Combinatorial Optimization 1990: Lecture Notes

Philip N. Klein, editor
P. Hubbard, P. Krishnan, G. Hillebrand,
D. Langworthy, M. Goldwasser, A. Mayer,
K. Basye, J. Borger, D. Luks, D.B. Conner,
D. Cervone, S. Sairam, R. Ravi,
A. Agrawal, R. Cohen, S. Rosenzweig

Department of Computer Science
Brown University
Providence, Rhode Island 02912

CS-91-59
September 1991

Combinatorial Optimization 1990: Lecture Notes

P. Klein, ed., P. Hubbard, P. Krishnan, G. Hillebrand,
D. Langworthy, M. Goldwasser, A. Mayer, K. Basye,
J. Borger, D. Luks, D.B. Conner, D. Cervone,
S. Sairam, R. Ravi, A. Agrawal, R. Cohen, S. Rosenzweig

Department of Computer Science
Brown University
Providence, Rhode Island 02912

October, 1991

Abstract

This technical report consists of lecture notes for a seminar on combinatorial optimization held in Fall 1990. The notes were edited by Philip Klein and written by members of the class. Editorial and organizational assistance was provided by Tina Cantor and Hsueh-I Lu.

Table of Contents

CS258: Combinatorial Optimization

Lecture 1, 9/4/90 | Philip Klein | Scribe: Philip Hubbard

1 Course Goals

Since this meeting was the first of the class, the discussion quickly focused on the goals of the course. The hope is that the class will answer two questions:

- What is combinatorial optimization?
- What is not combinatorial optimization?

At this point, a sample problem seemed appropriate.

2 A Sample Problem

The lazy daze of summer began to dwindle in the august halls of the Computer Science Department. Thoughts turned to serious matters, such as the assignment of TAs to courses. Each course must have one TA, and every student must be assigned to a course (no students were awarded fellowships that year, since all available funds were being channeled into the New Age Research Center that had just opened on campus). Since this department always catered to the wishes of its students, the professors struggled to ensure that each student would teach a course that he or she found interesting. The problem of assigning students to courses was so important, the professors represented it with a graph (see Figure 1). An edge from between a student and a course indicates

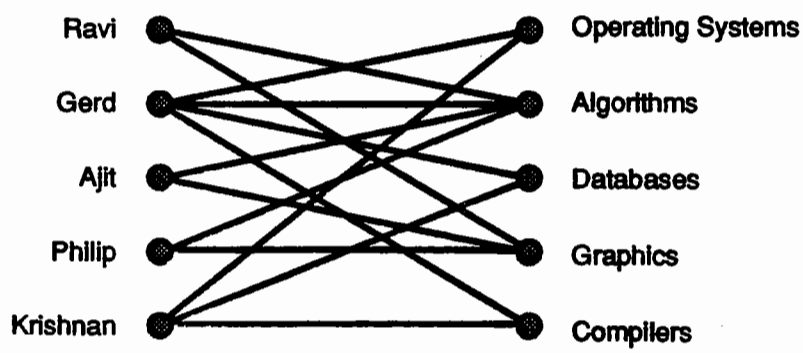


Figure 1: The student–course graph.

that the student would be interested in teaching that course.

Is it possible for every student to be assigned to a course he or she finds desirable? Unfortunately, it is not possible for this graph. The blame rests on Ajit, Philip and Ravi. Since the three of them all prefer the same two courses, there is no way to satisfy their wishes. The disappointment of

4.1 Augmenting Paths

It is possible to better than that. Notice that if a perfect matching exists, then any maximum matching will be a perfect matching. So a maximum matching algorithm can be used to find a perfect matching. An efficient algorithm to find a maximum matching can be based on the idea of *augmenting*. Consider the student–course graph again. Intuitively, if a student s_1 can TA only a course c already assigned to another student s_2, we'll give course c to student s_1 and try to find another course for s_2. The algorithm will be easier to explain once some definitions have been made:

- An *alternating path* in a graph G with respect to a subgraph M of G is a simple path (no repeated vertices) using edges alternately in M and not in M.
- An *augmenting path* with respect to M is an alternating path whose first and last edges are not in M.

When discussing these paths, we'll call an edge "matched" if it is an element of M, and "unmatched" if not. We'll call a vertex "matched" if it is an element of the subgraph *induced* by M, the subgraph constructed from the edges of M and all the vertices adjoined by these edges.

The augmenting path can be used to perform the augmenting operation discussed a moment ago. Let G be the student–course graph. Let M be a matching for some subset V' of the vertices of G. M indicates which students have already been assigned to courses. Make the first vertex s_1 of an augmenting path p be a student without a course to TA, i.e. $s_1 \notin M$. The second vertex will then be some course c_1 that the student would like to TA but has not been assigned to, because the edge out of the first vertex is unmatched. The third vertex will be the student s_2 who has been assigned to TA course c_1. The fourth vertex will be a course that student s_2 would have been willing to TA, but that has been assigned instead to student s_3, the fifth vertex. This pattern continues until the path reaches the last vertex, which must be a course without a TA. Notice that there must be an even number of vertices in the path, and hence an odd number of edges.

To do the augmenting, just set M to be the *symmetric difference* $M \oplus E_p$, where E_p is the set of edges from the path p. Since $M \oplus E_p \equiv (M - E_p) \cup (E_p - M)$, this operation adds to M every edge from the augmenting path p that is not already in M, and removes from M every edge from p that is currently in M. This "swapping" assigns student s_1 to course c_1, reassigns student s_2 from course c_1 to course c_2, and so on until every student and every course on the path have been assigned (see Figure 2). Notice that swapping maintains the essential property of a matching

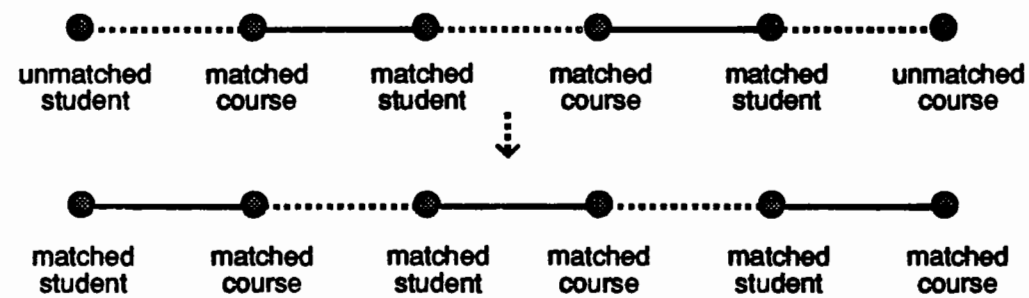


Figure 2: An augmenting path being swapped.

(that each vertex is adjoined by at most one edge from the matching) because each vertex gets a

Theorem 2 *Suppose a subgraph M of G is not a perfect matching. If there does not exist an augmenting path with respect to M, then there does not exist a perfect matching for G.*

The best way to prove this theorem is to prove is to prove a more general theorem about maximum matchings:

Theorem 3 *Suppose a subgraph M of G is a matching. If there is no augmenting path with respect to M, then M is a maximum matching.*

To prove this theorem, assume that there is no augmenting path with respect to M but M is not a maximal matching. There must therefore be a matching M^* that is bigger than M. Consider $M' = M^* \oplus M$, i.e. the set of edges in M but not in M^* and vice versa. M' consists of vertex-disjoint alternating paths and cycles (with respect to M). To see why, consider any vertex v in the subgraph induced by M'. Since both M and M^* are matchings, v adjoins at most one edge from M and one from M^*. So the degree of v (the number of edges it adjoins) is at most two. Hence, every vertex is either in the middle of a path or cycle, or at the end of a path. If a vertex has a degree of two, then one edge must be from M and one from M^*, as just mentioned. But any edge in M' and M^* must be in $M^* - M$ since M' is the symmetric difference, so a vertex of degree two must adjoin an edge in M and an edge not in M. So every path or cycle must be alternating with respect to M.

All even-length paths contain the same number of edges from M and M^* because all paths are alternating. But there must be at least one path that contains more edges from M^* because $|M^*| > |M|$. The only way an alternating path can contain more edges from M^* is to begin and end with an edge from M^*. This path is an augmenting path with respect to M. The assumption that there is no augmenting path has therefore been contradicted, and hence there cannot not be a matching M^* bigger than M. So M must be a maximum matching, and the theorem is proved.

Now we can prove Theorem 2. Since the subgraph M of G is not a perfect matching and there does not exist an augmenting path with respect to M, Theorem 3 indicates that M must be a maximum matching. If there were a perfect matching, it would have to be bigger than M, but then M would not be a maximum matching. So there cannot be a perfect matching, and Theorem 2 is proved.

5 k-Flows

Turning away from matching problems for a while, discussion focused on another graph problem. A vertex in a directed graph that adjoins only outgoing edges and no incoming edges is a ***source***. Correspondingly, a vertex that adjoins only incoming edges and no outgoing edges is a ***sink***. Consider a directed graph G with a single source a and a single sink b. The notion of a ***flow*** though G can be developed if the edges in G are considered to be pipes that can carry fluid. A flow involves the following rules:

- Each edge can carry at most one unit of fluid.
- Fluid flow originates only at the source a and is consumed only at the sink b. (Hence, there is *conservation* of fluid, i.e. the inflow at a vertex is equivalent to the outflow, and fluid cannot be lost or gained in any of the edges.)

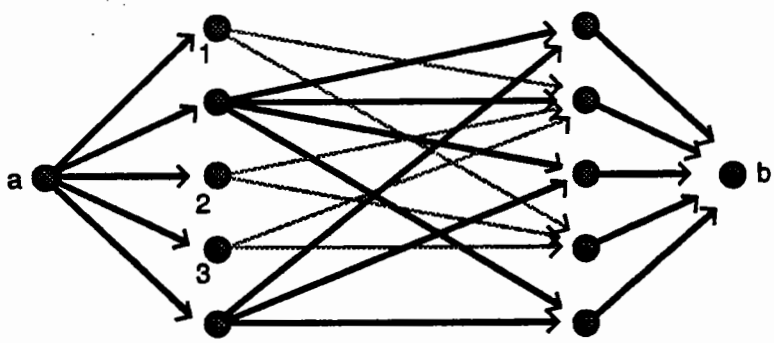


Figure 4: A directed graph.

$S' = \{1, 2, 3\}$ above, for which $|\Gamma(S')| < |S|$. Now consider the simple cut that partitions the graph into the vertices $A = S' \cup \Gamma(S') \cup \{a\}$ and $B = V - A$. The only edges from A to B are the edges from the vertices in $\Gamma(S')$ to b and the edges from a to $S - S'$, for a total of $n = |\Gamma(S')| + (k - |S'|)$ edges. Since $|\Gamma(S')| < |S'|$, it follows that $n < k$, so there can be no k-flow.

6 Tentative Conclusions

The point behind the examples given in this lecture was to illustrate the kinds of problems that will be the subject of this course. Finding a maximum matching is an optimization problem because we want to find the largest possible matching for the graph. It is also a combinatorial problem because graphs are a classic example of a combinatorial object. The k-flow problem can be thought of as an optimization problem if we try to find the largest k such that a graph admits a k-flow. It is not obvious, however, how the k-flow problem can be considered a combinatorial problem. It is true that the problem involves a graph, which is combinatorial, but the flow values are not constrained to be integral. Combinatorics always deals with discrete quantities (like integers), and does not concern itself with continuous quantities (like real numbers). We will see during later lectures, though, how to transform the k-flow problem so that it deals with only discrete quantities. So we can conclude that the two problems presented in this lecture exemplify combinatorial optimization, but we do not yet know enough to completely explain why.

We now state and prove Hall's theorem.

Theorem 1 (Hall) *Given a bipartite graph G with student vertices S and course vertices C and edges between S and C, there exists a perfect matching in the graph if and only if for every student subset S',*

$$|\Gamma(S')| \geq |S'|$$

3 Proof of Hall's theorem

($\Rightarrow$) : (Given a bipartite graph, if there is a perfect matching in the graph, then for every student subset S', $|\Gamma(S')| \geq |S'|$). This was done in the last class and is a fairly obvious necessary condition for the existence of a perfect matching.
($\Leftarrow$) : (If for every subset S', $|\Gamma(S')| \geq |S'|$ then there exists a perfect matching). We prove this result using induction on the size of the student set S (which is equal to the size of the course set C) .

- *Base case,* $|S| = 1$. In this case there is one student and one course and there must be an edge between them to make $|\Gamma(S)| \geq |S|$. This ensures a perfect matching.
- *Induction Hypothesis.* For all bipartite graphs with $|S| \leq n$, if for every subset S' of S, $|\Gamma(S')| \geq |S'|$ then there is a perfect matching for the graph.
- *Induction step.* To prove that for bipartite graphs with $|S| = n + 1$, if for every subset S' of S, $|\Gamma(S')| \geq |S'|$ then there is a perfect matching for the graph. To show this, we consider two cases.

  Case 1: Suppose that for every proper subset S' of S, $|\Gamma(S')| > |S'|$. Take student 1 and any neighbor of this student and pair them up. Remove these two nodes from the graph. The remaining graph has the property that for every subset of student vertices S', $|\Gamma(S')| \geq |S'|$ since at most one node (the node that student 1 was paired up with) is removed from $\Gamma(S')$ for each S'. Also, the remaining graph is of smaller size in the number of student nodes. Thus by the induction hypothesis, there is a perfect matching amongst these remaining nodes. This matching with the match for student 1 gives a perfect matching for the original graph.

  Case 2: There is a proper subset A of S such that $|\Gamma(A)| = |A|$. Refer to Figure 2. By the induction hypothesis applied to the graph induced on A and $\Gamma(A)$, there exists a perfect matching between A and $\Gamma(A)$. Now we remove A and $\Gamma(A)$ from the graph. Let us assume we are left with the student set A'. If the students in A' can be matched to courses in $\Gamma(A') - \Gamma(A)$, clearly we are done. Let A'' be any subset of A' and $\cup$ denote the disjoint union of two sets. We know from the property of the graph given to us that $|S'| \leq |\Gamma(S')|$ for every subset S' of S. This, in particular, implies that

$$|A| + |A''| = |(A \cup A'')| \leq |\Gamma(A \cup A'')| = |\Gamma(A)| + |\Gamma(A'') - \Gamma(A)|$$

  Since $|A| = |\Gamma(A)|$ for this case, the above implies that

$$|A''| \leq |\Gamma(A'') - \Gamma(A)|, \qquad \forall A'' \subseteq A'$$

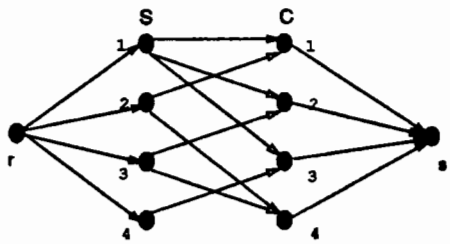


Figure 3: Solving the perfect matching problem using the k-flow problem

Introduce a source r and a sink s. Connect r to all nodes of S and all nodes of C to s. Direct edges between S and C in the sense from S to C. Now we ask the following question: "If we have a n-flow in this graph does that give us a perfect matching, where n is the number of students?" At first it would seem obviously true[1] since the edges between S and C that take part in the flow seem to give a perfect matching. This is indeed true if we require all flows to be integral. But recall the k-flow problem. It does not require an integral flow[2]. Where does this leave us now? If we can find a k-flow in the modified bipartite graph, does that imply anything about a perfect matching?

6 The Hillebrand conjecture

At this point in class Gerd made a conjecture which is stated below. This conjecture requires knowledge of some concepts in linear algebra and they lead us to the geometric relation between k-flow and perfect matchings, which is the topic of the rest of this handout.

Conjecture 1 (Hillebrand[3]) *Any n-flow is a convex combination of perfect matchings.*

The immediate question is: "What is a convex combination"? This is what we now investigate.

7 Scholium: Linear combinations

Let $\Re$ denote the set of real numbers. (Note that for this course, unless otherwise specified, we will not differentiate between reals and rationals). Let $v_1, v_2, \ldots, v_p \in \Re^m$. If $\lambda_1, \lambda_2, \ldots, \lambda_p \in \Re$ then we call

$$\lambda_1 v_1 + \lambda_2 v_2 + \cdots + \lambda_p v_p$$

a *linear combination.* We call it a *convex linear combination* if

$$\sum_{i=1}^{p} \lambda_i = 1 \qquad \text{and} \qquad \forall_i (\lambda_i \geq 0)$$

Some other interesting combinations include, *non-negative combinations* where for all i, $\lambda_i \geq 0$, *affine* combinations where $\sum_{i=1}^{p} = 1$, and *integer linear combinations* where for all i, λ_i is an integer. In particular, a convex linear combination is non-negative and affine.

[1] If it is not obvious, maybe you are correct

[2] So, if you didn't find it obvious, were you correct?

[3] The scribe named this conjecture after the student who sprung this in class.

Combinatorial Optimization

Lecture 3, Sep. 11, 1990 | **Philip Klein** | Scribe: Gerd Hillebrand

The subject of this lecture was, once again, the *perfect matching* problem. Continuing the geometrical approach brought up in the last lecture, the space of admissible matchings was identified with a certain convex polytope, the *matching polytope*. Then the notions and tools of convexity theory were used to prove the Birkhoff-von Neumann Theorem, which characterizes the perfect matchings as the vertices of the matching polytope.

1 Review of Convexity Theory.

Our setting will be the Euclidean space $\mathbf{R}^k$. Vectors $\mathbf{x} = (x_1, \ldots, x_k)$ will be denoted by lowercase boldface letters and matrices $\mathbf{A} = (a_{ij})$ by boldface capitals. A *halfspace* is the solution set of a linear inequality $\mathbf{a} \cdot \mathbf{x} \leq b$. A *polyhedron* is the solution set of a system of linear inequalities $\mathbf{Ax} \leq \mathbf{b}$, i.e. the intersection of a finite number of halfspaces. A bounded polyhedron is called a *polytope*. The *Krein-Milman Theorem* asserts that every polytope can be written as the set of convex combinations of a certain minimal set of vectors, called the *extreme points* or *vertices* of the polytope. These extreme points are exactly those points of the polytope that cannot be expressed as non-trivial convex combinations of other points.

2 The Matching Polytope.

With these definitions, we now proceed to describe the perfect matching problem in terms of convex polytopes and their vertices.

We consider a bipartite graph G with nodes $V \cup W$ and edges $E \subseteq V \times W$. A perfect matching of G can then be described as a vector $\mathbf{x} \in \{0,1\}^{|E|}$: For each edge $(v,w) \in E$, the corresponding component $x_{v,w}$ of $\mathbf{x}$ is 1 if (v,w) is included in the matching and 0 otherwise. However, not every vector in $\{0,1\}^{|E|}$ corresponds to a perfect matching. Instead, the perfect matchings are characterized by the condition

$$\sum_{(v,u)\in E} x_{v,u} = \sum_{(u,w)\in E} x_{u,w} = 1 \quad \forall v \in V, w \in W,$$

which says that each node has exactly one mate.

We have now described the perfect matchings as the solutions of a certain system of linear equations over the discrete space $\{0,1\}^{|E|}$. From a geometrical point of view however, it is much more natural to deal with the "continuous" space $\mathbf{R}^{|E|}$. Therefore, we introduce *fractional matchings* by allowing edges to participate in a matching "to a certain degree." In the TA-course assignment example, this means that we allow the "time-sharing" of TAs, so that a TA might spend 40% of his time teaching Algorithms and putting the remaining 60% into Graphics. Of course, we still want the workload for each TA and the total TA time for each course to sum up to 100%. Hence, we

the edges in C_o correspondingly. Note that all edges in C carry non-integral values and therefore δ and μ can be chosen to be strictly positive. We have the following relation:

$$\mathbf{x} = \frac{1}{\mu + \delta}(\mu \mathbf{y} + \delta \mathbf{z}),$$

because for each edge (v, w), either $y_{v,w} = x_{v,w} + \delta$, $z_{v,w} = x_{v,w} - \mu$, or $y_{v,w} = x_{v,w} - \delta$, $z_{v,w} = x_{v,w} + \mu$, or $y_{v,w} = z_{v,w} = x_{v,w}$, depending on whether $(v, w) \in C_e$, $(v, w) \in C_o$, or $(v, w) \notin C$. This expresses $\mathbf{x}$ as the proper convex combination of two other points of P, and thus $\mathbf{x}$ cannot be an extreme point of P. □

Note that the proof of the theorem actually gives rise to an algorithm for expanding any fractional matching into a convex combination of perfect matchings. To see this, observe that if δ and μ are given their maximum permissible values, the vectors $\mathbf{y}$ and $\mathbf{z}$ constructed during the proof will have at least one more integral component than $\mathbf{x}$—namely a zero on an edge in C_o resp. C_e. Therefore, if the same decomposition process is applied to $\mathbf{y}$ and $\mathbf{z}$, this will lead to a representation of $\mathbf{x}$ as a convex combination of four vectors having each two more integral components than $\mathbf{x}$, and so forth. Since a convex decomposition can be constructed as long as a matching has a non-integral component, this will eventually produce an expansion of $\mathbf{x}$ into a convex combination of matchings with all-integral components, i.e. perfect matchings. However, since the number of points under consideration doubles at each step, this algorithm may take exponential time. Still, a single vertex can be found in polynomial time by restricting the recursion to just one of the endpoints of the segment generated in each step.

4 The Birkhoff-von Neumann Theorem revisited.

The preceding proof of the Birkhoff-von Neumann Theorem uses a geometric characterization of extreme points, namely that they are the only points of a polytope that cannot be expressed as a proper convex combination of two other points. We will now give an equivalent algebraic characterization of extremality and use it to prove the Birkhoff-von Neumann Theorem in a purely algebraic way.

Definition. *Let* $\mathbf{Ax} \leq \mathbf{b}$ *be the defining system of inequalities for a polytope* P *and let* $\mathbf{A'x} \leq \mathbf{b'}$ *be a subset of these inequalities. Then the set* $\{\mathbf{x} \in P : \mathbf{A'x} = \mathbf{b'}\}$ *is called a* face *of* P.

In other words, a face of P is the intersection of P with some of its tangential hyperplanes. This makes the following lemma intuitively obvious.

Lemma. *The vertices of* P *are exactly the singleton faces.*

PROOF: Let $\{\mathbf{v}\}$ be a singleton face and let $\mathbf{A'x} \leq \mathbf{b'}$ be those inequalities of $\mathbf{Ax} \leq \mathbf{b}$ that are tight for $\mathbf{v}$. Take any convex combination $\mathbf{v} = \lambda\mathbf{u} + \mu\mathbf{w}$ with $\mathbf{u}, \mathbf{w} \in P$. Since $\mathbf{b'} = \mathbf{A'v} = \lambda\mathbf{A'u} + \mu\mathbf{A'w} \leq \lambda\mathbf{b'} + \mu\mathbf{b'} = \mathbf{b'}$, we have $\mathbf{A'u} = \mathbf{A'w} = \mathbf{b'}$. But by hypothesis, $\mathbf{v}$ is the only solution of $\mathbf{A'x} = \mathbf{b'}$ and hence $\mathbf{u}$ and $\mathbf{w}$ must equal $\mathbf{v}$. So there is no non-trivial convex decomposition of $\mathbf{v}$ and therefore $\mathbf{v}$ must be a vertex of P.

To prove the other direction, let $\mathbf{v}$ be any point of P but a singleton face and let again $\mathbf{A'x} \leq \mathbf{b'}$ be those inequalities of $\mathbf{Ax} \leq \mathbf{b}$ that are tight for $\mathbf{v}$. By hypothesis, there exists at least one more

CS258: Combinatorial Optimization

Lecture 4, 9/13/90 | **Philip Klein** | Scribe: David Langworthy

The class began with a review of lecture 3 of which I provide an outline. For the complete proofs of the Birkhoff - von Neuman Theorem see the notes for lecture 3. Then we applied the augmenting path approach that provided an algorithm for perfect matching to maximal flow. The remainder of the class was spent on this topic.

