b_i \geq 1, i \in I$).

Proof. The values of the left-hand side and the right hand side of the inequality vary during the algorithm's iteration.

We let the value of v_j and y_i after iteration t be denoted by $v_j^{(t)}$ and $y_i^{(t)}$ respectively. Let T be the number of iterations. We shall prove by induction on t that

$$\sum_{j\in\text{COVER}} v_j^{(t)} \leq \sum_{i\in I}(b_i - 1)y_i^{(t)}, \quad i = 1, \ldots, T.$$

For $t = 1$ the left-hand side is zero and the right-hand side nonnegative. We shall assume by induction that the inequality holds for $t = 1, \ldots, l-1$ and prove for l.

Let $M = w_k$ be the minimum column cost selected at iteration l; then the right-hand side increases by $(b_i - 1) \cdot M$ with y_i increasing by M. Each $v_j^{(l)}$ might be increased by at most M compared to the previous iteration but for no more than $(b_i - 1)$ columns. This is the case since a cost of a column could become negative (thus triggering the increase in $v_j^{(l)}$), only if it is already b_i columns or more covering row i in COVER, then this row would have been removed from the set I, and thus could not be considered at iteration l. Therefore the inequality is preserved at each iteration and hence the desired result. □

From the proof of the theorem it follows that the heuristic solution value does not in fact exceed $(\max_{i\in I} \sum_{i\in\text{COVER}} a_{ij})$ times the value of the optimum. This quantity could be much smaller than f.

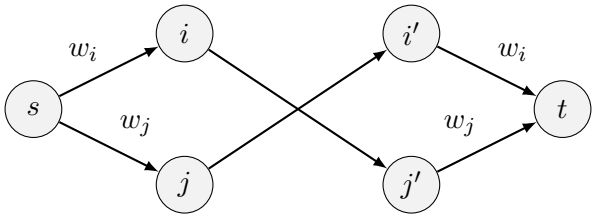
The derivation of the dual vector as a by-product of the heuristic also provides a certificate of optimality for the selected set COVER (or any other solution) satisfying $\sum_{i\in\text{COVER}} w_j = \sum_{i=1}^{n} b_i y_i - \sum_{j=1}^{n} v_j$.

3.3 A Persistency-Derived Approximation for Vertex Cover (VC) and Independent Sets (IS)

3.3.1 Preprocessing the graph

Let $G = (V,\ E)$ be a graph with node weight w_i for all $i \in V$.

1. Double all nodes such that for all $i \in V$ we add $i' \in V'$
2. For $(i,\ j) \in E$, add an edge between i and j' and between i' and j' with infinite weight
3. Add a source node s and a sink node t. For all node $i \in V$ add weighted edge (s, i) and (i', t) of weight w_i
4. Solve the minimum s-t cut problem on this graph. Let S be the source set and T the sink set. Let us define $\overline{C} = (V \cap T) \cup (V' \cap S)$



Let X^* be such that:

$$\begin{aligned} X_j^* &= 1 && \text{if } j \in \overline{C} \text{ and } j' \in \overline{C} \\ X_j^* &= \frac{1}{2} && \text{if } j \in \overline{C} \text{ or } j' \in \overline{C} \text{ but not both} \\ X_j^* &= 0 && \text{otherwise} \end{aligned}$$

X^* is a half integral optimal solution for the LP relaxation of (VC) (see monotonization)

Let us define three sets of nodes:

$$\begin{aligned} V_1 &= \{j | X_j^* = 1\} \\ V_{\frac{1}{2}} &= \{j | X_j^* = \frac{1}{2}\} \\ V_0 &= \{j | X_j^* = 0\} \end{aligned}$$

Let $H = (V_{\frac{1}{2}}, E_{\frac{1}{2}})$ be the subgraph induced by $V_{\frac{1}{2}}$.let C^A be the solution of solving (VC) on H with heuristic A. If C^* is a minimum weight vertex cover on $G = (V, E)$ we can prove:

$$\begin{aligned} \frac{w(V_1 \cup C^A)}{w(C^*)} &\leq \frac{w(V_1) + w(C^A)}{w(V_1) + \frac{1}{2}w(v_{\frac{1}{2}})} \\ &\leq \frac{w(V_1) + w(V_{\frac{1}{2}})}{w(V_1) + \frac{1}{2}w(v_{\frac{1}{2}})} \\ &\leq 2 \end{aligned}$$

Let IS^A be the solution of solving (IS) on H with heuristic A. If IS^* is a minimum weight

vertex cover on $G = (V, E)$ we can prove:

$$\frac{w(V_0 \cup IS^A)}{w(IS^*)} \leq \frac{w(V_0) + w(IS^A)}{w(V_0) + \frac{1}{2}w(v_{\frac{1}{2}})}$$
$$\leq 2\frac{w(IS^H)}{V_{\frac{1}{2}}}$$

3.3.2 Results on unweighted graphs

Let δ be the average degree on an unweighted graph $G = (V, E)$, we have a $\frac{2}{\delta+1}$-approximation for (IS)

Algorithm 14 Greedy

- Remove a node of smallest indegree
- At iteration i, chose V_i of minimum degree, delete V_i and all this neighbors from the graph, deleting $d_i + 1$ node from G
  The sum of the degrees over G is reduced by at least $d_i(d_i + 1)$

Let $g = |S|$ be the number of iteration of Greedy,

$$\sum_{i=1}^{g} d_i(d_i + 1) \leq n * \delta$$

$$\sum_{i=1}^{g} d_i + 1 = n$$

By summation Cauchy Schwarz gives $n(\delta + 1) \geq \sum_{i=1}^{g}(d_i + 1)^2 \geq \frac{n^2}{g}$. Therefore $g \geq \frac{n}{\delta+1}$

Theorem 8. *Using preprocessing (solve LP-relaxation)for any graph H with average degree δ, it takes $O(\delta n^{3/2})$ steps to find an IS of size greater or equal than $\frac{2}{\delta+1}$ times the optimal solution.*

Proof. To find IS, we solve a minimum cut problem on a simple unit capacity graph, Dinic's algorithm finds a solutipon in $O(m\sqrt{n})$, since m is $O(n\delta)$, the complexity is $O(\delta n^{3/2})$.

