18.415/6.854 Advanced Algorithms	November 1994

Approximation Algorithms

Lecturer: Michel X. Goemans

1 Introduction

Many of the optimization problems we would like to solve are NP-hard. There are several ways of coping with this apparent hardness. For most problems, there are straightforward exhaustive search algorithms, and one could try to speed up such an algorithm. Techniques which can be used include divide-and-conquer (or the refined branch-and-bound which allows to eliminate part of the search tree by computing, at every node, bounds on the optimum value), dynamic programming (which sometimes leads to pseudo-polynomial algorithms), cutting plane algorithms (in which one tries to refine a linear programming relaxation to better match the convex hull of integer solutions), randomization, etc. Instead of trying to obtain an optimum solution, we could also settle for a suboptimal solution. The latter approach refers to *heuristic* or "rule of thumb" methods. The most widely used such methods involve some sort of local search of the problem space, yielding a locally optimal solution. In fact, heuristic methods can also be applied to polynomially solvable problems for which existing algorithms are not "efficient" enough. A $\Theta(n^{10})$ algorithm (or even a linear time algorithm with a constant of 10^{100}), although efficient from a complexity point of view, will probably never get implemented because of its inherent inefficiency.

The drawback with heuristic algorithms is that it is difficult to compare them. Which is better, which is worse? For this purpose, several kinds of analyses have been introduced.

1. **Empirical analysis**. Here the heuristic is tested on a bunch of (hopefully meaningful) instances, but there is no guarantee that the behavior of the heuristic on these instances will be "typical" (what does it mean to be typical?).

2. **Average-case analysis**, dealing with the average-case behavior of a heuristic over some distribution of instances. The difficulty with this approach is that it can be difficult to find a distribution that matches the real-life data an algorithm will face. Probabilistic analyses tend to be quite hard.

3. **Worst-case analysis**. Here, one tries to evaluate the performance of the heuristic on the worst possible instance. Although this may be overly pessimistic, it gives a stronger guarantee about an algorithm's behavior. This is the type of analysis we will be considering in these notes.

To this end, we introduce the following definition:

Definition 1 *The performance guarantee of a heuristic algorithm for a minimization (maximization) problem is α if the algorithm is guaranteed to deliver a solution whose value is at most (at least) α times the optimal value.*

Definition 2 *An α-approximation algorithm is a polynomial time algorithm with a performance guarantee of α.*

Before presenting techniques to design and analyze approximation algorithms as well as specific approximation algorithms, we should first consider which performance guarantees are unlikely to be achievable.

2 Negative Results

For some hard optimization problems, it is possible to show a limit on the performance guarantee achievable in polynomial-time (assuming $P \neq NP$). A standard method for proving results of this form is to show that the existence of an α-approximation algorithm would allow you to solve some NP-complete decision problem in polynomial time. Even though NP-complete problems have equivalent complexity when exact solutions are desired, the reductions don't necessarily preserve approximability. The class of NP-complete problems can be subdivided according to how well a problem can be approximated.

As a first example, for the traveling salesman problem (given nonnegative lengths on the edges of a complete graph, find a tour — a closed walk visiting every vertex exactly once — of minimum total length), there is no α-approximation for any α unless $P = NP$. Indeed such an algorithm could be used to decide whether a graph (V, E) has an Hamiltonian cycle (simply give every edge e a length of 1 and every non-edge a very high or infinite length).

As another example, consider the bin packing problem. You have an integer T and weights $x_1, \ldots, x_n \in [0, T]$, and you want to partition them into as few sets ("bins") as possible such that the sum of the weights in each set is at most T. It is NP-complete to decide whether k bins are sufficient.

In fact, there is no α-approximation algorithm for this problem, for any $\alpha < 3/2$. To see this, consider the partition problem: given weights $x_1, \ldots, x_n \in [0, S]$ whose total sum is $2S$, is there a *partition* of the weights into two sets such that the sum in each set is S? This is the same as asking: are two bins sufficient when each bin has capacity S? If we had an α-approximation algorithm ($\alpha < 3/2$), we could solve the partition problem[1]. In general, if the problem of deciding whether a value is at most k is NP-complete then there is no α-approximation algorithm with $\alpha < \frac{k+1}{k}$ for the problem of minimizing the value unless $P = NP$.

[1] But wait, you exclaim — isn't there a polynomial-time approximation scheme for the bin packing problem? In fact, very good approximation algorithms can be obtained for this problem if you allow additive as well as multiplicative constants in your performance guarantee. It is our more restrictive model that makes this negative result possible. See Section 6 for more details.

Until 1992, pedestrian arguments such as this provided essentially the only known examples of non-approximability results. Then came a string of papers culminating in the result of Arora, Lund, Motwani, Sudan, and Szegedy[1] (based on the work of Arora and Safra [2]). They introduced a new characterization of NP in terms of probabilistically checkable proofs (PCP). In the new characterization, for any language L in NP, given the input x and a "proof" y of polynomial length in x, the verifier will toss $O(\log n)$ coins (where n is the size of x) to determine $k = O(1)$ positions or bits in the string y to probe; based on the values of these k bits, the verifier will answer "yes" or "no". The new characterization shows the existence of such a verifier V and a proof y such that (i) if $x \in L$ then there exists a proof y such that V outputs "yes" independently of the random bits, (ii) if $x \notin L$ then for every proof y, V outputs "no" with probability at least 0.5.

From this characterization, they deduce the following negative result for MAX 3SAT: given a set of clauses with at most 3 literals per clause, find an assignment maximizing the number of satisfied clauses. They showed the following:

Theorem 1 *For some $\epsilon > 0$, there is no $1-\epsilon$-approximation algorithm[2] for MAX 3SAT unless $P = NP$.*

The proof goes as follows. Take any NP-complete language L. Consider the verifier V given by the characterization of Arora et al. The number of possible output of the $O(\log n)$ toin cosses is $S = 2^{O(\log n)}$ which is polynomial in n. Consider any outcome of these coin tosses. This gives k bits, say $i_1, \ldots, i_k$ to examine in the proof y. Based on these k bits, V will decide whether to answer yes or no. The condition that it answers yes can be expressed as a boolean formula on these k bits (with the Boolean variables being the bits of y). This formula can be expressed as the disjunction ("or") of conjunctions ("and") of k literals, one for each satisfying assignment. Equivalently, it can be written as the conjunction of disjunction of k literals (one for each rejecting assignment). Since k is $O(1)$, this latter k-SAT formula with at most 2^k clauses can be expressed as a 3-SAT formula with a constant number of clauses and variables (depending exponentially on k). (More precisely, using the classical reduction from SAT to 3-SAT, we would get a 3-SAT formula with at most $k2^k$ clauses and variables.) Call this constant number of clauses $M \leq k2^k = O(1)$. If $x \in L$, we know that there exists a proof y such that all SM clauses obtained by concatenating the clauses for each random outcome is satisfiable. However, if $x \notin L$, for any y, the clauses corresponding to at least half the possible random outcomes cannot be all satisfied. This means that if $x \notin L$, at least $S/2$ clauses cannot be satisfied. Thus either all SM clauses can be satisfied or at most $SM - \frac{S}{2}$ clauses can be satisfied. If we had an approximation algorithm with performance guarantee better than $1-\epsilon$ where $\epsilon = \frac{1}{2M}$ we could decide whether $x \in L$ or not, in polynomial

[2]In our definition of approximation algorithm, the performance guarantee is less than 1 for *maximization* problems.

time (since our construction can be carried out in polynomial time). This proves the theorem.

The above theorem implies a host of negative results, by considering the complexity class MAX-SNP. defined by Papadimitriou and Yannakakis [21].

Corollary 2 *For any MAX-SNP-complete problem, there is an absolute constant* $\epsilon > 0$ *such that there is no* $(1-\epsilon)$*-approximation algorithm unless* $P = NP$*.*

The class of MAX-SNP problems is defined in the next section and the corollary is derived there. We first give some examples of problems that are complete for MAX-SNP.

1. MAX 2-SAT: Given a set of clauses with one or two literals each, find an assignment that maximizes the number of satisfied clauses.

2. MAX k-SAT: Same as MAX 2-SAT, but each clause has up to k literals.

3. MAX CUT: Find a subset of the vertices of a graph that maximizes the number of edges crossing the associated cut.

4. The Travelling Salesman Problem with the triangle inequality is MAX-SNP-hard. (There is a technical snag here: MAX-SNP contains only maximization problems, whereas TSP is a minimization problem.)

2.1 MAX-SNP Complete Problems

Let's consider an alternative definition of NP due to Fagin [9]. NP, instead of being defined computationally, is defined as a set of predicates or functions on structures G:

$$\exists S \forall x \exists y \; \psi(x, y, G, S)$$

where ψ is a quantifier free expression. Here S corresponds to the witness or the proof.

Consider for example the problem SAT. We are given a set of clauses, where each clause is the disjunction of literals. (A literal is a variable or its negation.) We want to know if there is a way to set the variables true or false, such that every clause is true. Thus here G is the set of clauses, S is the set of literals to be set to true, x represents the clauses, y represents the literals, and

$$\psi(x, y, G, S) = (P(G, y, x) \wedge y \in S) \vee (N(G, y, x) \wedge y \notin S)$$

Where $P(G, y, x)$ is true iff y appears positively in clause x, and $N(G, y, x)$ is true iff y appears negated in clause x.

Strict NP is the set of problems in NP that can be defined without the the third quantifier:

$$\exists S \forall x \ \psi(x, G, S)$$

where ψ is quantifier free.

An example is 3-SAT, the version of SAT in which every clause has at most 3 literals. Here $x = (x_1, x_2, x_3)$ (all possible combinations of three variables) and G is the set of possible clauses; for example $(x_1 \vee x_2 \vee x_3)$, $(\overline{x_1} \vee x_2 \vee \overline{x_3})$, and so forth. Then ψ is a huge conjunction of statements of the form: If (x_1, x_2, x_3) appears as $(\overline{x_1} \vee x_2 \vee \overline{x_3})$, then $x_1 \notin S \vee x_2 \in S \vee x_3 \notin S$.

Instead of asking that for each x we get $\psi(x, G, S)$, we can ask that the number of x's for which $\psi(x, G, S)$ is true be maximized:

$$\max_S |\{x : \psi(x, G, S)\}|$$

In this way, we can derive an optimization problem from an SNP predicate. These maximization problems comprise the class MAX-SNP (MAXimization, Strict NP) defined by Papadimitriou and Yannakakis [21]. Thus, MAX 3SAT is in MAX-SNP.

Papadimitriou and Yannakakis then introduce an *L-reduction* (L for linear), which preserverses approximability. In particular, if P L-reduces to P', and there exists an α-approximation algorithm for P', then there exists a $\gamma\alpha$-approximation algorithm for P, where γ is some constant depending on the reduction.

Given L-reductions, we can define MAX-SNP complete problems to be those $P \in$ MAX-SNP for which $Q \leq_L P$ for all $Q \in$ MAX-SNP. Some examples of MAX-SNP complete problems are MAX 3SAT, MAX 2SAT (and in fact MAX kSAT for any fixed $k > 1$), and MAX-CUT. The fact that MAX 3SAT is MAX-SNP-complete and Theorem 1 implies the corollary mentioned previously.

For MAX 3SAT, ε in the statement of Theorem 1 can be chosen can be set to $1/74$ (Bellare and Sudan [5]).

Minimization problems may not be able to be expressed so that they are in MAX-SNP, but they can still be MAX-SNP hard. Examples of such problems are:

- TSP with edge weights 1 and 2 (i.e., $d(i, j) \in \{1, 2\}$ for all i, j). In this case, there exists a 7/6-approximation algorithm due to Papadimitriou and Yannakakis.

- Steiner tree with edge weights 1 and 2.

- Minimum Vertex Cover. (Given a graph $G = (V, E)$, a vertex cover is a set $S \subseteq V$ such that $(u, v) \in E \Rightarrow u \in S$ or $v \in S$.)

3 The Design of Approximation Algorithms

We now look at key ideas in the design and analysis of approximation algorithms. We will concentrate on minimization problems, but the ideas apply equally well to maximization problems. Since we are interested in the minimization case, we know that an α-approximation algorithm H has cost $C_H \leq \alpha C_{OPT}$ where C_{OPT} is the cost of the optimal solution, and $\alpha \geq 1$.

Relating C_H to C_{OPT} directly can be difficult. One reason is that for NP-hard problems, the optimum solution is not well characterized. So instead we can relate the two in two steps:

1. $LB \leq C_{OPT}$
2. $C_H \leq \alpha LB$

Here LB is a lower bound on the optimal solution.

3.1 Relating to Optimum Directly

This is not always necessary, however. One algorithm whose solution is easy to relate directly to the optimal solution is Christofides' [6] algorithm for the TSP with the triangle inequality ($d(i,j) + d(j,k) \leq d(i,k)$ for all i, j, k). This is a $\frac{3}{2}$-approximation algorithm, and is the best known for this problem. The algorithm is as follows:

1. Compute the minimum spanning tree T of the graph $G = (V, E)$.
2. Let O be the odd degree vertices in T. One can prove that $|O|$ is even.
3. Compute a minimum cost perfect matching M on the graph induced by O.
4. Add the edges in M to E. Now the degree of every vertex of G is even. Therefore G has an Eulerian tour. Trace the tour, and take shortcuts when the same vertex is reached twice. This cannot increase the cost since the triangle inequality holds.

We claim that $Z_C \leq \frac{3}{2} Z_{TSP}$, where Z_C is the cost of the tour produced by Christofides' algorithm, and Z_{TSP} is the cost of the optimal solution. The proof is easy:

$$\frac{Z_C}{Z_{TSP}} \leq \frac{Z_T + Z_M}{Z_{TSP}} = \frac{Z_T}{Z_{TSP}} + \frac{Z_M}{Z_{TSP}} \leq 1 + \frac{1}{2} = \frac{3}{2}.$$

Here Z_T is the cost of the minimum spanning tree and Z_M is the cost of the matching. Clearly $Z_T \leq Z_{TSP}$, since if we delete an edge of the optimal tour a spanning tree results, and the cost of the minimum spanning tree is at most the cost of that tree. Therefore $\frac{Z_T}{Z_{TSP}} \leq 1$.

To show $\frac{Z_M}{Z_{TSP}} \leq \frac{1}{2}$, consider the optimal tour visiting only the vertices in O. Clearly by the triangle inequality this is of length no more than Z_{TSP}. There are an even number of vertices in this tour, and so also an even number of edges, and the tour defines two disjoint matchings on the graph induced by O. At least one of these has cost $\leq \frac{1}{2}Z_{TSP}$, and the cost of Z_M is no more than this.