1 Review of Perfect Matching

Theorem 1 (Birkhoff - von Neuman) *Every fractional matching is the convex combination of integral perfect matchings.*

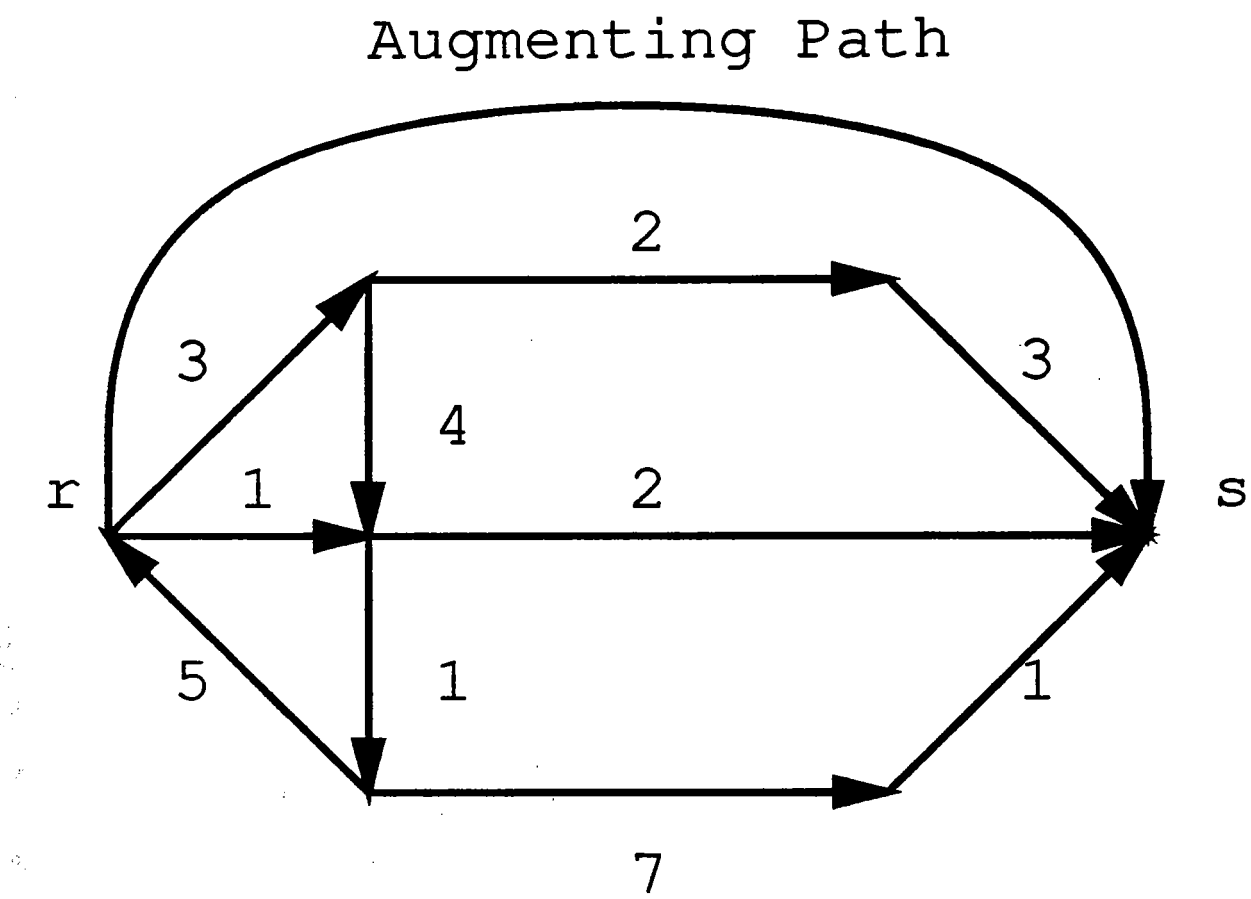
Proof: Consider the polytope, P, of fractional matchings where P is defined as below and A is a node-arc incidence matrix. There is one row for every vertex and one column for every edge. Each column has exactly two 1's in it because each edge has exactly two end points. Every matrix of this form is a node-arc incidence matrix because a graph can be easily constructed from this description. With a perfect matching x selects from A a set of edges so that every vertex is the end point of exactly one edge. P is bounded and therefore a polytope and can be described by a convex hull of its vertices. There are three equivalent definitions of vertices:

1. For a polytope Q vertices are the minimum cardinality set whose convex combinations are Q.
2. x_0 is a vertex if x_0 is in Q and cannot be written as a non-trivial convex combination of other points in Q.
3. x_0 is a vertex if $\{x_0\} = \{x_i | x \in Q, A'x = b'x\}$ where $Q = \{x : Ax \leq b\}$ and A' and b' are a subsystem of Q.

An integral element of P is a perfect matching. As was shown in lecture 3, every vertex is integral. From this fact the theorem follows. Since P contains all matchings and its vertices are perfect matchings then every fractional matching can be described as a convex combination of perfect matchings.

2 Maximum Flows via Augmenting Paths

We have shown that the augmenting path approach leads to a polynomial time algorithm for perfect matching. This does not directly yield an algorithm for Maximum Flow, but in this section we will apply the technique and derive an algorithm. Throughout this section I will use the following graph, G, as an example.



An augmenting path can be used to augment a flow in the following manner.

foreach edge, e, in the augmenting path, P **do**
    $f(e) \leftarrow f(e) + \delta$
    end
foreach reverse edge, e^R, in the augmenting path, P **do**
    $f(e) \leftarrow f(e) - \delta$
    end

In English, for each edge in the path increment the flow along that edge by δ. For each reverse edge in the path decrement the flow along the original edge by δ.

We now define a graph G_f which has all the nodes and edges G does plus some additional edges and the capacities of the edges may differ. We denote these capacities $u_f(e)$. G_f is called a residual graph because it represents the capacities available after a flow f is pushed through the graph. To define G_f we will need the notion of a reverse edge. Reverse edges never carry flow, they only hold capacity so that flow on the original edge can be reduced. Let e^R be e's reverse edge. That is if e points from a to b in G then e^R points from b to a. Now the definition.

$$u_f(e) = \begin{cases} u(e) - f(e) & \text{if } e \text{ is an original edge;} \\ f(e^R) & \text{if } e \text{ is a reverse edge.} \end{cases}$$

This approach generalizes to all max-min problems. Prove that one value must always be greater or equal to another value and then construct an algorithm to find two that are the same. The value that must be greater will be minimum and the value that must be lessor will be maximum.

Take any edge, $e = u \leftarrow v$, from A to B; the edge has no residual capacity for if it didn't v would be in A as well. We now must consider the two types of edges we have: original and reverse. If the edge is an original edge then we infer that $f(e) = u(e)$. Thus all original edges across the cut are at capacity. If the edge is a reverse edge. its original edge from B to A carries no flow since the reverse edge has no capacity. Thus there is no flow back across the cut from B to A.

The amount of flow going from A to B is

$$\sum_{e \in (E \cap (A \times B))} u(e)$$

because we have shown that each edge across the cut is at capacity and the amount of flow going from B to A is 0. We now have a found a flow equal to the value of a cut so the flow must be maximal and the cut must be minimal and the algorithm is correct.

References

[FF56] L. R. Ford Jr. and D. R. Fulkerson, Maximal flow through a network, *Canadian Journal of Mathematics*, Vol. 8, 1956 399-404.

[EFS56] P. Elias, A Feinstein, and C. E. Shannon, A note on the maximum flow through a network, *IRE Transactions on Information Theory* IT 2, 117-9.

Assume that wy is an edge from (a) to (b), unmatched in M. Since w is reachable, then this edge can be added to the augmenting path, and y would also be reachable, contradicting the fact that y is in (b). Therefore such an unmatched edge cannot exist.

Alternatively, assume that vx is an edge from (a) to (b), matched in M. Then consider how v is reachable. Either v is unmatched, but clearly this contradicts vx being matched, or else v is reached from a reachable right vertex via a matched edge. However, since vx is already matched, v cannot be matched to any other vertex, and x must be that reachable vertex leading to v, however by assumption x was unreachable. Therefore such a matched edge cannot exist, and therefore there are no edges between (a) and (b), and so VC is in fact a vertex covering.

Now we need to show that $|M| \geq |VC|$. First notice that every vertex in VC is incident to exactly one matched edge. Since all endpoints of matched edges are unique, clearly no vertex in VC is incident to more than one matched edge. For those vertices in (c), if they were unmatched, then they would be reachable, yet they are unreachable, so they must be incident to a matched edge. For any vertex in (d), since it is reachable, there is an augmenting path between an unmatched left node and it, and if it were unmatched, then our matching would not be maximum. However we assumed our matching is maximum, so every node in (d) must be matched. Therefore $|VC| = \sum_{e \in M} |\text{VC verticies incident to } e|$.

Now, note that no matched edge is incident to more than one VC vertex. Looking at the diagram, the only problem edges would be matched edges from (c) to (d). However since any vertex in (d) is reachable, if there were a matched edge from it to a node on the left half, then that node would also be reachable, and therefore could not be in (c). Using this fact and the above equation, $|VC| = \sum_{e \in M} |\text{VC verticies incident to } e| \leq \sum_{e \in M} 1 = |M|$.

Thus $|VC| \leq |M|$, and consequentially, |Max Matching| $\geq$ |Min Vertex Cover|.

2 Min-Max relations

What's the point of min-max relations??

2.1 You know when you're at optimality

If you find a feasible solution to two such programs whose objective functions are equal, then you know they are at optimality.

2.2 Good Characterization

For any problem, we want a quick way to prove a solution. For instance, for decision problems, there is a yes/no answer. If the problem is in NP, we can prove in polynomial time that a specific answer is "yes". If it is in $co - NP$, we can prove in polynomial time that a specific answer is "no". In other words, decisions problems with *good characterizations* are in NP or $co - NP$.

Now what about optimization problems, i.e. non-decision problems? This is where min-max relations help. For instance, vertex-cover is a good characterization for max matching, because if you can show a vertex cover of a certain size, we know the max-matching can't be bigger.

Combinatorial Optimization

Lecture 6, Sep. 25, 1990 | **Philip Klein** | Scribe: Alain Mayer

In this handout we will give an overview of the *Linear Programming Model*. First, we will define the problem mathematically, then we will give a first straight-forward algorithm to solve it and finally introduce the important *Duality-Theorem*.

1 A Short Description of the Problem

In this section we will describe the type of problem, that can be solved by linear programming, i.e. the *Linear Programming Problem*(LP-problem).

Informally, the LP-problem is to optimize a linear function over a polyhedron. Mathematically, this can be expressed as (from [GLS88]):

Definition. *Given a* $m \times n$ *matrix* $\mathbf{A}$, *a vector* $\mathbf{b} \in \mathbf{R}^m$, *and a (row-)vector* $\mathbf{c} \in \mathbf{R}^n$, *find a vector* $\mathbf{x} \in P = \{\mathbf{x} \in \mathbf{R}^n \mid \mathbf{Ax} \leq \mathbf{b}\}$ *maximizing the linear function* $f(\mathbf{x}) = \mathbf{cx}$.

The function f is usually called *objective function*. If we replace "maximizing" by "minimizing" in the above problem, the resulting problem is also called LP. Thus we will use from now on the term "optimize", that is, our goal is to optimize the objective function.

A vector $\mathbf{x}$ is said to be *feasible* if $\mathbf{x} \in P$. A feasible vector $\mathbf{x}'$ is said to be *at optimality* if $f(\mathbf{x}') = max\{\mathbf{cx} \mid \mathbf{Ax} \leq \mathbf{b}\}$.

Throughout the rest of this handout we will use $\mathbf{A}$, $\mathbf{b}$, $\mathbf{c}$, P and $\mathbf{x}$ as in the definition above.

2 History of Linear Programming

Before we investigate the LP-problem any further, we will give a short historical overview:

- The first hints of the idea of linear programming appear in the works of the French mathematician Fourier [Fou26].
- In the 1940's Dantzig, von Neumann, Kantorovich, and Koopmars developed the theory of linear programming, culminating 1951 in the *simplex method* developed by Dantzig ([Dan51]). The simplex method has proven to be quite practical for a large number of problems, although its worst case performance is not polynomial. This phenomenon has captured the attention of a lot of people in theoretical computer science. Stephen Cook said in his Turing Lecture ([Coo82]): "The identification of P with the tractable (or feasible) problems has been generally accepted in the field since the early 1970's. ...The most notable practical algorithm that has an exponential worst case running time is the simplex algorithm for linear programming. Smale ([Sma83]) attempts to explain this by showing that, in some sense, the average running time is fast, but it is also important to note that Khachian (see below) showed that linear programming is in P using another algorithm. Thus, our general thesis that P equals the feasible problems, is not violated."

Using the above lemma in a straightforward way, we arrive at the following algorithm for solving the LP-problem: Look at all vertices of P; pick the one that has highest value with respect to the objective function $\mathbf{cx}$.
The first refinement of the above is to identify a method to find all the vertices of P:

1. Choose a set of n independent rows of $\mathbf{A}$. (Remember n denotes the dimension of the solution-space) Denote this new $n \times n$ matrix by $\mathbf{A_0}$.

2. Solve the following linear equations $\mathbf{A_0 x} = \mathbf{b}$ and let the solution be denoted by $\mathbf{x_0}$. We can always find a solution since we have constructed $\mathbf{A_0}$ to be nonsingular.

3. Note that if the set $\{\mathbf{x} \in \mathbb{R}^n \mid x \in P,\ \mathbf{A_0 x} = \mathbf{b}\}$ is nonempty, it is a singleton face of P. Thus all we have to do is to check if $\mathbf{x_0} \in P$. If the answer is positive, then $\{\mathbf{x_0}\}$ is a singleton face of P and thus $\mathbf{x_0}$ is a vertex of P.

Obviously there are $\binom{m}{n}$ possibilities to construct $\mathbf{A_0}$. In order to obtain all vertices we have no other choice than to try out all these possibilities. Following the above arguments, we know that $\binom{m}{n} \geq$ (no of vertices of P). When we consider the example of an n-dimensional hypercube, which has 2^n vertices, we can easily see that our algorithm will have an exponential time-performance. We will certainly explore more efficient solutions in the lectures to come.

4 Equivalence of Different Forms of LP-Problems

We mentioned in Section 1 that regardless of whether we are minimizing or maximizing the objective function, we still have a LP-problem. In this section we will show that this and other transformations yield equivalent problems. We will denote the form of the problem, as introduced in Section 1, somewhat arbitrarly as the "canonical" form:

min If we want to put a minimizing problem into "canonical" form, we just have to change the sign of $\mathbf{c}$.

equality If we are given a system of m linear equalities, then we can replace each equality $\mathbf{ax} = b$ by the following two inequalities: $\mathbf{ax} \geq b$ and $-\mathbf{ax} \geq -b$. Thus, in our new problem, which is in "canonical" form, the matrix $\mathbf{A}$ will be have the dimensions $2m \times n$.

constraint If we have an additional constraint on the variable-vector to be nonnegative, i.e. $\mathbf{x} \geq 0$, then we can eliminate this contraint by extending $\mathbf{A}$ with n rows of an negative identity matrix and $\mathbf{b}$ with n zeros. In our new problem, which is in "canonical" form, the matrix $\mathbf{A}$ will be have the dimensions $(m+n) \times n$.

The reader is encouraged to investigate the reverse directions of the above and further variants of the LP-problem, such as mixed equalities/inequalities.

[GLS88] M. Groetschel, L. Lovasz and A. Schrijver, "Geometric Algorithms and Combinatorial Optimization," *Springer-Verlag Berlin* (1988).

[Kar84] N. Karmakar, "A new polynomial-time algorithm for linear programming," *Combinatorica* 4 (1984), pp. 373–396.

[Kha79] L. Khachian, "A Polynomial Algorithm for Linear Programming", *Doklady Akad. Nauk USSR* 244, **5** (1979). Translated in *Soviet Math. Doklady*, 20.

[YL82] Boris Yamnitsky and Leonid A. Levin, 'An old linear programming algorithm runs in polynomial time," 23rd FOCS (1982), pp. 327-32.

[Sma83] S. Smale, "On the average number of steps in the simplex method of linear programming", *Mathematical Programming* **27** (1983), pp. 241-262.

[vNeu] J. von Neumann is credited with being the first to state the duality problem. Private notes circulated at least as early as 1947.

In the simple case presented in the figure, $\gamma_{i\neq 1,2} = 0$ and we may write:

$$c + \gamma_1(-a_1) + \gamma_2(-a_2) = 0$$

or

$$c = \gamma_1 a_1 + \gamma_2 a_2$$

Since all the γ_i are non-negative, c has now been written as a non-negative combination of the rows of A. Also, the only γ_i which are not 0 are those for which $a_i x \leq b_i$ is *tight* at $\hat{x}$, i.e., where $a_i \hat{x} = b_i$. In general,

$$c = \gamma_1 a_1 + \gamma_2 a_2 + \cdots + \gamma_m a_m$$

where many of the γ_i may be 0. So we may write:

$$c\hat{x} = \gamma_1 a_1 \hat{x} + \gamma_2 a_2 \hat{x} + \cdots + \gamma_m a_m \hat{x}$$

and we may replace the $a_i \hat{x}$ in each term with b_i because either the γ_i is 0 or the inequality is tight, giving:

$$c\hat{x} = \gamma_1 b_1 + \gamma_2 b_2 + \cdots + \gamma_m b_m$$

Because the γ_i are non-negative, they may be used as $\check{y}$ and we have $c\hat{x} = \check{y}b$.

The physical analogy suggests a more rigorous proof method based on the idea of moving the point along the exterior of the polytope defined by the inequalities until the optimum point is reached. This is the idea behind the Simplex method.

The algorithm begins at some vertex in the polytope defined by the constraints, and proceeds by moving in some direction that increases the objective function until another constraint becomes tight. At this point, the constraint that was moved away from is discarded, the new constraint is added, and the resulting tight subsystem is used to repeat the process until the optimum is reached.

Let x_0 be a vertex (assume for the time being that we can find one). Let $A_k x = b_k$ be a tight, nonsingular, subsystem defining x_0. Let n be the number of constraints in this system. We wish to choose a direction to move away from this vertex which "loosens" just one of the constraints, and move away from the vertex in this direction until we reach another vertex. To do this, we write c as a linear combination of the rows of A_k, e.g., by using Gaussian elimination. So now we have:

$$c = \gamma_1 a_1 + \gamma_2 a_2 + \cdots + \gamma_n a_n$$

Now if all the $\gamma_i \geq 0$ then we are done since then γ is a dual feasible point such that $cx_0 = \gamma b$.

Otherwise, there is at least one $\gamma_i < 0$, and we choose a direction to move. In order to cope with degeneracy, we order all the constraints at the beginning of the algorithm, and then move away from the lowest ordered constraint that is in our tight subsystem and has corresponding $\gamma_i < 0$. Let a_{i*} be the constraint we choose to move away from. Then Δx, the direction we will move in, is the result of solving the system:

1. if a_j is not in A_p, then $\lambda_j = 0$.
   if a_j is in A_p, then a_j must be either
2. lower-ordered than a_r, in which case $\lambda_j \geq 0$ but $a_j \Delta x \leq 0$, or
3. higher-ordered than a_r, in which case $\lambda_j < 0$ but $a_j \Delta x > 0$, or
4. $j = r$, in which case $a_j \Delta x = 0$, or

Figure 2: Joe Polyhedron

References

[Sch86] A. Schrijver, "Theory of Linear and Integer Programming". John Wiley & Sons, Inc, New York (1986).

2 The Lecture – Duality

Given a linear program $\{ x : Ax \leq b \}$, and an objective function c, a common goal is to find $max\{ cx : Ax \leq b \}$. To solve this, we will create a second linear program inherently related to the first. We will call the original program the *primal* program, and we will create what is called the *dual* program. For the above linear program, the dual is the $min\{ yb : yA = c, y \geq 0 \}$. The importance of the dual is brought out by the following theorem

Theorem 1 (Duality Theorem[1]) *Let (P) denote the linear program $max\{ cx : Ax \leq b \}$, and let (D) denote the dual, $min\{ yb : yA = c, y \geq 0 \}$. If (P) and (D) both have feasible solutions then both problems have optimal solutions and the optimum values of the objective functions are equal.*

Proof: Proving the duality theorem is equivalent to proving the following two conditions hold:

(1) For any feasible x and y, $cx \leq yb$.

(2) There exists some $\hat{x}$ and $\check{y}$, such that $c\hat{x} = \check{y}b$.

(1) follows directly from the definition of the primal and dual spaces. Since $c = yA$ then $cx = yAx$. Also, since $Ax \leq b$ and $y \geq 0$, then $yAx \leq yb$. Combining these two, $cx \leq yb$.
(2) The existence of optimal solutions can be proven by an algorithm. Namely, the simplex method will provide an optimal $\hat{x}$ and $\check{y}$. By running the simplex method on the primal linear program, at termination, a subset of the rows of A will be maintained as a basis, and since the inequalities are tight, by solving the linear systems $Ax = b$ and $yA = c$, $\hat{x}$ and $\check{y}$ can be found. The simplex method will guarantee that these are optimal.

2.1 Complementary slackness

The following result gives conditions which are necessary and sufficient for two vectors to be optimal for a linear program and its dual.

Theorem 2 (Complementary Slackness Theorem[1]) *Suppose x is a feasible solution to the primal linear programming problem (P), $max\{ cx : Ax \leq b \}$ and y is a feasible solution for the dual (D), $min\{ yb : yA = c, y \geq 0 \}$. A necessary and sufficient condition for x and y to be optimal for (P) and (D), respectively is that $\forall i, (y_i > 0 \Rightarrow a_i x = b_i)$. An equivalent form for the condition is that $y \cdot (b - Ax) = 0$.*

Proof: Assume that $\forall i : (y_i > 0 \Rightarrow a_i x = b_i)$. Clearly, if $y_i = 0$, then $y_i a_i x = y_i b_i$. Also if $y_i > 0$, then by the assumption $a_i x = b_i$, and thus $y_i a_i x = y_i b_i$. So $\forall i : y_i a_i x = y_i b_i$. Since this is true for all i, $yAx = yb$, and since $c = yA$, then $cx = yb$. Since the inequality is tight, then x and y are optimal solutions. Therefore if $\forall i, (y_i > 0 \Rightarrow a_i x = b_i)$, then x and y are optimal.

Conversely, Assume x and y are optimal. By duality, the inequality $cx \leq yb$ must be tight. That is $cx = yb$. Since $c = yA$, then the equality can be written as $yAx = yb$. Rewriting this, $\sum_i y_i(a_i x) = \sum_i y_i b_i$, or $0 = \sum_i y_i(b_i - a_i x)$. From the dual, $y_i \geq 0$. From the primal problem, $a_i x \leq b_i$, that is $(b_i - a_i x) \geq 0$. Since both factors are non-negative, then each term in the summation must be non-negative. Since the sum of these non-negative numbers is equal to 0, then each term, $y_i(b_i - a_i x) = 0$. So, for each component, $y_i a_i x = y_i b_i$. Now if $y_i > 0$ then, both sides can be divided by y_i, and thus $a_i x = b_i$. Therefore, if x and y are optimal, then $\forall i, (y_i > 0 \Rightarrow a_i x = b_i)$.

3 Combinatorial Applications of duality

This section discusses how to use duality in combinatorial applications. First a general "game plan" describes the steps to take for using duality in a general application. Following this, the game plan is used on some example combinatorial applications.

3.1 The game plan

1. Let S = Combinatorial structure
2. Let P_S = convex hull of S. (where we interpret S as a set of incidence vectors)
3. Express P_S in terms of linear constraints. (e.g. $P_S = \{ x : Ax \leq b \}$) and express the objective function, c in these terms.
4. Apply duality. (e.g. $max\{ cx : Ax \leq b \} = min\{ yb : yA = c, y \geq 0 \}$)
5. Interpret the dual polyhedron combinatorially

3.2 Example – Bipartite Matching

Now, let's look at how we would use the duality "game plan" on our favorite combinatorial application, bipartite matching.

1. Let S = All matching of a bipartite graph
2. Let P_S = convex hull of matchings = "bipartite matching polytope"
3. Matchings (not necessarily perfect) on a bipartite graph are characterized by the polytope:
   $P_M = \{ cx : Ax \leq \vec{1}, x \geq 0 \}$, where A is the vertex-edge incidence matrix.
   To maximize the matching, the objective function, $c = \vec{1}$.
4. Maximum matchings: $max\{ \vec{1}x : Ax \leq \vec{1}, x \geq 0 \} = min\{ y\vec{1} : yA \geq \vec{1}, y \geq 0 \}$
5. What is $min\{ y\vec{1} : yA \geq \vec{1}, y \geq 0 \}$??
   Let's Pretend y is integral. (this will be discussed in detail next lecture)
   Then, $yA \geq \vec{1}$ means that for each edge, $e = v_1v_2 : \sum_v y_v A_{ve} \geq 1$
   Since A_{ve} is zero for everything but v_1 and v_2, the summation reduces, so that $\sum_v y_v A_{ve} = \sum_{v_1, v_2} y_v A_{ve} = (y_{v_1} + y_{v_2})$. So for each edge, $(y_{v_1} + y_{v_2}) \geq 1$.
   This corresponds to the vertex covering problem, and thus from duality, we get the **pretend** theorem that: Max Matching = Min Vertex Covering.

References

[1] M. Grötschel, L. Lovász, A. Schrijver, *Geometric Algorithms and Combinatorial Optimization*, pp.14-16, 1980.