The size of ths IS delivered b preprocessing and greedy on $V_{\frac{1}{2}}$ is $|V_0| + \frac{|V_{\frac{1}{2}}|}{\delta_H+1}$ □

We can state the following facts:

1. Since G is connected, $\delta \geq \frac{\delta_H|V_{\frac{1}{2}}|+|V_0|+|V_1|}{n}$
2. $|V_0| \geq |V_1|$

Proof. Suppose not, the objective value at optimal is: $|V_0| + \frac{1}{2}|V_{\frac{1}{2}}|$. Note that $x_i = \frac{1}{2}$ is feasible

$$\frac{1}{2}(|V_0| + |V_1| + |V_{\frac{1}{2}}) > |V_0| + \frac{1}{2}|V_{\frac{1}{2}}$$

Our first solution is not optimal. Contradiction □

Proof of approximation bound

By fact 1 in class, we know that

$$2\frac{1}{\frac{\delta_H|V_{\frac{1}{2}}|+|V_1|+|V_0|}{|V_{\frac{1}{2}}|+|V_1|+|V_0|}+1} \geq \frac{2}{\delta+1}.$$

Now want to show

$$\frac{|V_0| + \frac{|V_{\frac{1}{2}}|}{\delta_H+1}}{|V_0| + \frac{1}{2}|V_{\frac{1}{2}}|} \geq 2\frac{1}{\frac{\delta_H|V_{\frac{1}{2}}|+|V_1|+|V_0|}{|V_{\frac{1}{2}}|+|V_1|+|V_0|}+1},$$

which is equivalent to

$$\frac{|V_0| + \frac{|V_{\frac{1}{2}}|}{\delta_H+1}}{|V_0| + \frac{1}{2}|V_{\frac{1}{2}}|} \geq 2\frac{|V_{\frac{1}{2}}| + |V_1| + |V_0|}{(\delta_H+1)|V_{\frac{1}{2}}| + 2|V_1| + 2|V_0|}.$$

That is,

$$\begin{aligned}|V_0|\left((\delta_H+1)|V_{\frac{1}{2}}| + 2|V_1| + 2|V_0|\right) + |V_{\frac{1}{2}}|\left(|V_{\frac{1}{2}}| + \frac{2|V_1|}{\delta_h+1} + \frac{2|V_0|}{\delta_h+1}\right)\\ \geq |V_0|\left(2|V_{\frac{1}{2}}| + 2|V_1| + 2|V_0|\right) + |V_{\frac{1}{2}}|\left(|V_{\frac{1}{2}}| + |V_1| + |V_0|\right).\end{aligned}$$

After canceling the terms, the inequality becomes

$$(\delta_h - 1)|V_0||V_{\frac{1}{2}}| - \frac{\delta_H - 1}{\delta_H + 1}|V_{\frac{1}{2}}||V_1| - \frac{\delta_H - 1}{\delta_H + 1}|V_{\frac{1}{2}}||V_0| \geq 0.$$

Multiplying both side by $(\delta_H + 1)$:

$$(\delta_h^2 - 1)|V_0||V_{\frac{1}{2}}| - (\delta_H - 1)|V_{\frac{1}{2}}|(|V_1| + |V_0|) \geq 0.$$

By fact 2: $|V_0| \geq |V_1|$, we can see that

$$\begin{aligned}(\delta_h^2-1)|V_0||V_{\frac{1}{2}}|-(\delta_H-1)|V_{\frac{1}{2}}|(|V_1|+|V_0|) &\geq (\delta_h^2-1)|V_0||V_{\frac{1}{2}}|+2(1-\delta_H)|V_{\frac{1}{2}}||V_0| \\ &= (\delta_h^2-2\delta_H+1)|V_{\frac{1}{2}}||V_0| \\ &\geq 0.\end{aligned}$$

Therefore, the inequality is true.

3.4 Results for easily colorable graphs

An easily colorable graph is a graph $G=(V,E)$ where it is possible to find in a polynmial time a k-coloration $(V_1,V_2,\ldots,V_k)$ partitioning V such that no two adjacent nodes are contained in the same subset subset V_i.

Let V_k be such that : $w(V_k)=\max\limits_{i\in\{1\ldots k\}}\{w(V_i)\}$. Then, $V_1\cup V_2\cup\cdots\cup V_{k-1}$ is a vertex cover of weight less than or equal to $\frac{k-1}{k}w(V)$

Let C^A be this solution, we have the following result:

$$\frac{w(C^A)}{C^*_{\frac{1}{2}}}\leq\frac{\frac{k-1}{k}w(V_{\frac{1}{2}})}{\frac{1}{2}w(V_{\frac{1}{2}})}\leq 2-\frac{2}{k}$$

Graph of minimum degree d

For $G=(V,E)$ a graph and $D(G)$ the largest d such that G contains a subgraph in which all vertices have a degree larger than d, it has been proven that every g can be colored in $D(G)+1$ colors. Let us consider the following algorithm:

Algorithm 15 NeighborLimit

0. $d=0$ $L=\emptyset$
1. if $G=\emptyset$
   Then STOP
   Else pick $v\in V$ of smallest degree
2. $d\leftarrow max(d,deg(v))$
3. Remove v and all edges incident to v of G
   Append v to L

This algorithm outputs a sequence of $(v_1,\ldots,v_n)$ such that v_i has been removed at the i^{th} iteration and has at most d neighbors in $v_{i+1},\ldots,v_n$.