3.2 Using Lower Bounds

Let

$$C_{OPT} = \min_{x \in S} f(x).$$

A lower bound on C_{OPT} can be obtained by a so-called *relaxation*. Consider a related optimization problem $LB = \min_{x \in R} g(x)$. Then LB is a lower bound on C_{OPT} (and the optimization problem is called a relaxation of the original problem) if the following conditions hold:

$$S \subseteq R \tag{1}$$

$$g(x) \leq f(x) \text{ for all } x \in S. \tag{2}$$

Indeed these conditions imply

$$LB = \min_{x \in R} g(x) \leq \min_{x \in S} f(x) = C_{OPT}.$$

Most classical relaxations are obtained by using linear programming. However, there are limitations as to how good an approximation LP can produce. We next show how to use a linear programming relaxation to get a 2-approximation algorithm for Vertex Cover, and show that this particular LP relaxation cannot give a better approximation algorithm.

3.3 An LP Relaxation for Minimum Weight Vertex Cover (VC)

A vertex cover U in a graph $G = (V, E)$ is a subset of vertices such that every edge is incident to at least one vertex in U. The vertex cover problem is defined as follows: Given a graph $G = (V, E)$ and weight $w(v) \geq 0$ for each vertex v, find a vertex cover $U \subseteq V$ minimizing $w(U) = \sum_{v \in U} w(v)$. (Note that the problem in which nonpositive weight vertices are allowed can be handled by including all such vertices in the cover, deleting them and the incident edges, and finding a minimum weight cover of the remaining graph. Although this reduction preserves optimality, it does not maintain approximability; consider, for example, the case in which the optimum vertex cover has 0 cost (or even negative cost).)

This can be expressed as an integer program as follows. Let $x(v) = 1$ if $v \in U$ and $x(v) = 0$ otherwise. Then

$$C_{OPT} = \min_{x \in S} \sum_{v \in V} w(v)x(v)$$

where

$$S = \left\{ x \in R^{|V|} : \begin{array}{ll} x(v) + x(w) \geq 1 & \forall (v,w) \in E \\ x(v) \in \{0,1\} & \forall v \in V \end{array} \right\}.$$

We now relax S, turning the problem into a linear program:

$$LB = \min_{x \in R} \sum_{v \in V} w(v)x(v)$$

$$R = \left\{ x \in R^{|V|} : \begin{array}{ll} x(v) + x(w) \geq 1 & \forall (v,w) \in E \\ x(v) \geq 0 & \forall v \in V \end{array} \right\}.$$

In order to show that R is a relaxation, we must show that it satisfies conditions 1 and 2. Condition 1 clearly holds, as $0, 1 \geq 0$. Furthermore, condition 2 also holds, since the objective function is unchanged. Thus, we can conclude that $LB \leq C_{OPT}$, and we can prove that an algorithm H is an α-approximation algorithm for VC by showing $C_H \leq \alpha LB$.

The limitation of this relaxation is that there are instances where $LB \sim \frac{1}{2} C_{OPT}$. This implies that it cannot be used to show any $\alpha < 2$, since if we could then H would give a better answer than the optimum. One such instance is K_n: the complete graph on n vertices, with all vertices weight 1. All the nodes but one must be in the cover (otherwise there will be an edge between two that are not, with neither in the cover set). Thus, $C_{OPT} = n - 1$. The relaxation, on the other hand, can have $x(v) = \frac{1}{2}, \forall v \in V$. Thus, $LB \leq \frac{n}{2}$, which means $LB \sim \frac{1}{2} C_{OPT}$.

3.4 How to use Relaxations

There are two main techniques to derive an approximately optimal solution from a solution to the relaxed problem.

1. **Rounding**
   Find an optimal solution x^* to the relaxation. Round $x^* \in R$ to an element $x' \in S$. Then prove $f(x') \leq \alpha g(x^*)$ which implies

   $$f(x') \leq \alpha LB \leq \alpha C_{OPT}$$

   Often randomization is helpful, as we shall see in later sections. In this case $x^* \in R$ is randomly rounded to some element $x' \in S$ so that $E[f(x')] \leq \alpha g(x^*)$. These algorithms can sometimes be derandomized, in which case one finds an x'' such that $f(x'') \leq E[f(x')]$.

2. **Primal-Dual**
   Consider some weak dual of the relaxation:
   $$\max\{h(y) : y \in D\} \leq \min\{g(x) : x \in R\}$$
   Construct $x \in S$ from $y \in D$ such that
   $$f(x) \leq \alpha h(y) \leq \alpha h(y_{\max}) \leq \alpha g(x_{\min}) \leq \alpha C_{OPT}.$$
   Notice that y can be any element of D, not necessarily an optimal solution to the dual.

We now illustrate these techniques on the minimum weight vertex cover problem.

3.4.1 Rounding applied to VC

This is due to Hochbaum [16]. Let x^* be the optimal solution of the LP relaxation. Let
$$U = \{v \in V : x^*(v) \geq \frac{1}{2}\}$$
We claim U is a 2-approximation of the minimum weight VC. Clearly U is a vertex cover, because for $(u, v) \in E$ we have $x^*(u) + x^*(v) \geq 1$, which implies $x^*(u) \geq 1/2$ or $x^*(v) \geq 1/2$. Also
$$\sum_{v \in U} w(v) \leq \sum_{v \in V} w(v) 2x^*(v) = 2LB$$
since $2x^*(v) \geq 1$ for all $v \in U$.

3.4.2 Primal-Dual applied to VC

This is due to Bar-Yehuda and Even [4]. First formulate the dual problem. Let $y \in R^{|E|}$; the elements of y are $y(e)$ for $e = (u, v) \in E$. The dual is:
$$\max \sum_{e \in E} y(e)$$
$$\sum_{u:e=(v,u)\in E} y(e) \leq w(v) \quad \forall v \in V \tag{3}$$
$$y(e) \geq 0 \quad \forall e \in E. \tag{4}$$

Initialize C (the vertex cover) to the empty set, $y = 0$ and $F = E$. The algorithm proceeds by repeating the following two steps while $F \neq \emptyset$:

1. Choose some $e = (u, v) \in F$. Increase $y(e)$ as much as possible, until inequality (3) becomes tight for u or v. Assume WLOG it is tight for u.

2. Add u to C and remove all edges incident to u from F.

Clearly C is a vertex cover. Furthermore
$$\sum_{v \in C} w(v) = \sum_{v \in C} \sum_{u:e=(v,u)\in E} y(e) = \sum_{e=(v,u)\in E} |C \cap \{v, u\}| y(e) \leq \sum_{e \in E} 2y(e) \leq 2LB.$$

4 The Min-Cost Perfect Matching Problem

In this section, we illustrate the power of the primal-dual technique to derive approximation algorithms. We consider the following problem.

Definition 3 *The* Minimum-Cost Perfect Matching Problem (MCPMP) *is as follows: Given a complete graph $G = (V, E)$ with $|V|$ even and a nonnegative cost function $c_e \geq 0$ on the edges $e \in E$, find a perfect matching M such that the cost $c(M)$ is minimized, where $c(M) = \sum_{e \in M} c_e$.*

The first polynomial time algorithm for this problem was given by Edmonds [8] and has a running time of $O(n^4)$ where $n = |V|$. To date, the fastest strongly polynomial time algorithm is due to Gabow [10] and has a running time of $O(n(m + n \lg n))$ where $m = |E|$. For dense graphs, $m = \Theta(n^2)$, this algorithm gives a running time of $O(n^3)$. The best weakly polynomial algorithm is due to Gabow and Tarjan [12] and runs in time $O(m\sqrt{n\alpha(m,n)\log n}\log nC)$ where C is a bound on the costs c_e. For dense graphs with $C = O(n)$, this bound gives an $O^*(n^{2.5})$ running time.

As you might suspect from these bounds, the algorithms involved are fairly complicated. Also, these algorithms are too slow for many of the instances of the problem that arise in practice. In this section, we discuss an approximation algorithm by Goemans and Williamson [13] that runs in time $O(n^2 \lg n)$. (This bound has recently been improved by Gabow, Goemans and Williamson [11] to $O(n(n + \sqrt{m \lg \lg n}))$.) Although MCPMP itself is in PTIME, this algorithm is sufficiently general to give approximations for many NP-hard problems as well.

The algorithm of Goemans and Williamson is a 2-approximation algorithm — it outputs a perfect matching with cost not more than a factor of 2 larger than the cost of a minimum-cost perfect matching. This algorithm requires that the costs c_e make up a metric, that is, c_e must respect the triangle inequality: $c_{ij} + c_{jk} \geq c_{ik}$ for all triples i, j, k of vertices.

4.1 A linear programming formulation

The basic idea used in the 2-approximation algorithm of Goemans and Williamson is linear programming and duality. The min-cost perfect matching problem can be formulated as a linear program. The algorithm does not directly solve the linear program, but during its operation, it can compute a feasible solution to the dual program. This dual feasible solution actually certifies the factor of 2 approximation. Before writing down the linear program, we start with an observation.

Consider a matching M and a set $S \subset V$ of vertices with $|S|$ odd. If M is a perfect matching, then since $|S|$ is odd, there must be some edge in the matching that has one endpoint inside S and the other outside. In other symbols, let $\delta(S)$ be the set of edges in E with exactly one endpoint in S; if M is a perfect matching and $|S|$ is odd, then $M \cap \delta(S) \neq \emptyset$.

With this observation, we can now formulate MCPMP as a linear program:

$$
\begin{aligned}
Z = \text{Min} \sum_{e \in E} c_e x_e & \\
\text{subject to:} \quad \sum_{e \in \delta(S)} x_e \geq 1 & \qquad \text{for all } S \subset V \text{ with } |S| \text{ odd} \\
x_e \geq 0 & \qquad \text{for all } e \in E.
\end{aligned}
$$

We can now see that the value Z of this linear program is a lower bound on the cost of any perfect matching. In particular, for any perfect matching M, we let

$$
x_e = \begin{cases} 1 & \text{if } e \in M; \\ 0 & \text{otherwise.} \end{cases}
$$

Clearly, this assignment is a feasible solution to the linear program, so we know that $Z \leq c(M)$. This bound also applies to a minimum-cost perfect matching M^*, so we have $Z \leq c(M^*)$.

Note that this is a huge linear program having one constraint for each $S \subset V$ of odd cardinality. Though it is too large to be solved in polynomial time by any of the linear programming algorithms we have seen, the ellipsoid method can actually solve this program in polynomial time. We do not consider this solution technique; rather we let the linear program and its dual serve as a tool for developing and analyzing the algorithm.

We now consider the dual linear program:

$$
\begin{aligned}
Z = \text{Max} \sum_{\substack{S \subset V, \\ |S| \text{ odd}}} y_S & \\
\text{subject to:} \quad \sum_{\substack{S \subset V, \\ e \in \delta(S)}} y_S \leq c_e & \qquad \text{for all } e \in E \\
y_S \geq 0 & \qquad \text{for all } S \subset V \text{ with } |S| \text{ odd.}
\end{aligned}
$$

Note that by strong duality, the value Z of this dual linear program is the same as the value Z of the primal program.

This dual linear program is used to verify that the perfect matching output by the algorithm is actually within a factor of 2 of optimal. The algorithm outputs two things:

1. a perfect matching M', and

2. a dual feasible solution y such that

$$
c(M') \leq 2 \sum_{\substack{S \subset V, \\ |S| \text{ odd}}} y_S.
$$

Since y is dual feasible, we know that

$$\sum_{\substack{S \subset V, \\ |S| \text{ odd}}} y_S \leq Z \leq c(M)$$

where M is any perfect matching. Thus we have

$$c(M') \leq 2Z \leq 2c(M^*)$$

where M^* is a min-cost perfect matching. The algorithm is therefore (given that it runs in polynomial time) a 2-approximation algorithm for MCPMP.

To be precise, the algorithm need not actually output the dual feasible solution y — it is only needed as an analysis tool to prove the factor of 2 approximation bound. In spite of the fact that there are an exponential number of y_S variables, the algorithm could actually compute and output the y_S values since it turns out that only a polynomial number of them are non-zero. When we finally get to exhibiting the algorithm, we will include the computation of the y_S values.

4.2 From forest to perfect matching

Rather than directly compute the perfect matching M', the algorithm first computes a forest F' from which M' can be derived. In the forest F', all components have even size, and furthermore, F' is edge-minimal in the sense that if any edge of F' is removed, then the resulting forest has an odd size component. Additionally, the cost of F' is bounded by twice the value of the dual feasible solution; that is,

$$c(F') \leq 2 \sum_{\substack{S \subset V, \\ |S| \text{ odd}}} y_S.$$

We now show how to convert F' into a perfect matching M' such that $c(M') \leq c(F')$. The idea is as follows. Starting from the forest F', consider any vertex v with degree at least 3. Take two edges (u, v) and (v, w); remove them and replace them with the single edge (u, w). Since the edge costs obey the triangle inequality, the resulting forest must have a cost not more than $c(F')$. Thus, if we can iterate on this operation until all vertices have degree 1, then we have our perfect matching M'.

The only thing that can get in the way of the operation just described is a vertex of degree 2. Fortunately, we can show that all vertices of F' have odd degree. Notice then that this property is preserved by the basic operation we are using. (As a direct consequence, the property that all components are even is also preserved.) Therefore, if all vertices of F' have odd degree, we can iteration the basic operation to produce a perfect matching M' such that $c(M') \leq c(F')$. Notice that M' is produced after $O(n)$ iterations.

Lemma 3 *All vertices of F' have odd degree.*

Proof: Suppose there is a vertex v with even degree, and let v be in component A of F'. Removing v and all its incident edges partitions A into an even number k of smaller components $A_1, A_2, \ldots, A_k$. If all k of these components have odd size, then it must be the case that A has odd size. But we know that A has even size — all components of F' have even size — so there must be a component A_i with even size. Let v_i denote the vertex in A_i such that (v, v_i) is an edge of F'. Now if we start from F' and remove the edge (v, v_i), we separate A into two even size components. This contradicts the edge-minimality of F'. □

4.3 The algorithm

The algorithm must now output an edge-minimal forest F' with even size components and be able to compute a dual feasible solution y such that $c(F') \leq 2 \sum y_S$.

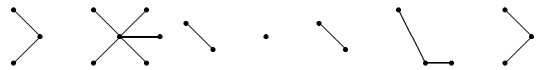
At the highest level, the algorithm is:

1. Start with $F = \emptyset$.

2. As long as there exists an odd size component of F, add an edge between two components (at least one of which has odd size).

Note that the set of components of F is initially just the set of vertices V.