3 Unimodularity

Definition. *A square matrix is unimodular if its determinant is 0,±1. Any matrix is totally unimodular if every square submatrix is unimodular.*

Lemma. *If* $\mathbf{A}$ *is totally unimodular, then so are* $\mathbf{A}^T, -\mathbf{A}, [\mathbf{I}\ \mathbf{A}]$, *and* $[\mathbf{A}\ \mathbf{I}]$, *where* $[\mathbf{M}\ \mathbf{N}]$ *is the matrix formed by "concatenating" the columns of* $\mathbf{N}$ *to the columns of* $\mathbf{M}$.

PROOF: Show the definition of total unimodularity directly by evaluating the determinant of every submatrix by expanding into cofactors. □

It is worth noting that by transposing the concatenated matrices, total unimodularity is closed under vertical concatenation as well.

4 When are polyhedra integral?

Theorem 1 *For integral* $\mathbf{A}$, *the polyhedron* $\{\mathbf{x} : \mathbf{Ax} \leq \mathbf{b}\}$ *is integral for every integral* $\mathbf{b}$ *if and only if A is totally unimodular.*

Since we will use this theorem primarily to show that polyhedra are integral, only the if direction will be proved.

PROOF: (If)

We will prove that every face of the polyhedron contains an integral point, which by lemma 1 is equivalent to saying the polyhedron is integral.

Take a minimal face of the polyhedron, $F = \{\mathbf{x} : \mathbf{A'x} = \mathbf{b'}\}$, where $\mathbf{A'x} = \mathbf{b'}$ is a subsystem of $\mathbf{Ax} = \mathbf{b}$. Let m' be the number of rows in $\mathbf{A'}$. We may assume without loss of generality that the rows of $\mathbf{A'}$ are independent since there is always an independent subsystem with an equivalent solution set which we can consider. Since the rank of A' is m', there are m' independent columns. We can always rearrange the columns in $\mathbf{A'}$ as long as we do the same to the components of $\mathbf{x}$, so assume that the m' independent columns are the first columns of $\mathbf{A}$. Let $\mathbf{A''}$ denote the submatrix of $\mathbf{A'}$ that consists exactly of the first m' columns, and let $\mathbf{x'}$ denote the vector that consists of the first m' components of $\mathbf{x}$. Now if $\mathbf{x}_0''$ is a solution to $\mathbf{A''x''} = \mathbf{b'}$ we can find a vector $\mathbf{x}_0$ that satisfies $\mathbf{A'x} = \mathbf{b'}$ simply by adding enough zero components (through the inclusion mapping) to $\mathbf{x}_0''$ to make the equation $\mathbf{A'x}_0 = \mathbf{b'}$ syntactically correct. Thus to show that $\mathbf{x}$ is integral, it suffices to show that $\mathbf{x}_0''$ is integral. We do this as follows.

Since $\mathbf{x}_0'' = \mathbf{A''^{-1}b'}$ (Note: We can invert $\mathbf{A''}$ since its columns are independent and it is square.) and since $\mathbf{b'}$ is integral, it suffices to show that every entry in $\mathbf{A''^{-1}}$ is integral. We do this by using Cramer's rule to evaluate each entry.

Cramer's Rule: If M is invertible, the $(i,j)^{th}$ element of $\mathbf{M}^{-1}$ is $(-1)^{i+j}\det(\mathbf{M}_{[i,j]})/\det(\mathbf{M})$ where $\mathbf{M}_{[i,j]}$ is the matrix formed by eliminating the i^{th} column and j^{th} row from $\mathbf{M}$.

Since $\mathbf{A}$ is totally unimodular, $\mathbf{A''}$ is unimodular which implies its determinant is -1, 0, or 1. Since the columns of $\mathbf{A''}$ are independent, its determinant is -1 or 1. Thus, the denominator of every entry is ±1. Since the determinant of any integral matrix (specifically $\mathbf{M}_{[i,j]}$) is integral, the numerator, and hence the value, of any entry in the inverse is integral. Thus there is an integral point on any arbitrary face, F. □

Theorem 2 *The maximum weight of the matchings in an integrally weighted bipartite graph is equal to the minimum multiset cardinality of the c-vertex covers.*

PROOF: Let $\mathbf{A}$ be the node-edge incidence matrix for the graph. The matching polytope is $M = \{\mathbf{x} : \mathbf{Ax} \leq \mathbf{1}, \mathbf{x} \geq \mathbf{0}\}$ and the c-vertex cover polytope is $VC = \{\mathbf{y} : \mathbf{yA} \geq \mathbf{c}, \mathbf{y} \geq \mathbf{0}\}$ where $\mathbf{c}$ is the cost vector of the edges. By LP-duality, $\max\{\mathbf{c} \cdot \mathbf{x} : \mathbf{x} \in M\} = \min\{\mathbf{y} \cdot \mathbf{1} : \mathbf{y} \in VC\}$. These expressions are maximum weight matching and minimum c-vertex cover, respectively. Since $\mathbf{A}$ is totally unimodular, both optimal faces contain an integral point. Since these optimal points are integral, each is a valid solution to its problem. □

6 References

Hoffman and Kruskal, Integral boundary points of convex polyhedra,*Linear Inequalities and Related Systems*, Princeton Univ. Press, Princeton, NJ, 1956, pp 223-246.

Egervary, On combinatorial properties of matrices, *Mat. Lappk.*, 38, 1931, pp 16-28.

We are looking for maximum weight perfect matching: Formally, (by LP-Duality), we are looking for the following conditions to hold:

$$max\{\,\mathbf{cx} : \mathbf{Ax} = \mathbf{1}, \mathbf{x} \geq \mathbf{0}\,\} = min\{\,\mathbf{y1} : \mathbf{yA} \geq \mathbf{c}\,\}$$

where **A** is the node-edge incidence matrix. (Note: Since we have equality on left, we do not need equality on right of the expression.)

We will need to have a dual variable for each vertex. In order to do this, recall the previously mentioned alternative characterization of a face as an optimization for some vector, **c**. Everything that was given in this form is a face.

2 Complementary Slackness Conditions: [review]

Fact: The conditions that hold for feasible x and y, can only hold if x and y are both optimal. To rederive this fact, use easy direction:

$$\begin{aligned} max &\leq min \\ \mathbf{cx} &\geq \mathbf{y1} = \mathbf{yAx} \\ \mathbf{(yA - c)x} &\geq \mathbf{0} \\ Either : \mathbf{(c)}_e = \mathbf{(yA)}_e &\ or \quad \mathbf{(x)}_e = 0. \end{aligned}$$

That is, e is *not* in the matching, or e is covered exactly c_e times. (Note, specifically, that if it is an edge is in the matching, it falls under the latter condition)

3 Reduced Costs

For an edge, v_1v_2, $\overline{c_{v_1v_2}} = c_{v_1v_2} - y_{v_1} - y_{v_2}$: if y is a feasible vector, then every reduced cost is non-positive.

$\forall$ Perfect Matchings, General Matchings:

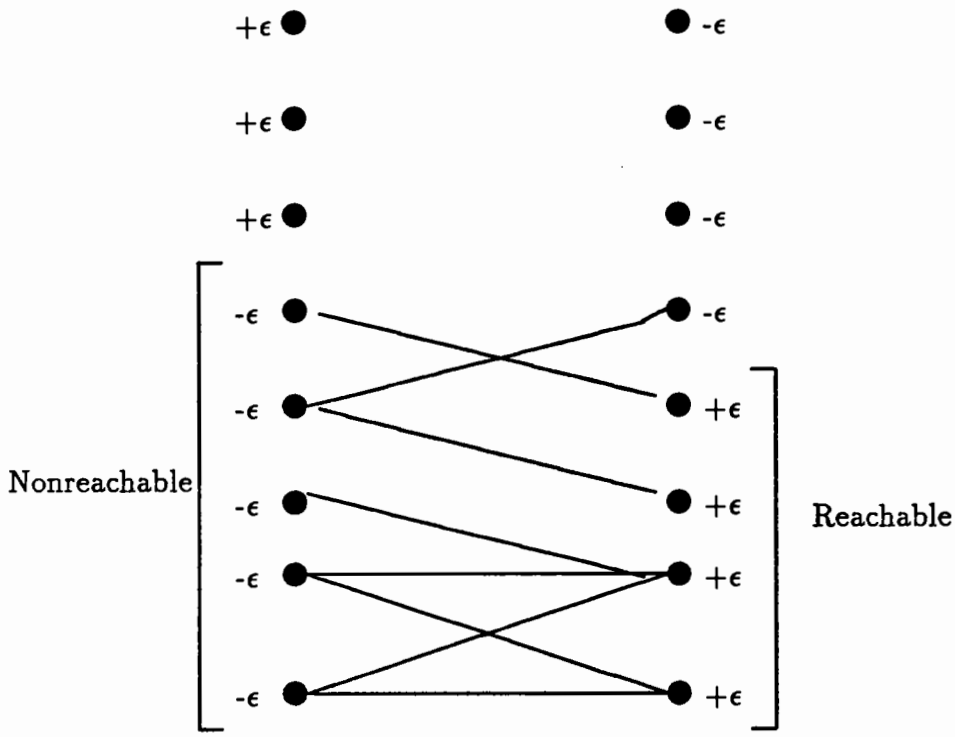
$$\begin{aligned} \textstyle\sum_{(v_1,v_2)\in M} \overline{c_{v_1v_2}} &= \textstyle\sum_{(v_1,v_2)\in M}(c_{v_1v_2} - y_{v_1} - y_{v_2}) \\ &= \textstyle\sum_{(v_1,v_2)\in M} c_{v_1v_2} - \sum_{(v_1,v_2)\in M}(y_{v_1} + y_{v_2}) \\ &= \textstyle\sum_{(v_1,v_2)\in M} c_{v_1v_2} - \sum_{v\in M}(y_v) \\ &= \textstyle\sum_{(v_1,v_2)\in M} c_{v_1v_2} - y_{v_1} - y_{v_2} \end{aligned}$$

$\forall v \in V, w \in W,$

As we optimize reduced costs, we simultaneously optimize original costs. (order of Perfect Matchings by cost is unchanged)

Question: For reduced costs, in order to obtain complementary slackness, we will need all non-negative weights. In other words, we have matching such that reduced costs are all zero. (Complementary Slackness dictates that edges in matching are zero cost [zero reduced cost]. In other words, every edge must be covered exactly. Any matching satisfying complementary slackness, has zero cost.)

Now, we need a perfect matching with zero cost. We have found a matching, but it is not necessarily perfect.



where $\epsilon = \frac{1}{2}(mininumslackness)$

Key observations:

Matched edges are not affected, from unreachable → unreachable

This will only affect edges from reachable → unreachable, or unreachable → reachable (The unreachable can only get **more negative**)

6 Timing of Algorithm Discussion

Two algorithms already known find the weight of a maximum matching faster than the algorithm that will be provided here. Let n = number of nodes, m = number of edges. One algorithm does exists for finding the weight, which does not depend on cost and can be done in $O(nmlogn)$. A different algorithm exists which runs in $O(n(m + nlogn))$.

Open Problem: Is there an algorithm for finding matchings which will run faster then this and is also strongly polynomial???

a negative cycle exists, we can produce an infinitely "short" path by running around the cycle forever. Any path can always be made shorter by running around the negative cycle another time.

With that out of the way, it isn't too hard to realize that we need to add some value to each edge in order to guarantee that all edges are positive. With this done, we can simply apply Dijkstra, and we are finished. However, a simple "add the smallest negative integer to everything" scheme does not maintain the ordering of paths, and our shortest path would tell us nothing about the shortest path in the original graph.

Instead, we derive the *reduced costs* or *reduced lengths* of an edge l between vertices u and v as follows:

$$\bar{l}(uv) = l(uv) + p(u) - p(v) \tag{1}$$

We require that, for $\bar{l}(uv) \geq 0$, the following shall hold true:

$$p(v) \leq l(uv) + p(u) \tag{2}$$

We call $p(u)$ and $p(v)$ the "prices" of u and v, thinking of $p(v)$ as the price you must pay to leave v or the amount you gain by entering v. Thus, when leaving a node, you pay a penalty, since you are adding length to your path.

In order to obtain non-negativity, we choose (that is to say, compute) prices to satisfy the constraint given in Equation 2. Note that this constraint allows us some flexibility, as we can add to both sides and still maintain the constraint. We can now give an algorithm.

For convenience, we set the lowest price to zero. The prices are related to the shortest path lengths. For a path P given by $(v_0v_1)(v_1v_2)\ldots(v_{n-1}v_n)$, we have the following:

$$\begin{aligned} \bar{l}(P) &= l(v_0v_1) + p(v_0) - p(v_1) \\ &\quad + l(v_1v_2) + p(v_1) - p(v_2) \\ &\quad + \cdots \\ &= l(P) + p(v_0) - p(v_n) \end{aligned} \tag{3}$$

Thus, the reduced cost is different from the original cost by the difference in the costs of the beginning and end of the path, $p(v_0) - p(v_n)$.

But why is this useful? If we use the length of a path as its ordering value, then the order of paths from v_0 to v_k is unchanged by the reduced costs.

It turns out that $p(v_0)$ and $p(v_k)$ are directly related to the y's in duality. We see this by making the graph bipartite in the following way. Duplicate each vertex. From each vertex to its new duplicate, create an edge with weight zero. Old edges now go from the original source node to the duplicate of the destination node.

Such a graph, with associated weights for some sample edges is given in Figure 1. We can see the edge from node v_0 to node v_0', marked with its weight of 0. An edge from v_0 to v_k in the real graph is given by an edge from v_0 to v_k' in the new bipartite graph. This edge is marked with its weight, which is the negative of its length, or $-l(v_0, v_k)$. Thus, we can represent a directed, weighted graph as a bipartite graph. Since bipartite graphs are what we have been developing the duals for, we immediately have a large number of useful tools to apply to the graph.

Conjecture 2 *There exists a perfect matching of weight zero in the given bipartite graph.*

since the cost of matched edges is zero, from Equation 4.

We choose the dual variable y as the price p for each node, since we have more dual variables than prices (we have one price per original node, and one dual per edge in the new graph). In particular, if v_1v_2 is an original edge, we can rewrite Equations 1 and 2 as follows:

$$\begin{aligned} l(v_1v_2) &\leq y(v_1) - y(v_2) \\ y(v_2) &\leq y(v_1)l(v_1v_2) \end{aligned}$$

Gabow and Tarjan have shown that we can thus find a matching in a graph with negative edge lengths in $O(\sqrt{v}e\log(vC))$ time.

3 Other Applications

Other applications of the dual variables include input and output within a cosntant order of optimal. Minimum cost can be implemented as a subroutine. Duals can also be applied to problems involving degree-constrained subgraphs, such as paths and T-joins, both of which will be discussed in future lectures.

4 Forging Ahead: The Man, The Myth, The Legend, Edmonds

During the second half of the class, we delved into new territory. Probing deep into the jungles of graph theory, we left the neat and orderly realm of bipartite graphs far behind. And who did we find in these jungles? "Dr. Edmonds, I presume?"

Actually, the main point of this part of the lecture was that we handle non-bipartite matching problems by pretending that the graph is actually bipartite. How we manage such a piece of graphical flim-flammery is due to Edmonds.

The legend has it that Edmonds saw a lecture on bipartite matching, and promptly went home and deduced how to solve non-bipartite matching. Furthermore, he figured out how to do it well (i.e. in polynomial time). In order to show that he was indeed doing non-bipartite matching well, he had to come up with what it meant to have a good algorithm (this occuring in the years Before Polynomiality). Along the way, he described a couple of other heretofore unheard of notions, like NP and Co-NP.

The scribe's playfulness notwithstanding (only fair, since we're talking about a paper called "Trees, Paths, and Flowers"), Edmonds was an important figure in graph theory. We will now discuss Edmonds's method for forming a bipartite graph from a non-bipartite one. Once this is done, we can apply the same tools we have been developing all along.

4.1 Finding an Augmenting Path

Recall our theorem:

Theorem 1 *For any graph G and matching M, M is maximum if and only if there is no augmenting path in G with respect to M.*

We have a simple algorithm for maximum matching in a graph G. We start with an empty matching $M = \emptyset$. While there exists an augmenting path, we augment the matching M.

Edmonds termed such an almost alternating cycle a "blossom."[?]. With respect to a matching M, a blossom is an odd, almost-alternating cycle of matched and unmatched nodes. We term the cycle proper the blossom, and the alternating path leading to it the stem. The class termed the entire structure a "flyswatter" for reasons that can best be attributed to Figure 3. Note that the stem may be empty.

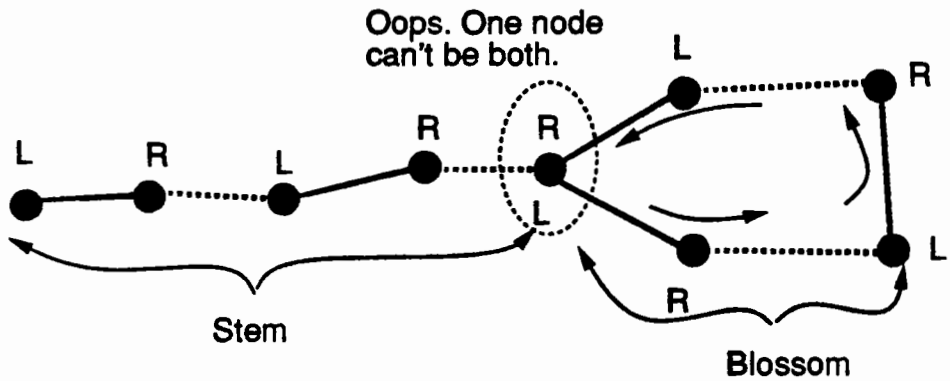


Figure 3: A blossom with its stem.

We can now "shrink" the blossom ("wilting"). All nodes of the cycle go "shwup" (cue sound effects). The edges leading into the blossom remain the same and still connect to the blossom, now a single node. The resulting graph is bipartite (until we find another blossom) and is thus suitable for the matching algorithms we have been developing. We use G/B to denote the graph G after the blossom B has been shrunk, and b to denote the node in G/B produced from B (see Figure 4).

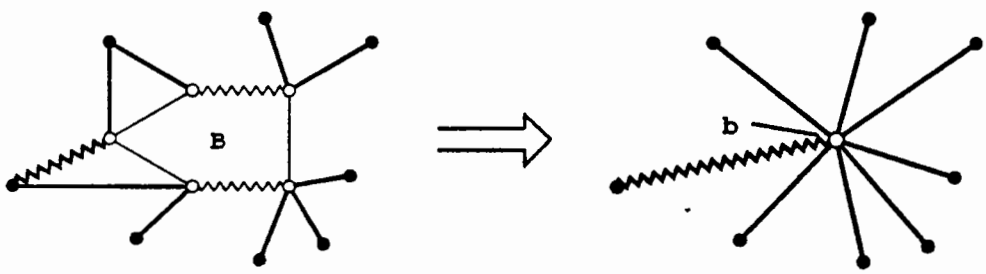


Figure 4: Shrinking a blossom B into a node b in the new graph G/B.

4.3 How Do Blossoms Help?

In order to really make use of blossoms, we need a condition relating an augmenting path in the graph with the shrunken blossoms to an augmenting path in the original graph.

Theorem 2 *For any graph G and matching M and blossom B, there exists an augmenting path in G if and only if there exists an augmenting path in G/B.*

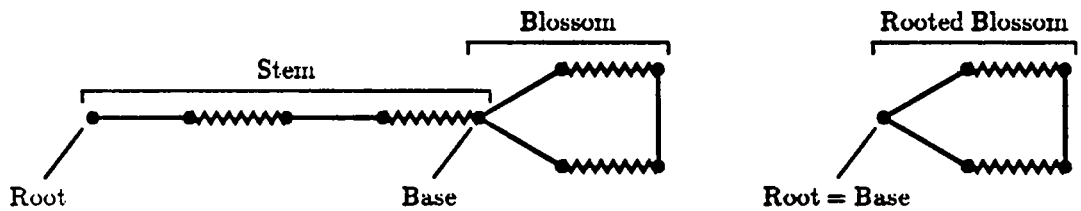
Combinatorial Optimization

Lecture 12, 16 October 1990 | **Philip Klein** | Scribe: Davide P. Cervone
Lecture 13, 18 October 1990

We continue our discussion of non-bipartite matchings be reviewing blossoms, and then use these to produce an algorithm for finding a maximum matching in non-bipartite graphs.

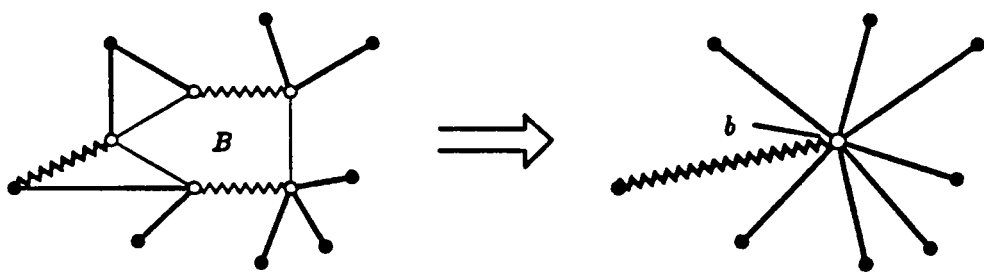
1 Blossoms, Stems, and Roots

Given a graph G with a matching M, a *blossom* B with respect to this matching is an odd-length alternating path starting and ending at the same node (called the *base* of the blossom) such that the first and last edges of B are not in M and such that there is an even-length alternating path from an unmatched node, called the *root*, to the base. This path is called the *stem*, and it may be of length zero, in which case the blossom is said to be *rooted*.

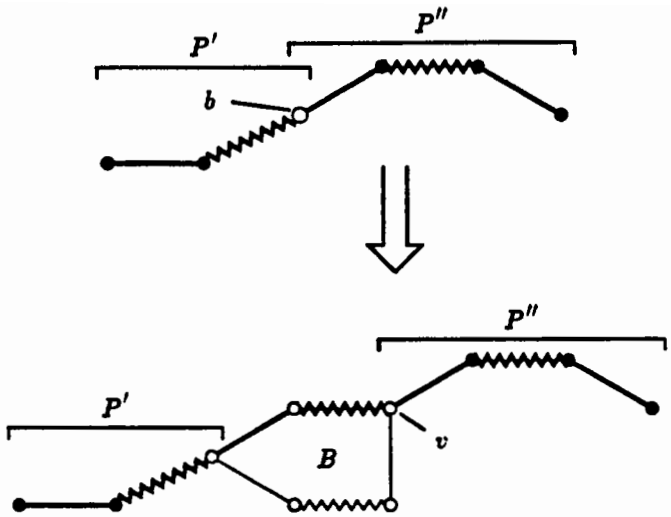


Observe that the only matched edge adjacent to the blossom is the one that is part of the stem (all the nodes except the base are matched within the blossom itself). If the blossom is rooted, then there are no matched edges adjacent to B.

Given a graph G with a matching M and a blossom B with respect to M, we define a new graph G/B, the *contraction* of G with respect to B, in the following way: remove from G the edges of B and identify all the nodes of B to a single node in G/B (multiple edges can be collapsed if desired). This new node will be called B/B, or simply b.



The matching M induces a matching M/B in G/B: matched edges in the un-modified portion of G remain unchanged, and all edges adjacent to B/B are unmatched except for the edge corresponding to the matched edge joining the stem to the base, when the blossom has a stem. Thus b is a matched node in G/B if and only if B has a stem in G.



The path P' in G ends at some node v of the blossom, and there are two paths within the blossom itself that join v to the base. One of these paths begins with a matched edge, since B is an alternating path (if v is the base itself, theis path will be of length zero). We can join this portion of the blossom to the path P to obtain an alternating path in G that ends at the base of B. This, together with P'', is an augmenting path in G. ■

Note that, although the first half of the proof is existential, the second half is algorithmic. We will use this in the design of the Edmund's algorithm as follows:

- Look for an augmenting path, pretending that G is bipartite.
- If we encounter a blossom, contract it and continue the search.
- If we do not find an augmenting path, then we have a maximum matching.
- If we find an augmenting path, it may be in a contracted graph. Turn this into an augmenting path in G *via* the procedure given in the proof of lemma 2.2 above.
- Augment the matching using this path.
- Repeat this procedure until no augmenting path is found.

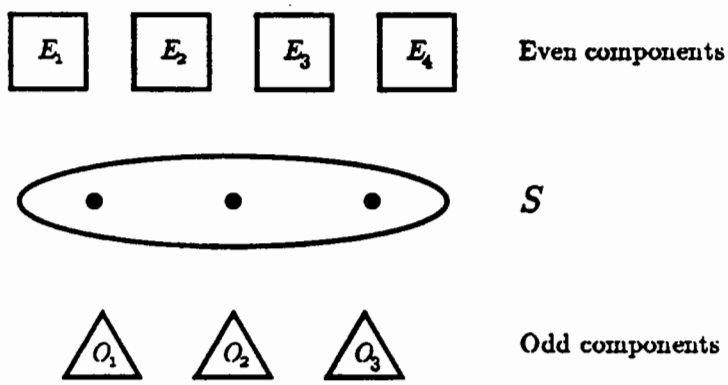
In the following section we describe this algorithm more completely, and hint at some of the implementation details and efficiency bounds.