Therefore by working backward, we can assign colors to each vertex with $D(G)+1$ colors. Starting from v_n, assign to each node the smallest index color possible. as v_i has at most d neighbors colored already, there exist a feasible color for v_i

Theorem 9 (Brook's Theorem)**.** *If $G=(V,E)$ is a connected graph of max degree Δ, $\Delta \geq 3$ and G is not a complete graph, then there exists a Δ-coloration of G*

There exists a constructive proof, an algorithm in $O(\Delta n)$ that finds the Δ-coloration

$\rightarrow$ $2-\frac{2}{\Delta}$-approximation for (VC)

$\rightarrow$ $\frac{2}{\Delta}$-approximation for (IS)

Planar Graphs

As another instance, planar graphs are 4-colorable (this is not easily proven, but a proof for the 5-colorability of planar graphs is given in a subsequent section). Therefore we get:

$\rightarrow$ 1.5-approximation for (VC)

$\rightarrow$ $\frac{1}{2}$-approximation for (IS)

3.5 A Shifting Strategy for Covering and Packing

We are given a set of n points in d dimensions. We would like to cover them with hyper-spheres that have diameter D. We are seeking the covering which uses the fewest hyper-spheres.

Theorem 10. *There is a polynomial approximation scheme H^d such that, for every $\ell \geq 1$, H^d_ℓ delivers a cover of the given n points in $\mathcal{R}^d$ by d-dimensional balls of diameter D in time $o(\ell^d(\ell\sqrt{d})^d(2n)^{d(\ell\sqrt{d})^d+1})$, with approx factor $(1+\frac{1}{\ell})^d$.*

We will prove this result in a series of steps. We will show that if we divide up a dimension into strips of width ℓD and have an algorithm that solves the problem approximately on each strip with factor α, then the solution we get by combining the solutions on the strips makes the approximation factor no worse than $(1+\frac{1}{\ell})\alpha$. This recursion, coupled with an appropriate base case, is sufficient for the theorem.

Lemma 8. *In a square of width ℓD, we can solve to optimality in time $o(n^{4\ell^2})$.*

Proof. We use brute force. With $(\ell\sqrt{2})^2 = 2\ell^2$ disks we can cover the side length ℓd completely. Two points determine a disk, so there are at most $2\binom{n}{2}$ disk positions to worry about. Thus, we need to check at most $\binom{n^2}{2\ell^2}$ subsets. □

Now assume we have an algorithm $\mathcal{A}$ which solves the problem approximately on strips of width ℓD. Assume that $\mathcal{A}$ has approximation ratio α. We start by dividing up the box into strips of

length ℓD. We call this partition S_0 and calculate the result of $\mathcal{A}$ on strips in this partition. Then we *shift* the strips over by D and repeat. We call this partition S_1. We continue until we have done this ℓ times. Since we have shifted ℓ times, we have the same strips as S_0. We will consider the best solution among

$$\{\mathcal{A}(S_0), \mathcal{A}(S_1), \ldots, \mathcal{A}(S_{\ell-1})\}.$$

Lemma 9. *The best solution among* $\{\mathcal{A}(S_0), \mathcal{A}(S_1), \ldots, \mathcal{A}(S_{\ell-1})\}$ *will have approximation ratio* $(1 + \frac{1}{\ell})\alpha$.

Proof. Fix an arbitrary partition S_i. Let $|\text{OPT}_j|$ be the number of disks of OPT that cover points in the j^{th} strip. Let $|\text{OPT}^{(j)}|$ be the optimal number of disks required to cover the j^{th} strip. Naturally, $|\text{OPT}^{(j)}| \leq |\text{OPT}_j|$. Furthermore, we have

$$\sum_j |\text{OPT}_j| \leq |\text{OPT}| + |\text{DBL}_i|.$$

Here, $|\text{DBL}_i|$ is the set of disks in OPT which are double counted because they cross between two strips. Since the balls have diameter D and we are shifting by D every time, it must be the case that every disk is double-counted at most once, so

$$\sum_i |\text{DBL}_i| \leq |OPT|.$$

Thus, by pigeonhole principle, there exists i such that $|\text{DBL}_i| \leq \frac{1}{\ell}|\text{OPT}|$. Thus, for this i,

$$\sum_j |\text{OPT}_j| \leq (1 + \frac{1}{\ell})|\text{OPT}|.$$

If we have an α approximation for each strip, then

$$\mathcal{A}(S_i) \leq \alpha \sum_j |\text{OPT}_j| \leq \alpha(1 + \frac{1}{\ell})|\text{OPT}|.$$

□

This lemma implies the theorem directly since we can solve recursively by breaking down the problem in each dimension.

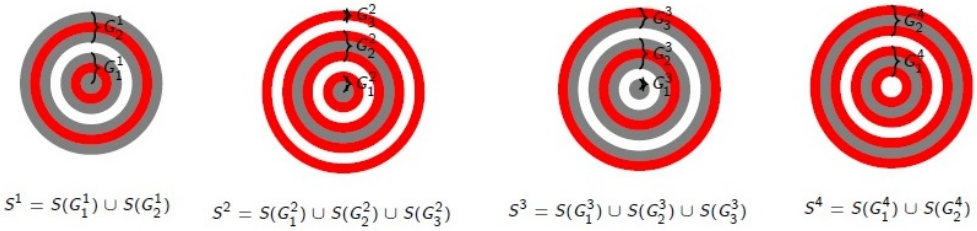
3.6 Approximation Algorithms for NP-Complete Problems on Planar Graphs

This section comes from the presentation "Approximation Algorithms for NP-Complete Problems on Planar Graphs" by Marius Knabben. Essentially, we apply the idea of shifting to planar graphs. The "shifting" occurs between levels of outerplanarity.

Definition 8. *A graph is p-outerplanar if it can be drawn in the plane without crossing of edges and by removing all vertices that belong to the outer face the resulting graph is* $p-1$*-outerplanar.*

Algorithm 16 Shifting for Planar Graphs

Input: The graph G and k
for $i = 1, \ldots, k+1$ **do**
  G^i is constructed by deleting all the level$(i, (k+1)+i, 2(k+1)+i, \ldots)$ nodes of G.
  G^i is composed of multiple components G^i_1 to G^i_m which are at most k outerplanar.
  for $j = 1, \ldots, m$ **do**
    S^i_j = Max Independent Set(G^i_j).
  end for
  $S^i = \cup_j S^i_j$.
end for
$S = \max_i S^i$
Output: S.