The choice of edges is guided by the dual linear program shown earlier. We start with all the dual variables equal to zero; $y_S = 0$. Suppose at some point in the execution we have a forest F as shown below and a dual solution y. Look at the



1 2 3 4 5 6 7

components S of odd cardinality (components 1, 4, 6 and 7, in this case). For these components, increase y_S by some δ, leaving all other values of y_S unchanged. That is,

$$y_S \leftarrow \begin{cases} y_S + \delta & \text{if } S \text{ is an odd size component of } F \\ y_S & \text{otherwise.} \end{cases}$$

Make δ as large as possible while keeping y_S dual feasible. By doing this, we make the constraint on some edge e tight; for some e the constraint

$$\sum_{\substack{S \subset V, \\ e \in \delta(S)}} y_S \leq c_e$$

becomes

$$\sum_{\substack{S \subset V, \\ e \in \delta(S)}} y_S = c_e.$$

This is the edge e that we add to F. (If more than one edge constraint becomes tight simultaneously, then just arbitrarily pick one of the edges to add.)

We now state the algorithm to compute F'. The steps that compute the dual feasible solution y are commented out by placing the text in curly braces.

$F \leftarrow \emptyset$
$\mathcal{C} \leftarrow \{\{i\} \mid i \in V\}$ $\quad$ $\{$*The components of* $F\}$
$\{$*Let* $y_S \leftarrow 0$ *for all* S *with* $|S|$ *odd.*$\}$
$\forall i \in V$ do $d(i) \leftarrow 0$ $\quad$ $\{d(i) = \sum_{S \ni i} y_S\}$
while $\exists C \in \mathcal{C}$ with $|C|$ odd do
    Find edge $e = (i, j)$ such that $i \in C_p$, $j \in C_q$, $p \neq q$
        which minimizes $\delta = \frac{c_e - d(i) - d(j)}{\lambda(C_p) + \lambda(C_q)}$
        where $\lambda(C) = \begin{cases} 1 & \text{if } |C| \text{ is odd} \\ 0 & \text{otherwise} \end{cases}$ (*i.e.*, the parity of C).
    $F \leftarrow F \cup \{e\}$
    $\forall C \in \mathcal{C}$ with $|C|$ odd do
        $\forall i \in C$ do $d(i) \leftarrow d(i) + \delta$
        $\{$*Let* $y_C \leftarrow y_C + \delta$.$\}$
    $\mathcal{C} \leftarrow \mathcal{C} \setminus \{C_p, C_q\} \cup \{C_p \cup C_q\}$
$F' \leftarrow$ edge-minimal F

4.4 Analysis of the algorithm

Lemma 4 *The values of the variables y_S computed by the above algorithm constitute a dual feasible solution.*

Proof: We show this by induction on the *while* loop. Specifically, we show that at the start of each iteration, the values of the variables y_S are feasible for the dual linear program. We want to show that for each edge $e \in E$,

$$\sum_{\substack{S \subset V, \\ e \in \delta(S)}} y_S \leq c_e.$$

The base case is trivial since all variables y_S are initialized to zero and the cost function c_e is nonnegative. Now consider an edge $e' = (i', j')$ and an iteration. There are two cases to consider.

In the first case, suppose both i' and j' are in the same component at the start of the iteration. In this case, there is no component $C \in \mathcal{C}$ for which $e' \in \delta(C)$. Therefore, since the only way a variable y_S gets increased is when S is a component, none of the variables y_S with $e' \in \delta(S)$ get increased at this iteration. By the induction hypothesis, we assume the iteration starts with

$$\sum_{\substack{S\subset V,\\ e'\in\delta(S)}} y_S \le c'_e,$$

and therefore, since the left-hand-side of this inequality does not change during the iteration, this inequality is also satisfied at the start of the next iteration.

In the second case, suppose i' and j' are in different components; $i' \in C_{p'}$ and $j' \in C_{q'}$ at the start of the iteration. In this case, we can write

$$\begin{aligned}\sum_{\substack{S\subset V,\\ e'\in\delta(S)}} y_S &= \sum_{\substack{S\subset V,\\ i'\in S}} y_S + \sum_{\substack{S\subset V,\\ j'\in S}} y_S \\ &= d(i') + d(j'),\end{aligned}$$

where $d(i)$ is defined by

$$d(i) = \sum_{\substack{S\subset V,\\ i\in S}} y_S.$$

The equality follows because since i' and j' are in different components, if S contains both i' and j', then S is not and never has been a component; hence, for such a set S, we have $y_S = 0$. We know that during this iteration $d(i')$ will be incremented by δ if and only if $y_{C_{p'}}$ is incremented by δ, and this occurs if and only if $\lambda(C_{p'}) = 1$. Let $d'(i')$ and $d'(j')$ be the new values of $d(i')$ and $d(j')$ after this iteration. Then we have

$$\begin{aligned}d'(i') &= d(i') + \delta\lambda(C_{p'}), \quad \text{and}\\ d'(j') &= d(j') + \delta\lambda(C_{q'}).\end{aligned}$$

Now, by the way δ is chosen, we know that

$$\delta \le \frac{c_{e'} - d(i') - d(j')}{\lambda(C_{p'}) + \lambda(C_{q'})}.$$

Thus, at the beginning of the next iteration we have

$$\begin{aligned}\sum_{\substack{S\subset V,\\ e'\in\delta(S)}} y_S &= d'(i') + d'(j') \\ &= d(i') + \delta\lambda(C_{p'}) + d(j') + \delta\lambda(C_{q'}) \\ &= d(i') + d(j') + \delta[\lambda(C_{p'}) + \lambda(C_{q'})] \\ &\le d(i') + d(j') + \frac{c_{e'} - d(i') - d(j')}{\lambda(C_{p'}) + \lambda(C_{q'})}[\lambda(C_{p'}) + \lambda(C_{q'})] \\ &= c_{e'}.\end{aligned}$$

Finally, for completeness sake, we note that the constraint $y_S \ge 0$ is satisfied because $y_S = 0$ initially and $\delta \ge 0$. □

As a final observation, we note that when the algorithm selects an edge e', the corresponding constraint in the dual linear program becomes tight. This means that for all edges $e \in F$, we have

$$\sum_{\substack{S \subset V,\\ e \in \delta(S)}} y_S = c_e.$$

4.5 A simulated run of the algorithm

Since the algorithm as given can be difficult to visualize, here is an example of how it would execute. See Figure 1.

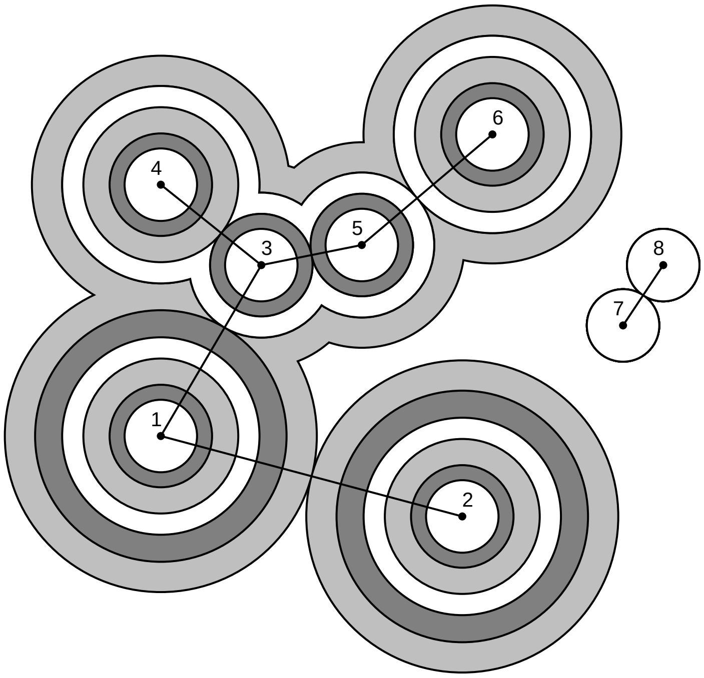


Figure 1: A sample run of the algorithm. The various values of $d(i)$ are indicated by the shaded regions around the components.

We'll assume a Euclidean distance metric to ease visualization. Now, initially, all points (1 through 8) are in separate components, and $d(i)$ is 0 for all i. Since the metric is Euclidean distance, the first edge to be found will be $(7,8)$. Since both components are of odd size, δ will be half the distance between them $((c_e - 0 - 0)/(1+1))$. Since, in fact, *all* components are of odd size, every $d(i)$ will be increased by this amount, as indicated by the innermost white circle around each point. The set $\{7,8\}$ now becomes a single component of even size.

In general, we can see the next edge to be chosen by finding the pair of components whose boundaries in the picture can be most easily made to touch. Thus, the next edge is $(3,5)$, since the boundaries of their regions are closest, and the resulting values of $d(i)$ are represented by the dark gray bands around points 1 through 6. Note that the component $\{7, 8\}$ does not increase its $d(i)$ values since it is of even size.

We continue in this way, expanding the "moats" around odd-sized components until all components are of even size. Since there is an even number of vertices and we always expand odd-sized components, we are guaranteed to reach such a point.

4.6 Final Steps of Algorithm and Proof of Correctness

Let $F' = \{e \in F : F \setminus \{e\}$ has an odd sized component$\}$. We will show that the F' so obtained is a forest with components of even cardinality and that it is edge minimal with respect to this property. It is obvious that F' is a forest since $F' \subset F$ and F is a forest. We will also show that the cost of this forest (F'), is less than or equal to twice the dual solution. In section 4.2 we showed how to build a matching from this forest with the cost of the matching less than or equal to the cost of the forest. Thus, this gives us a 2-approximation algorithm for matching. As an example see the figure given below.

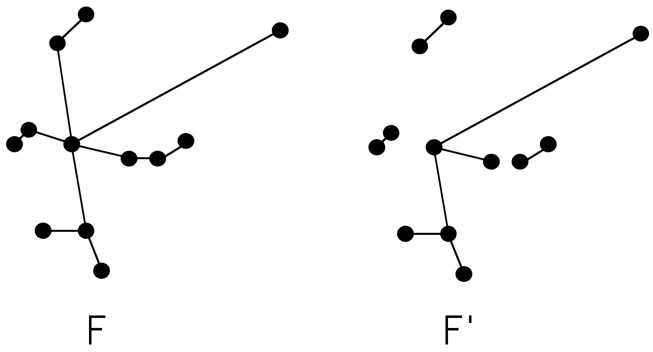


Figure 2: Example showing how to get F' from F.

Theorem 5 *Let $F' = \{e \in F : F \setminus \{e\}$ has an odd sized component$\}$. Then*

1. *every component of F' has an even number of vertices and F' is edge minimal with respect to this property..*

2. $\sum_{e \in F'} c_e \leq 2 \sum_S y_S$.

Proof:

Let us first show that every component of F' has an even number of vertices. Suppose not. Then consider the components of F. Every component of F has an even number of vertices by design of the algorithm. Consider a component of F' which has an odd number of vertices and let us denote it as T_i'. Let T_i be the component that T_i' belongs to in F. Let $N_1, \ldots, N_j$ be the components of F within T_i obtained by removing T_i' (see figure 3). T_i has an even number of vertices. N_k with $1 \leq k \leq j$ has an even number of vertices because, otherwise, the edge from N_k to T_i' would belong to F' by definition. But this implies that T_i' is even, which is a contradiction.

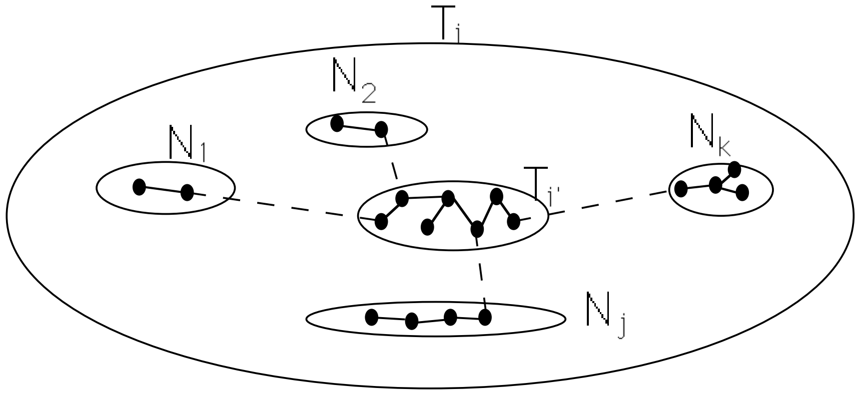


Figure 3: Every component of F' has an even # of vertices.

A simple proof by contradiction shows that F' is edge minimal. Suppose F' is not edge minimal. Then there is an edge or set of edges which can be removed which leave even sized components. Consider one such edge e. It falls into one of two categories:

1. Its removal divides a component into two even sized components. But this means that e was already removed by the definition of F'.

2. Its removal divides a component into two odd sized components. Despite the fact that other edges may be removed, as well, an two odd sized component will remain in the forest. Thus, e cannot be removed.

Now let us prove the second portion of the theorem. In what follows, though we do not explicitly notate it, when we refer to a set S of vertices, we mean a set S of vertices with $|S|$ odd. We observe that by the choice of the edges e in F, we have

$$c_e = \sum_{S: e \in \delta(S)} y_S$$

for all $e \in F$. Thus,

$$\begin{aligned} \sum_{e \in F'} c_e &= \sum_{e \in F'} \sum_{S: e \in \delta(S)} y_S \\ &= \sum_{S} y_S |F' \cap \delta(S)| \end{aligned}$$

Thus we need to show,

$$\sum_{S} y_S |F' \cap \delta(S)| \leq 2 \sum_{S} y_S$$

We will show this by induction. In what follows, bear in mind that F' is what we have, at the end of the algorithm. We will show the above relation holds at every iteration.

Initially, $y_S = 0$. Thus the LHS and RHS are both zero. Thus, this is true initially.

Let us suppose it is true at any intermediate stage in the algorithm. We will show that it will remain true in the next iteration. From one iteration to the next the only y_S that change are those with $C \in \mathcal{C}$ with $|C|$ odd. Thus if we show the increase in the LHS is less than the RHS we are done. i.e.

$$\delta \sum_{C \in \mathcal{C}, |C| \text{ odd}} |F' \cap \delta(C)| \leq 2\delta |\{C \in \mathcal{C}, |C| \text{ odd}\}|$$

or

$$\sum_{C \in \mathcal{C}, |C| \text{ odd}} |F' \cap \delta(C)| \leq 2 |\{C \in \mathcal{C}, |C| \text{ odd}\}|$$

Now, define a graph H with $C \in \mathcal{C}$ as vertices, with an edge between C_p and C_q if there exists an edge in $F' \cap \{\delta(C_p) \cap \delta(C_q)\}$. We can partition these vertices into two groups based on their cardinality. Those that have even cardinality and those that have odd cardinality. Remove from this graph all vertices that have even cardinality and are isolated (they have no edges incident to them). We will denote the resulting set of vertices of odd and even cardinality by Odd and $Even$ respectively.