3 Edmond's Algorithm for Maximum Matchings

In more detail, the algorithm is the following:

I) Start with $M = \emptyset$. We will maintain a stack of nodes, and a list of blossoms that have been contracted.

Suppose G is a graph and M is a perfect matching in G. Let S be any subset of nodes of G, and let $G-S$ be the subgraph of G formed by removing S and any edges incident to S from G. $G-S$ may not be connected, so let $E_1, \ldots, E_l$ be the connected components of $G-S$ that have an even number of nodes, and $O_1, \ldots, O_k$ be the connected components of H that have an odd number of nodes.



Now none of the O_j can be perfectly matched internally (by Goldwasser's Lemma), so if M is a perfect matching, at least one node in O_j must be matched to a node outside of O_j. But the O_j and E_i are connected components of $G-S$, so the only edges leading out of O_j must end in S. For M to be a perfect matching, there must be at least one node in S for each component O_j; that is, $|S| \geq k$.

Generalizing this result, we see that the number of unmatched nodes in *any* matching is at least $k - |S|$. This is the first part of Tutte's theorem, which we now state.

Theorem 4.2 (Tutte) *The minimum number of unmatched nodes (in a graph with a maximum matching) equals* $\max_{S \subseteq V_G} [(\textit{number of odd components of } G-S) - |S|]$.

Proof: The observations above show that the minimum number of unmatched nodes is at least this big, so we have one direction completed already.

For the other direction, we will provide an algorithmic proof using Edmond's unweighted matching algorithm. We perform the Edmond's algorithm and produce a maxmimal matching. Recall that in this algorithm, we may be contracting blossoms as we go; we will use $\tilde{G}$ to represent the final contracted graph. In the last step of the algorithm, we have attempted to find an augmenting path starting at each of the unmatched nodes in $\tilde{G}$. As we test the possible paths from an unmatched node, we mark their nodes "to be ignored" so that we don't re-test them later. When all paths from a given node are exhausted, the entire subgraph of $\tilde{G}$ containing that node will be marked as ignored.

The order and labelling that we have performed during the algorithm imposes a special structure on these subgraphs: we will have traced out a spanning subtree of the subgraph, and the nodes will be labelled alternately LEFT and RIGHT. We will call these subgraphs ***incomplete subgraphs.*** When no augmenting path is found, we will have decomposed $\tilde{G}$ into a collection of incomplete subgraphs rooted at unmatched nodes, together perhaps with some perfectly matched subgraphs (*i.e.*, every node in them is matched). These latter we will call ***completed subgraphs.***

since the blossoms are connected, and they also remain odd, since every blossom contains an odd number of nodes (the proof of this is by induction: the first blossom has an odd number of nodes since we are contracting an odd-length cycle; later, if we contract a blossom that has contracted blossoms as its nodes, we are combining an odd number of odd numbers, which is an odd number).

Thus each LEFT-node remains an odd component as we uncontract, and since only LEFT-nodes are blossoms, the number of odd components of $\tilde{G} - S$ equals the number of odd components of $G - S$ (each node of S is an original node of G, and it is valid to consider $G - S$).

Putting all this together, we have

$$\begin{aligned}
&\big|\{\text{odd components in } G - S\}\big| - |S| \\
&\quad = \big|\{\text{odd components in } \tilde{G} - S\}\big| - |S| \\
&\quad = \big|\{\text{incomplete subgraphs of } \tilde{G}\}\big| \\
&\quad = \big|\{\text{nodes in } \tilde{G} \text{ unmatched by } M\}\big| \\
&\quad = \big|\{\text{nodes in } G \text{ unmatched by } M\}\big|.
\end{aligned}$$

Thus

$$\begin{aligned}
&\big|\{\text{nodes in } G \text{ unmatched by } M\}\big| \\
&\quad \geq \min_{M'} \Big[\big|\{\text{nodes in } G \text{ unmatched by } M'\}\big|\Big] && \text{(true for any } M\text{)} \\
&\quad \geq \max_{S'} \Big[\big|\{\text{odd components in } G - S'\}\big|\Big] && \text{(easy direction of theorem)} \\
&\quad \geq \big|\{\text{odd components in } G - S\}\big| && \text{(true for any } S\text{)} \\
&\quad = \big|\{\text{nodes in } G \text{ unmatched by } M\}\big| && \text{(from above)}
\end{aligned}$$

from which we can conclude that the inequalities above are actually equalities. In particular, the second and third lines are equal, which concludes the proof of the theorem. ■

5 References

[1] J. Edmunds, "Paths, Trees, and Flowers," *Canad. J. Math.* **17** (1965), 449–467.

[2] W. T. Tutte, "The Factorization of Linear Graphs," *J. London Math. Soc.* **22** (1947), 107–111.

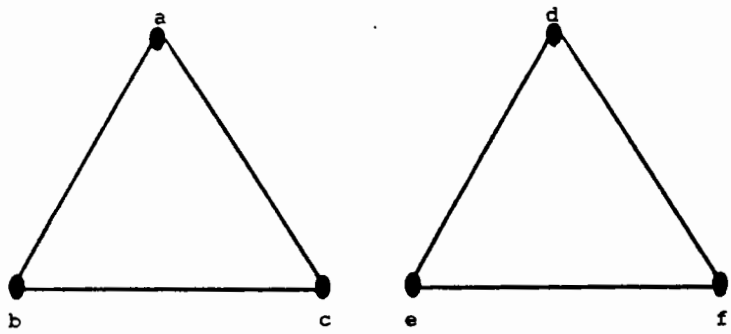


Figure 1: Counter Example

algorithm to solve the following linear programming problem: $\min\{cx : x \in P\}$. We know from exercises in the past that for any polyhedron P and a vertex v there exists a cost function c such that cx is minimized only at v. Therefore if we can demonstrate that for any given cost function c, $\min\{cx : x \in P\}$ gives us an x which is a perfect matching we would've proved that every vertex of P is a perfect matching. Which would in turn establish the claim. The algorithm we are going to use for this purpose is given below

The Min-weight Matching Algorithm

Maintain dual variables and thus reduced costs.
Keep some version of Complementary Slackness.
Repeat
  Repeat
    Get a maximum matching (using an unweighted matching algorithm).
    on the subgraph containing only "allowed" edges.
    If the current matching is not perfect, perform "dual updates".
  Until you find an augmenting path of allowed edges.
  Augment the path just found.
Until the matching is perfect.

This is very similar to the algorithm for the bipartite case, the difference so far is that we are using Edmond's maximum matching algorithm for general graphs, instead of the one for bipartite graphs. We now need to write down P in terms of LP-inequalities, so that we may get the dual variables, and find the appropriate slackness criteria. To do that we define two matrices A the usual node-edge incidence matrix, and B the oddset-edge incidence matrix, the definition now becomes

$$P = \{x : Ax = \bar{1},\ Bx \geq \bar{1}, x \geq 0\}$$

The LP is therefore

$$\min\{cx : Ax = \bar{1},\ Bx \geq \bar{1}, x \geq 0\}$$

and the dual

$$\max\{[y|z]\bar{1} : yA + zB \leq c,\ z \geq 0\}$$

To get the complementary slackness condition we look at the easy direction of the duality theorem, which gives us

$$y\bar{1} + z\bar{1} \leq yAx + zBx = (yA + zB)x \leq cx$$

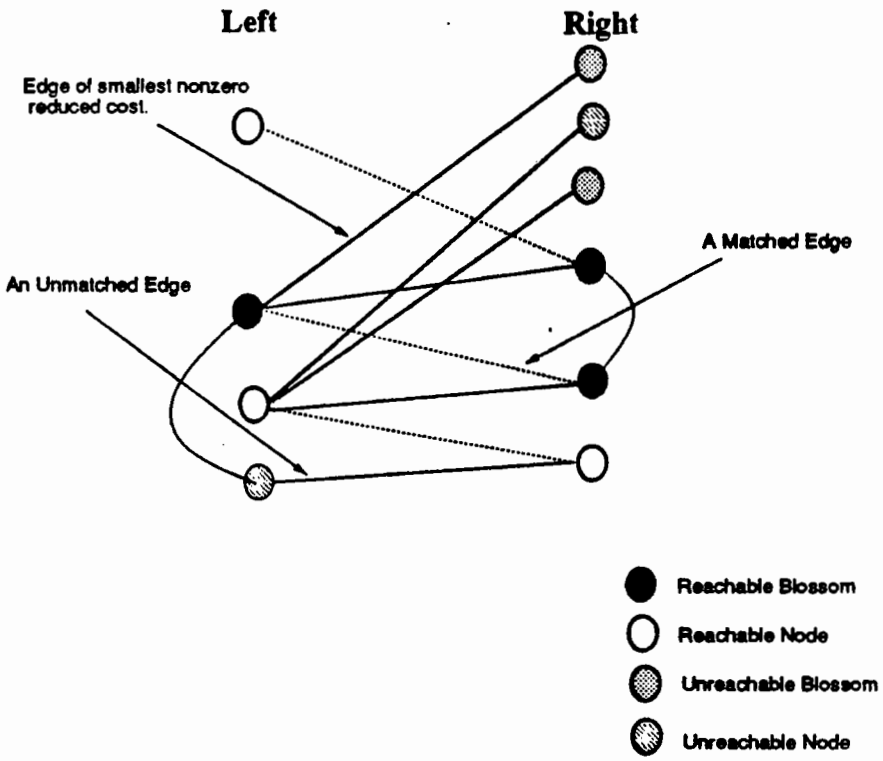


Figure 2: Pictorial view of the augmenting step

we want to reduce the cost of edges from left reachable nodes to unreachable nodes exactly as in the bipartite case (since that would result in the growth of some potential augmenting paths). Let ϵ be the minimum reduced cost edge between left reachable and unreachable nodes. As before we could increase all the left reachable nodes' y (or z if it is a blossom) values by ϵ; and to compensate, we could decrease all the y (or z if it is a blossom) values of the right reachable nodes by ϵ. Such a change would result in at least one more zero-(reduced)cost edge from a left reachable node to an unreachable node, thus increasing a potential augmenting path. Unfortunately this can lead us to two kinds of problems

1. An edge between two left reachable nodes now has negative reduced cost, since the y (or z) values of both its end points were increased.
2. A right blossom S now has z_S less than zero, since we subtracted ϵ from all the right reachable nodes.

To take care of these two cases we do the following:

1. Let ϵ_1 be the minimum reduced cost edge between left reachable and unreachable nodes.
2. Let $\epsilon_2 = 1/2 \min(c_{uv} | u \text{and } v \text{are both reachable left nodes})$.
3. Let $\epsilon_3 = \min(S : S \text{ is a right blossom})$.

We now let $\epsilon = \min(\epsilon_1, \epsilon_2, \epsilon_3)$. As before we add ϵ to all the left reachable nodes and subtract the same from all the right reachable nodes. We are now sure to maintain the feasibility of the new

2. For each S such that $z_s > 0$, $(Bx)_S \leq 1$, that is for each odd set S in the optimal solution for the dual problem, there is at most one edge leaving that set in the optimal matching.

With this revised slackness condition it is easy to see that the algorithm will find the minimum cost perfect matching if one exists.

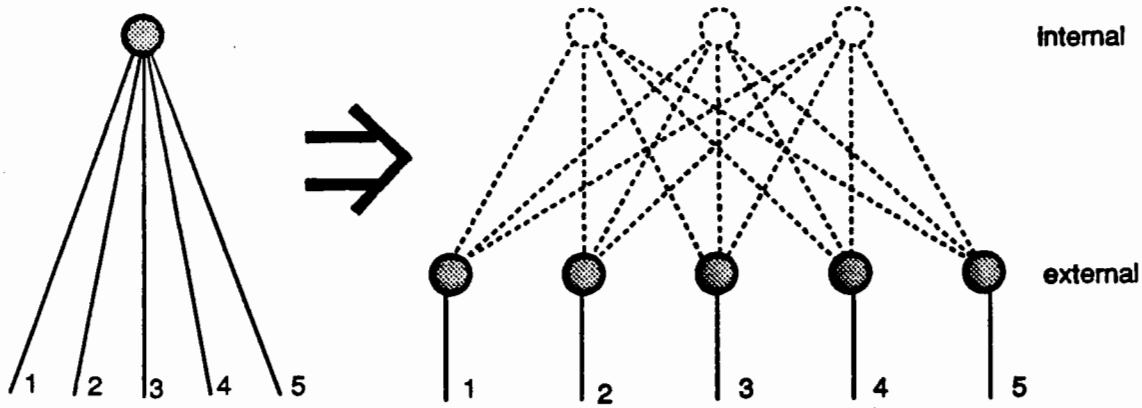


Figure 1: The construction for the exact min-weight DCS problem with $U(v) = 2$.

for v will be given a zero-weight edge to each of the external vertices for v. This construction is illustrated in Figure 1.

Now we can observe that any perfect matching in G' corresponds to an exact DCS solution in G. Consider how a perfect matching affects the vertices in G' that were built from a vertex $v \in V_G$. Every internal vertex of G' must be matched to one of its corresponding external vertices, because there are no other vertices connected to the internal vertices. Since there are $\delta_G(v) - U(v)$ internal vertices, there are now $\delta_G(v) - [\delta_G(v) - U(v)] = U(v)$ unmatched external vertices. Each of these remaining external vertices must each be matched to an external vertex built from some other vertex $v' \in V$. So the external vertices for v are matched with $U(v)$ other external vertices in G'. The construction of the solution subgraph H will give an edge between v and each of the G-vertices corresponding to these other external vertices. So v will have degree $U(v)$ in H, which is what we wanted.

The construction preserves the weights of any edges that it copies from G to G'. So if the perfect matching on G has minimum weight, the subgraph H will also have minimum weight.

2.2 Another DCS Problem

The other example we discussed does not have a particular name, so we'll call it The Problem. For this example, the function U specifies a range of allowable degrees, i.e.,

$$U(v) = \{n \in \mathbf{Z} | l(v) \leq n \leq u(v)\}$$

for a lower-bound function l and and upper-bound funciton u. The degree constraints of The Problem are less stringent than those for the exact DCS problem. The extra flexibility may allow a lower-weight solution H to be found. Or it may allow a solution to be found for a graph that had no solution before.

The approach for solving The Problem will be to construct a graph G' such that a solution to the min-weight exact DCS problem on G' will be a solution for The Problem on G. The construction involves making two copies G^1 and G^2 of G. A vertex v from G will be represented by a vertex v^1 in G^1 and a vertex v^2 in G^2. Both v^1 and v^2 will get a complete copy of the edges adjoining v, and we'll call these edges the *external* edges. The graph G' will be the union of the two graphs G^1 and G^2 with some extra edges added. There will be $[l(v) - u(v)]/2$ extra edges added between each v^1

Combinatorial Optimization

Lecture No.16, 10/30/90 | **Philip N. Klein** | Scribe: R. Ravi

1 Review : A generalization of shortest $r - s$ path

First, we review what was done during the last class. We considered the problem of finding a shortest $r - s$ path between a pair of nodes r and s in an undirected graph with costs on its edges. These edge cost are allowed to be negative. If the graph is directed, we could reduce the problem to one with no negative weights after preprocessing the graph as follows. First, we check that there are no negative cycles in the graph for this could lead to an unbounded solution. Then, we use the notion of reduced cost discussed in an earlier lecture to come up with reduced lengths that are all nonnegative. We do this in a way that the shortest path in the resulting graph is still the shortest path of the original and that these differ in value by a quantity that we can determine easily. We then find a shortest $r - s$ path in the resulting graph that has nonnegative weights to complete the solution.

The story is different in the case of undirected graphs. We need more elaborate machinery to solve this. Consider the following two problems.

Problem P1:
Given an undirected graph G with arbitrary weights on its edges and two distinguished nodes r and s and given that G has no negative cost cycles, find an edge-subgraph of minimum weight in G such the degree of both r and s in this subgraph is exactly one and that of every other node is either zero or two.

Problem P2:
Given an undirected graph G with arbitrary weights on its edges and two distinguished nodes r and s and given that G has no negative cost cycles, find an edge-subgraph of minimum weight in G such both r and s have odd degree in this subgraph and every other node is of even dgree in this subgraph.

Firstly, note that the problem P1 is exactly the shortest $r-s$ path problem in G that we started this section with. We claim that P2 is more general than P1 in that P1 can be reduced to P2. Consider a solution to the problem P2. Let this edge-subgraph be F, say.

1. The edge-subgraph F contains a simple path between r and s. This is easy because if we start at r and follow any path, we keep eating off a degree of exactly two for each node we traverse in the path. Since all degrees are even (including that of r where we used one of its degree to start on this path), the only place we could end is s. This path we found can be easily trimmed into a simple path (containing no cycles). We call this simple path P.

2. Clearly, the edge-subgraph $F - P$ has even degree at every node in G. This is because removing the path P reduced the degree of each of r and s by an odd quantity (in particular, by one) and of every other vertex by an even quantity (i.e., two). Since every node in G except r and s had even degree to begin with, the claim follows.

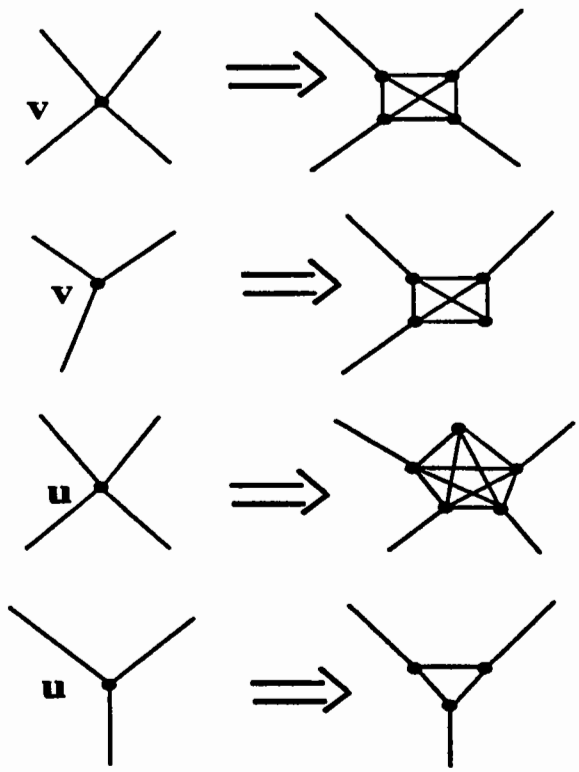


Figure 1: Transforming G to G' : If a node $v \in V - \{r, s\}$ is of even degree, we replace it by a clique of size equal to the degree of v and pair off the incident edges of v with the nodes in this clique. If v is of odd degree, we add an extra node in the clique. For the case of $u = r, s$, we replace u by a clique of size equal to its degree if u is of odd degree and add an extra node in this clique if u is of even degree.

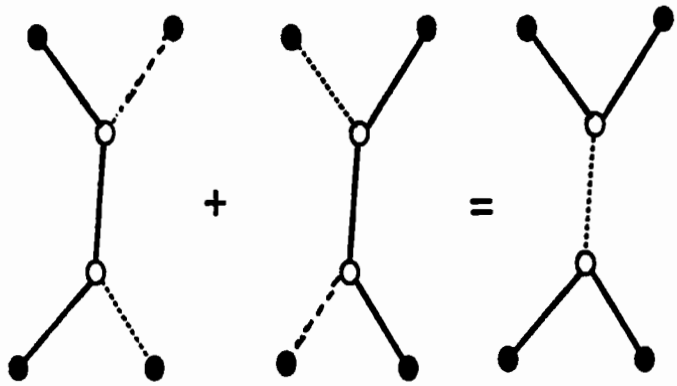


Figure 3: The modulo two sum of paths between the darkened T-nodes forms a valid T-join. The paths are shown darkened in the picture and the other edges not in the path are shown dotted. An edge like the edge between the two non-T-nodes in this figure, occuring in both the paths is cancelled in taking the modulo two sum.

3.1 Perfect Matching

Notice that the min-cost perfect matching problem is just a minimum weight T-join problem with $T = V$. However, this is cheating for we really used min-cost perfect matchings to find a solution to the T-join problem. However, the formalism of T-joins captures the notions of perfect matching and shortest $r - s$ path trivially as illustrated in section 1. [1]

3.2 The Chinese postman problem or the ethnic mail-carrier

Here, we are presented with a model of a city with the edges representing streets which a postman must tour and deliver mail. We may assume that the edges are weighted by the distance of the streets they represent and that the graph representing the city is connected. The objective is to find a minimum cost tour that visits all the edges at least once. As background, we call a graph *Eulerian* if each node in the graph is of even degree. It is not hard to see that this is a necessary and sufficient condition for the graph to have a tour that visits each edge exactly once and returns to its starting point. We call such a tour of the graph an *Euler tour*. Neccesity is easy to see. For sufficiency, notice that once we start at a vertex, each time we cross any other vertex in the tour, we always have an edge to leave by. Also, each time we go through an edge like this, we are reducing its degree by an even amount. So the only place we can end such a tour is the starting vertex itself. Notice that this also follows trivially from an earlier observation that every graph with nodes of even degree only is just a collection of cycles. If the graph is connected, it is easy to combine these cycles into a single tour that is an Euler tour.

The solution to the Chinese postman problem on an Eulerian graph is just an Euler tour. If the graph is not Eulerian, we need to find a tour that visits all the edges at least once and maybe visits a few edges more than once. Notice that this corresponds to finding a minimum cost duplication of some edges in the graph to make it Eulerian. Also, duplicating an edge r times is equivalent to duplicating it only (r *mod* 2) times. This is because duplicating any edge twice would achieve

[1] It is also intersting that the polyhedral charachterisation of the T-join polytope yields a characterisation of the polytope (or more correctly, the dom) of perfect matchings of a graph.

Combinatorial Optimization

Lecture No. 17, 11/01/90 | **Philip N. Klein** | Scribe: Ajit Agrawal

1 Overview

In the previous lectures we saw that the problem of T-join can be solved in polynomial time. We also saw that using T-join we can solve the problem of finding the shortest path between a given pair of nodes in a graph with positive and negative edge weights. In this lecture, we present more applications of T-join.

2 The Problem of T-join

Recall that the problem of T-join is the following. We are given a graph G with costs on the edges, and a subset T of the nodes of G. The problem is to find a minimum cost subgraph G' of G such that the nodes of T, and only these nodes, have odd degree in G'.

3 Applications of T-join

Here are some of the applications of T-join.

1. Min-Wt Perfect Matching : By choosing T to be the complete node set V of G, the T-join corresponds to the min weight perfect matching.

2. Chinese Postman Problem : The problem is to find a minimum cost tour of a graph such that each edge is traversed at least once. If each edge is traversed exactly once, then such a tour is called an Euler tour of the graph. For an Euler tour to exist, each node must have even degree as the tour must enter a node using an edge and then leave using another edge not used so far. It is well known that this condition is also sufficient. Thus if any of the nodes in the graph have odd degree then the Euler tour does not exist and hence the Chinese Postman tour must traverse some edges more than once.

   In the Chinese Postman tour, no edge will be traversed more than twice, for then the tour can be made shorter. Hence the cost of the tour is the sum of the costs on the edges plus the sum of the costs of the edges that were traversed twice. Finding the minimum cost subgraph that must be traversed twice in the tour can be done using T-joins by considering the set T to consist of all the nodes with odd degree. This was shown in the last lecture.

3. Shortest undirected path between two nodes r and s: By setting the set T to consist of the nodes r and s, the minimum cost T-join corresponds to the shortest path between r and s. Note that the edge costs can be negative.

4. Finding the minimum weight cut in a planar graph G: This is the main subject of this lecture and we elaborate on it in the coming sections.

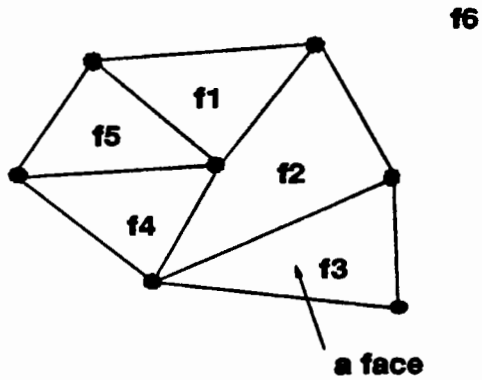


Figure 2: Example of a planar graph: The planar graph in this example divides the plane into 6 regions (including the region outside the boundary of the graph). Each such region is called a *face*.