- We now have four approximate solutions (S^1 to S^4).
- In at least one of those four subgraphs at most 1/4 of all nodes of G are deleted.
- This subgraph's solution is at least 3/4 optimal.
- Reintroducing k: the solution is $k/(k+1)$ optimal.

This idea applies to many problems. For example:

- Minimum Vertex Cover
- Minimum Dominating Set
- Minimum Edge Dominating Set

3.7 A FPTAS for Knapsack Problem

Consider the Knapsack problem:

$$\text{(KP)} \qquad \begin{array}{rrcl} \max & \sum_{j=1}^n p_j x_j & & \\ \text{s.t.} & \sum_{j=1}^n a_j x_j & \leq & B \\ & x_j & \in & \mathbb{B}, \quad j = 1, \ldots, n \end{array}$$

The Knapsack Problem is solved with Dynamic Programming in time $o(nB)$.

Let $p_j(k) = \lfloor \frac{p_j}{k} \rfloor$. The Scaled Knapsack Problem is as follows:

$$\text{(SKP)} \qquad \begin{array}{rlll} \max & \sum_{j=1}^{n} p_j(k) x_j & & \\ \text{s.t.} & \sum_{j=1}^{n} a_j x_j & \leq & B \\ & x_j & \in & \mathbb{B}, \quad j = 1, \ldots, n \end{array}$$

Let $S(k)$ be the optimal solution to the scaled Knapsack problem. Let S^* be the optimal solution to the original knapsack problem.

$$\begin{aligned} \sum_{j \in S(k)} p_j &\geq \sum_{j \in S(k)} k \left\lfloor \frac{p_j}{k} \right\rfloor \\ &\geq \sum_{j \in S^*} k \left\lfloor \frac{p_j}{k} \right\rfloor \\ &\geq \sum_{j \in S^*} (p_j - k) \\ &\geq \sum_{j \in S^*} p_j - k|S^*| \end{aligned}$$

The approximation ratio is

$$\epsilon \leq \frac{k|S^*|}{P^*} \leq \frac{kn}{P_{\max}}.$$

The running time is

$$o(n^2 \frac{P_{\max}}{k}) = o(n^3 \frac{1}{\epsilon}).$$

4 Planar Graphs

4.1 Heawood's 5-color theorem (1890)

Proposition 3. *If a graph is k-critical, then the minimum degree node in the graph has degree* $\geq k - 1$.

Proposition 4. *In a planar graph, there exists a node of degree* ≤ 5.

Theorem 11. *Every planar graph is 5-colorable.*

Proof. The proof is by induction on n, the number of nodes in graph G. If $n \leq 5$ then the graph is 5-colorable. Now, assume all graphs with n nodes are 5-colorable and let $G(V, E)$ be a graph with $n + 1$ nodes.

By the induction hypothesis, $G - \{v\}$ is 5-colorable for any $v \in V$. Assuming, by way of contradiction, that G is not 5-colorable, we can say that it is 6-critical, so that the minimum degree node has degree ≥ 5. However, since G is also planar there exists a node with degree ≤ 5. Then the minimum degree node $v \in G$ has degree exactly 5.

Since $G - \{v\}$ is 5-colorable, there exists a partitioning of $V - \{v\}$ into 5 sets of different colors, $(V_1, V_2, V_3, V_4, V_5)$. G is not 5-colorable, v must be adjacent to at least one vertex in each set. Since v has degree of 5, it is adjacent to exactly one vertex in each set. G is a planar graph, therefore we can label neighbors of v in clockwise order as v_1, v_2, v_3, v_4 and v_5, where $v_i \in V_i$.

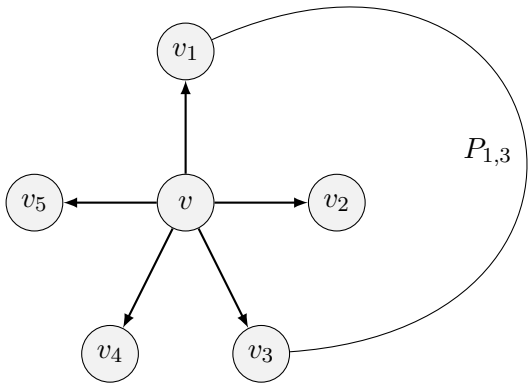


Figure 12: There exists a path from v_1 to v_3.

Let $G_{i,j}$ be the the subgraph of G induced by nodes in $V_i \cup V_j$. v_i and v_j must be in the same connected component of $G_{i,j}$, since otherwise we may flip the colors in one of the components to assign v_i and v_j the same color and thus contradict the statement that v must be adjacent to exactly one node of each color. Thus, there exists a path $P_{i,j}$ from v_i to v_j in $G_{i,j}$, for all $i, j \in \{1, 2, 3, 4, 5\}, i \neq j$.

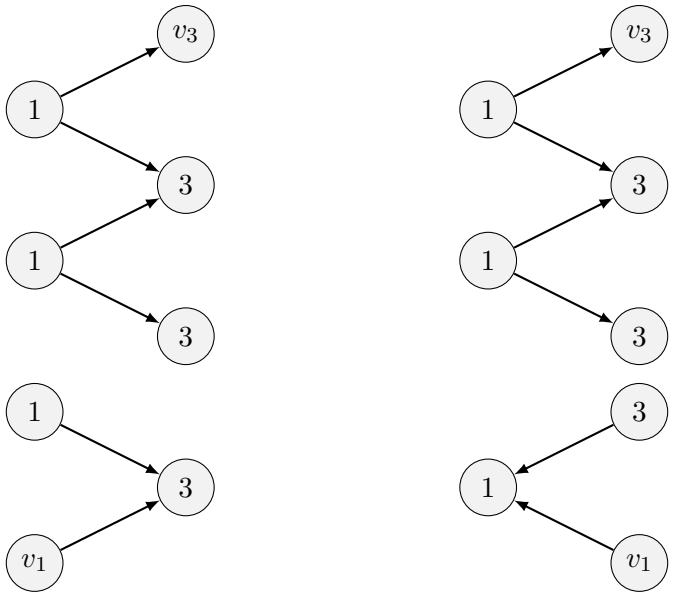


Figure 13: If v_1 and v_3 are in two different connected components of $G_{1,3}$ we can flip colors in one component to assign the same color to v_1 and v_3.