Now $\sum_{C \in \mathcal{C}, |C| \text{ odd}} |F' \cap \delta(C)|$ corresponds to the sum of the degrees of vertices in Odd in the graph H. And, $|\{C \in \mathcal{C}, |C| \text{ odd}\}|$, corresponds to the number of vertices in odd. Thus we need to show:

$$\sum_{v \in Odd} d_H(v) \leq 2|Odd|$$

where $d_H(v)$ denotes the degree of node v in the graph H. Since F' is a forest, H is also a forest and we have:

Number of edges in $H \leq$ number of vertices in H. Or

$$\frac{\sum_{v \in Odd} d_H(v) + \sum_{v \in Even} d_H(v)}{2} \leq |Odd| + |Even|$$

or

$$\sum_{v \in Odd} d_H(v) \leq 2|Odd| + \sum_{v \in Even} (2 - d_H(v))$$

We now claim that if $v \in Even$ then v is not a leaf. If this is true then $(2 - d_H(v)) \leq 0$ for $v \in Even$ and so we are done.

Suppose there is a $v_i \in Even$ which is a leaf. Consider the component C in H that v_i is contained in. By the construction of H, each tree in F' is either contained solely in the vertices represented by C or it is strictly outside C. Since each tree in F' contains an even number of vertices C does (w.r.t. the original graph), as well. So v_i and $C - v_i$ each contains an even number of vertices. As a result, removing the edge between v_i and $C - v_i$ would leave even sized components, thus contradicting the minimality of F'.

$\square$

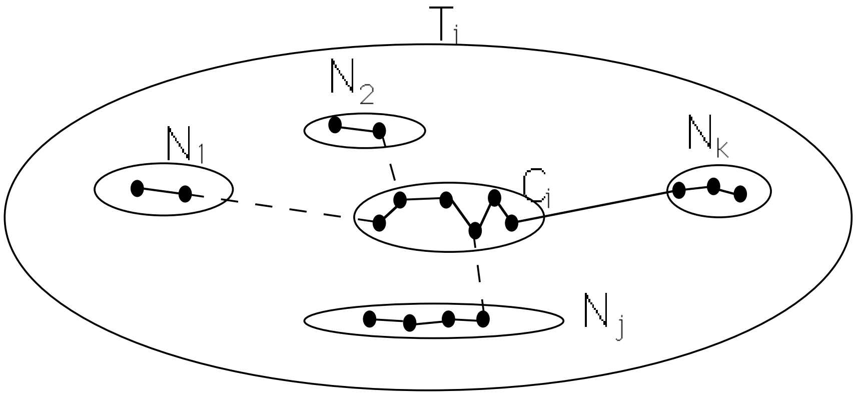


Figure 4: $d_H(v) \geq 2$ for $v \in Even$

4.7 Some implementation details

The algorithm can be implemented in $O(n^2 \log n)$. For this purpose, notice that the number of iterations is at most $n - 1$ since F is a forest. The components of F can be maintained as a union-find structure and, therefore, all mergings of components take $O(n\alpha(n,n))$ where α is the inverse Ackermann function. In order to get the $O(n^2 \log n)$ bound, we shall show that every iteration can be implemented in $O(n \log n)$ time.

In order to find the edge e to add to F, we maintain a priority queue containing the edges between different components of F. This initialization of this priority queue takes $O(n^2 \log n)$ time. In order to describe the key of an edge, we need to introduce a notion of *time*. The time is initially set to zero and increases by δ in each iteration (the time can be seen to be the maximum of d_i over all vertices i). The key of an edge $e = (i, j)$ is equal to the time at which we would have $c_e = d_i + d_j$ if the parity of the components containing i and j don't change. The edge to be selected is therefore the edge with minimum key and can be obtained in $O(n \log n)$. When two components merge, we need to update the keys of edges incident to the resulting component (since the parity might have changed). By keeping track of only one edge between two components (the one with smallest key), we need to update the keys of $O(n)$ edges when two components merge. This can be done in $O(n \log n)$ ($O(\log n)$ per update).

To complete the discussion, we need to show how to go from F to F'. By performing a post-order traversal of the tree and computing the parity of the subtrees encountered (in a recursive manner), this step can be implemented in $O(n)$ time.

5 Approximating MAX-CUT

In this section, we illustrate the fact that improved approximation algorithms can be obtained by considering relaxations more sophisticated than linear ones. At the same time, we will also illustrate the fact that rounding a solution from the relaxation in a randomized fashion can be very useful. For this purpose, we consider approximation algorithms for the MAX-CUT problem. The unweighted version of this problem is as follows:

Given: A graph $G = (V, E)$.
Find: A partition $(S, \bar{S})$ such that $d(S) := |\delta(S)|$ is maximized.

It can be shown that this problem is NP-hard and MAX SNP-complete and so we cannot hope for an approximation algorithm with guarantee arbitrarily close to 1 unless $P = NP$. In the weighted version of the problem each edge has a weight w_{ij} and we define $d(S)$ by,

$$d(S) = \sum_{(i,j)\in E: i\in S, j\notin S} w_{ij}.$$

For simplicity we focus on the unweighted case. The results that we shall obtain will also apply to the weighted case.

Recall that an α-approximation algorithm for MAX-CUT is a polynomial time algorithm which delivers a cut $\delta(S)$ such that $d(S) \geq \alpha z_{MC}$ where z_{MC} is the value of the optimum cut. Until 1993 the best known α was 0.5 but now it is 0.878 due to an approximation algorithm of Goemans and Williamson [14]. We shall first of all look at three (almost identical) algorithms which have an approximation ratio of 0.5.

1. **Randomized construction.** We select S uniformly from all subsets of V. i.e. For each $i \in V$ we put $i \in S$ with probability $\frac{1}{2}$ (independently of $j \neq i$).

$$\begin{aligned} E\left[d(S)\right] &= \textstyle\sum_{(i,j)\in E} Pr\left[(i,j) \in \delta(S)\right] \qquad \text{by linearity of expectations} \\ &= \textstyle\sum_{(i,j)\in E} Pr\left[i \in S, j \notin S \text{ or } i \notin S, j \in S\right] \\ &= \tfrac{1}{2}|E|. \end{aligned}$$

But clearly $z_{MC} \leq |E|$ and so we have $E\left[d(S)\right] \geq \frac{1}{2} z_{MC}$. Note that by comparing our cut to $|E|$, the best possible bound that we could obtain is $\frac{1}{2}$ since for K_n (the complete graph on n vertices) we have $|E| = \binom{n}{2}$ and $z_{MC} = \frac{n^2}{4}$.

2. **Greedy procedure.** Let $V = \{1, 2, \ldots, n\}$ and let $E_j = \{i : (i,j) \in E \text{ and } i < j\}$. It is clear that $\{E_j : j = 2, \ldots, n\}$ forms a partition of E. The algorithm is:

```
Set S = {1}
For j = 2 to n do
    if |S ∩ E_j| ≤ ½|E_j|
        then S ← S ∪ {j}.
```

If we define $F_j = E_j \cap \delta(S)$ then we can see that $\{F_j : j = 2, \ldots, n\}$ is a partition of $\delta(S)$. By definition of the algorithm it is clear that $|F_j| \geq \frac{|E_j|}{2}$. By summing over j we get $d(S) \geq \frac{|E|}{2} \geq \frac{z_{MC}}{2}$. In fact, the greedy algorithm can be obtained from the randomized algorithm by using the method of conditional expectations.

3. **Local search.** Say that $\delta(S)$ is locally optimum if $\forall i \in S : d(S - \{i\}) \leq d(S)$ and $\forall i \notin S : d(S \cup \{i\}) \leq d(S)$.

   Lemma 6 *If $\delta(S)$ is locally optimum then $d(S) \geq \frac{|E|}{2}$.*

   Proof:

$$
\begin{aligned}
d(S) &= \frac{1}{2} \sum_{i \in V} \{\text{number of edges in cut incident to } i\} \\
&= \frac{1}{2} \sum_{i \in V} |\delta(S) \cap \delta(i)| \\
&\geq \frac{1}{2} \sum_{i \in V} \frac{1}{2} d(i)| \\
&= \frac{2}{4} |E| \\
&= \frac{|E|}{2}.
\end{aligned}
$$

   The inequality is true because if $|\delta(S) \cap \delta(i)| < \frac{1}{2}|\delta(i)|$ for some i then we can move i to the other side of the cut and get an improvement. This contradicts local optimality. □

   In local search we move one vertex at a time from one side of the cut to the other until we reach a local optimum. In the unweighted case this is a polynomial time algorithm since the number of different values that a cut can take is $O(n^2)$. In the weighted case the running time can be exponential. Haken and Luby [15] have shown that this can be true even for 4-regular graphs. For cubic graphs the running time is polynomial [22].

Over the last 15-20 years a number of small improvements were made in the approximation ratio obtainable for MAX-CUT. The ratio increased in the following manner:

$$
\frac{1}{2} \rightarrow \frac{1}{2} + \frac{1}{2m} \rightarrow \frac{1}{2} + \frac{1}{2n} \rightarrow \frac{1}{2} + \frac{n-1}{4m}
$$

where $m = |E|$ and $n = |V|$, but asymptotically this is still 0.5.

5.1 Randomized 0.878 Algorithm

The algorithm that we now present is randomized but it differs from our previous randomized algorithm in two important ways.

- The event $i \in S$ is not independent from the event $j \in S$.
- We compare the cut that we obtain to an upper bound which is better that $|E|$.

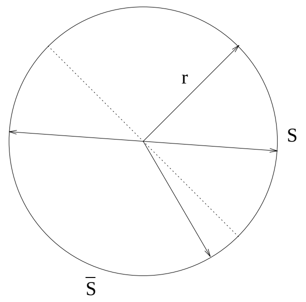


Figure 5: The sphere S_n.

Suppose that for each vertex $i \in V$ we have a vector $v_i \in \mathbb{R}^n$ (where $n = |V|$). Let S_n be the unit sphere $\{x \in \mathbb{R}^n : ||x|| = 1\}$. Take a point r uniformly distributed on S_n and let $S = \{i \in V : v_i \cdot r \geq 0\}$ (Figure 5). (Note that without loss of generality $||v_i|| = 1$.) Then by linearity of expectations:

$$E\left[d(S)\right] = \sum_{(i,j)\in E} Pr\left[\text{sign}(v_i \cdot r) \neq \text{sign}(v_j \cdot r)\right]. \tag{5}$$

Lemma 7

$$\begin{aligned} Pr\left[sign(v_i \cdot r) \neq sign(v_j \cdot r)\right] &= Pr\left[\textit{random hyperplane separates } v_i \textit{ and } v_j\right] \\ &= \frac{\alpha}{\pi} \end{aligned}$$

where $\alpha = \arccos(v_i \cdot v_j)$ *(the angle between* v_i *and* v_j*).*

Proof: This result is easy to see but it is a little difficult to formalize. Let P be the 2-dimensional plane containing v_i and v_j. Then $P \cap S_n$ is a circle. With probability 1, $H = \{x : x \cdot r = 0\}$ intersects this circle in exactly two points s and t (which are diametrically opposed). See figure 6. By symmetry s and t are uniformly distributed on the circle. The vectors v_i and v_j are separated by the hyperplane H if and only if either s or t lies on the smaller arc between v_i and v_j. This happens with probability $\frac{2\alpha}{2\pi} = \frac{\alpha}{\pi}$. $\square$

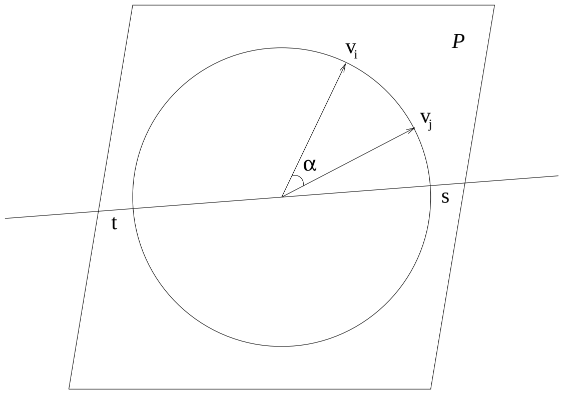


Figure 6: The plane P.

From equation 5 and lemma 7 we obtain:

$$E\left[d(S)\right] = \sum_{(i,j)\in E} \frac{\arccos(v_i \cdot v_j)}{\pi}.$$

Observe that $E\left[d(S)\right] \leq z_{MC}$ and so

$$\max_{v_i} E\left[d(S)\right] \leq z_{MC},$$

where we maximize over all choices for the v_i's. We actually have $\max_{v_i} E\left[d(S)\right] = z_{MC}$. Let $\delta(T)$ be a cut such that $d(T) = z_{MC}$ and let e be the unit vector whose first component is 1 and whose other components are 0. If we set

$$v_i = \begin{cases} e & \text{if } i \in T \\ -e & \text{otherwise.} \end{cases}$$

then $\delta(S) = \delta(T)$ with probability 1. This means that $E\left[d(S)\right] = z_{MC}$.

Corollary 8

$$z_{MC} = \max_{||v_i||=1} \sum_{(i,j)\in E} \frac{\arccos(v_i \cdot v_j)}{\pi}.$$

Unfortunately this is as difficult to solve as the original problem and so at first glance we have not made any progress.

5.2 Choosing a good set of vectors

Let $f : [-1, 1] \to [0, 1]$ be a function which satisfies $f(-1) = 1$, $f(1) = 0$. Consider the following program:

$$
\begin{array}{llr}
 & \text{Max} \quad \displaystyle\sum_{(i,j)\in E} f(v_i \cdot v_j) & \\
 & \text{subject to:} & \\
(P) & \qquad ||v_i|| = 1 & i \in V.
\end{array}
$$

If we denote the optimal value of this program by z_P then we have $z_{MC} \leq z_P$. This is because if we have a cut $\delta(T)$ then we can let,

$$v_i = \begin{cases} e & \text{if } i \in T \\ -e & \text{otherwise.} \end{cases}$$

Hence $\sum_{(i,j)\in E} f(v_i \cdot v_j) = d(T)$ and $z_{MC} \leq z_P$ follows immediately.

5.3 The Algorithm

The framework for the 0.878 approximation algorithm for MAX-CUT can now be presented.

1. Solve (P) to get a set of vectors $\{v_1^*, \ldots, v_n^*\}$.
2. Uniformly select $r \in S_n$.
3. Set $S = \{i : v_i^* \cdot r \geq 0\}$.