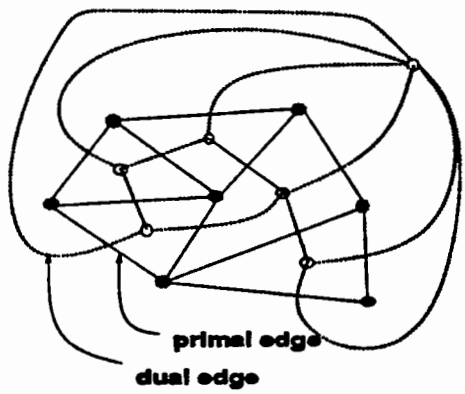


Figure 3: Example of the dual of a planar embedded graph: Solid lines indicate edges of the original graph G. There is a node in the dual for each face in the embedding of G. There is an edge between two nodes in the dual if the faces corresponding to the nodes in the primal share a common boundary edge. The edges of the dual are shown by dotted lines. Notice that the original graph is also the dual of the dual graph.

T. Having found the minimum weight edge disjoint set of cycles in the dual graph G^*, we can recover the minimum weight cut in G.

5.1 The Dual Graph

Consider a planar graph G. Let D be an embedding of G in the plane. An embedding is a representation of how the graph G can be drawn in a plane. There are efficient algorithms known both for checking if a given graph is planar, and for finding an embedding if it is. The graph when drawn in the plane divides the plane into regions, and each such minimal region is called a *face* (see fig. 2).

Given the embedding D, we define the dual graph G^* of G as follows. For each face f in the embedding of G, there corresponds a node v_f in G^*. If faces f and g have a common edge e (share a common boundary in the embedding), then there is a corresponding edge between v_f and v_g in

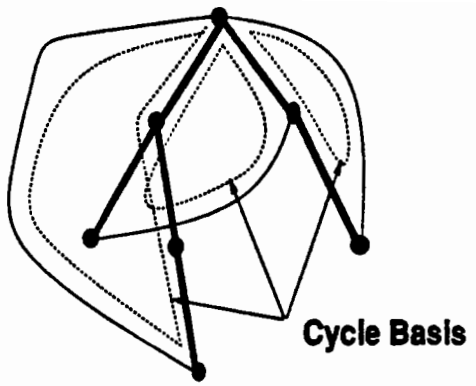


Figure 4: Forming the cycle basis: Solid edged denote the edges of the graph. Darkened edges denote a spanning tree. Each of the non-tree edges form a cycle. These cycles are denoted by dashed loops in the figure. The edge vectors corresponding to these cycles form a set of basis cycle vectors for the graph.

non-spanning cycles is $|E| - (|V| - 1)$, because that is exactly the number of non-tree edges in the graph.

A cocycle basis is formed as follows. Call a cocycle a *star cut* corresponding to a node v if the cut can be represented as $\Gamma(S)$ where S consists of just the single node v. We will show that the set of star cuts corresponding to all nodes except one form a cocycle basis (fig. 5). Note that there are $|V| - 1$ such star cuts. The node whose star cut is excluded can be any node. Let this node be v_0. Any cocycle $\Gamma(S)$, where S does not contain v_0, can be represented as the sum of the star cuts of the nodes belonging to S. If S contains v_0, then $\Gamma(S)$ has another representation, namely $\Gamma(V - S)$, where the set $V - S$ does not contain v_0.

We prove by contradiction that the set of star cuts are independent. Assume that this was not true. Then some set of star cuts add up to 0. Consider the set of nodes corresponding to this set of star cuts. Let this set be denoted by S. The sum of the star cuts represents the set of edges going between the set S and the set $V - S$, where V is the set of nodes of the graph. By assumption there are no edges going out of S. However, the set $V - S$ contains the node v_0 by construction and hence is non-empty (fig. 6). If the graph is connected there must be at least one edge going out of the set S, which leads to a contradiction of our assumption. This shows that the star cuts are indeed independent.

We now show that the star cuts and the non-spanning cycles form a basis respectively for the cocycle and cycle space. We show that the cycle space and the cocycle space are orthogonal, and hence independent, in the edge vector space. Since we have found a total of $|E|$ independent vector elements ($|E| - (|V| - 1)$ cycle basis elements and $|V| - 1$ cocycle basis elements) in an E-dimensional vector space, these elements must form a set of basis elements (fig. 7).

To show that the cycle and the cocycle spaces are orthogonal to each other, consider the scalar product $x \cdot y$ of a vector x in the cycle space and a vector y in the cocycle space. Each edge that is common between the cocycle represented by y and the set of edge disjoint cycles represented by y, contributes 1 in the product $x \cdot y$. Since any cycle must intersect any cocyle cut an even number of times, the number of edges in the intersection between a cocycle and a set of edge disjoint cycles must be even. Thus the value of $x \cdot y$ must be 0 under modulo 2 addition. Hence the cycle and the

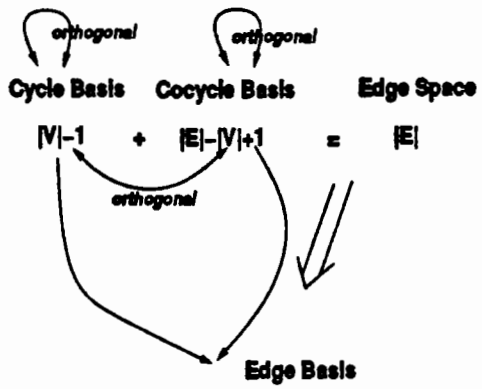


Figure 7: Star cuts and cycles using non-tree edges together form a basis for the edge space: The star cuts are orthogonal to the cycles in question, and together are $|E|$ in number, which implies that they form an edge basis.

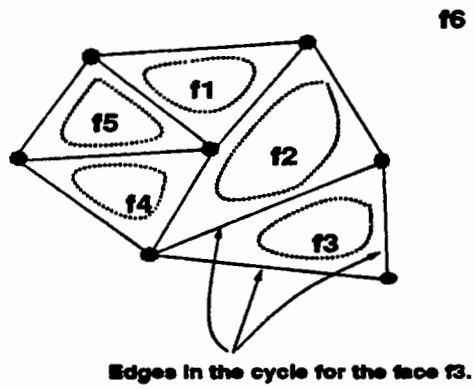


Figure 8: Cycle basis for a planar graph: Cycles corresponding to all nodes except one form a cycle basis. In this figure, the cycle corresponding to the face f_6 is not in the basis.

cocycle spaces are orthogonal to each other.

5.3.2 Forming the Cycle Basis Elements for a Planar Graph G

In the previous section we showed how to construct the cycle basis elements for any graph G. We now show another construction for the cycle basis elements of a planar graph G that will be useful in showing the duality between the cuts of G and the cycles of its dual planar graph G^*.

Consider the set of cycles corresponding to all faces except one in the planar graph. These faces are dual to the star-cuts in the dual graph which we know by arguments presented earlier to be independent. By Euler formula, the number of faces in an embedding of a planar graph equals $|E| - |V| + 2$, where E and V are the set of edges and nodes in the planar graph. It then follows that we have constructed $|E| - |V| + 1$ independent vectors in the cycle space. From the proof in the previous section, the cycle space is of dimension $|E| - |V| + 1$ implying that the edge vectors corresponding to the faces form a cycle basis.

Any set of edge disjoint cycles can be represented as the sum of the basis elements corresponding

Combinatorial Optimization

Lecture 18 and 19, Nov. 6/8, 1990 **Philip Klein** Scribe: Gerd Hillebrand

In this handout, we look at the ellipsoid algorithm—the first linear programming algorithm to run in polynomial time. We'll describe the basic idea, the intricacies of its implementation, refinements of the algorithm for oracle computations, and some applications.

1 The History of LP Algorithms

The standard algorithm for solving linear programs, the simplex method, was devised by G. Dantzig back in 1947. It is still by far the most commonly used algorithm, although it requires exponential time in the worst case (the first such examples were given by Klee and Minty [KM72] in 1972). The good practical performance of the simplex algorithm and some theoretical evidence that LP should not be inherently intractable stimulated a lot of research trying to find a polynomial time LP algorithm, but no progress was made until 1979. In that year, the Russian mathematician L.G. Khachyian published a note describing how an algorithm originally devised a few years earlier for non-linear non-differentiable optimization, the so-called ellipsoid method, could be used to solve linear programs in polynomial time. The announcement created a lot of excitement, but it turned out that its practical implications were rather small. Although a theoretical breakthrough, the ellipsoid method performs much worse than the simplex algorithm on real world problems.

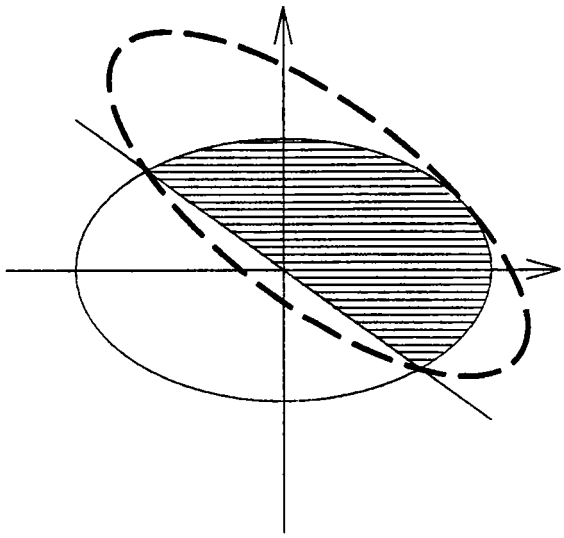
More progress was made in 1984, when N. Karmarkar proposed an interior point method for linear programming that not only runs in polynomial time, but also—at least according to Karmarkar—performs as well as simplex in practice. In the same year, N. Megiddo showed that linear programming can be done in *linear* time for any fixed dimension. His algorithm has since been improved and simplified several times, but it seems unclear whether it is of any practical value.

2 The Ellipsoid Method—An Overview

The ellipsoid algorithm by itself is a method for checking a system of linear inequalities for feasibility. More precisely, the algorithm takes a set of linear inequalities and either produces a vector satisfying these inequalities or reports that the system has no solution. The running time is polynomial in the encoding length of the input, i.e., the total number of bits needed to represent the system of inequalities. It is interesting to note that the algorithm depends in a nontrivial way on the finiteness of this encoding length—it cannot be extended to work on arbitrary real numbers, even if there were a machine that could store and manipulate infinite precision numbers.

The basic idea of the algorithm is as follows. Assume, for simplicity, that the polyhedron described by the given system of inequalities is either fully-dimensional and bounded or empty. Then the algorithm starts out by computing an ellipsoid that contains the entire polyhedron. This is possible because of the finite encoding length of the system of inequalities, which puts a bound on the magnitude of the coefficients and, by Cramer's rule, on the magnitude of the

Löwner-John ellipsoids are hard to compute in general, but there is an explicit formula if K is the intersection of an ellipsoid and a halfspace:



If $E = M\mathbb{B} + a$ and $H = \{ x \in \mathbb{R}^n : x^T c \leq a^T c \}$, then the Löwner-John ellipsoid of $E \cap H$ is given by $E' = M'\mathbb{B} + a'$, where

$$a' = a - \frac{1}{n+1} \frac{M^T M c}{\sqrt{(Mc)^T (Mc)}}, \tag{1}$$

$$(M')^T M' = \frac{n^2}{n^2-1} \left(M^T M - \frac{2}{n+1} \frac{(M^T Mc)(M^T Mc)^T}{(Mc)^T (Mc)} \right). \tag{2}$$

We will use this formula in the iteration step of the algorithm to determine the next ellipsoid. Note, however, a few caveats of this formula:

1. M' is only implicitly given via the product $(M')^T M'$. One can show, however, that the right hand side of (2) is positive definite, which implies that there is a unique solution M' of the equation (see [PS, Thm. 8.3]). In practice, M' never needs to be computed, because only the product $(M')^T M'$ is needed during the next step.

2. Equation (1) for a' involves a square root, which means that in practice only an approximation of a' can be computed. We will see later how to deal with this.

3. It is not obvious from the formulas that E' is indeed smaller than E. By a fairly tedious computation [PS, Thm. 8.3] one obtains, however, that $\text{vol}(E')/\text{vol}(E) = |\det M'|/|\det M| < 2^{-\frac{1}{2(n+1)}} < 1$.

Before we are ready to put the ellipsoid method together, we have to deal with one last issue, namely initialization and termination. Here is where the encoding length of the input system comes into play, because it allows us to give upper and lower bounds on the volume of the corresponding polyhedron:

Lemma 3.2 *Let $P \subset \mathbb{R}^n$ be a non-empty, bounded polyhedron given by a system of strict inequalities $Ax < b$ with integral coefficients. Let L be the encoding size of (A, b) in bits. Then*

General step:

(4) If e_k satisfies $Ax < b$, halt and output e_k.

(5) If $k > N$, halt and print "P is empty."

(6) Otherwise, choose an inequality $\alpha x < \beta$ of $Ax < b$ that is violated by e_k. Compute the Löwner-John ellipsoid E_{k+1} of the intersection of E_k with the halfspace determined by $\alpha x < \alpha e_k$, and let e_{k+1} be its center. Increment k and go back to step (4).

Theorem 4.1 *The algorithm given above correctly decides whether P is feasible.*

Proof: By Lemma 3.2, the initial ellipsoid is large enough to contain all vertices of P, if there are any. This property is preserved at each iteration, because P is always wholly contained in the halfspace $\alpha x < \alpha e_k$. The volume of the current ellipsoid decreases by a factor of at least $2^{-1/(2n+2)}$ at each step and it is easy to see that after the given number of steps, it becomes smaller than the lower bound of Lemma 3.2. Therefore, if no feasible point was found until then, the polyhedron must be empty. □

5 Implementation of the Algorithm

There is one problem with the ellipsoid algorithm as given above: It requires infinite precision arithmetic! (Recall that the computation of the Löwner-John ellipsoid involves a square root.) In practice, computations will be carried out with a certain fixed precision, i.e., all results will be rounded to some number p of bits. This causes the following problem: E_{k+1} is defined to be the smallest ellipsoid containing the intersection of E_k and the halfspace $\{x : \alpha x < \alpha e_k\}$. If E_{k+1} is perturbed because of rounding, it might not contain this intersection (and hence the polyhedron) anymore.

One way of getting around this problem is the following: At each step, blow up the computed ellipsoid by a factor $\zeta > 1$. If ζ is large enough, this will offset the rounding effects and guarantee that all vertices of the polyhedron are still inside the ellipsoid. However, ζ cannot be too large, because the ellipsoids must still shrink sufficiently fast.

It turns out that the following works (see [GLS, page 81]):

- Do $N := 50(n+1)^2 L$ iterations.
- Keep $8N$ bits of precision.
- Blow up by $\xi := 1 + \frac{1}{4(n+1)^2}$ at each step.

With these modifications, the ellipsoid method becomes a practical algorithm; and by estimating the magnitude of the numbers occurring at each step it can be seen that not only the total number of arithmetic operations is polynomial, but also the number of bits processed in each operation, so that the algorithm as a whole is indeed polynomial.

6 Applications to Linear Programming

The first step in applying the ellipsoid algorithm to linear programming is to get rid of the *strict* inequalities. This is accomplished by again exploiting the "granularity" imposed by the finite encoding length of the input.

3. If the problem is feasible and bounded, determine the optimum value of cx with an error not larger than 2^{-2L} by using binary search between -2^{2L} and 2^{2L}. This can be done with at most $4L+1$ checks of the feasibility of systems $Ax \leq b, cx \geq \gamma$ for suitable γ.
4. Once the optimum value is known to this precision, the corresponding vertex can be found by determining the hyperplanes it lies on (by checking the feasibility, for $k = 1, \ldots, m$, of the system $Ax \leq b, cx \geq opt, A_k x = b_k, A_j x = b_j \ \ (j \in H(k))$, where A_k is the kth row of A and $H(k)$ is the set of those j for which the answer was "yes" previously). Choosing a set of n pairwise different hyperplanes and determining their intersection (i.e. solving a certain $n \times n$ system of linear equations) then gives the solution.

As a corollary, we have

Theorem 6.3 *There exist a polynomial time algorithm for solving linear programs.*

7 Oracle Computations

The main feature of the ellipsoid algorithm, besides its polynomial running time, is the fact that it does not require knowledge of the full system of inequalities defining the polyhedron. All that is really needed is a *separation oracle:* a subroutine that checks whether a given point x is inside the polyhedron, and if not, produces a hyperplane separating x and P. This subroutine could be called during each iteration with the center of the current ellipsoid as an argument and if it returned with a negative answer, the accompanying hyperplane and the current ellipsoid would be used to construct the new ellipsoid. Of course, we would still need some bound on the encoding complexity of P in order to compute the initial ellipsoid and the maximum number of iterations, but such a bound might be available even without knowing the inequalities defining the polyhedron.

In this section, we will elaborate on oracle computations on polyhedra. We will describe three different ways of characterizing a polyhedron via oracles and we will see that by virtue of the ellipsoid algorithm, all three are PTIME-equivalent.

Definition 7.1 *Let P be a polyhedron. Instead of giving its facets ($P = \{x : Ax \leq b\}$) or vertices ($P = \mathrm{conv}\{v_1, \ldots, v_k\}$), P can also be described by oracles:*

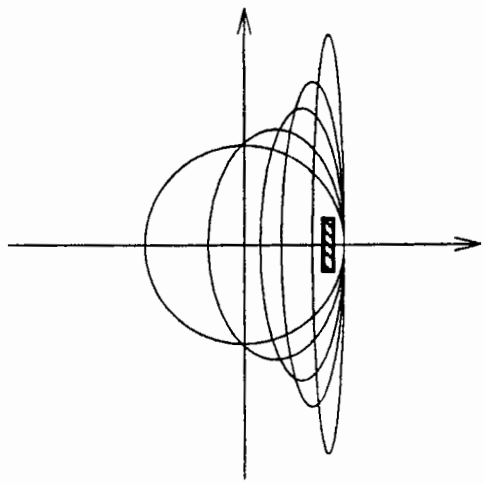
Separation Oracle: *For any vector $x \in \mathbf{R}^n$, either assert that $x \in P$ or produce a separating hyperplane.*

Optimization Oracle: *For any $c \in \mathbf{R}^n$, either assert that P is empty, or that cx is unbounded over P, or produce a vector $x_0 \in P$ maximizing cx over P.*

Violation Oracle: *For any vector $c \in \mathbf{R}^n$ and any $\gamma \in \mathbf{R}$, either assert that $cx \leq \gamma$ for all $x \in P$ or produce a vector $x_0 \in P$ such that $cx_0 > \gamma$.*

Clearly, any optimization oracle can also be used as a violation oracle, by just determining the maximum value of cx over P and either asserting that $cx \leq \gamma$ is true on P, if $\max_P cx \leq \gamma$, or outputting an optimal vector x_0 as a counterexample, if $\max_P cx = cx_0 > \gamma$.

The remaining reductions are more involved and before we describe them, we have to talk about encoding complexities first.



Also, their short axis will get arbitrarily close to the normal of H. The crux of the matter is: If the ellipsoid volume is less than a certain threshold depending on the dimension and the vertex complexity of P, the normal of H can be found by rounding the short axis using the simultaneous diophantine approximation technique developed in lecture 25. The reader is referred to [GLS, Lemma 6.4.2] for the (somewhat involved) details of this rounding. The bottom line is that we can find in oracle-polynomial time either a vector in P or a hyperplane containing P.

This technique can now be repeated. Whenever the ellipsoid method produces a hyperplane H containing P, one dimension of the problem can be eliminated, by projecting P onto some coordinate plane not orthogonal to H. The ellipsoid method is then applied again to the projection. In the end, either a feasible point was found, or the polyhedron has been reduced to zero dimensions, i.e., a single point, whose feasibility can be checked by a single oracle call.

To prove (2), note that SEPARATION can be viewed as a linear program: For $y \in \mathbf{R}^n$, we want a hyperplane $cx = \gamma$ such that $cx \leq \gamma$ for all $x \in P$ and $cy > \gamma$. Thus, check if

$$\max\{\, cy - \gamma : (c,\gamma) \in P^\circ \,\} > 0,$$

where

$$P^\circ := \{\, (c,\gamma) \in \mathbf{R}^{n+1} : cx \leq \gamma \;\; \forall x \in P \,\}$$

is the *polar* of P.

Note that if $P = \mathrm{conv}(V) + \mathrm{cone}(U)$, then

$$(c,\gamma) \in P^\circ \iff cv \leq \gamma, cu = 0 \quad \forall v \in V, u \in U,$$

i.e., the facet-complexity of P° is not higher than the vertex-complexity of P.

Now, by definition the violation oracle for P is a separation oracle for P°:

$$(c,\gamma) \notin P^\circ \iff \exists x \in P : \;\; cx - \gamma > 0 \geq \max\{\, c'x - \gamma' : (c',\gamma') \in P^\circ \,\}$$

Thus the claim follows by applying the OPTIMIZATION→SEPARATION reduction just proved to P°. □

CS258: Combinatorial Optimization

Lecture 20, November 13, 1991 **Philip Klein** Scribe: Bob Cohen

We show how to characterize T-joins polyhedrally in this section. We answer the question: What is the dual of a minimum-weight T-join?

Given two sets A and B, the *symmetric difference* of A and B written $A \oplus B = \{x | x \in A \cup B \text{ and } x \notin A \cap B\}$. The following result is useful in our characterization:

Claim 1 *We can assume, without loss of generality, that the edge costs of a T-join are non-negative.*

Proof: Suppose $G = (V, E)$ is a graph with some negative edge weights and $T \subseteq V$ is a vertex set. Let $w(e)$ be the weight of edge e and $w(a)$ be the sum of the weights of the edges in T-join a. Let $w_a(e)$ be the weight of e with respect to a, that is

$$w_a(e) = \begin{cases} w(e) & \text{if } e \in a \\ 0 & \text{if } e \notin a \end{cases}$$

We show an additional edge weight function w' with $w'(e) \geq 0$ for all $e \in E$ and vertex set T' such that there is a one-to-one mapping δ between T-joins and T'-joins and $w(a) = w'(\delta(a)) + c$ for each T-join a and constant c.

The edge weight function w' is defined to be $w'(e) = |w(e)|$. Let $E^- \subseteq$ be the set of edges with negative edge weights with w. Let V^- be the vertices incident to an odd number of edges in E^-. Let $T' = T \oplus V^-$. Consider a T-join a. We define $\delta(a) = a \oplus E^-$. It is easy to verify that $\delta(a)$ is a T'-join and if a' is a T'-join, then $\delta(a')$ is a T-join (δ is its own inverse).

Finally, we need to show that $w(a) = w'(\delta(a)) + c$ for each T-join a and constant c. Suppose $e \in a \cup E^-$. If $w(e) \geq 0$ then $e \in \delta(a)$ since $e \notin E^-$. If $w(e) < 0$, then there are two cases:

- If $e \in a$ then $e \notin \delta(a)$ so $w_a(e) = w'_{\delta(a)}(e) + w(e)$.
- If $e \notin a$ then $e \in \delta(a)$ so $w_a(e) = w'_{\delta(a)}(e) + w(e)$.

Therefore we get, $w(a) = w'(\delta(a)) + \sum_{e \in E^-} w(e)$. □

Given a graph $G = (V, E)$ and a set of nodes $T \subset V$ with $|T|$ even, a ***blocker*** of the T-joins is a minimal set of edges $B \subset E$ such that for any minimal T-join h, $B \cap h \neq \emptyset$. For example, suppose $T = \{u, v\}$ for some nodes u and v. Then a set minimal T-join is a path between u and v. A blocker of the T-joins would be any cut separating u and v. A ***T-cut*** is a minimal set of edges $f \subset E$ that separates the nodes of T such that each side of the separation has an odd number of nodes. A T-cut is a blocker for the T-joins.

The polyhedral characterization of T-joins is based on the theory of blockers. The space we are using is $R^{|E|}$ with $x \geq 0$ for all vectors x. Since every T-cut f contains an edge of every T-join x

($\supseteq$): We need to show that every vertex of $\{x \geq 0 : Ax \geq 1\}$ is a T-join.

Suppose this is not true. Let graph G and node set T be the smallest counterexample, i.e. G has the fewest nodes and edges. Let v be a vertex of $\{x \geq 0 : Ax \geq 1\}$ that is not in $P_{T\text{-join}}$. We will show that in fact v is a convex combination of T-joins plus a non-negative value (and hence in $P_{T\text{-join}}$)

Since v is a vertex, $|E|$ of the inequalities must be tight. Notice that for any edge e the inequality $x_e \geq 0$ cannot be tight at v since otherwise we can remove e from G to get a smaller counterexample.

A trivial T-cut separates out exactly one vertex from T. There are at most $|T|$ such T-cuts. Assume that $|E| > |T|$. Then there must exist a non-trivial T-cut a such that $a \cdot x \geq 1$ is tight at v, i.e. $a \cdot v = 1$. (Note if $|E| \leq |T|$, then $|T| = n$ or $|T| = n - 1$. These cases are easily handled.)