In this spirit, consider $G_{1,3}$ and $G_{2,4}$. As we have numbered the nodes in a clockwise fashion, v_2 is separated from v_4 in G by the cycle $\{v, P_{1,3}, v\}$. Specifically, the cycle will encircle v_2, so, due to the planarity of G, there cannot exist a path from v_2 to v_4 in $G_{2,4}$. Therefore v_2 and v_4 must be in two different connected components, leading in a contradiction. Thus, G is 5-colorable.

□

4.2 Geometric Dual of Planar Graph

The geometric dual of a planar graph is defined by construction:

1. Place a node in each face, including the exterior face.
2. Two nodes are adjacent if the corresponding faces share an edge. Self-loops are allowed (when the same face appears on both sides of an edge.

The original planar graph G is called the primal and the newly constructed graph G^* is called the dual. The dual of a planar graph is still planar.

The following are some interesting points regarding the primal and dual relationship:

1. The number of nodes in the dual = the number of faces in the primal.
2. For simple graphs, the number of edges in the primal $\leq$ the number of edges in the dual.
3. If there exists no path of length ≥ 2 that separates any 2 faces, then the dual of the dual graph is the primal.

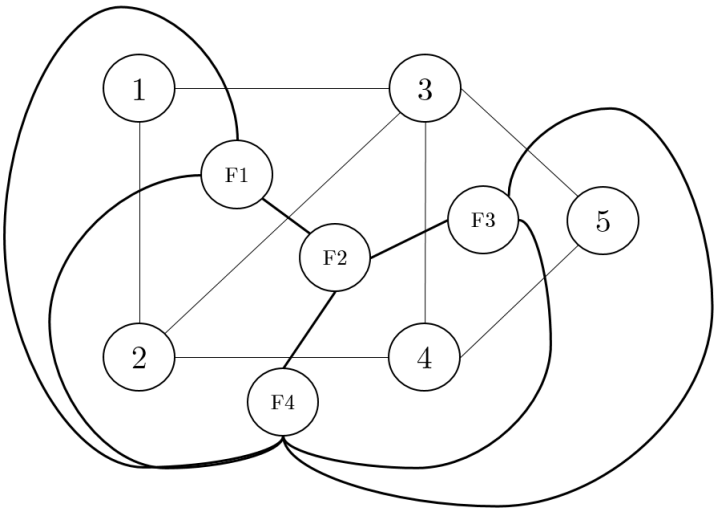


Figure 14: Geometric Dual of a Graph

4.3 Minimum s-t cut: Undirected planar graph

Given a graph $G = (V, E)$, an undirected planar embedded graph, $c : E \to \mathbf{R}$, a weight function on edges $e \in E$ such that no negative cost cycles are defined inside G and G^*, and two nodes $s, t \in V$, consider a cut $(S, V \setminus S)$ such that $s \in S$, $t \notin S$. Let's define $\delta(S)$ as the set of edges crossing the cut. We want to find a cut that minimizes:

$$c(\delta(S)) \quad = \quad \sum_{e \in \delta(S)} c(e) \tag{19}$$

The idea here is to use the dual graph G^*. Assume without loss of generality that G is connected. A min-cut in G is simple (i.e. both sides of the cut are connected) so in the dual G^* it corresponds to a minimum length simple cycle that separates nodes s and t. Here we consider $c(e)$ as the length/weight of the edge in the dual graph, in a one-to-one correspondence. Hence, in order to obtain the minimum cut in the primal graph G, we are looking for the minimum length separating s-t cycle in G^*.

Let f and g be two faces in G^* such that f is incident to node s and g to node t. Let P^* be a shortest f-to-g path in G^*. We can make the following observations:

1. Any cycle separating s and t crosses P^*.
2. There exists a minimum length cycle that separates s and t and crosses P^* exactly once.

In order to find such a minimum length cycle, we duplicate the nodes along P^*, obtaining two copies of every node of P^*. Performing this transformation, the new graph has a unique face containing nodes s and t. Then, we can compute the shortest path P_i for every pair of copies of nodes i along P^*. The shortest P_i defines the minimum length cycle separating s and t in G^* and thus, it corresponds to a minimum s-t cut in the primal G.