Theorem 9

$$E\left[d(S)\right] \geq \alpha z_P \geq \alpha z_{MC}$$

where,

$$\alpha = \min_{-1\leq x\leq 1} \frac{\arccos(x)}{\pi f(x)}.$$

Proof:

$$
\begin{aligned}
E\left[d(S)\right] &= \sum_{(i,j)\in E} \frac{\arccos(v_i^* \cdot v_j^*)}{\pi} \\
&\geq \alpha \sum_{(i,j)\in E} f(v_i^* \cdot v_j^*) \\
&= \alpha z_P \\
&\geq \alpha z_{MC}.
\end{aligned}
$$

$\square$

We must now choose f such that (P) can be solved in polynomial time and α is as large as possible. We shall show that (P) can be solved in polynomial time whenever f is linear and so if we define,

$$f(x) = \frac{1-x}{2}$$

then our first criterion is satisfied. (Note that $f(-1) = 1$ and $f(1) = 0$.) With this choice of f,

$$\begin{aligned}\alpha &= \min_{-1 \le x \le 1} \frac{2\arccos(x)}{\pi(1-x)} \\ &= \frac{2\arccos(-0.689)}{\pi(1-(-0.689))} \\ &= 0.87856.\end{aligned}$$

(See figure 7.)

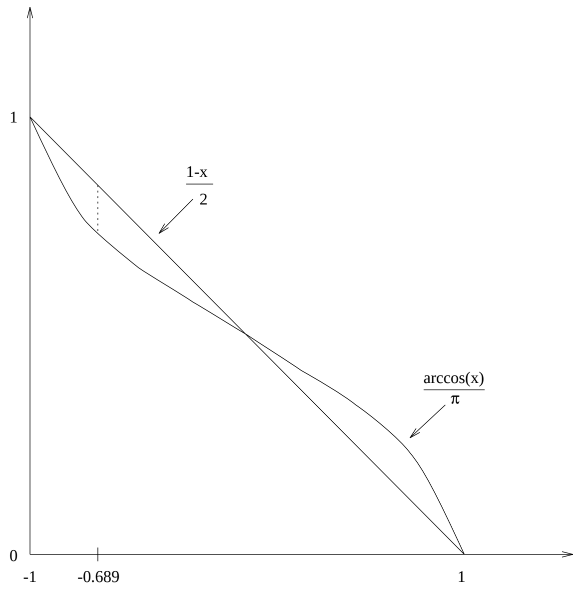


Figure 7: Calculating α.

5.4 Solving (P)

We now turn our attention to solving:

$$\text{Max} \quad \sum_{(i,j)\in E} \frac{1}{2}(1 - v_i \cdot v_j)$$

subject to:

$$(P) \qquad ||v_i|| = 1 \qquad i \in V.$$

Let $Y = (y_{ij})$ where $y_{ij} = v_i \cdot v_j$. Then,

- $||v_i|| = 1 \Rightarrow y_{ii} = 1$ for all i.
- $y_{ij} = v_i \cdot v_j \Rightarrow Y \succeq 0$, where $Y \succeq 0$ means that Y is positive semi-definite: $\forall x : x^T Y x \geq 0$.) This is true because,

$$\begin{aligned} x^T Y x &= \sum_i \sum_j x_i x_j (v_i \cdot v_j) \\ &= \left\| \sum_i x_i v_i \right\|^2 \\ &\geq 0. \end{aligned}$$

Conversely if $Y \succeq 0$ and $y_{ii} = 1$ for all i then it can be shown that there exists a set of v_i's such that $y_{ij} = v_i \cdot v_j$. Hence (P) is equivalent to,

$$(P') \qquad \begin{aligned} \text{Max} \quad & \sum_{(i,j) \in E} \frac{1}{2}(1 - y_{ij}) \\ \text{subject to:} \quad & \\ & Y \succeq 0 \\ & y_{ii} = 1 \qquad\qquad i \in V. \end{aligned}$$

Note that $Q := \{Y : Y \succeq 0, y_{ii} = 1\}$ is convex. (If $A \succeq 0$ and $B \succeq 0$ then $A + B \succeq 0$ and also $\frac{A+B}{2} \succeq 0$.) It can be shown that maximizing a concave function over a convex set can be done in polynomial time. Hence we can solve (P') in polynomial time since linear functions are concave. This completes the analysis of the algorithm.

5.5 Remarks

1. The optimum Y could be irrational but in this case we can find a solution with an arbitrarily small error in polynomial time.

2. To solve (P') in polynomial time we could use a variation of the interior point method for linear programming.

3. Given Y, v_i can be obtained using a Cholesky factorization ($Y = VV^T$).

4. The algorithm can be derandomized using the method of conditional expectations. This is quite intricate.

5. The analysis is very nearly tight. For the 5-cycle we have z_{MC} and $z_P = \frac{5}{2}(1 + \cos\frac{\pi}{5}) = \frac{25+5\sqrt{5}}{8}$ which implies that $\frac{z_{MC}}{z_P} = 0.88445$.

6 Bin Packing and $P \parallel C_{max}$

One can push the notion of approximation algorithms a bit further than we have been doing and define the notion of approximation schemes:

Definition 4 *A* polynomial approximation scheme (pas) *is a family of algorithms $A_\epsilon : \epsilon > 0$ such that for each $\epsilon > 0$, A_ϵ is a $(1+\epsilon)$-approximation algorithm which runs in polynomial time in input size for fixed ϵ.*

Definition 5 *A* fully polynomial approximation scheme (fpas) *is a* pas *with running time which is polynomial both in input size and $1/\epsilon$.*

It is known that if π is a strongly NP-complete problem, then π has no *fpas* unless $P = NP$. From the result of Arora et al. described in Section 2, we also know that there is no *pas* for any $MAX - SNP$ hard problem unless $P = NP$.

We now consider two problems which have a very similar flavor; in fact, they correspond to the same NP-complete decision problem. However, they considerably differ in terms of approximability: one has a *pas*, the other does not.

- **Bin Packing**: Given item sizes $a_1, a_2, \ldots, a_n \geq 0$ and a bin size of T, find a partition of $I_1, \ldots, I_k$ of $1, \ldots, n$, such that $\sum_{i \in I_l} a_i \leq T$ and k is minimized (the items in I_l are assigned to bin l).
- $\boldsymbol{P \parallel C_{max}}$: Given n jobs with processing times $p_1, \ldots, p_n$ and m machines, find a partition $\{I_1, \ldots, I_m\}$ of $\{1, \ldots, n\}$, such that the makespan defined as $\max_i(\sum_{j \in I_i} p_j)$ is minimum. (The makespan represents the maximum completion time on any machine given that the jobs in I_i are assigned to machine i).

The decision versions of the two problems are identical and NP-complete. However when we consider approximation algorithms for the two problems we have completely different results. In the case of the bin packing problem there is no α-approximation algorithm with $\alpha < 3/2$, unless $P = NP$.

Proposition 10 *There is no α-approximation algorithm with $\alpha < 3/2$, for bin packing, unless $P = NP$, as seen in Section 2.*

However, we shall see, for $P \parallel C_{max}$ we have α-approximation algorithms for any α.

Definition 6 *An algorithm* **A** *has an asymptotic performance guarantee of α if*

$$\alpha \geq \limsup_{k \to \infty} \alpha_k$$

where

$$\alpha_k = \sup_{I:OPT(I)=k} \frac{A(I)}{OPT(I)}$$

and $OPT(I)$ denotes the value of instance I and $A(I)$ denotes the value returned by algorithm **A**.

For $P \parallel C_{max}$, there is no difference between an asymptotic performance and a performance guarantee. This follows from the fact that $P \parallel C_{max}$ satisfies a scaling property : an instance with value $\beta OPT(I)$ can be constructed by multiplying every processing time p_j by β.

Using this definition we can analogously define a *polynomial asymptotic approximation scheme (paas)*. And a *fully polynomial asymptotic approximation scheme (fpaas)*.

Now we will state some results to illustrate the difference in the two problems when we consider approximation algorithms.

1. For bin packing, there does not exist an α-approximation algorithm with $\alpha < 3/2$, unless $P = NP$. Therefore there is no *pas* for bin packing unless $P = NP$.

2. For $P \parallel C_{max}$ there exists a *pas*. This is due to Hochbaum and Shmoys [17]. We will study this algorithm in more detail in today's lecture.

3. For bin packing there exists a *paas*. (Fernandez de la Vega and Lueker [7]).

4. For $P \parallel C_{max}$ there exists no *fpaas* unless $P = NP$. This is because the existence of a *fpaas* implies the existence of a *fpas* and the existence of a *fpas* is ruled out unless $P = NP$ because, $P \parallel C_{max}$ is strongly NP-complete.

5. For bin packing there even exists a *fpaas*. This was shown by Karmarkar and Karp [18].

6.1 Approximation algorithm for $P||C_{max}$

We will now present a polynomial approximation scheme for the $P||C_{max}$ scheduling problem.

We analyze a pas for $P||C_{max}$, discovered by Hochbaum and Shmoys [17]. The idea is to use a relation similar to the one between an optimization problem and its decision problem. That is, if we have a way to solve decision problem, we can use binary search to find the exact solution. Similarly, in order to obtain a $(1+\epsilon)$-approximation algorithm, we are going to use a so-called $(1+\epsilon)$-relaxed decision version of the problem and binary search.

Definition 7 *$(1+\epsilon)$-relaxed decision version of $P||C_{max}$ is a procedure that given ϵ and a deadline T, returns either:*

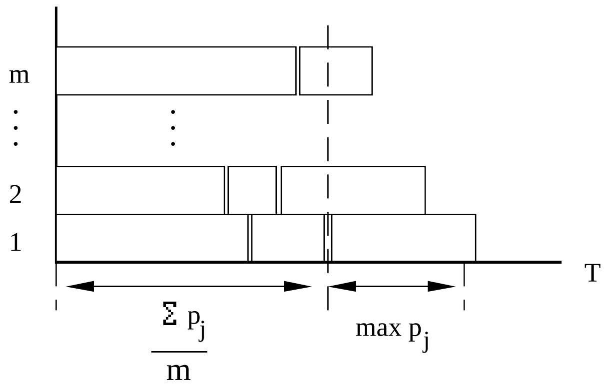


Figure 8: List scheduling.

- "NO" — *if there does not exist a schedule with makespan* $\leq T$*, or*
- "YES" — *if a schedule with makespan* $\leq (1+\epsilon)T$ *exists.*

In case of "yes", the actual schedule must also be provided.

Notice that in some cases both answers are valid. In such a case, we do not care if the procedure outputs "yes" or "no". Suppose we have such a procedure. Then we use binary search to find the solution. To begin our binary search, we must find an interval where optimal $C_{\max}$ is contained. Notice that $\left(\sum_j p_j\right)/m$ is an average load per machine and $\max_j p_j$ is the length of the longest job. We can put a bound on optimum $C_{\max}$ as follows:

Lemma 11 *Let*

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

then $L \leq C_{\max} < 2L$*.*

Proof: Since the longest job must be completed, we have $\max_j p_j \leq C_{\max}$. Also, since $\left(\sum_j p_j\right)/m$ is the average load, we have $\left(\sum_j p_j\right)/m \leq C_{\max}$. Thus, $L \leq C_{\max}$.

The upper bound relies on the concept of list scheduling, which dictates that a job is never processed on some machine, if it can be processed earlier on another machine. That is, we require that if there is a job waiting, and an idle machine, we must use this machine to do the job. We claim that for such a schedule $C_{\max} < 2L$. Consider the job that finishes last, say job k. Notice that when it starts, *all* other machines are busy. Moreover, the time elapsed up to that point is no more than the average load of the machines (see Figure 8). Therefore,

$$
\begin{aligned}
C_{\max} &\leq \frac{\sum_{j \neq k} p_j}{m} + p_k \\
&< \frac{\sum_j p_j}{m} + \max_j p_j \\
&\leq 2 \max\left(\max_j p_j, \frac{\sum_j p_j}{m} \right) \\
&= 2L.
\end{aligned}
$$

□

Now we have an interval on which to do a (logarithmic) binary search for $C_{\max}$. By T_1 and T_2 we denote lower and upper bound pointers we are going to use in our binary search. Clearly, $T = \sqrt{T_1 T_2}$ is the midpoint in the logarithmic sense. Based on Lemma 11, we must search for the solution in the interval $[L, \ldots, 2L]$. Since we use logarithmic scale, we set $\log T_1 = \log_2 L$, $\log T_2 = \log_2 L + 1$ and $\log T = \frac{1}{2}(\log_2 T_1 + \log_2 T_2)$.

When do we stop? The idea is to use different value of ϵ. That is, the approximation algorithm proceeds as follows. Every time, the new interval is chosen depending on whether the procedure for the $(1+\epsilon/2)$-relaxed decision version returns a "no" or (in case of "yes") a schedule with makespan $\leq (1+\epsilon/2)T$, where $T = \sqrt{T_1 T_2}$ and $[T_1, \ldots, T_2]$ is the current interval. The binary search continues until the bounds T_1, T_2 satisfy the relation $\frac{T_2(1+\epsilon/2)}{T_1} \leq (1+\epsilon)$, or equivalently $\frac{T_2}{T_1} \leq \frac{1+\epsilon}{1+\epsilon/2}$. The number of iterations required to satisfy this relation is $O(\lg(1/\epsilon))$. Notice that this value is a *constant* for a *fixed* ϵ. At termination, the makespan of the schedule corresponding to T_2 will be within a factor of $(1+\epsilon)$ of the optimal makespan.

In order to complete the analysis of the algorithm, it remains to describe the procedure for the $(1+\epsilon/2)$-relaxed decision procedure for any ϵ. Intuitively, if we look at what can go wrong in list scheduling, we see that it is "governed" by "long" jobs, since small jobs can be easily accommodated. This is the approach we take, when designing procedure that solves the $(1+\epsilon/2)$-relaxed decision version of the problem. For the rest of our discussion we will denote $\epsilon/2$ by ϵ'.

Given $\{p_j\}$, ϵ' and T, the procedure operates as follows:

Step 1: *Remove all (small) jobs with $p_j \leq \epsilon' T$.*

Step 2: *Somehow (to be specified later) solve the $(1+\epsilon')$-relaxed decision version of the problem for the remaining (big) jobs.*

Step 3: *If answer in step 2 is "no", then return that there does not exist a schedule with makespan $\leq T$.*
If answer in step 2 is "yes", then with a deadline of $(1+\epsilon')T$ put back all small jobs using list scheduling (i.e. greedy strategy), one at a time. If all jobs are

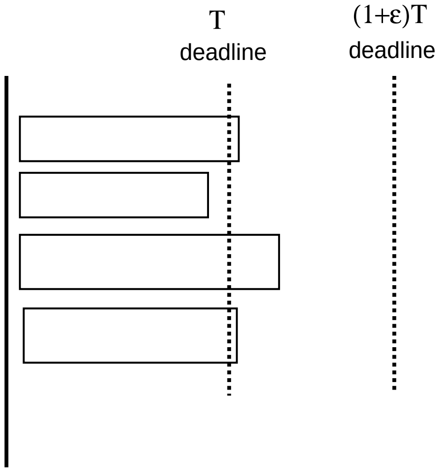


Figure 9: Scheduling "small" jobs.

accommodated then return that schedule, else return that there does not exist a schedule with makespan $\leq T$*.*

Step 3 of the algorithm gives the final answer of the procedure. In case of a "yes" it is clear that the answer is correct. In case of a "no" that was propagated from Step 2 it is also clear that the answer is correct. Finally, if we fail to put back all the small jobs we must also show that the algorithm is correct. Let us look at a list schedule in which some small jobs have been scheduled but others couldn't (see Figure 9).