Consider the graph G_1 that results from contracting one side of T-cut a into a single vertex u_1. Let $T_1 = (T \cap V_1) \cup \{u_1\}$. Define G_2, u_2, and T_2 similarly for the graph that results from contracting the other side of a. Since G_1 and G_2 are both smaller than G, we have by assumption:

$$P_{T_1\text{-join}} = \{x : x \geq 0, A_1 x \geq 1\}$$

$$P_{T_2\text{-join}} = \{x : x \geq 0, A_2 x \geq 1\}$$

where A_1 and A_2 are the matrices of T_1-cuts and T_2-cuts.

Let v_1 be the projection of v into $P_{T_1\text{-join}}$ and v_2 be the projection of v into $P_{T_2\text{-join}}$. First we show $v_1 \in P_{T_1\text{-join}}$. To do this we need to show that v_1 satisfies the inequalities that describe $P_{T_1\text{-join}}$. We know $v_1 \geq 0$ since $v \geq 0$. Suppose a_1 is a T_1-cut. Cut a_1 will also be a T-cut in G since we replace the single node u_1 with an odd number of nodes on the u_1 side of T-cut a. Therefore, there will be an odd number of nodes from T on each side of a_1, so it is a T-cut in G. Then we know that $a_1 v \geq 1$ so $a_1 v_1 \geq 1$, which implies that $v_1 \in P_{T_1\text{-join}}$. Similarly $v_2 \in P_{T_2\text{-join}}$.

Suppose $t_1, \ldots t_k$ are vertices of $P_{T_1\text{-join}}$ and $u_1, \ldots u_k$ are vertices of $P_{T_2\text{-join}}$. We have:

$$v_1 = \lambda_1 t_1 + \cdots + \lambda_k t_k + \text{positive stuff} \tag{1}$$

$$v_2 = \gamma_1 u_1 + \cdots + \gamma_h u_h + \text{positive stuff} \tag{2}$$

where $\sum \lambda_i = 1$ and $\sum \gamma_i = 1$. We also know since a is a T-cut:

$$\begin{aligned} a v_1 &= 1 = a\lambda_1 t_1 + \cdots + a\lambda_k t_k + \text{positive stuff} \\ &\geq \lambda_1 + \cdots + \lambda_k + \text{positive stuff} \\ &= 1 + \text{positive stuff} \end{aligned}$$

So, $a t_i = 1$ for all i. Since t_i is a T-join, it is a 0-1 vector. Hence, t_i has exactly one edge in T-cut a. The same result holds for v_2.

We can clear denominators in both equation (1) and (2) by multiplying by some integer K.

$$\begin{aligned} K v_1 &= t_1 + \cdots + t_\ell + \text{positive stuff} \\ K v_2 &= u_1 + \cdots + u_m + \text{positive stuff} \end{aligned}$$

where ℓ and m are some integers. Note that t_i and u_j may occur multiple times in these equations. Additionally, they only appear if they have positive coefficients in equations (1) and (2).

Combinatorial Optimization

Lecture Nos. 21 and 22, November 15 & 20, 1990 **Philip Klein** Scribe: Seth Rosenzweig

1 Blocking Theory

1.1 Definitions

Definition. For a finite set, N, a *clutter* over N is a collection of subsets of N such that no one subset contains another.

Definition. For any collection C of subsets of N, $S \in C$ is called a *minimal element* of C if no other element of C is contained in it (see figure 1). We denote the set of minimal elements of C by minimal(C).

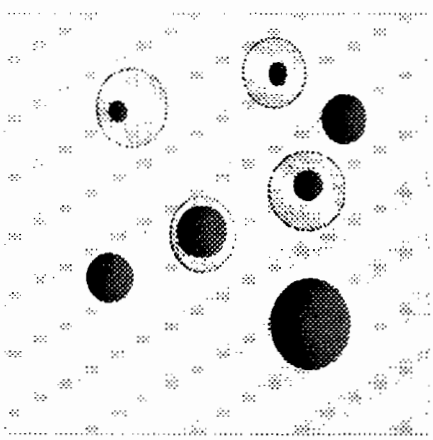


Figure 1: The square represents the set, N; The circles represent an arbitrary set C; and the dark circles are the minimal elements of C. The dark circles are also a clutter over N.

For any clutter, H, minimal(H) = H. We can also write $(C)^+ = \{S \subseteq N$ for some $S' \in C, S \supseteq S'\}$ (S is a set of subsets of N).

1.2 Blockers

For a clutter, H, we attempt to define a blocker of H by $BL(H) = \{f | f$ intersects every $h \in H\}$. However, this definition is no good. Using Blockers and intersects interchangeably, we need to take the minimal $\{f | f$ intersects every $h \in H\}$. So, $BL(H)$ = minimal$\{f | f$ intersects every $h \in H\}$.

Claim 2 If h' is not in H^+, then h is not in $BL(BL(H))^+$.

PROOF:
Since h' is not in H^+, then no element of H is contained in h'. For every $h \in H$, $h \cap h' \neq \emptyset$. $\overline{h'} \in BL(H)^+$, so, h' is not in $BL(BL(H))^+$. □

We have shown that

$$H^+ = BL(BL(H))^+$$

and

$$\text{minimal}(H^+) = \text{minimal}(BL(BL(H))^+)$$

thus,

$$H = BL(BL(H))$$

1.4 Polyhedral Domain

So far, we have defined the clutter, H, and the blocker of H. We also proved that $H = BL(BL(H))$. Now, we will do this in the polyhedral domain.

Polyhedra $P = \{X | Ax \geq 1, x \geq 0\}$ is of *blocking type* where A is 0-1.
We require that A has no zero rows. We define the analogous blocking polyhedra.

Definition. If P is of blocking type, then

$$BL(P) = \{z \geq 0 | z \cdot x \geq 1 \forall x \in P\}$$

Theorem 1 We claim the following:
1. $BL(P)$ is of blocking type.
2. Facets (defining inequalities) of $BL(P)$ are the facets of P.
3. Vertices of $BL(P)$ are the vertices of P.
4. $BL(BL(P)) = P$.

Remember: There are two ways of defining polyhedra:
1. $P = \{x | Ax \geq 1\}$ inequalities or facets (inequalities define the polyhedra).
2. Convex combination of a bunch of vertices + con($w_i \ldots$). $P = \text{conv}(v_1 \ldots v_k) + \text{con}(w_i \ldots)$, where $\mathrm{v} \cdot z \geq 1$.

Lemma. $BL(P) = \{z \geq 0 | z \cdot \mathrm{v} \geq 1\}$

PROOF: Suppose that H has the property. Then,

$$\text{dom}(H) = BL(\text{dom}(BL(H)))$$

$$BL(\text{dom}(H)) = BL(BL(\text{dom}(BL(H))))$$

$$BL(\text{dom}(BL(BL(H)))) = \text{dom}(BL(H))$$

$$BL(\text{dom}(BL(F))) = \text{dom}(F)$$

where $F = BL(H)$. □

Theorem 3 Suppose that H has the property, and that we can find a minimum cost element of H for any given cost-vector, $c \geq 0$ (i.e., we can solve $\min\{cx|x \in H\}$. If this is the case, then for any $w \geq 0$, we can solve $\min\{wz|z \in BL(H)\}$ (e.g., $H = \{\text{T} - \text{join}\}BL(Q)$).

PROOF: It is sufficient to solve separation for $\{z|z \cdot h \geq 1 \forall h \in H\}$. Given a candidate, z_0, check if z_0 satisfies all of the inequalities. If it does not, then output a separating hyperplane. Take z_0 as cost and find $\min\{z_0 \cdot x|x \in H\}$. Get x_0 such that if $z_0 - x_0 < 1$ then $x_0 z \geq 1$ is violated. If $z_0 \cdot x_0 \geq 1$, then $z_0 \cdot x \geq 1, \forall x \in H$. So, z_0 is in the polyhedron. □

(2) facets of $abl(P)$ are Maximal vertices of P

(3) maximal vertices of $abl(P)$ are facets of P

(4) $abl(abl(P)) = P$

Definition. P and $abl(P)$ are an **anti-blocking pair** of a polyhedra.

Definition. Suppose P a non-negative polyhedra. Then $\mathbf{subdom(P)} = \{ z \geq 0 : \exists x \in P, z \leq x \}$

This definition is very similar to the definition of the dominator mentioned in previous lectures. The elements of subdom(P) can be reached by taking an element of P, and adding in some negative vector. The key difference to note is that the dominator was unbounded because any positive vector could be added in, however the sub-dominator is bounded because the restriction that $z \geq 0$, limits what can be subtracted out.

Theorem 3 If clutter H has the **anti-property** then so does $ABL(H)$

Again, refer to previous lectures to see a complete definition of the **property**.

2 "Perfect" Graphs

The following definitions are for a undirected graph $G = (V, E)$.

Definition. The **complement** of $G = (V, E)$, written $\overline{G}$ is equal to the graph $(V, \overline{E})$, where $\overline{E} = \{(u, v) : (u, v) \notin E\}$.

The following characteristics of graph G are defined:

- $\omega(G)$ = Clique number
- $\chi(G)$ = Coloring number
- $\alpha(G)$ = Independent set number
- $\theta(G)$ = Clique cover number

The following relations are trivially true:

- $\chi(G) \geq \omega(G)$

  If you have a clique of 10 nodes, then each of those nodes needs a different color.

- $\theta(G) \geq \alpha(G)$

  If you have an independent set of 10 nodes, then you will need at least 10 cliques to cover the set.

- $\alpha(G) = \omega(\overline{G})$

  Independent sets become cliques in the complement, and vice versa.

($\Rightarrow$)

Assume G is perfect. Let w be integral and c_w be the maximal weight of a clique. Show that

$$c_w = \max\{\, wx : x \geq \vec{0}, AX \leq \vec{1}\,\} \text{for all } w$$

Since $Cl(G) \subseteq I(G)$, then $c_w \leq \max\{\, wx : x \geq \vec{0}, AX \leq \vec{1}\,\}$.
By induction on w: If w is a 0-1 vector, than result from observation.
Induction step:

Let $u \in V$ with $wu > 1$
Consider $w' = w - u$
By induction, $\exists y \geq 0 : yA \geq w'$ and $y1 = c_{w'}$
Therefore, $\exists$ independent set $s : su = 1$
Let $w'' = w - s$
Let q be a clique with $w''q = c_{w''}$
Suppose $sq > 0$, then $c''_w = w''q \leq wq - 1 \leq c_w - 1$
If $sq = 0$, then $c_{w''} = w''q \leq w'q \leq y1 - 1$ since $qs = 0$

$$\begin{aligned} c_{w''} &= c_{w'} - 1 \leq c_w - 1 \\ \text{Therefore, } c_w &\geq 1 + c_{w''} \\ &= 1 + \max\{\, (w-s)x : x \geq 0, Ax \leq 1\,\} \\ &\geq \max\{\, wx : x \geq 0, AX \leq 1\,\} \\ c_w &= \max\{\, wx : x \geq 0, Ax \leq 1\,\} \end{aligned}$$

($\Leftarrow$)

Suppose $Cl(G) = I(G)$
Then $\max\{\, wx : x \geq 0, Ax \leq 1\,\}$ is a clique for all 0-1 w

We need to show $\min\{\, y1 : y \geq 0, yA \leq w\,\}$ has optimum solutions for 0-1 w.
By Induction on w: (Base case $w = 0$),

Suppose w is 0-1 and y' and optimal solution
Let s be an independent set with $y's > 0$

$$\begin{aligned} \max\{\, (w-s)x : x > 0, Ax \leq 1\,\} &= \min\{\, y1 : yA \geq w - s\,\} \\ &< \min\{\, y1 : yA \geq w\,\} \\ &= \max\{\, wx : x \geq 0, Ax \leq 1\,\} \end{aligned}$$

All values are integer by assumption, so differ by 1 since $sx \leq 1$, and so

$$\max\{\, (w-s)x : x > 0, Ax \leq 1\,\} = \max\{\, wx : x \geq 0, Ax \leq 1\,\} - 1$$

By induction integral y^* is a solution of $\min\{\, y1 : yA \geq w - s\,\}$, so increase $y^*(s)$ by 1 to give an integer optimum solution to $\min\{\, y1 : yA \geq w\,\}$.

Combinatorial Optimization

Lecture 24, Nov 29 '90 | **P. Krishnan** | Scribe: P. Krishnan

We look at how to find a maximum clique and a minimum coloring of a perfect graph in polynomial time. This is accomplished using the ellipsoid algorithm that was discussed in earlier lectures. The original paper discussing this is by Grötschel, Lovász, and Shrijver (GLS) [Lóvasz 84]. Some portions of this script have been lifted from [Klein 88].

1 Introduction, basics, and notation

All graphs we consider will be simple, undirected and finite. A graph is denoted by $G = (V(G), E(G))$, where $V(G)$ is the node set and $E(G)$ is the edge set of the graph. Two nodes i and j are *adjacent* if they are connected by an edge. The *complement graph* of G is defined as the graph $\overline{G}$, with $V(\overline{G}) = V(G)$ and in which two different nodes are adjacent if and only if they are not adjacent in G. We will denote a vector x by $\vec{x}$, and its ith component by x_i. The ith row of a matrix A will be denoted by a_i, and the (i,j)th element by a_{ij} or $(A)_{ij}$.

Definitions

An *independent set* of a graph G is a set of nodes $I \subseteq V(G)$ such that no two nodes of I are adjacent in G. A *clique* of G is a set of nodes $C \subseteq V(G)$, such that any two nodes of C are adjacent in G. The maximum cardinality of an independent set in G is called the ***independent set number*** of G and is denoted by $\alpha(G)$. The maximum cardinality of a clique in G is called the ***clique number*** of G and is denoted by $\omega(G)$. Clearly, a clique of G is an independent set of $\overline{G}$ and thus, $\omega(G) = \alpha(\overline{G})$. We can also assign weights to nodes of a graph and define the weight of a clique to be the sum of weights of nodes in the clique. The *maximum weight clique* is then just the clique with maximum weight.

A *k-coloration* of a graph is a mapping of its nodes to colors such that no two nodes which are adjacent have the same color. In other words, a k-coloration of G is a partition of $V(G)$ into k independent sets of G. The least integer k for which G admits a k-coloration is called the ***chromatic number*** of G, denoted by $\chi(G)$. A ***k-clique cover*** of G is a partition of $V(G)$ into k cliques of G. The least number k for which G admits a k-clique cover is called the ***clique cover number*** of G and is denoted by $\theta(G)$. It should be clear that every k-coloration of G is a k-clique cover of $\overline{G}$, and hence $\chi(G) = \theta(\overline{G})$.

We consider the problems of finding a maximum clique, an optimal coloring, a maximum independent set, and a minimum clique cover of a graph. These optimization problems are in general NP-complete. However, they can be solved in polynomial time for *perfect graphs* defined below. Note that the following two inequalities hold for all graphs G:

$$\omega(G) \leq \chi(G) \tag{1}$$

$$\alpha(G) \leq \theta(G) \tag{2}$$

3 Finding the independent set

To find an independent set I intersecting every maximum clique, we use the ellipsoid method. Let $P(G)$ be the *clique polytope* of the n-node graph G defined as,

$$P(G) = \text{conv}\{\vec{x} \mid \vec{x} \in \Re^n, \vec{x} \text{ is an incidence vector of a clique in } G\}.$$

Let A be a matrix such that each row of A is an incidence vector of an independent set of G. Note that the number of rows of A will be the number of independent sets of G, and this may be very large. Determining A may therefore not be possible. A clique and an independent set intersect in at most one node. It was proved in earlier lectures that the clique polytope $P(G)$ is:

$$P(G) \quad = \quad \{\vec{x} \mid \vec{x} \geq \vec{0},\ A \cdot \vec{x} \leq \vec{1}\}. \tag{3}$$

Finding a maximum size clique in a perfect graph could in principle be done using linear programming over the polytope $P(G)$. The LP problem would be

$$\max\{\vec{1} \cdot \vec{x} \mid \vec{x} \geq \vec{0},\ A \cdot \vec{x} \leq \vec{1}\}. \tag{4}$$

The optimal dual to the above LP will give us an independent set that intersects every maximum clique.

The million dollar question however is: Why is the above true, and if true, how do we find the optimal dual? This is where the ellipsoid algorithm comes in. GLS have shown that if one can optimize over a polytope for a given linear objective function, then one can find an optimal dual as well[1]. The primal LP problem of (4) finds a maximum size clique. By Lemma 1 and from what we will see in later sections, we can optimize over $P(G)$ and hence the ellipsoid algorithm can get us an optimal dual. The dual for the LP of (4) is

$$\min\{\vec{y} \cdot \vec{1} \mid \vec{y} \geq \vec{0},\ \vec{y} \cdot A \geq \vec{1}\}. \tag{5}$$

The ellipsoid algorithm can be made to return a set of pairs, the non-zero $\vec{y}$ entries and the corresponding rows of matrix A for the optimal dual. The nonzero entries of the optimal dual $\vec{y}$ can be interpreted as a selection of some independent sets (rows of A). We claim that any one such selected independent set (row of A) satisfies the property we are looking for as shown in the lemma below.

Note. Actually, to get the optimal dual, the primal must be optimizable for every objective function $\vec{w}$. The primal problem when the objective function can be arbitrary is

$$\max\{\vec{w} \cdot \vec{x} \mid \vec{x} \geq 0,\ A \cdot \vec{x} \leq \vec{1}\},$$

which corresponds to finding the maximum weight clique. We shall see later in Section 5 an outline of why this can be done in polynomial time.

Lemma 3 *Any row of A which is selected by the dual optimal $\vec{y}$ is an independent set which intersects with every maximum clique.*

[1] Not quite. See note following paragraph

4.1 Linear algebra revisited

We state some definitions of matrices and some of their properties without proof. The suspicious reader may refer to any book on linear algebra.

Definition 1 λ is an eigenvalue of matrix A, if it is a solution to the equation $\det(A - \lambda I) = 0$.

Definition 2 The *trace* of a matrix is the sum of its diagonal elements.

Definition 3 An $n \times n$ matrix B is called *positive semi-definite* if for every vector $x \in \Re^n$, $x^T Bx \geq 0$. It is called *positive definite* if the inequality is strict.

Definition 4 A *principal submatrix* of a matrix is obtained by deleting rows and columns in matching pairs.

Lemma 4 *A matrix is positive semi-definite iff all its eigenvalues are non-negative. A matrix is positive semi-definite iff all its principal submatrices have a non-negative determinant.*

4.2 Proving existence of $\vartheta(G)$

Let $\gamma(G)$ be the set of $n \times n$ symmetric matrices A such that

$$(i = j \quad \text{or} \quad i \text{ adjacent to } j) \Rightarrow a_{ij} = 1.$$

For a matrix A, let $\Lambda(A)$ denote the maximum eigenvalue of A. We define $\vartheta_A(G)$ as

$$\vartheta_A(G) = \min\{\Lambda(A) \mid A \in \gamma(G)\}.$$

Let $\beta(G)$ be the set of $n \times n$ symmetric, positive semi-definite matrices B, with trace 1 such that

$$(i \neq j \quad \text{and} \quad i \text{ not adjacent to } j) \Rightarrow b_{ij} = 0.$$

We define $\vartheta_B(G)$ as

$$\vartheta_B(G) = \max\left\{\sum_{i,j} b_{ij} \mid B \in \beta(G)\right\}.$$

We will use the ellipsoid method to find $\vartheta_B(G)$ by optimizing over the set $\beta(G)$.

Note that there is a correlation between a matrix in $\gamma(G)$ or $\beta(G)$ and the adjacency matrix of G. A matrix in $\gamma(G)$ has ones on its diagonal and at all places where there is a 1 in the adjacency matrix of G. The other entries (which are zero in the adjacency matrix) could be arbitrary; the matrix should however be symmetric. A matrix in $\beta(G)$ has zeros at all places that are 0 in the adjacency matrix of G. The other entries could be arbitrary; the matrix should however be symmetric, positive semi-definite and have a trace of one.

We will now prove the inequalities in (7) step by step.

- $\omega(G) \leq \vartheta_B(G)$.
  To show this we need to exhibit one matrix $B \in \beta(G)$ such that $\sum_{i,j} b_{ij} \geq k$, where k is the

m, of nodes. We will remove this restriction later. So our graph has mk nodes, with each of the k colors coloring m nodes each.

Let J be the $mk \times mk$ matrix of all ones. Let K be the $mk \times mk$ matrix such that

$$k_{ij} = \begin{cases} 1 & \text{if } i \neq j \ \& \ i \text{ and } j \text{ have the same color} \\ 0 & \text{otherwise.} \end{cases}$$

Let $A = J + tK$ for some as yet undetermined t. We will now find a t such that $A \in \gamma(G)$ and $\Lambda(A) \leq \chi(G)$. It is easy to see that $A \in \gamma(G)$ for any t, since A is symmetric and for any position of A that should be a 1, J has a 1 and K has a 0.

Because J and K are symmetric, each has an orthonormal basis of eigenvectors. Because J and K commute, we can take the same basis for the two matrices. It follows that there is a correspondence between the eigenvalues of J and K, such that the eigenvalues of $J + K$ are obtained by adding the eigenvalues of J to the corresponding eigenvalues of K. If v is an eigenvector of J and K, with corresponding eigenvalues λ and λ', then

$$(J + K)v = Jv + Kv = \lambda v + \lambda' v = (\lambda + \lambda')v$$

We use this observation to determine the eigenvalues of $J + tK$.

One eigenvector of J is the all-one's vector $\vec{1}$, which has its corresponding eigenvalue mk. The eigenvalue of K corresponding to the all-one's vector is $m - 1$, so we get $mk + t(m-1)$ as one of A's eigenvalues. Since the columns of J are all identical, J has an $(n-1)$-dimensional null space; thus the remaining $n - 1$ eigenvectors all have eigenvalue 0. One can check that each eigenvalue of K is either $m - 1$ or -1. Therefore, the eigenvalues of $A = J + tK$ are $mk + t(m-1)$, $0 + t(-1)$, and $0 + t(m-1)$. If we choose $t = -k$, the maximum eigenvalue is $km - k(m-1) = k$. This shows that $\Lambda(A) \leq$ chromatic-number(G) for the case in which each color class has the same number of nodes.

Suppose now that the sizes of the color classes vary. We add isolated nodes to the color classes until they all have the same size, obtaining a graph G'. Clearly we have not changed the chromatic number. Construct the matrix A' as described above, and let A be the matrix obtained from A' by deleting rows and columns corresponding to the added nodes. We claim that $\Lambda(A) \leq \Lambda(A')$. To see this, let $\lambda = \Lambda(A')$. Then as observed before, $\lambda I' - A'$ is positive semi-definite, where I' is the identity matrix of the same size as A'. It follows that $\lambda I - A$ is also positive semi-definite; if $x^{\mathrm{T}}(\lambda I - A)x$ were negative, then $x'^{\mathrm{T}}(\lambda I' - A')x'$ would be negative, where x' is obtained from x by inserting some zeroes. Since $\lambda I - A$ is positive semi-definite, all the eigenvalues of A are $\leq \lambda$. That is, $\Lambda(A) \leq \Lambda(A')$. This completes the proof.

We have thus proved the existence of a $\vartheta(G)$ which lies between $\omega(G)$ and $\chi(G)$. We now need to compute it.

4.3 Computing $\vartheta_B(G)$

We compute $\vartheta_B(G)$ using the ellipsoid method. Recall that $\vartheta_B(G) = \max\{\sum_{i,j} b_{ij} \mid B \in \beta(G)\}$. We will maximize over the set $\beta(G)$ to compute $\vartheta_B(G)$. To apply the ellipsoid algorithm, we need to ensure that

- *$\beta(G)$ is bounded*
  For showing that $\beta(G)$ is bounded we will show that each element of the matrix $B \in \beta(G)$ is bounded from above by some value. Note that by the definition of positive semi-definiteness (Lemma 4), every principal submatrix must have a positive determinant. Each diagonal element is a principal submatrix by itself. So it must be non-negative. Hence each diagonal element is bounded by the trace which is 1. For non-diagonal element b_{ij}, consider the determinant of the principal submatrix containing b_{ij} as shown below, which must be non-negative. Noting that B is symmetric we get,

$$\begin{vmatrix} b_{ii} & b_{ij} \\ b_{ji} & b_{jj} \end{vmatrix} = b_{ii}b_{jj} - b_{ij}^2 \geq 0.$$

  The above implies that b_{ij}, the off-diagonal element, is bounded by $\sqrt{b_{ii}b_{jj}}$, which by the argument before is bounded by 1.