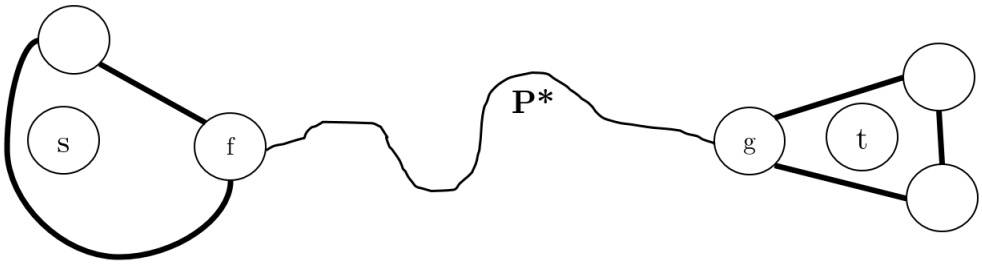


Figure 15: Shortest path from face f to g in G^*

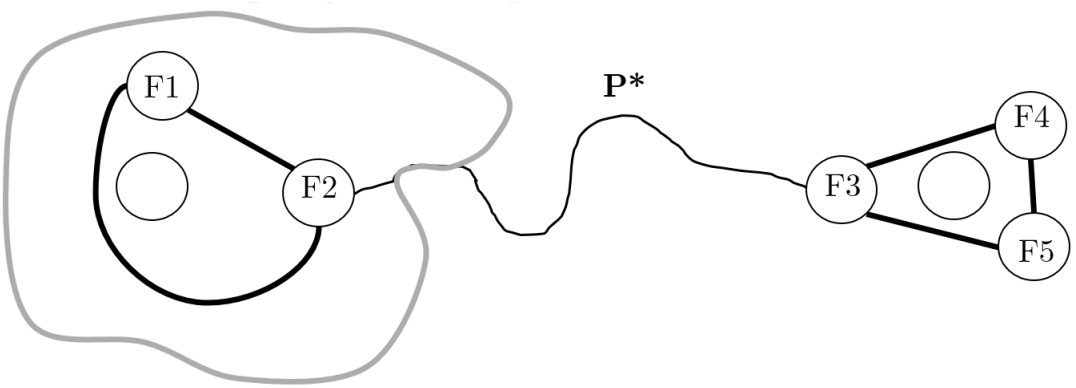


Figure 16: Minimum length cycle separating s and t that crosses P^* exactly once

4.3.1 Reif's Algorithm (1983)

Reif's algorithm consists of a "Divide and conquer" approach for solving the shortest path problems. The central idea is that shortest cycles do not cross so the original problem can be separated into sub-problems.

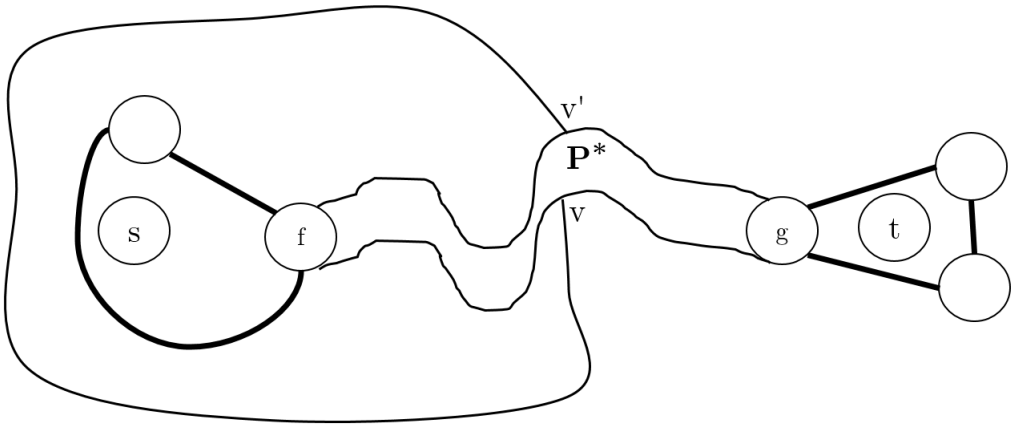


Figure 17: Reif's algorithm "Divide and conquer" approach.

Starting with the middle vertex v of P^*, we separate the problem into two new sub-problems. A new source $s^{'}$ and sink $t^{'}$ nodes are added to each sub-problem and the procedure is repeated.

Running time:

1. Paths are shorter by a factor of 2.
2. Depth of the recursion $\leq logn$.
3. Single source shortest path (SSSP) algorithm for planar graphs $O(nlogn)$.

Overall complexity $O(nlogn)$.

4.3.2 Modified Multiple source shortest path

We can use a modified version of the Multiple source shortest path (MSSP) algorithm for planar graphs developed in [**?**] to compute the shortest path between every pair of copies of each node of P^* in a complexity of $O(nlogn)$. This particular implementation exploits the fact that each pair of nodes lies on the same face of the planar graph.

Running time:

1. Find P^* in $O(n)$.
2. MSSP (planar graph implementation) in $O(nlogn)$.
3. Number of pairs $O(n)$.

Overall complexity $O(nlogn)$, same as Reif's "Divide and conquer" algorithm.

Currently, the best known algorithm is due to [**?**] with a complexity of $O(nloglogn)$. It uses the same overall idea and also applies a decomposition scheme for solving the multiple shortest path problems and FR-Dijkstra's algorithm.

4.4 s-t Planar Flow Network

We define a s-t planar flow network as a planar graph where nodes s and t are on the exterior face (or on the same face which can then be embedded as an exterior face).

Dual construction

1. Add a (t, s) arc with infinite capacity.
2. Obtain the dual G^* of the graph.
3. Call the two nodes in the two faces created by the new edge s^* (new face), and t^* (external face).

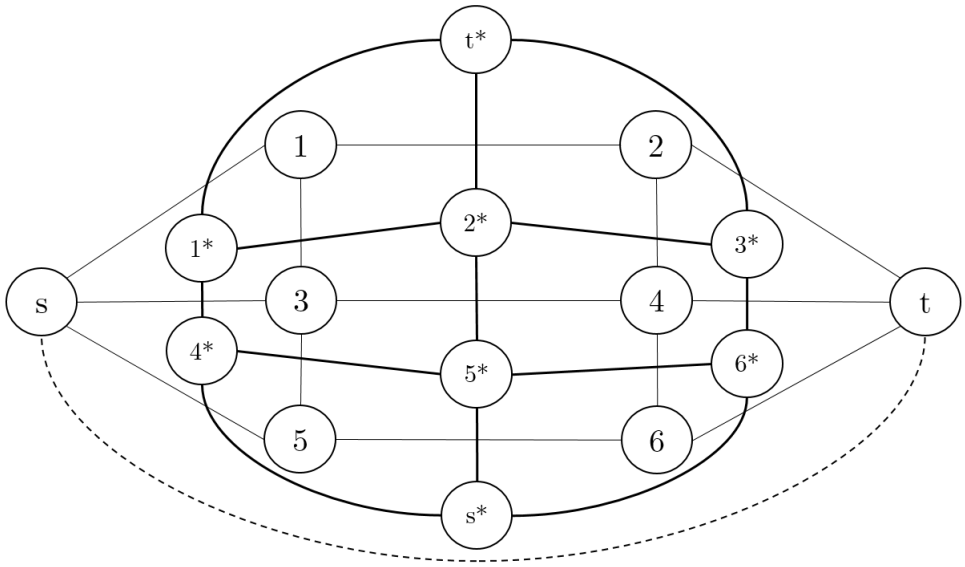


Figure 18: s-t Planar Flow Network and its dual graph G^*

4.5 Minimum s-t cut in an Undirected Weighted Planar Flow Network

In an undirected s-t planar flow network we can exploit the structure of the dual graph G^* in order to obtain the minimum s-t cut in the primal G, transforming the original minimum length separating cycle problem into a shortest path problem.