If we cannot accomodate all small jobs with a deadline of $(1+\epsilon')T$, it means that all machines are busy at time T since the processing time of each small job is $\leq \epsilon' T$. Hence, the average load per processor exceeds T. Therefore, the answer "no" is correct.

Now, we describe Step 2 of the algorithm for $p_j > \epsilon' T$. Having eliminated the small jobs, we obtain a constant (when ϵ is fixed) upper bound on the number of jobs processed on one machine. Also, we would like to have only a small number of distinct processing times in order to be able to enumerate in polynomial time all possible schedules. For this purpose, the idea is to use rounding. Let q_j be the largest number of the form $\epsilon' T + k\epsilon'^2 T \leq p_j$ for some $k \in \mathbf{N}$. A refinement of Step 2 is the following.

2.1 Address the decision problem: Is there a schedule for $\{q_j\}$ with makespan $\leq T$?

2.2 If the answer is "no", then return that there does not exist a schedule with makespan $\leq T$.
If the answer is "yes", then return that schedule.

The Lemma that follows justifies the correctness of the refined Step 2.

Lemma 12 *Step 2 of the algorithm is correct.*

Proof: If Step 2.1 returns "no", then it is clear that the final answer of Step 2 should be "no", since $q_j \leq p_j$.

If Step 2.1 returns "yes", then the total increase of the makespan due to the replacement of q_j by p_j is no greater than $(1/\epsilon')\epsilon'^2 T = \epsilon' T$. This is true, because we have at most $T/(\epsilon' T) = 1/\epsilon'$ jobs per machine, and because $p_j \leq q_j + \epsilon'^2 T$ by definition. Thus, the total length of the schedule with respect to $\{p_j\}$ is at most $T + \epsilon' T = (1 + \epsilon')T$. □

It remains to show how to solve the decision problem of Step 2.1. We can achieve this in polynomial time using dynamic programming. Note that the input to this decision problem is "nice": We have at most $P = \lfloor 1/\epsilon' \rfloor$ jobs per machine, and at most $Q = 1 + \left\lfloor \frac{1-\epsilon'}{\epsilon'^2} \right\rfloor$ distinct processing times. Since ϵ' is considered to be fixed, we essentially have a constant number of jobs per machine and a constant number $q'_1, \ldots, q'_Q$ of processing times. Let $\vec{n} = \{n_1, \ldots, n_Q\}$, where n_i denotes the number of jobs whose processing time is q_i. We use the fact that the decision problems of $P||C_{max}$ and the bin packing problems are equivalent. Let $f(\vec{n})$ denote the minimum number of machines needed to process $\vec{n}$ by time T. Finally, let $R = \{\vec{r} = (r_1, \ldots, r_Q) : \sum_{i=1}^{Q} r_i q'_i \leq T, r_i \leq n_i, r_i \in \mathbb{N}$. R represents the sets of jobs that can be processed on a single machine with a deadline of T. The recurrence for the dynamic programming formulation of the problem is

$$f(\vec{n}) = 1 + \min_{\vec{r} \in R} f(\vec{n} - \vec{r}),$$

namely we need one machine to accomodate the jobs in $\vec{r} \in R$ and $f(\vec{n} - \vec{r})$ machines to accomodate the remaining jobs. In order to compute this recurrence we first have to compute the at most Q^P vectors in R. The upper bound on the size of R comes from the fact that we have at most P jobs per machine and each job can have one of at most Q processing times. Subsequently, for each one of the vectors in R we have to iterate for n^Q times, since $n_i \leq n$ and there are Q components in $\vec{n}$. Thus, the running time of Step 2.1 is $O(n^{1/\epsilon'^2}(1/\epsilon'^2)^{(1/\epsilon')})$.

From this point we can derive the overall running time of the pas in a straightforward manner. Since Step 2 iterates $O(\lg(1/\epsilon))$ times and since $\epsilon = 2\epsilon'$, the overall running time of the algorithm is $O(n^{1/\epsilon^2}(1/\epsilon^2)^{(1/\epsilon)} \lg(1/\epsilon))$.

7 Randomized Rounding for Multicommodity Flows

In this section, we look at using randomness to approximate a certain kind of multicommodity flow problem. The problem is as follows: given a directed graph $G = (V, E)$, with sources $s_i \in V$ and sinks $t_i \in V$ for $i = 1, \ldots, k$, we want to find a path P_i from s_i to t_i for $1 \leq i \leq k$ such that the "width" or "congestion" of any edge is as small as possible. The "width" of an edge is defined to be the number of paths using that edge. This multicommodity flow problem is NP-complete in general.

The randomized approximation algorithm that we discuss in these notes is due to Raghavan and Thompson [24].

7.1 Reformulating the problem

The multicommodity flow problem can be formulated as the following integer program:

$$
\begin{array}{llll}
 & \text{Min} \quad W & & \\
 & \text{subject to:} & & \\
(6) & \sum_w x_i(v,w) - \sum_w x_i(w,v) = \begin{cases} 1 & \text{if } v = s_i \\ -1 & \text{if } v = t_i \\ 0 & \text{otherwise} \end{cases} & & i = 1,\ldots,k, v \in V, \\
 & x_i(v,w) \in \{0,1\} & & i = 1,\ldots,k, (v,w) \in E, \\
(7) & \sum_i x_i(v,w) \leq W & & (v,w) \in E.
\end{array}
$$

Notice that constraint (6) forces the x_i to define a path (perhaps not simple) from s_i to t_i. Constraint (7) ensures that every edge has width no greater than W, and the overall integer program minimizes W.

We can consider the LP relaxation of this integer program by replacing the constraints $x_i(v,w) \in \{0,1\}$ with $x_i(v,w) \geq 0$. The resulting linear program can be solved in polynomial time by using interior point methods discussed earlier in the course. The resulting solution may not be integral. For example, consider the multicommodity flow problem with one source and sink, and suppose that there are exactly i edge-disjoint paths between the source and sink. If we weight the edges of each path by $\frac{1}{i}$ (i.e. set $x(v,w) = \frac{1}{i}$ for each edge of each path), then $W_{LP} = \frac{1}{i}$. The value W_{LP} can be no smaller: since there are i edge-disjoint paths, there is a cut in the graph with i edges. The average flow on these edges will be $\frac{1}{i}$, so that the width will be at least $\frac{1}{i}$.

The fractional solution can be decomposed into paths using *flow decomposition*, a standard technique from network flow theory. Let x be such that $x \geq 0$ and

$$\sum_w x(v,w) - \sum_w x(w,v) = \begin{cases} a & \text{if } v = s_i \\ -a & \text{if } v = t_i \\ 0 & \text{otherwise.} \end{cases}$$

Then we can find paths $P_1, \ldots, P_l$ from s_i to t_i such that

$$
\begin{aligned}
\alpha_1, \ldots, \alpha_l &\in \mathbb{R}^+ \\
\sum_i \alpha_i &= a \\
\sum_{j:(v,w)\in P_j} \alpha_j &\leq x(v,w).
\end{aligned}
$$

To see why we can do this, suppose we only have one source and one sink, s and t. Look at the "residual graph" of x: that is, all edges (v, w) such that $x(v, w) > 0$. Find some path P_1 from s to t in this graph. Let $\alpha_1 = \min_{(v,w)\in P_1} x(v, w)$. Set

$$x'(v, w) = \begin{cases} x(v, w) - \alpha_1 & (v, w) \in P_1 \\ x(v, w) & \text{otherwise.} \end{cases}$$

We can now solve the problem recursively with $a' = a - \alpha_1$.

7.2 The algorithm

We now present Raghavan and Thompson's randomized algorithm for this problem.

1. Solve the LP relaxation, yielding W_{LP}.
2. Decompose the fractional solution into paths, yielding paths P_{ij} for $i = 1, \dots, k$ and $j = 1, \dots, j_i$ (where P_{ij} is a path from s_i to t_i), and yielding $\alpha_{ij} > 0$ such that $\sum_j \alpha_{ij} = 1$ and
$$\sum_i \sum_{j:(v,w)\in P_{ij}} \alpha_{ij} \le W_{LP}.$$
3. (Randomization step) For all i, cast a j_i-faced die with face probabilities α_{ij}. If the outcome is face f, select path P_{if} as the path P_i from s_i to t_i.

We will show, using a Chernoff bound, that with high probability we will get small congestion. Later we will show how to derandomize this algorithm. To carry out the derandomization it will be important to have a strong handle on the Chernoff bound and its derivation.

7.3 Chernoff bound

For completeness, we include the derivation of a Chernoff bound, although it already appears in the randomized algorithms chapter.

Lemma 13 *Let X_i be independent Bernoulli random variables with probability of success p_i. Then, for all $\alpha > 0$ and all $t > 0$, we have*

$$Pr\left[\sum_{i=1}^k X_i > t\right] \le e^{-\alpha t} \prod_{i=1}^k E\left[e^{\alpha X_i}\right] = e^{-\alpha t} \prod_{i=1}^k \left(p_i e^\alpha + (1 - p_i)\right).$$

Proof:

$$Pr\left[\sum_{i=1}^k X_i > t\right] = Pr\left[e^{\alpha \sum_{i=1}^k X_i} > e^{\alpha t}\right]$$

for any $\alpha > 0$. Moreover, this can be written as $Pr[Y > a]$ with $Y \geq 0$. From Markov's inequality we have

$$Pr[Y > a] \leq \frac{E[Y]}{a}$$

for any nonnegative random variable. Thus,

$$\begin{aligned} Pr\left[\sum_{i=1}^{k} X_i > t\right] &\leq e^{-\alpha t} E\left[e^{\alpha \sum_i X_i}\right] \\ &= e^{-\alpha t} \prod_{i=1}^{k} E\left[e^{\alpha X_i}\right] \text{ because of independence.} \end{aligned}$$

The equality then follows from the definition of expectation. □

Setting $t = (1+\beta)E[\sum_i X_i]$ for some $\beta > 0$ and $\alpha = \ln(1+\beta)$, we obtain:

Corollary 14 *Let X_i be independent Bernoulli random variables with probability of success p_i, and let $M = E[\sum_{i=1}^{k} X_1] = \sum_{i=1}^{k} p_i$. Then, for all $\beta > 0$, we have*

$$Pr\left[\sum_{i=1}^{k} X_i > (1+\beta)M\right] \leq (1+\beta)^{-(1+\beta)M} \prod_{i=1}^{k} E\left[(1+\beta)^{X_i}\right] \leq \left[\frac{e^{\beta}}{(1+\beta)^{(1+\beta)}}\right]^M.$$

The second inequality of the corollary follows from the fact that

$$E\left[(1+\beta)^{X_i}\right] = p_i(1+\beta) + (1-p_i) = 1 + \beta p_i \leq e^{\beta p_i}.$$

7.4 Analysis of the R-T algorithm

Raghavan and Thompson show the following theorem.

Theorem 15 *Given $\epsilon > 0$, if the optimal solution to the multicommodity flow problem W^* has value $W^* = \Omega(\log n)$ where $n = |V|$, then the algorithm produces a solution of width $W \leq W^* + c\sqrt{W^* \ln n}$ with probability $1-\epsilon$ (where c and the constant in $\Omega(\log n)$ depends on ϵ, see the proof).*

Proof: Fix an edge $(v,w) \in E$. Edge (v,w) is used by commodity i with probability $p_i = \sum_{j:(v,w)\in P_{ij}} \alpha_{ij}$. Let X_i be a Bernoulli random variable denoting whether or not (v,w) is in path P_i. Then $W(v,w) = \sum_{i=1}^{k} X_i$, where $W(v,w)$ is the width of edge (v,w). Hence,

$$E[W(v,w)] = \sum_i p_i = \sum_i \sum_{j:(v,w)\in P_{ij}} \alpha_{ij} \leq W_{LP} \leq W^*.$$

Now using the Chernoff bound derived earlier,

$$Pr[W(v,w) \geq (1+\beta)W^*] \leq \left[\frac{e^{\beta}}{(1+\beta)^{(1+\beta)}}\right]^{W^*} = e^{[\beta-(1+\beta)\ln(1+\beta)]W^*}.$$

Assume that $\beta \le 1$. Then, one can show that

$$\frac{e^{\beta}}{(1+\beta)^{(1+\beta)}} = e^{\beta-(1+\beta)\ln(1+\beta)} \le e^{-\beta^2/3}.$$

Therefore, for

$$\beta = \sqrt{\frac{3\ln\frac{n^2}{\varepsilon}}{W^*}},$$

we have that

$$Pr[W(v,w) \ge (1+\beta)W^*] \le \frac{\varepsilon}{n^2}.$$

Notice that our assumption that $\beta \le 1$ is met if

$$W^* \ge 6\ln n - 3\ln\varepsilon.$$

For this choice of β, we derive that

$$(1+\beta)W^* = W^* + \sqrt{3W^*\ln\frac{n^2}{\varepsilon}}.$$

We consider now the maximum congestion. We have

$$Pr[\max_{(v,w)\in E} W(v,w) \ge (1+\beta)W^*] \le \sum_{(v,w)\in E} Pr[W(v,w) \ge (1+\beta)W^*] \le |E|\frac{\varepsilon}{n^2} \le \varepsilon,$$

proving the result. □

7.5 Derandomization

We will use the method of conditional probabilities. We will need to supplement this technique, however, with an additional trick to carry through the derandomization. This result is due to Raghavan [23].

We can represent the probability space using a decision tree. At the root of the tree we haven't made any decisions. As we descend the tree from the root we represent the choices first for commodity 1, then for commodity 2, etc. Hence the root has j_1 children representing the j_1 possible paths for commodity 1. Each of these nodes has j_2 children, one for each of the j_2 possible paths for commodity 2. We continue in the manner, until we have reached level k. Clearly the leaves of this tree represent all the possible choices of paths for the k commodities.