- *Separation oracle*
  The ellipsoid algorithm will from time to time, make a guess and want the Separation oracle to say whether the guess is a feasible solution, or give a hyperplane which proves that the guess is wrong. So, we want to determine this Separation oracle. We will be given an $n \times n$ matrix B, and we need to determine if $B \in \beta(G)$ and if not, provide a separating hyperplane. What would a separating hyperplane look like? It would be a vector ϕ such that every feasible solution obeys some condition with ϕ which the given guess B does not. What could go wrong? The given matrix B may not have its trace equal to 1, or it may not be symmetric, or it may not have zeros where it should. We can easily check these conditions and form a hyperplane ϕ which separates the guess B from the polytope. The non trivial condition is positive semi-definiteness. For any n vector ϕ, if matrix U is positive semi-definite, $\phi^T U \phi$ must be nonnegative (Definition 4). In other words,

$$\sum_{i,j} \phi_i \phi_j u_{ij} \geq 0$$

  So, if the guess B is not positive semi-definite, we will come up with a vector ϕ such that the above rule is not obeyed.

  We state here a fact without proof. To determine whether a symmetric matrix B of rank r is positive semi-definite, it turns out to be sufficient to find a nonsingular $r \times r$ principal submatrix of B and test the submatrix for positive-definiteness. Finding such a submatrix can be done using Gaussian elimination, swapping rows (and corresponding columns) as needed to avoid zero pivots. After r stages of Gaussian elimination, the r^{th} leading submatrix is nonsingular and rows $r+1$ through n are all zero. This submatrix corresponds to some nonsingular principal submatrix of the original matrix B, depending on what row-swaps were performed.

  Assume that the rank of the guess B is k. We can perform Gaussian elimination as mentioned above (in polynomial time) and determine a reordering of nodes such that the submatrix $B' = \{B_{ij}^{(G)} \mid i,j \leq k\}$ is non-singular; k being the rank of matrix B and $B^{(G)}$ being the matrix after Gaussian elimination. Now, B is positive semi-definite iff $\det(B_l) > 0$ for $1 \leq l \leq k$; B_l being the $l \times l$ "corner" submatrix of $B^{(G)}$. If B is not positive semi-definite, there must

where w_i is the weight of node i, is equal to $\vartheta_B(G')$. We can then use the following lemma, similar to Lemma 1, to find the maximum weight clique along the lines discussed in Section 2 for the maximum size clique.

Lemma 5 *Let $\omega_w(G)$ be the weight of the maximum weight clique, and $\chi_w(G)$ be something we could call the* weighted coloring number. *Then, for any graph G with weights associated with nodes, we can compute $\vartheta_B^w(G)$ in polynomial time such that*

$$\omega_w(G) \leq \vartheta_B^w(G) \leq \chi_w(G).$$

In perfect graphs, the above holds with equality throughout.

The details of why $\vartheta_B^w(G)$ equals $\vartheta_B(G')$ was not discussed in the lecture. The interested reader is referred to [Lóvasz 84].

6 Conclusions

We have seen how the optimization problems of finding the maximum size clique, minimum coloring, maximum independent set and minimum clique cover, which are NP-complete for general graphs, can be solved in polynomial time when the graph is perfect. These algorithms make use of the ellipsoid method and hence, although theoretically good, may not be good in practice, since the "numerical instability" of the ellipsoid algorithm percolates to these algorithms too. To quote from the original paper [Lóvasz 84]

> "...should be viewed as a theoretical contribution showing that certain programming problems for perfect graphs are indeed polynomially solvable...(Future) algorithms should have a more combinatorial nature and should not suffer from the numerical instability (due to our present day computer technology of fixed precision arithmetic) of the ellipsoid method..."

References

[Lóvasz 84] Grötschel, Lovász, Shrijver; *Polynomial algorithms for perfect graphs.* Annals of Discrete Mathematics 21 (1984) pp. 325–356.

[Lóvasz 79] Lóvasz; *On the Shannon Capacity of a Graph.* IEEE Transactions on Information Theory., Vol. IT-25, No. 1, Jan '79.

[Klein 88] Klein; *How to solve NP-complete problems on graphs you can't even recognize, or Some applications of the Ellipsoid Method.* Scribe notes in a class on Algorithms and Complexity, MIT.

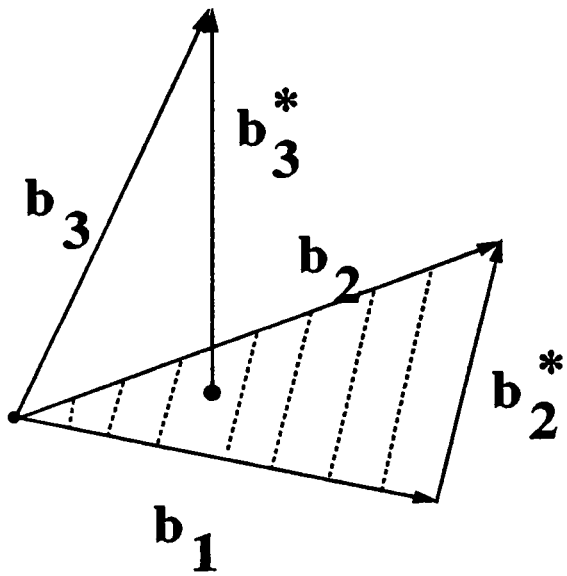


Figure 1: In the basis b_1, b_2, b_3, $b_1^* = b_1$, b_2^* is the part of b_2 orthogonal to b_1 and b_3^* is the part of b_3 orthogonal to the plane spanned by b_1 and b_2.

4. $|det[b_1 \ldots b_n]|$ *is the volume of the parallelopiped generated by* $b_1, \ldots, b_n$.

Proof:

1. If another basis $b'_1, \ldots, b'_n$ generates the same lattice as $b_1, \ldots, b_n$, then, we can write the following.

$$\begin{aligned} b'_i &= \sum_{j=1}^{n} \lambda_{ij} b_j \ \forall i \text{ with } \lambda_{ij} \in Z \text{ and} \\ b_i &= \sum_{j=1}^{n} \eta_{ij} b'_j \ \forall i \text{ with } \eta_{ij} \in Z \end{aligned}$$

We can rewrite these into the following martix equation.

$$\begin{aligned} B' &= \Lambda B \\ B &= \Gamma B' \end{aligned}$$

where the matrices Λ and Γ are integral. From the above two equations, we can write

$$B = \Gamma B' = \Gamma \Lambda B$$

Taking determinants, we get

$$det(B) = det(\Gamma) det(\Lambda) det(B)$$

Cancelling off $det(B)$ on both sides, we get

$$det(\Gamma) det(\Lambda) = 1$$

3.2 A lower bound

We now derive a lower bound on the shortest vector in the lattice. Let $\lambda(\mathcal{L})$ denote the vector in $\mathcal{L}$ with the smallest norm.

Observation 3.2 $\lambda(\mathcal{L}) \geq \min_i \|b_i^*\|$.

Proof: Consider any non-zero $x \in \mathcal{L}$. We can write x as

$$\begin{aligned} x &= \lambda_k b_k + \sum_{i=1}^{k-1} \lambda_i b_i \text{ where } k \text{ is the largest non-zero coefficient in the expansion} \\ &= \lambda_k b_k^* + \sum_{i=1}^{k-1} \lambda_i' b_i \end{aligned}$$

by expanding b_k using the Gram-Schmidt equations. The λ_i''s are defined by

$$\begin{aligned} \lambda_i b_i + \frac{b_k \cdot b_i^*}{b_i^* \cdot b_i^*} b_i^* &= \lambda_i' b_i \text{ since} \\ b_k &= b_k^* + \sum_{i=1}^{k-1} \frac{b_k \cdot b_i^*}{b_i^* \cdot b_i^*} b_i^* \end{aligned}$$

by the Gram-Schmidt Orthogonalization equations. Taking norm of both sides in 1 and noting that the two summands are orthogonal, we can write this as follows.

$$\begin{aligned} \|x\| &= |\lambda_k \|b_k^*\|| + \|\sum_{i=1}^{k-1} \lambda_i' b_i\| \\ &\geq \|b_k^*\| \\ &\geq \min_i \|b_i^*\| \end{aligned}$$

□

4 The problem

In section 3.1, we proved that $det(\mathcal{L}) = \Pi_{i=1}^n \|b_i^*\|$. Since, $\|b_i^*\| \leq \|b_i\|$ trivially, we have that $\Pi_{i=1}^n \|b_i\| \geq det(\mathcal{L})$ and how close this inequality is to equality is a measure of how orthogonal the basis is. This is a preview of how the goodness of a basis, which is roughly how close it is to the quantity $det(\mathcal{L})$ which we know is independent of the basis vectors itself, is related to 'how much orthogonal' it is. The following theorems were shown by Minkowski.

Theorem 4.1 *1. For any lattice $\mathcal{L}$ generated by n independent vectors, there is a basis $b_1', \ldots, b_n'$ such that $\Pi_{i=1}^n \|b_i'\| \leq n^n det(\mathcal{L})$.*

2. For any lattice $\mathcal{L}$, $\lambda(\mathcal{L}) \leq \sqrt{n}(det(\mathcal{L}))^{\frac{1}{n}}$.

We don't prove the theorem here but notice that this leads us to the following natural problems on lattices. We are given a basis $b_1, \ldots, b_n$ generating the lattice $\mathcal{L}$.

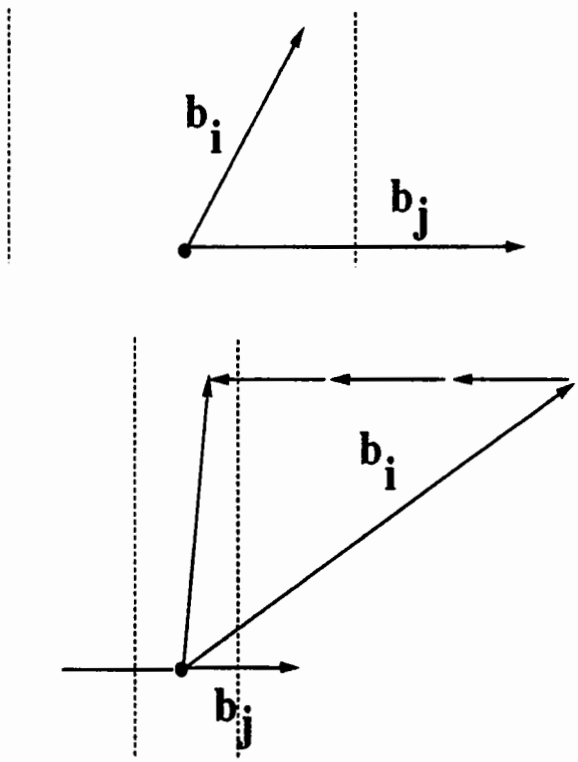


Figure 2: The figure above illustrates that we can always fix the coefficients for any pair of vectors b_i and b_j by illustrating the two essential cases. Remember that $\mu_{i,j}$ is just the fraction of the projection of b_i onto b_j with respect to the total length of b_j itself. When this rato is less than a half, then these two vectors are pseudo-orthogonal. In the first figure above, $\mu_{i,j}$ is already less than a half to begin with. The dotted lines represent the region within which the head of b_j must lie so as to be psuedo-orthogonal to b_i. In the second figure, it is seen that we can subtract off copies of b_i from b_j so as to make its head lie within the dotted region and hence fix the coefficient $\mu_{i,j}$.

Proof of claim: Since the basis is pseudo-ordered, we have for all $i > 1$,

$$\begin{aligned}
\frac{3}{4}\|b_{i-1}^*\|^2 &\leq \|b_i^* + \mu_{i,i-1}b_{i-1}^*\|^2 \\
&= \|b_i^*\|^2 + \mu_{i,i-1}^2\|b_{i-1}^*\|^2 \text{ since } b_i^* \text{ and } b_{i-1}^* \text{ are orthogonal.} \\
&\leq \|b_i^*\|^2 + \frac{1}{4}\|b_{i-1}^*\|^2 \text{ since } |\mu_{i,i-1}| \leq \tfrac{1}{2}.
\end{aligned}$$

Rearranging and taking square roots yields the claim. □

Now we go ahead and prove the theorem.

1. We can repeatedly apply claim 6.1 above to get. $\|b_1^*\| = \|b_1\| \leq 2^{\frac{i-1}{2}}\|b_i^*\|$ for any i. We know from Observation 3.2 that $\lambda(\mathcal{L}) \geq \min_i \|b_i^*\|$. Putting these together, we get $\|b_1\| \leq 2^{\frac{i-1}{2}}\|b_i^*\| \leq 2^{\frac{n-1}{2}} \min_i \|b_i^*\| \leq 2^{\frac{n-1}{2}}\ \lambda(\mathcal{L})$.

2. For this, we first note that we can rewrite Claim 6.1 as $\|b_1\| \leq 2^{\frac{i-1}{2}}\|b_i^*\|$ for any i. Multiplying the n such inequalities, one for each i, we get

$$\begin{aligned}
\|b_1\|^n &\leq \Pi_{i=1}^n 2^{\frac{i-1}{2}}\|b_i^*\| \\
&= 2^{\frac{n(n-1)}{4}}\Pi_{i=1}^n\|b_i^*\| \\
&= 2^{\frac{n(n-1)}{4}} det(\mathcal{L}) \text{ using the result in Observation 3.1}
\end{aligned}$$

   Taking the n^{th} root of this yields 2.

3. First, we note that Claim 6.1 can be written as $\|b_j^*\|^2 \leq 2^{i-j}\|b_i^*\|^2$. Since we need to bound the product of the norms of the b_i's by the value of $det(\mathcal{L})$ which we know is the product of the norms of the corresponding orthogonal vectors, we must try and express the b_i's in terms of the b_i^*'s. For this, we begin with the Gram-Schmidt expression for b_i.

$$\begin{aligned}
\|b_i\|^2 &= \|b_i^* + \sum_{j=1}^{i-1}\mu_{i,j}b_j^*\|^2 \\
&= \|b_i^*\|^2 + \sum_{j=1}^{i-1}\mu_{i,j}^2\|b_j^*\|^2 \text{ since these are orthogonal vectors.} \\
&\leq \|b_i^*\|^2 + \frac{1}{4}\sum_{j=1}^{i-1}\|b_j^*\|^2 \text{ since } |\mu_{i,j}| \leq \tfrac{1}{2}
\end{aligned}$$

   Substituting $\|b_j^*\|^2 \leq 2^{i-j}\|b_i^*\|^2$ for all j's and simplifying, we get $\|b_i\|^2 \leq 2^{i-1}\|b_i^*\|^2$. Taking the product of the n such inequalities, one for each i, we get

$$\begin{aligned}
\Pi_{i=1}^n\|b_i\|^2 &\leq \Pi_{i=1}^n\|b_i^*\|^2 2^{i-1} \\
&= (det(\mathcal{L}))^2\Pi_{i=1}^n 2^{i-1} \text{ using the results in Observation 3.1.}
\end{aligned}$$

   Simplifying and taking square roots yields 3.

□

by a factor of $\frac{3}{4}$ at every application of (SWAP) and it must be integral, the maximum number of applications of (SWAP) is $O(\log(\Pi_{i=1}^{n-1} d_i))$ which is atmost $O(n^2 \log(nb_{max}))$. □

Note that the number of times the swap operation is performed and hence the running time of the whole algorithm is polynomial in the 'size of the input' by which we mean the bit-size of the maximum number in the input which is $O(\log b_{max})$.

8 An application

We illustrate an application of this algorithm in rounding a set of real numbers such that they obey certain nice properties like for instance some equations. The Best approximation problem of a real number is, given a real number α, to find the best rational $\frac{p}{q}$ approximating it with $q < Q$ for some fixed Q. When we have a set of real numbers that obey an equation for instance, then finding the best approximations of these reals independently is not guarenteed to preserve some equation between these approximating rationals. For example consider $\alpha_1 = 0.1422, \alpha_2 = 0.2213, \alpha_3 = 6359$ obeying the equation that $\alpha_1 + \alpha_2 + \alpha_3 = 1$. If we independently find the best approximations for these numbers with $Q = 100$, we get the rational $\frac{1}{7}, \frac{2}{9}, \frac{7}{11}$ approximating them respectively. But these rations don't sum up to 1. Thus, we need a way of simultaneously approximating these numbers such that they preseve such equations. To this end, we turn to simultaneous Diophantine approximations.

8.1 Simultaneous Diophantine Approximationm:

We first state a theorem due to Dirichlet without proof.

Theorem 8.1 *Given $\alpha_1, \ldots, \alpha_n \in R^n$ and a small ϵ, there exists integers $q, p_1, \ldots, p_n$ such that $1 \leq q \leq \epsilon^{-n}$ and $\left|\alpha_i - \frac{p_i}{q}\right| \leq \frac{\epsilon}{q}$ for all i.*

The proof of the thorem is non-constructive and does not furnish us with the appropriate q and the p_i's. Using the basis reduction algorithm, we shall show how to come up with these q and p_i's obeying these conditions but with the relaxed condition that $1 \leq q \leq 2^{\frac{n(n+1)}{4}} \epsilon^{-n}$. We shall first demonstrate how such an approximation can be used in rounding reals preserving an equation obeyed by them.

8.2 Applying Diophantine approximation to rounding

In this section, we show the following lemma that demonstrates the use of simultaneous Diophantine approximation in rounding reals such that this preserves an equation obeyed by these reals.

Lemma 8.1 *If q and p_i's are integers obtained by simoltaneous Diaphontine approximation of the reals $\alpha_1, \ldots, \alpha_n \in R^n$ and if $\sum_{i=1}^n a_i \alpha_i = b$ and $\sum_{i=1}^n |a_i| < \epsilon^{-1}$, then $\sum_{i=1}^n a_i \frac{p_i}{q} = b$.*

Proof: From the conditions of simultaneous Diaphontine approximation that $\left|\alpha_i - \frac{p_i}{q}\right| \leq \frac{\epsilon}{q}$, we can derive that

$$b - \sum i = 1^n |a_i| \frac{\epsilon}{q} \leq \sum_{i=1}^{n} a_i \frac{p_i}{q} \leq b + \sum i = 1^n |a_i| \frac{\epsilon}{q}$$

References

[1] A. K. Lenstra, H. W. Lenstra Jr., L. Lovász, "Factoring polynomials with rational coefficients", *Math. Ann. 261*, pp. 515-534, 1982.

[2] L. Lovász, "An Algorithmic theory of Numbers, Graphs and Convexity", Regional Conference series in Applied maths, SIAM (1986).

Lemma 2.1 *Let* $P^* = \{\mathbf{x} \mid 2\mathbf{A}\mathbf{x} \leq 2\mathbf{b} + 1\}$. *Then*

- $\mathbf{x} \in Z^n$, $\mathbf{x} \in P$ *iff* $\mathbf{x} \in Z^n$, $\mathbf{x} \in P^*$.
- *If* P *is not empty, then* P^* *is full-dimensional.*

PROOF:

1. If $\mathbf{x}$ is in P then clearly it is also in P^*. On the other hand, note that if $\mathbf{x}$ is integer, $2\mathbf{b} + \mathbf{1}$ has only odd entries and $2\mathbf{A}\mathbf{x}$ has only even entries, and therefore if $2\mathbf{A}\mathbf{x} \leq 2\mathbf{b} + \mathbf{1}$ then $2\mathbf{A}\mathbf{x} \leq 2\mathbf{b}$.

2. Let $\mathbf{y} \in P$ and hence $\mathbf{y} \in P^*$. Consider the n vectors $(\mathbf{y}^1, \mathbf{y}^2, \ldots, \mathbf{y}^n)$, where $\mathbf{y}^i = \mathbf{y} + \epsilon * \mathbf{e}_i$, $\mathbf{e}_i$ being the unit vector pointing in direction i. If we can show that all $\mathbf{y}^i$ are in P^* then, because $\{\mathbf{y}\} \cup \{\mathbf{y}^i\}$ is a set of affinely independent vectors, we can conclude that $\dim(P^*) = n$. Let $\mathbf{y} = [y_1, y_2, \ldots, y_n]^T$. Since $\mathbf{y} \in P$ we know that for $1 \leq i \leq m$: $2a_{i,1}y_1 + 2a_{i,2}y_2 + \ldots 2a_{i,n}y_n \leq 2b_i$. For $\mathbf{y}^j = \mathbf{y} + \epsilon * \mathbf{e}_j$ being in P, we have to make sure that: $2a_{i,1}y_1 + \ldots + 2a_{i,j}(y_j + \epsilon) + \ldots + 2a_{i,n}y_n \leq 2b_i + 1$ and thus we have to set $\epsilon \leq 1/(2a_{i,j})$. Clearly, $\epsilon \leq \frac{1}{2\max(a_{i,j})}$ insures that all n vectors are in P.

□

2.2 Geometric Definitions

Definition 2 *A parallelepiped in the n-dimensional space, generated by a set of n linearly independent vectors (*$\Gamma = \{\mathbf{b_1}, \mathbf{b_2}, \ldots \mathbf{b_n}\}$*) is the convex hull of* $\mathbf{0}$ *and all vectors that are sums of different vectors of* Γ.

Example: the parallelepiped spanned by $\Gamma = \{(0,1),(0,1)\}$ is the interior of the 2-dimensional unit square. In [Scri2] it was shown that the volume of a parallelepiped is the product $\|\mathbf{b_1^*}\| \ldots \|\mathbf{b_n^*}\|$, where we denote by $\mathbf{b_i^*}$ the part of $\mathbf{b_i}$ orthogonal to the space spanned by the vectors $\mathbf{b_1}, \ldots, \mathbf{b_{i-1}}$. In our example $\mathbf{b_i} = \mathbf{b_i^*}$ and thus the volume is equal to 1 (as expected).

Definition 3 *A simplex in the n-dimensional space is the convex hull of* $n+1$ *affinely independent points. We will implicitly assume a simplex has always one vertex at the origin.*

Example: If $n = 2$, then the set of points $V = \{(0,0),(0,1)),(0,1)\}$ are affinely independent and the simplex defined by V is the interior of the triangle defined by the vertices in V. The volume of the simplex is obviously $\frac{1}{2}$. It can be shown that in general the volume of a simplex is $\frac{1}{n!}*$ (Volume of the paralellepiped spanned by $V - \{\mathbf{0}\}$).

3 Preliminaries for Lenstra's Algorithm

Our overall goal is to find an integer point in the polytope P. So we would like to start with a (non-integer) point that is as "deep" as possible in the the polytope P. This can be achieved by constructing two ellipsoids E, E', such that $E \subseteq P \subseteq E'$ and E is obtained by shrinking E' by (only) a polynomial factor. The intuition behind this is that since we only have to shrink E' "a little", E' must be rather tight around P and thus its center must be well within P. Hence,

As the next theorem states, we have now achieved our intermediate goal:

Theorem 3.1 *Let ξ and r be defined as before. Then there is an R such that $B_1 \subseteq P' \subseteq B_2$, $B_2 = B(\xi, R)$ and $\frac{R}{r} \leq 4n^{\frac{3}{2}}$*

PROOF: We have proved the first containment in Lemma 3.2. About the second containment: Assume for a moment that B_2 is centered at $\mathbf{y_0}$. Then $R = \sqrt{n}$ would be sufficient to enclose the cube of Lemma 3.3, which in turn encloses P'. However, B_2 is not centered at $\mathbf{y_0}$. Still, we know that B_2's center is somewhere in this cube, and thus a radius of length $2\sqrt{n}$ (length of the diagonal of the cube) is sufficient. Hence we have $\frac{R}{r} \leq \frac{2\sqrt{n}}{1/2n} = 4n^{\frac{3}{2}}$ □

We have now demonstrated that our choice of $\mathbf{W}$ has been a good one. Now we should also show that we can make this choice in polynomial time. In [Scri1] it was shown that the number of vertex candidates is $\binom{m}{n}$. Thus we can compute the volume of every possible simplex in time proportional to $\binom{\binom{m}{n}}{n+1}$. Since n is a constant, the above, "dumb" method of choosing $\mathbf{W}$ is polynomial!

4 Lenstra's Algorithm

In the last section, we have shown how to find a point (ξ) which is "deep" in the interior of P'. Now we show how to round its coordinates to the nearest lattice point. If we are lucky, the new point will be still in P' and we are done. If the rounded point is not in P' then we know that the grid defined by the actual basis of the lattice must be rather coarse and we will use this knowledge to reduce the problem at hand to a number of simpler, i.e. lower-dimensional problems.