We observe:

1. Weight of edge in dual = Capacity of edge in primal.
2. Any path between s^* and t^* corresponds to a cut separating s and t.
3. Capacity of cut in primal G = Cost of path in dual G^*.
4. Shortest path between s^* and t^* corresponds to a minimum cut separating s and t in G.

Assuming that there are no negative cost cycles in G^*, Dijkstra's Algorithm can be applied to find the shortest path from s^* to t^*. The best running time for Dijkstra's Algorithm (Fibonacci heaps implementation) is $O(m + nlogn)$. Then, since $m < 3n$ (planar graph, Euler's formula), we have that the total complexity of the algorithm is $O(3n + nlogn) = O(nlogn)$.

Note that the above algorithm for minimum cut in planar graphs makes no use of the max flow and thus, the cut does not by itself give the max flow in the network.

4.6 Max flow in s-t planar graph

Shown by Hassin in 1981 [**?**] that max flow can also be derived from shortest path labels. In the original graph we define:

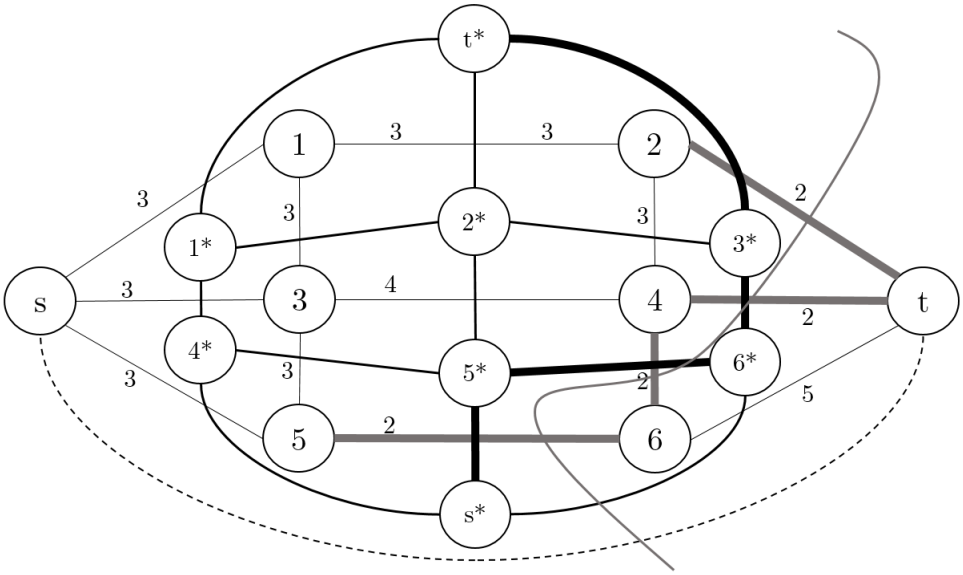


Figure 19: s-t Planar Flow Network minimum cut obtained as shortest s^*-t^* path in G^*.

1. U_{ij} = flow capacity of arc (i, j).
2. X_{ij} = flow from node i to node j.

In an undirected graph, the flow capacity for an edge is $(0, U_{ij})$ for both directions. Therefore $-U_{ij} \leq X_{ij} \leq U_{ij}$.

Consider the edge (i, j) in the direction i to j, let i^* be a dual node in the face to the left of the arc and j^* to the right. Define $d(i^*)$ the shortest path from s^* to i^* in G^*, then:

$$X_{ij} \quad = \quad d(i^*) - d(j^*) \tag{20}$$

Proposition 5. *$X_{ij} = d(i^*) - d(j^*)$ are feasible flows.*

Proof.

1. The capacity constraints are satisfied: Since $d(i^*)$ is the shortest path from s^* to i^* in the dual graph, $d(i^*)$ satisfies the shortest path optimality condition:

$$\begin{aligned}
d(i^*) &\leq d(j^*) + c_{i^*,j^*} \quad \text{(capacity of cut in primal = cost of path in dual)} && (21)\\
d(i^*) &\leq d(j^*) + U_{ij} && (22)\\
d(j^*) &\leq d(i^*) + U_{ij} && (23)\\
-U_{ij} &\leq X_{ij} \leq U_{ij} && (24)
\end{aligned}$$

2. The flow balance constraints are satisfied: For a node i, let's label its neighbors $1, 2, .., p$ and create the dual graph with nodes $1^*, 2^*, .., p^*$ (see Figure 20).

Thus, we have:

$$X_{i1} = d(1^*) - d(2^*) \tag{25}$$
$$X_{i2} = d(2^*) - d(3^*) \tag{26}$$
$$... \tag{27}$$
$$X_{ip} = d(p^*) - d(1^*) \tag{28}$$

Sum up all the equation we have: $\sum_{k=1}^{p} X_{ik} = 0$.

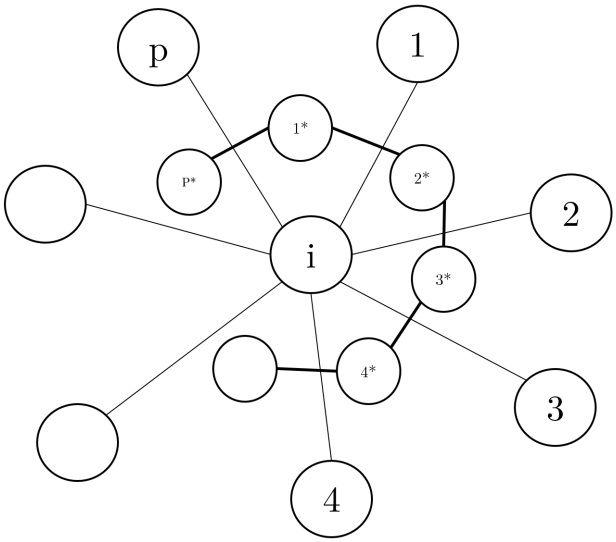


Figure 20: Dual graph for max-flow.