A node at level i (the root is at level 0) is labeled by the i choices of paths for commodities $1 \ldots i : l_1 \ldots l_i$. Now we define:

$$g(l_1 \ldots l_i) = Pr\left[\max_{(v,w)\in E} W(v,w) \ge (1+\beta)W^* \;\middle|\; \begin{array}{l} l_1 \text{ for commodity } 1 \\ l_2 \text{ for commodity } 2 \\ \vdots \\ l_i \text{ for commodity } i \end{array}\right].$$

By conditioning on the choice of the path for commodity i, we obtain that

$$g(l_1 \dots l_{i-1}) = \sum_{j=1}^{j_i} \alpha_{ij} g(l_1, \dots, l_{i-1}, j) \tag{8}$$

$$\geq \min_j g(l_1, \dots, l_{i-1}, j) \tag{9}$$

If we could compute $g(l_1, l_2, \dots)$ efficiently, we could start from $g(\emptyset)$ and by selecting the minimum at each stage construct a sequence $g(\emptyset) \geq g(l_1) \geq g(l_1, l_2) \geq \dots \geq g(l_1, l_2, \dots, l_k)$. Unfortunately we don't know how to calculate these quantities. Therefore we need to use an additional trick.

Instead of using the exact value g, we shall use a *pessimistic estimator* for the probability of failure. From the derivation of the Chernoff bound and the analysis of the algorithm, we know that

$$(10)\ Pr\left[\max_{(v,w)\in E} W(v,w) \geq (1+\beta)W^*\right] \leq (1+\beta)^{-(1+\beta)W^*} \sum_{(v,w)\in E} E\left[\prod_{i=1}^{k} e^{\beta X_i^{(v,w)}}\right],$$

where the superscript on X_i denotes the dependence on the edge (v, w), i.e. $X_i^{(v,w)} = 1$ if (v, w) belongs to the path P_i. Letting $h(l_1, \dots, l_i)$ be the RHS of (10) when we condition on selecting path P_{jl_j} for commodity j, $j = 1, \dots, i$, we observe that:

1. $h(l_1, \dots, l_i)$ can be easily computed,
2. $g(l_1, \dots, l_i) \leq h(l_1, \dots, l_i)$ and
3. $h(l_1, \dots, l_i) \geq \min_j h(l_1, \dots, l_{i-1}, j)$.

Therefore, selecting the minimum in the last inequality at each stage, we construct a sequence such that $1 > \varepsilon \geq h(\emptyset) \geq h(l_1) \geq h(l_1, l_2) \geq \dots \geq h(l_1, l_2, \dots, l_k) \geq g(l_1, l_2, \dots, l_k)$. Since $g(l_1, l_2, \dots, l_k)$ is either 0 or 1 (there is no randomness involved), we must have that the choice of paths of this deterministic algorithm gives a maximum congestion less than $(1 + \beta)W^*$.

8 Multicommodity Flow

Consider an undirected graph $G = (V, E)$ with a capacity u_e on each edge. Suppose that we are given k commodities and a demand for f_i units of commodity i between two points s_i and t_i. In the area of multicommodity flow, one is interested in knowing whether all commodities can be shipped simultaneously. That is, can we find flows of value f_i between s_i and t_i such that the sum over all commodities of the flow on each edge (in either direction) is at most the capacity of the edge.

There are several variations of the problem. Here, we consider the concurrent flow problem: Find α^* where α^* is the maximum α such that for each commodity we can ship αf_i units from s_i to t_i. This problem can be solved by linear programming since all the constraints are linear. Indeed, one can have a flow variable for each edge and each commodity (in addition to the variable α), and the constraints consist of the flow conservation constraints for each commodity as well as a capacity constraint for every edge. An example is shown in figure 8. The demand for each commodity is 1 unit and the capacity on each edge is 1 unit. It can be shown that $\alpha^* = \frac{3}{4}$.

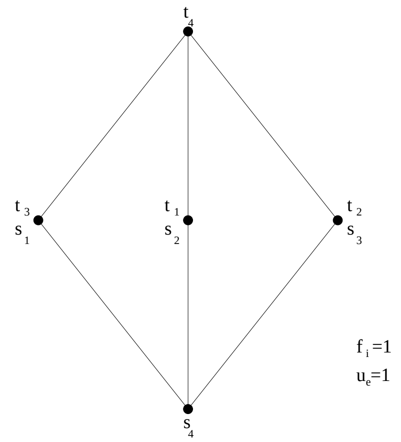


Figure 10: An example of the multi-commodity flow problem.

When there is only one commodity, we know that the maximum flow value is equal to the minimum cut value. Let us investigate whether there is such an analogue for multicommodity flow. Consider a cut $(S, \bar{S})$. As usual $\delta(S)$ is the set of edges with exactly one endpoint in S. Let,

$$f(S) = \sum_{i:|S \cap \{s_i, t_i\}|=1} f_i.$$

Since all flow between S and $\bar{S}$ must pass along one of the edges in $\delta(S)$ we must have,

$$\alpha^* \leq \frac{u(\delta(S))}{f(S)}$$

where $u(\delta(S)) = \sum_{e \in \delta(S)} u_e$. The multicommodity cut problem is to find a set S which minimizes $\frac{u(\delta(S))}{f(S)}$. We let β^* be the minimum value attainable and so we have $\alpha^* \leq \beta^*$. But, in general, we don't have equality. For example, in Figure 8, we have $\beta^* = 1$. In fact, it can be shown that the multicommodity cut problem is NP-hard. We shall consider the following two related questions.

1. In the worst case, how large can $\frac{\beta^*}{\alpha^*}$ be?

2. Can we obtain an approximation algorithm for the multicommodity cut problem?

In special cases, answers have been given to these questions by Leighton and Rao [19] and in subsequent work by many other authors. In this section, we describe a very recent, elegant and general answer due to Linial, London and Rabinovitch [20]. The technique they used is the embedding of metrics. The application to multicommodity flows was also independently obtained by Aumann and Rabani [3].

We first describe some background material on metrics and their embeddings.

Definition 8 *(X, d) is a metric space or d is a metric on a set X if*

1. $\forall x, y : d(x, y) \geq 0$.
2. $\forall x, y : d(x, y) = d(y, x)$.
3. $\forall x, y, z : d(x, y) + d(y, z) \geq d(x, z)$.

Strictly speaking we have defined a semi-metric since we do not have the condition $d(x, y) = 0 \Rightarrow x = y$. We will be dealing mostly with finite metric spaces, where X is finite. In $\mathbb{R}^n$ then the following are all metrics:

$$\begin{array}{rclcll} d(x,y) & = & \|x-y\|_2 & = & \sqrt{\sum (x_i - y_i)^2} & \ell_2 \text{ metric} \\ d(x,y) & = & \|x-y\|_1 & = & \sum |x_i - y_i| & \ell_1 \text{ metric} \\ d(x,y) & = & \|x-y\|_\infty & = & \max_i |x_i - y_i| & \ell_\infty \text{ metric} \\ d(x,y) & = & \|x-y\|_p & = & (\sum |x_i - y_i|^p)^{\frac{1}{p}} & \ell_p \text{ metric} \end{array}$$

Definition 9 *(X, d) can be embedded into (Y, ℓ) if there exists a mapping $\varphi : X \to Y$ which satisfies $\forall x, y : \ell(\varphi(x), \varphi(y)) = d(x, y)$.*

Definition 10 *(X, d) can be embedded into (Y, ℓ) with distortion c if there exists a mapping $\varphi : X \to Y$ which satisfies $\forall x, y : d(x, y) \leq \ell(\varphi(x), \varphi(y)) \leq c d(x, y)$.*

The following are very natural and central questions regarding the embedding of metrics.

ISIT-ℓ_p: Given a finite metric space (X, d), can it be embedded into $(\mathbb{R}^n, \ell_p)$ for some n?

EMBED-ℓ_p: Given a finite metric space (X, d) and $c \geq 1$, find an embedding of (X, d) into $(\mathbb{R}^n, \ell_p)$ with distortion at most c for some n.

As we will see in the following theorems, the complexity of these questions depend critically on the metric themselves.

Theorem 16 *Any (X, d) can be embedded into $(\mathbb{R}^n, \ell_\infty)$ where $n = |X|$. (Thus, the answer to ISIT-ℓ_∞ is always "yes".)*

Proof: We define a coordinate for each point $z \in X$. Let $d(x, z)$ be the z coordinate of x. Then,

$$\ell_\infty(\varphi(x), \varphi(y)) = \max_{z \in X} |d(x, z) - d(y, z)| = d(x, y),$$

because of the triangle inequality. $\square$

Theorem 17 *ISIT-$\ell_2 \in P$. (i.e., ISIT-ℓ_2 can be answered in polynomial time.)*

Proof: Assume that there exists an embedding of $\{1, 2, \ldots, n\}$ into $\{v_1 = 0, v_2, \ldots, v_n\}$. Consider one such embedding. Then,

$$d^2(i,j) = ||v_i - v_j||^2 = (v_i - v_j)(v_i - v_j) = v_i^2 - 2v_i \cdot v_j + v_j^2.$$

But $v_i^2 = d^2(1,i)$ and $v_j^2 = d^2(1,j)$ which means that,

$$v_i \cdot v_j = \frac{d^2(1,i) + d^2(1,j) - d^2(i,j)}{2}.$$

We now construct $M = (m_{ij})$ where,

$$m_{ij} = \frac{d^2(1,i) + d^2(1,j) - d^2(i,j)}{2}.$$

Hence if M is not positive semi-definite then there is no embedding into ℓ_2. If M is positive semidefinite then we carry out a Cholesky decomposition on M to express M as $M = VV^T$. From the rows of V we can obtain an embedding into $(\mathbb{R}^n, \ell_2)$. $\square$

Theorem 18 *ISIT-ℓ_1 is NP-complete.*

This theorem is given without proof. The reduction is from MAX CUT, since as we will see later there is a very close relationship between l_1-embeddable metrics and cuts. We also omit the proof of the following theorem.

Theorem 19 *Let $X \subseteq \mathbb{R}^n$. (X, ℓ_2) can be embedded into $(\mathbb{R}^m, \ell_1)$ for some m.*

The converse of this theorem is not true as can be seen from the metric space $(\{(0,0), (-1,0), (1,0), (0,1)\}, \ell_1)$.

8.1 Reducing multicommodity flow/cut questions to embedding questions

In this section, we relate α^* and β^* through the use of metrics.

Claim 20

$$\beta^* = \min_{\ell_1\text{-embeddable metrics } (V,\ell)} \frac{\sum_{(x,y)\in E} u_{xy}\ell(x,y)}{\sum_{i=1}^k f_i \ell(s_i, t_i)}.$$

Proof:

($\geq$) Given S, let

$$\varphi(x) = \begin{cases} 0 & \text{if } x \notin S \\ 1 & \text{if } x \in S. \end{cases}$$

Let ℓ be the ℓ_1 metric on the line, i.e. $\ell(a,b) = |a-b|$. Then,

$$\begin{aligned} u(\delta(S)) &= \sum_{(x,y)\in E} u_{xy}\ell(x,y) \\ f(S) &= \sum_{i=1}^{k} f_i \ell(s_i,t_i) \end{aligned}$$

since $\ell(x,y) = 1$ if and only if x is separated from y by S and 0 otherwise.

($\leq$) We can view any ℓ_1 embeddable metric ℓ as a combination of cuts. See figure 11 for the 2-dimensional case.

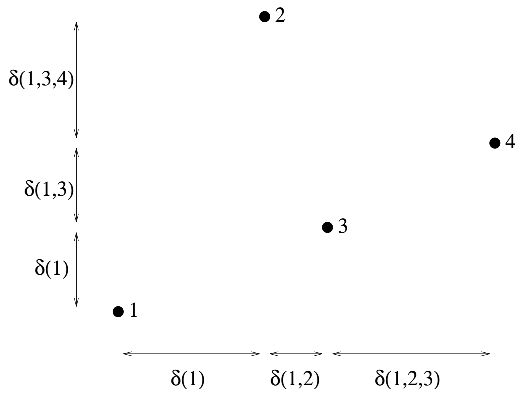


Figure 11: Viewing an ℓ_1-metric as a combination of cuts.

For any set S define a metric 1_S by,

$$1_S = \begin{cases} 1 & \text{if } x, y \text{ are separated by } S \\ 0 & \text{otherwise.} \end{cases}$$

Then we can write ℓ as,

$$\ell(x,y) = \sum_i \alpha_i 1_{S_i}(x,y),$$

where the α_i's are nonnegative. Hence,

$$\frac{\sum_{(x,y)\in E} u_{xy}\ell(x,y)}{\sum_{i=1}^{k} f_i \ell(s_i,t_i)} = \frac{\sum_i \alpha_i u(\delta(S_i))}{\sum_i \alpha_i f(S_i)} \geq \min_S \frac{u(\delta(S))}{f(S)}.$$

$\square$

Claim 21

$$\alpha^* = \min_{\ell_\infty\text{-embeddable metrics } (V,d)} \frac{\sum_{(x,y)\in E} u_{xy} d(x,y)}{\sum_i f_i d(s_i,t_i)}.$$

Note that by theorem 16 we actually minimize over all metrics.
Proof:

($\leq$) For any metric d let the volume of an edge (x,y) be $u_{xy}d(x,y)$. The total volume of the graph is $\sum_{(x,y)\in E} u_{xy}d(x,y)$. If we send a fraction α of the demand then the amount of volume that we use is at least $\alpha \sum_i f_i d(s_i,t_i)$. Hence $\alpha \sum_i f_i d(s_i,t_i) \leq \sum_{(x,y)\in E} u_{xy}d(x,y)$.

($\geq$) We use the strong duality of linear programming. α^* can be formulated as a linear program in several different ways. Here we use a formulation which works well for the purpose of this proof although it is quite impractical. Enumerate the paths from s_i to t_i, let P_{ij} be the jth such path and let x_{ij} be the flow on P_{ij}. The linear program corresponding to multicommodity flow is,

$$\begin{aligned}
\text{Max} \quad & \alpha \\
\text{subject to:} \quad & \\
& \alpha f_i - \sum_j x_{ij} \leq 0 && i \in \{1,\ldots,k\} \\
& \sum_i \sum_{j: e\in P_{ij}} x_{ij} \leq u_e && e \in E \\
& \alpha \geq 0 \\
& x_{ij} \geq 0
\end{aligned}$$

The dual of this linear program is:

$$\begin{aligned}
\text{Min} \quad & \sum_{e\in E} u_e \ell_e \\
\text{subject to:} \quad & \\
& \sum_{i=1}^{k} f_i h_i \geq 1 \\
& \sum_{e\in P_{ij}} \ell_e - h_i \geq 0 && \forall i,j \\
& h_i \geq 0 \\
& \ell_e \geq 0
\end{aligned}$$

The second constraint in the dual implies that h_i is at most the shortest path length between s_i and t_i with respect to ℓ_e. By strong duality if ℓ is an optimum solution to the dual then,

$$\begin{aligned}\alpha^* &= \sum_{e\in E} u_e \ell_e \\ &\geq \frac{\sum_{e\in E} u_e \ell_e}{\sum_{i=1}^k f_i h_i} \\ &\geq \frac{\sum_{(i,j)\in E} u_{ij} d(i,j)}{\sum_{i=1}^k f_i d(s_i,t_i)},\end{aligned}$$

where $d(a,b)$ represents the shortest path length with respect to ℓ_e. The first inequality holds because $\sum_{i=1}^k f_i h_i$ is constrained to be at least 1.