□

For the edges on the shortest path in G^*, we identify the minimum cut. Then, the flow on the cut edges is saturated:

$$\text{Flow value} = \text{cut capacity} = \text{max flow}$$

We use Dijkstra's Algorithm to find the shortest path, which is also the max flow.

The complexity of the algorithm is $O(nlogn)$ as we justified in minimum cut algorithm. Hence, we obtain the same complexity when solving min cut or max flow in a undirected weighted s-t planar flow network.

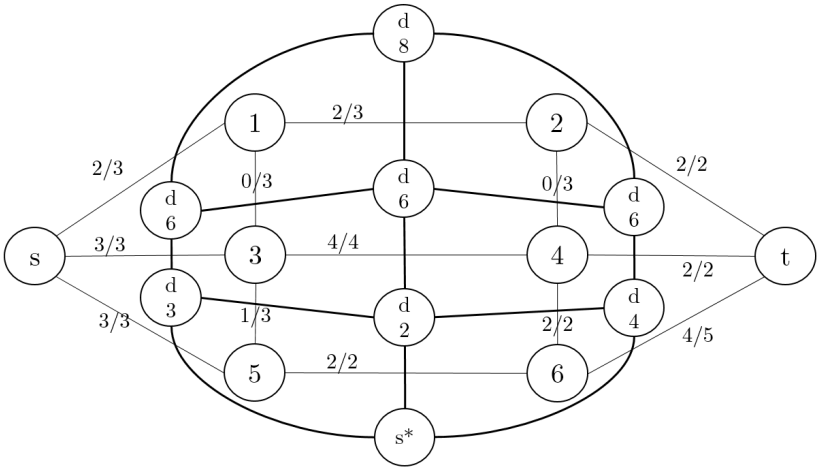


Figure 21: Maximum s-t flow.

References

[Bal95] R. Baldick (1995). A unified approach to polynomially solvable cases of integer non-separable quadratic optimization. *Discrete Applied Mathematics* 61:195–212

[BW90] R. Baldick and F.F. Wu (1990). Efficient integer optimization algorithms for optimal coordination of capacitators and regulators. *IEEE Transactions on Power Systems* 5:805–812

[BYE81] R. Bar-Yehuda and S. Even (1981). A linear time approximation algorithm for the weighted vertex cover problem. *J. of Algorithms* 2 198–203

[BM76] J. A. Bondy and U. S. R. Murty, *Graph theory with applications* Vol. 290. London:Macmillan, 1976.

[Chv79] V. Chvátal (1979). A Greedy Heuristic for the Set-Covering Problem *Math. of Oper. Res.* Vol. 4, 3, 233–235

[F95] U. Feige. A threshold of $\ln n$ for approximating set cover. manuscript, 1995.

[GGT89] G. Gallo, M. D. Grigoriadis, and R. E. Tarjan (1989). A fast parametric flow algorithm and applications. *SIAM Journal on Computing* 18(1):30–55

[GJ79] M. R. Garey and D. S. Johnson, *Computers and Intractability,* W. H. Freeman, San Francisco, 1979.

[HH86] N. G. Hall and D. S. Hochbaum (1986). A fast approximation algorithm for the multicovering problem. *Discrete Applied Mathematics* 15 35–40

[Hoc82] D. S. Hochbaum. Approximation algorithms for the set covering and vertex cover problems. *SIAM J. Comput.* 11(3) 1982, an extended version: W.P. #64-79-80, GSIA, Carnegie-Mellon University, April 1980.

[Hoc94] D. S. Hochbaum (1994). Lower and upper bounds for the allocation problem and other nonlinear optimization problems. *Mathematics of Operations Research* 19:390–409

[Hoc01] D. S. Hochbaum (2001). An efficient algorithm for image segmentaion, Markov random fields and related problems. *J. of ACM*, 48(4):686-701

[Hoc13] D. S. Hochbaum (2013). Multi-label Markov random fields as an efficient and effective tool for image segmentation, total variations and regularization. *Numerical Mathematics: Theory, Methods and Applications* 6:169-198

[HOC] D. S. Hochbaum, *Lecture Notes for IEOR 266: Graph Algorithms and Network Flows* Berkeley, 2016.

[HQ03] D. S. Hochbaum and M. Queyranne (1995). Minimizing a convex cost closure set. *SIAM Journal of Discrete Math* 16(2):192–207

[HS90] D. S. Hochbaum and J. G. Shanthikumar (1990). Convex separable optimization is not much harder than linear optimization. *Journal of the ACM* 37:843–862

[DW86] D. S. Hochbaum and E. Wigderson (1986). The Linzertorte problem, or a unified approach to painting, baking and weaving. *Discrete applied mathematics* 14:17–32

[Joh74] D. S. Johnson (1974). Approximation Algorithms for Combinatorial Problems. *J. Comput. System Sci.*, 9, 256–278

[KS80] R. Kindermann and J. L. Snell (1980). *Markov random fields and their applications.* Providence,RI: The American Mathematical Society.

[Lov75] L. Lovász (1975). On the Ratio of Optimal Integral and Fractional Covers. *Discrete Math.* 13 383–390

[MST91] Y. Mansour, B. Schieber and P. Tiwari (1991). Lower bounds for computations with the floor operation. *SIAM Journal on Computing* 20:315–327

[R87] J. Renegar (1987). On the worst case arithmetic complexity of approximation zeroes of polynomials. *Journal of Complexity* 3:90–113