□

Linial, London, and Rabinovitch and Aumann and Rabani use the following strategy to bound $\frac{\beta^*}{\alpha^*}$ and approximate the minimum multicommodity cut.

1. Using linear programming, find α^* and the corresponding metric d as given in claim 21.
2. Embed d into $(\mathbb{R}^m, \ell_1)$ with distortion c. Let ℓ be the resulting metric.

By claim 20 this shows that $\frac{\beta^*}{\alpha^*} \leq c$ since,

$$\beta^* \leq \frac{\sum_{(x,y)\in E} u_{xy}\ell(x,y)}{\sum_{i=1}^k f_i \ell(s_i,t_i)} \leq \frac{c\sum_{(x,y)\in E} u_{xy} d(x,y)}{\sum_{i=1}^k f_i d(s_i,t_i)} = c\alpha^*.$$

In order to approximate the minimum multicommodity cut, we can use the proof of claim 20 to decompose ℓ into cuts. If S is the best cut among them then,

$$\frac{u(\delta(S))}{f(S)} \leq \frac{\sum_{(x,y)\in E} u_{xy}\ell(x,y)}{\sum_{i=1}^k f_i \ell(s_i,t_i)}.$$

Our remaining two questions are:

- How do we get an embedding of d into ℓ? Equivalently, how can we embed ℓ_∞ into ℓ_1.
- What is c?

8.2 Embedding metrics into ℓ_1

The following theorem is due to Bourgain.

Theorem 22 *For all metrics d on n points, there exists an embedding of d into ℓ_1 which satisfies:*

$$d(x,y) \leq ||x-y||_1 \leq O(\log n) d(x,y).$$

Proof: Let k range over $\{1, 2, 4, 8, \ldots, 2^j, \ldots, 2^p\}$ where $p = \lfloor \log n \rfloor$. Hence we have $p+1 = O(\log n)$ different values for k. Now choose n_k sets of size k. At first take all sets of size k, i.e., $n_k = \binom{n}{k}$. Introduce a coordinate for every such set. This implies that points are mapped into a space of dimension $\sum_{j=0}^{p} n_{2^j} < 2^n$. For a set A of size k the corresponding coordinate of a point x is,

$$\frac{\alpha d(x, A)}{n_k}$$

where $d(x, A) = \min_{z \in A} d(x, z)$ and α is a constant which we shall determine later. Suppose that $d(x, A) = d(x, s)$ and $d(y, A) = d(y, t)$, where s and t are in A. Then,

$$d(x, A) - d(y, A) = d(x, s) - d(y, t) \leq d(x, t) - d(y, t) \leq d(x, y).$$

Exchanging the roles of x and y, we deduce that $|d(x, A) - d(y, A)| \leq d(x, y)$. Hence,

$$\begin{aligned} ||x - y||_1 &= \sum_A \frac{\alpha}{n_{|A|}} |d(x, A) - d(y, A)| \\ &\leq \alpha \sum_A \frac{1}{n_{|A|}} d(x, y) \\ &= \alpha(p+1) d(x, y) \\ &= O(\log n) d(x, y). \end{aligned}$$

We now want to prove that $||x - y||_1 \geq d(x, y)$. Fix two points x and y and define,

$$\begin{aligned} B(x, r) &= \{z : d(x, z) \leq r\}, \\ \bar{B}(x, r) &= \{z : d(x, z) < r\}, \\ \rho_0 &= 0, \\ \rho_t &= \inf\{r : |B(x, r)| \geq 2^t, |B(y, r)| \geq 2^t\}. \end{aligned}$$

Let ℓ be the least index such that $\rho_\ell \geq \frac{d(x,y)}{4}$. Redefine ρ_ℓ so that it is equal to $\frac{d(x,y)}{4}$. Observe that for all t either $|\bar{B}(x, \rho_t)| < 2^t$ or $|\bar{B}(y, \rho_t)| < 2^t$. Since $B(x, \rho_{\ell-1}) \cap B(y, \rho_{\ell-1}) = \emptyset$ we have $2^{\ell-1} + 2^{\ell-1} \leq n \Rightarrow \ell \leq p$. Now fix $k = 2^j$ where $p-1 \geq j \geq p-\ell$ and let $t = p - j$ (thus, $1 \leq t \leq l$). By our observation we can assume without loss of generality that $|\bar{B}(x, \rho_t)| < 2^t$. Let A be a set of size k and consider the following two conditions.

1. $A \cap \bar{B}(x, \rho_t) = \emptyset$.
2. $A \cap B(y, \rho_{t-1}) \neq \emptyset$.

If 1. and 2. hold then $d(x, A) \geq \rho_t$ and $d(y, A) \leq \rho_{t-1}$ and so $|d(x, A) - d(y, A)| \geq \rho_t - \rho_{t-1}$. Let

$$R_k = \{A : |A| = k \text{ and } A \text{ satisfies conditions 1. and 2.}\}$$

Lemma 23 *For some constant $\beta > 1$ (independent of k), there are at least $\frac{n_k}{\beta}$ sets of size k which satisfy conditions 1. and 2, i.e. $|R_k| \geq \frac{n_k}{\beta}$.*

From this lemma we derive,

$$\begin{aligned}
||x-y||_1 &\geq \sum_{j=p-\ell,k=2^j}^{p-1} \sum_{R_k} \frac{\alpha}{n_k}|d(x,A)-d(y,A)| \\
&\geq \sum_{j=p-\ell,k=2^j}^{p-1} \frac{n_k}{\beta}\frac{\alpha}{n_k}(\rho_{p-j}-\rho_{p-j-1}) \\
&= \frac{\alpha}{\beta}\sum_{j=p-\ell}^{p-1}(\rho_{p-j}-\rho_{p-j-1}) \\
&= \frac{\alpha}{\beta}\rho_\ell \\
&= \frac{\alpha}{4\beta}d(x,y).
\end{aligned}$$

Hence if we choose $\alpha = 4\beta$ then we have $||x-y||_1 \geq d(x,y)$. We now have to prove lemma 23.

Proof of lemma 23: Since $|\bar{B}(x,\rho_t)| < 2^t$, $|B(y,\rho_{t-1})| \geq 2^{t-1}$ and we are considering all sets of size k the following is a restatement of the lemma: Given disjoint sets P and Q with $a = |P| < 2^t$ and $b = |Q| \geq 2^{t-1}$, if E is the event that a uniformly selected A misses P and intersects Q then $\Pr[E] \geq \frac{1}{\beta}$. We calculate this probability as follows:

$$\begin{aligned}
\Pr[E] &= \frac{\binom{n-a}{k} - \binom{n-a-b}{k}}{\binom{n}{k}} \\
&= \frac{(n-a)!(n-k)!}{(n-a-k)!n!} - \frac{(n-a-b)!(n-k)!}{(n-a-b-k)!n!} \\
&= \frac{n-a}{n}\frac{n-a-1}{n-1}\cdots\frac{n-a-k+1}{n-k+1} - \\
&\quad \frac{n-a-b}{n}\frac{n-a-b-1}{n-1}\cdots\frac{n-a-b-k+1}{n-k+1} \\
&= \left(1-\frac{a}{n}\right)\left(1-\frac{a}{n-1}\right)\cdots\left(1-\frac{a}{n-k+1}\right) - \\
&\quad \left(1-\frac{a+b}{n}\right)\left(1-\frac{a+b}{n-1}\right)\cdots\left(1-\frac{a+b}{n-k+1}\right).
\end{aligned}$$

As an approximation (this can be made formal), we replace $(1-\frac{a}{n-j})$ by $e^{-a/n}$, and $(1-\frac{a+b}{n-j})$ by $e^{-(a+b)/n}$. Thus,

$$\Pr[E] \approx e^{-\frac{ak}{n}} - e^{-\frac{(a+b)k}{n}} \geq e^{-\frac{ak}{n}}\left(1-e^{-\frac{bk}{n}}\right).$$

This for example shows that if a, b and k are all $\sqrt{n}$ then this probability is a constant, which may seem a bit paradoxical. Using our bounds on a and b, we get

$$\begin{aligned}\Pr[E] &\approx e^{-\frac{ak}{n}}\left(1-e^{-\frac{bk}{n}}\right) \geq e^{-\frac{2^t 2^j}{n}}\left(1-e^{-\frac{2^{t-1}2^j}{n}}\right)\\ &= e^{-\frac{2^p}{n}}\left(1-e^{-\frac{2^{p-1}}{n}}\right) \geq e^{-1}\left(1-e^{-\frac{1}{4}}\right).\end{aligned}$$

We now choose $\beta \approx \left(e^{-1}\left(1-e^{-\frac{1}{4}}\right)\right)^{-1}$ and the proof is complete. □

Bourgain's proof is not quite algorithmic since the dimension is exponential. Linial, London and Rabinovitch just sample uniformly with $n_k = O(\log n)$ and show that with high probability the embedding has the required properties. This follows from a Chernoff bound.

We have thus shown that the distortion c can be chosen to be $O(\log n)$. We can do even better by proving the following variant to Bourguain's theorem.

Theorem 24 *Let d be a metric on a set V of n points. Suppose that $T \subseteq V$ and $|T| = k$. Then there exists an embedding of d into ℓ_1 which satisfies:*

$$\begin{array}{ll} ||x-y||_1 \leq O(\log k) d(x,y) & \forall x,y \in V. \\ ||x-y||_1 \geq d(x,y) & \forall x,y \in T. \end{array}$$

In order to prove this theorem we restrict the metric to T and then embed the restricted metric. If we look at the entire vertex set V then the first part of the original proof still works.

This new theorem is enough to show that $\frac{\beta^*}{\alpha^*} \leq O(\log k)$ and to approximate the multicommodity cut to within $O(\log k)$. This result is best possible in the sense that we can have $\frac{\beta^*}{\alpha^*} = \Theta(\log k)$.

References

[1] S. Arora, C. Lund, R. Motwani, M. Sudan, and M. Szegedy. Proof verification and hardness of approximation problems. In *Proceedings of the 33rd Annual Symposium on Foundations of Computer Science*, pages 14–23, 1992.

[2] S. Arora and S. Safra. Probabilistic checking of proofs. In *Proceedings of the 33rd Annual Symposium on Foundations of Computer Science*, pages 2–13, 1992.

[3] Y. Aumann and Y. Rabani. An $O(\log k)$ approximate min-cut max-flow theorem and approximation algorithm. Manuscript, 1994.

[4] R. Bar-Yehuda and S. Even. A linear time approximation algorithm for the weighted vertex cover problem. *Journal of Algorithms*, 2:198–203, 1981.

[5] M. Bellare and M. Sudan. Improved non-approximability results. In *Proceedings of the 26th Annual ACM Symposium on Theory of Computing*, pages 184–193, 1994.

[6] N. Christofides. Worst-case analysis of a new heuristic for the travelling salesman problem. Technical Report 388, Graduate School of Industrial Administration, Carnegie Mellon University, Pittsburgh, PA, 1976.

[7] W. F. de la Vega and G. S. Luecker. Bin packing can be solved within $(1+\epsilon)$ in linear time. *Combinatorica*, 1(4), 1981.

[8] J. Edmonds. Maximum matching and a polyhedron with 0,1-vertices. *Journal of Research of the National Bureau of Standards B*, 69B:125–130, 1965.

[9] R. Fagin. Generalized first-order spectra, and polynomial-time recognizable sets. In R. Karp, editor, *Complexity of Computations*. AMS, 1974.

[10] H. N. Gabow. Data structures for weighted matching and nearest common ancestors with linking. In *Proceedings of the 1st ACM-SIAM Symposium on Discrete Algorithms*, pages 434–443, 1990.

[11] H. N. Gabow, M. X. Goemans, and D. P. Williamson. An efficient approximation algorithm for the survivable network design problem. In *Proceedings of the Third MPS Conference on Integer Programming and Combinatorial Optimization*, pages 57–74, 1993.

[12] H. N. Gabow and R. E. Tarjan. Faster scaling algorithms for general graph matching problems. Technical Report CU-CS-432-89, University of Colorado, Boulder, 1989.

[13] M. X. Goemans and D. P. Williamson. A general approximation technique for constrained forest problems. In *Proceedings of the 3rd Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 307–316, 1992.

[14] M. X. Goemans and D. P. Williamson. Improved approximation algorithms for maximum cut and satisfiability problems using semidefinite programming. In *Proceedings of the 26th Annual ACM Symposium on Theory of Computing*, pages 422–431, 1994.

[15] A. Haken and M. Luby. Steepest descent can take exponential time for symmetric connection networks. *Complex Systems*, 2:191–196, 1988.

[16] D. Hochbaum. Approximation algorithms for set covering and vertex cover problems. *SIAM Journal on Computing*, 11:555–556, 1982.

[17] D. Hochbaum and D. Shmoys. Using dual approximation algorithms for scheduling problems: theoretical and practical results. *Journal of the ACM*, 34(1), Jan. 1987.

[18] N. Karmarkar and R. Karp. An efficient approximation scheme for the one-dimensional bin-packing problem. In *Proceedings of the 23rd Annual Symposium on Foundations of Computer Science*, 1982.

[19] T. Leighton and S. Rao. An approximate max-flow min-cut theorem for uniform multicommodity flow problems with applications to approximation algorithms. In *Proceedings of the 29th Annual Symposium on Foundations of Computer Science*, pages 422–431, 1988.

[20] N. Linial, E. London, and Y. Rabinovich. The geometry of graphs and some of its algorithmic applications. In *Proceedings of the 35th Annual Symposium on Foundations of Computer Science*, 1994.

[21] C. H. Papadimitriou and M. Yannakakis. Optimization, approximation, and complexity classes. *Journal of Computer and System Sciences*, 43:425–440, 1991.

[22] S. Poljak. Integer linear programs and local search for max-cut. Preprint, 1993.

[23] P. Raghavan. Probabilistic construction of deterministic algorithms: approximating packing integer programs. *Journal of Computer and System Sciences*, 37:130–143, 1988.

[24] P. Raghavan and C. D. Thompson. Randomized rounding: a technique for provably good algorithms and algorithmic proofs. *Combinatorica*, 7:365 – 374, 1987.