4.1 The Algorithm

1. Find a reduced basis for $\mathcal{L}(\mathbf{W^{-1}})$. Call the basis vectors $\mathbf{b_1}, \mathbf{b_2}, \ldots, \mathbf{b_n}$.
2. Since the vectors in the reduced basis are linearly independent, we can write $\xi = \sum_{i=1}^{n} \lambda_i \mathbf{b_i}$. Let $w_i = [\lambda_i]$, where $[t]$ is the integer nearest to t. Now let $\mathbf{w} = \sum_{i=1}^{n} w_i \mathbf{b_i}$.
3. If $\mathbf{w} \in P'$ then we are done.
4. Otherwise: Since $\mathbf{w}$ is not in the polytope, it is also not in the ball which is enclosed in the polytope:

$$\|\mathbf{w} - \xi\| > r \tag{1}$$

In addition we have:

$$\|\mathbf{w} - \xi\| = \|\sum_{i=1}^{n}([\lambda_i] - \lambda_i)\mathbf{b_i}\| \leq (\frac{n}{2} \cdot \max_i \|\mathbf{b_i}\|) = \frac{n}{2} \cdot \|\mathbf{b_1}\| \tag{2}$$

For any basis of a lattice the following holds:

$$det(\mathcal{L}) \leq \Pi_{i=1}^{n} \|\mathbf{b_i}\| \tag{11}$$

The value of $det(\mathcal{L})$ can be computed as follows:

$$det(\mathcal{L}) = |det[\mathbf{b_1} \ldots \mathbf{b_n}]| = |det[\mathbf{b_1} \ldots \mathbf{b_{l-1}}, \mathbf{b^*}, \mathbf{b_{l+1}} \ldots \mathbf{b_n}]| \tag{12}$$

We now simultaneously combine (10) & (11), replace $\mathbf{b_l}$ by $\mathbf{b^*}$ (okay by (12)) and factor out $\mathbf{b^*}$:

$$\|\mathbf{b_l}\| \cdot (\Pi_{i \neq l}^{n} \|\mathbf{b_i}\|) \leq (2^{\frac{n(n-1)}{4}}) \cdot (\|\mathbf{b^*}\|) \cdot (\Pi_{i \neq l}^{n} \|\mathbf{b_i}\|) \tag{13}$$

and we finally get:

$$\|\mathbf{b_l}\| \leq 2^{\frac{n(n-1)}{4}} \cdot \|\mathbf{b^*}\| \tag{14}$$

□

4.2 Complexity

Let us briefly review the algorithm, sketch its complexity and check whether it is really polynomial: First we have to find the matrix $\mathbf{W}$. As shown, this takes time proportional to $\left(\binom{m}{n} \right)$. We also need to have the inverse of $\mathbf{W}$. The cost of this is $O(n^3)$. Then we must find the reduced basis of the lattice $\mathcal{L}(\mathbf{W}^{-1})$. Following [Scri2] this requires $O(n^2 \log(nb_{max}))$, where b_{max} is is the biggest component in $\mathbf{W}^{-1}$. Finally, we have to find the coefficients of ξ in the given basis and to test membership of the rounded point. This requires additionial costs of $O(n^2)$. At the end we either recurse to $2 \cdot 2^{n^2+1}$ problems in $n-1$ dimensions or we stop. Bearing in mind that n is a constant, we see that the algorithm has a polynomial running time. From a practical point of view however, we have to be aware that the huge number of problems we have to recurse to in case we fall outside of P' when rounding, will restrict the use of this algorithm to problems in small dimensions.

5 A Slightly More General Problem

Remember the general problem introduced in Section 1: $\max\{\mathbf{cx} \mid \mathbf{Ax} \leq \mathbf{b};\ \mathbf{x} \text{ integral}\}$. A similar fact as for the simpler problem holds here too (see [GS78]): If there is a solution $\mathbf{x}$ to this problem, then the size of $\mathbf{x}$ is polynomially bounded in the input. Thus, by using binary search and Lenstra's algorithm, we also obtain a polynomial algorithm for this problem.

References

[GS78] J. von zur Gathen and M. Sieveking, "A bound on solutions of linear integer equalities and inequalities", *Proc. of the AMS*, **72** (1978), pp. 155 - 158.

[Len83] H. W. Jr. Lenstra, "Integer programmming with a fixed number of variables", *Mathematics of Operations Research* **8** (1983), pp. 538 - 548.

Combinatorial Optimization

Lecture No. 27, 12/11/90 | **Ajit Agrawal** | Scribe: Ajit Agrawal

1 Overview

In this class, we will be studying a factor of 2 approximation algorithm for scheduling unrelated jobs on parallel machines. The algorithm is due to Lenstra, Shmoys, and Tardos [1].

2 The Problem

We consider the following scheduling problem. There are m parallel machines and n independent jobs. Each job is to be assigned to one of the machines. The processing of job j on machine i requires integral time p_{ij}. The total time used by a machine j is the sum of the processing times of the jobs assigned to the machine, and the *makespan* of the schedule is the maximum total time used by any machine. The objective is to find a schedule that has the minimum makespan over all schedules. The problem is known to be NP-complete.

3 Results

Since the makespan problem is NP-complete, we cannot find the best schedule in polynomial time. However, Lenstra, Shmoys and Tardos give a polynomial time algorithm that constructs a schedule with a makespan no more than twice the minimum makespan over all schedules. They also show that no polynomial algorithm can guarantee better than $\frac{3}{2}$-approximation to the makespan problem, unless P = NP. The paper also presents a polynomial algorithm for the case when the processing times are restricted to be either 1 or 2. However, we shall not be presenting that here.

4 A special case

To motivate ourselves to understand the problem, consider the makespan problem when the processing time of every job on every machine is 1. This problem has a simple polynomial algorithm. Map job j to the machine j mod m. Since the total work to be done is of n units divided among m machines, the makespan must be at least $\lceil \frac{n}{m} \rceil$, which is exactly the makespan given by the algorithm.

5 The general case

We shall now show how to go about getting a factor of 2 approximation for the general case. Note that in the general case, there are no restrictions on the values of p_{ij} except that they must be non-negative integers.

says YES, it has a schedule that has a makespan at most $2u$. We now set u to d. In either case, we reduce the range between l and u to about half its original range. We repeat this procedure until l equals u. At that point we know that the makespan is at least l and by construction, the decision procedure will output a schedule with makespan at most $2l$, and hence at most twice the optimal makespan. Thus we have a 2-approximation algorithm to the makespan problem by using $O(\log m)$ calls to the 2-relaxed decision oracle.

7 2-relaxed decision procedure

We now show how to construct the 2-relaxed decision procedure. When given an instance of the makespan problem and a value d, this procedure must produce a schedule if there exists one with makespan no more than $2d$.

We shall define a 0-1 variable x_{ij}, where i ranges over the jobs from 1 through n, and j ranges over the machines from 1 through m. x_{ij} is set to 1 if the job i is assigned to machine j by the schedule, and is 0 otherwise. An assignment of 0-1 to the variables x_{ij} defines an acceptable schedule provided each job is assigned to exactly one machine, and the total time taken by a machine is no more than d. These constraints can more formally be written in the following manner :

1) Job constraints: every job must be executed.

$$\sum_{1 \le j \le m} x_{ij} = 1, \ \forall \text{ job } 1 \le i \le n$$

2) Machine constraints: the total time taken by every machine must be less than the deadline d.

$$\sum_{1 \le i \le n} p_{ij} x_{ij} \le d, \ \forall \text{ machine } 1 \le j \le m$$

3) Trivial constraint: nonnegativity constraints.

$$x_{ij} \ge 0, \ \forall\, 1 \le i \le n, \forall\, 1 \le j \le m,$$

4) Integrality constraints.

$$x_{ij} \in \{0,1\}, \ \forall\, 1 \le i \le n, \forall\, 1 \le j \le m,$$

Note that the constraints above define a bounded polytope since all the variables must be in the range between 0 and 1. We shall call the above polytope an integral d-polytope. There exists a schedule with makespan at most d if there is a feasible point within the integral d-polytope. However, checking for a feasible point in the integral polytope is hard. If we relax the integrality constraints and allow x_{ij} to be a fractional value between 0 and 1, then the ellipsoid algorithm will find a vertex in this polytope. We shall call the polytope defined by dropping the integrality constraints (constraint type 4) as the fractional d-polytope. While it is easy to determine a vertex in this fractional d-polytope, we cannot guarantee that the vertices of this polytope also lie within the corresponding integral d-polytope. However, we shall show that a vertex of the fractional d-polytope can be "rounded" to give a feasible point in the integral $2d$-polytope.

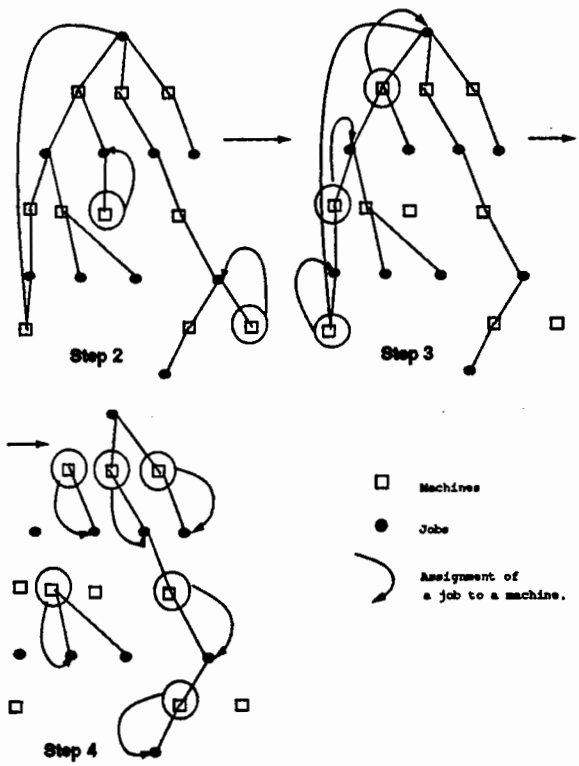


Figure 3: The rounding algorithm: The above example shows pictorially how a vertex $\tilde{x}$ in the fractional d-polytope is rounded to a feasible point in the integral $2d$-polytope. If a job is assigned completely to a processor in the fractional solution, then it is also assigned to the same processor in the integral solution (step 2). The cycles are removed from the bipartite graph by finding a trivial matching of jobs to processors (step 3). The cycles are of even length and there is a maximum of one cycle per connected component. The rest of the jobs are assigned to the machines by finding a trivial matching of jobs to machines using the tree structure (step 4).

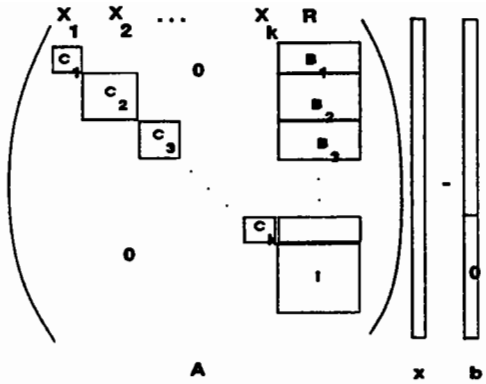


Figure 4: The set of tight constraints for $\tilde{x}$: If the variables are permuted according to the components that they lie in, then the permuted matrix A has the above form, where there are blocks of non-zero elements along the diagonal. After eliminating the non-zero elements in the blocks $B_1, \ldots, B_k$, the matrix has non-zero elements only along the diagonal blocks, which proves that each of the diagonal blocks must be of full rank.

bipartite graph G'. We shall show that for each connected component, the total number of edges is at most the total number of nodes in the component, which will imply that G' is a pseudoforest, since each component can then have at most one cycle.

Let G' have k nontrivial components $C_1, \ldots, C_k$. The nontrivial components are the components with at least one edge. We say that a variable x_{ij} belongs to a component if the nodes i and j belong to the component. Consider the set of variables X_l belonging to the component C_l. Let R be the set of variables not belonging to any of the components. Then there must be a trivial constraint for each variable in R. Let us rearrange the constraints C so that the variables are arranged in the order $X_1, \ldots, X_k$, and then R, the rest of the variables. We claim that the permuted matrix A has a structure shown in figure 4. If a constraint is a trivial constraint, then the variable corresponding to this trivial constraint cannot belong to any of the nontrivial components, and hence must belong to R. For a non-trivial constraint, the constraint is either a job constraint or a machine constraint. If it is a job constraint, then there are nonzero entries in A corresponding to the job's in G. Each such variable corresponds either to the same component as the job node or goes between two components. If it goes between two components, it belongs to R. Thus the only non-zero entries in a job constraint correspond either to the to the variables belonging to the same component as the job or correspond to variables in R. A similar statement can be shown for each machine constraint.

The matrix A after permutation then has blocks of nonzero matrices along the diagonal and has non-zero elements in the columns corresponding to variables in R. But since each variable in R has a trivial constraint in C, this constraint can be used in a gaussian elimination step to convert all the nonzero elements in its column to 0 without affecting any other columns. We will then be left with a matrix which has nonzero elments only in blocks along the diagonal. However, the rank of this matrix is still v. The only way it is possible is if each of the block matrices corresponding to the components are of full rank. Thus for each component, the number of nontrivial tight constraints equals the number of variables corresponding to the component, and hence equals the number of edges in G'. Since each job node and machine node have exactly one nontrivial constraint, the number of nontrivial contraints for the nodes of each component can be at most the number of

9 Review

We showed that we if we can find a 2-relaxed decision procedure for makespan, then we can construct a factor of 2 approximation algorithm. We constructed the 2-relaxed decision procedure by defining an integral d-polytope for a given d. We relaxed the integrality constraints to obtain a fractional d-polytope. We found a vertex in this polytope and gave a rounding procedure that produced a point in the integral $2d$-polytope, and thus defines a schedule with makespan at most $2d$. We also showed that getting better than $\frac{3}{2}$ approximation is NP-complete.

References

[1] K. K. Lenstra, D. B. Shmoys and E. Tardos, "Approximation algorithms for scheduling unrelated parallel machines," *28th Symposium on Foundations of Computer Science* (1987), pp. 217-224.

Definition 4 For any set A, span of A (denoted by $s(A)$) is the maximal set $S \subseteq M$ which contains A such that $r(S) = r(A)$. We say that A spans K if $K \subseteq s(A)$.

Definition 5 A set A is a base of M if it is a maximal set in $\mathcal{I}$.

We now give two lemmas without proof, since these lemmas are just alternative ways of defining matroids

Lemma 1 *Axiom 2 in the definition 1 is quivalent to the following statement.*

- *The union of an independent set and any element contains atmost one circuit.*

Lemma 2 *The two axioms in definition 1 are equivalent to the following two axioms.*

- *Axiom* $1'$ *No circuit contains another circuit.*
- *Axiom* $2'$ *If distinct circuits* C_1 *and* C_2 *both contain an element* e *then* $C_1 \cup C_2 - e$ *contains a circuit.*

Now using axioms $1'$ and $2'$ we will prove a lemma which abt circuits in a matroid, will be useful in proving the main theorem of the lecture.

Lemma 3 *If* C_1 *and* C_2 *are circuits in a matroid* M *with an elment* e *in common and an element* a *such that* $a \in C_1 - C_2$, *then there is a circuit* C *such that* $a \in C \subset C_1 \cup C_2 - e$.

Proof: Suppose C_1, C_2, a and e are such that the lemma is false and $C_1 \cup C_2$ is minimal. By axiom $2'$ there is a circuit $C_3 \subset C_1 \cup C_2 - e$ but a is not an element of C_3. By axiom $1'$ there is an element $b \in C_3 \cap (C_2 - C_1)$. By minimality of $C_1 \cup C_2$, there is a circuit C_4 such that $e \in C_4 \subset C_2 \cup C_3 - b$. Since $a \in C_1 - C_2$, it is not in C_4. Using minimality again we see that there is a circuit C such that $a \in C \subset C_1 \cup C_4 - e \subset C_1 \cup C_2 - e$. □

2 Partitioning a Matroid

Theorem 1 *The elements of a matroid* M *can be partitioned into* k *or fewer independent sets if and only if there is no subset* A *of* M *for which* $|A| > k \times r(A)$.

The rest of the lecture will be devoted to proving theorem 1. We will see a non-constructive proof for the only if part and a constructive proof for the if part. The constructive part of the proof gives us an algorithm to partition a matroid into disjoint independent sets, if we have an algorithm to determine for any $A \subset M$ and $e \in M$ whether there exists a circuit $C \subset A \cup e$ such that $e \in C$, and if so output such a circuit. For many common matroids such as the graph matroids this is a trivial task. Using theorem 1 we can for example, partition a graph into the smallest number of edge-disjoint forests.

Only-if: Suppose we have a collection C of k independent sets $\{I_1, I_2, \ldots, I_k\}$ which partitions M, and suppose that A is any subset of M. Let $A_i = I_i \cap A$. Since $\cup_{i=1}^{k} I_i = M$ $A = A \cap M = \cup_{i=1}^{k} A_i$. Also by axiom 1 each A_i is an independent set. This implies that $r(A_i) \leq r(A)$, therefore $|A| \leq k \times r(A)$. □

Combinatorial Optimization

Lecture: Dec. 18, 1990 Scribe: David Luks
Number 29

Lecturers: Michael Goldwasser & Seth Rosenzweig

1 Motivation

Schriver's Theorem of Min-Max relations generalizes over many other significant results. In this set of notes, some of these "consequences" of the theorem will be discussed. Each of the examples given, begins with some specified graph, and uses the theorem to derive some notable corollaries. We assume to prove these consequences that the theorem is, in fact, true.

The theorem that follows is not proven in this set of notes. However, a proof is given in the notes that accompany these.

2 Statement of Theorem

Theorem 1 *Let $D' = (V, A')$ be a directed graph such that d' is acyclic (not strongly connected) and each source of D' is connected to each sink of D' by a dipath. Then for each digraph $D = (V, A)$, we have the following:*

(a) For any length function $l : A \to \mathbf{Z}_+$, the minimum length of a strong connector for D' is equal to the maximum number of (not necessarily distinct) strong cuts such that each arc a of D is contained in at most $l(a)$ of these strong cuts. (In other words, we will try to show that the minimum size strong connector is the same as the maximum number of packed strong cuts.

(b) For any capacity function $c : A \to \mathbf{Z}_+$, the minimum capacity of a strong cut is equal to the maximum number of (not necessarily distinct) strong connectors such that each arc of D is contained in at most $c(a)$ of these strong connectors.

3 Discussion and Background

Before we mention any of the five consequences that will be presented, some background is necessary to clarify our examples. For all of our examples, let $D = (V, A)$ and $D' = (V, A')$ both be directed graphs with the same vertex set, V.

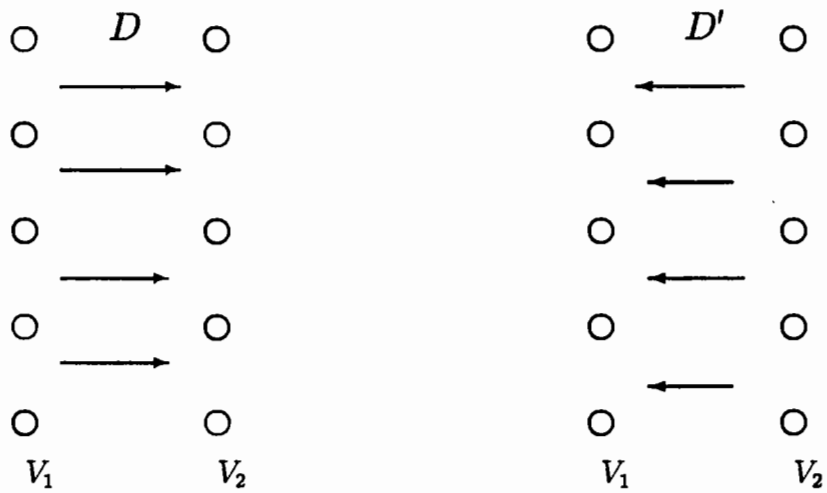
We define a **strong connector** for D' to be any subset $A'' \subseteq D$ such that the directed graph $(V, A' \cup A'')$ is strongly connected. (i.e. Take edges from D and consider these together with the orginal edges of D'. If this gives rise to a strongly connected graph, then the edges that we have added comprise a strong connector.)

we will be interested in those edges between the subset containing s and the other one, containing r. This is exactly the $r - s$ cut.

We have concluded that a strong-connector in this case is exactly an $r-s$ path and that a strong cut is exactly an $r - s$ cut. Applying the theorem, we obtain the fact that the minimum $r - s$ path must be the same as the maximum number of packing cuts. (how many cuts one can make without overfilling the edges). By the second part of the theorem, we obtain that the minimum $r - s$ is maximum $r - s$ path. Or, that the minimum cuts, is actually equal to the maximum flow. (The maximum r-s path is exactly the r-s flow)

4.2 Consequence 2

Let $G = (V, E)$ be a bipartite graph where V_1 and V_2 partition the vertices of V. (Namely, $V_1 \cup V_2 = V$.) Let $D = (V, A)$ where $A = \{(v_1, v_2) \mid v_1 \in V_1, v_2 \in V_2, (v_1, v_2) \in E\}$. Further, let $D' = (V, A')$ where $A' = \{(v_2, v_1) | v_2 \in V_2, v_1 \in V_1, (v_2, v_1) \in E\}$. ($D$ consists of all edges from the group of vertices V_1 to V_2 and D' is all of the opposite edges) This is illustrated in the following picture:

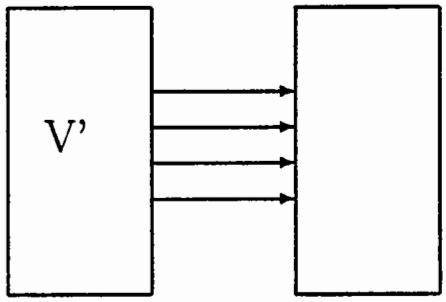


Once again, we will try to identify the strong connector and the strong cuts for the example. Clearly, in order to make the graph of D' strongly connected we will need to add edges out of everything in V_1, but also into everything in V_2. This is exactly the edge covering number by definition. (In an edge covering, every vertex is touched by at least one edge)

To find a strong cut, induced by D', consider a set of vertcies, V' such that no arrow of D' enters. This would work with any subset of the set V_2. To find a strong cut for this set, simply find all edges from V_1 that enter. The smallest such set would be the edges out of some vertex in V_1. The smallest possible such set would be found by finding the vertex with the smallest degree in D. To find the maximum strong cut, recall the definition of clique number. The clique number is the maximum number of vertices such that no two vertices are adjacent. In the case of the bipartite graph, the clique number will be exactly one half of the vertices or the other. (Note: The clique number is often referred to as the stability number)

From the above explanation and the theorem, we resolve the following two results which can be attributed to König and Gupta, respectively:

no arrow from D' enters V'. By definition, we have a cut of V. Consider, those edges that exit V'. To find a strong cut, we are trying to find those edges which enter V'. Consider the following picture:



Directed Cut

To find any of these edges in D, recall that any edge of D is just an edge of D' reversed. Therefore, all of the edges of the strong cut must have been in C originally. We also note, that the edges must have been also in some Directed cut of the graph, as in the above picture. From all of this, we notice that, a strong cut = (Directed Cut $\cap C)^R$.

To use the theorem, first let k = the minimum size strong cut. From our theorem, we must have a partition of D into k different strong connectors. (Notice, that though every edge may not be the member of a minimum strong connector, adding edges to a strong connector, will always keep a graph strongly connected. Therefore, any edge that is not included, add onto some strong connector) So, we have partitioned our edges, A into k partitions. Furthermore, each of these partition pieces must intersect a directed cut, from what we have shown above.

We can extend this example, to obtain the result of Lucchesi-Younger. If we let C be our entire graph, then it follows directly from the definitions and by the above argument:

Lucchesi-Younger Theorem For any digraph, the minimum cardinality of a dijoin is equal to the maximum number of pairwise disjoint dicuts.

5 Conclusion

Using Schriver's Theorem as stated in an early section of these notes, we have generalized to several important results. The notes that accompany these, will prove the theorem, but will as shown above prove all of the above results, as well.

Theorem 3 Let f_1 and f_2 be integral submodular set-functions on a set X, such that $f_i(X') \geq \max\{|X'|, k\}$ for each nonempty subset X' of X, and $i = 1, 2$. Then X can be partitioned into classes $X_1, \ldots, X_k$ such that $f_i(X') \geq \sum_{j=1}^{k} \max\{|X_j \cap X'|, 1\}$ for each nonempty subset X' of X, and $i = 1, 2$.

Theorem 4 (bi-branching theorem) Let $D = (V, A)$ be a directed graph, and let V be split into classes V_1 and V_2, such that any nonempty subset of V_1 (of V_2, respectively) is entered (left, respectively) by at least k arrows of D. Then A can be split into k bi-branchings.

Theorem 5 Let $D = (V, A)$ be an acyclic directed graph, such that any pair of source and sink is connected by a directed path, and let C be a subset of A such that each directed cut of D intersects C in at least k arrows. Then C can be split into classes $C_1, \ldots, C_k$ such that each class C_i intersects each directed cut.

Theorem 6 (Schrijver's min-max theorem for directed graphs) Let $D' = (V, A')$ be an acyclic directed graph, such that each source/sink pair is connected by a directed path. Let $D = (V, A)$ be a directed graph. Then the maximum number of pairwise disjoint strong connectors for D' is equal to the minimum size of a strong cut induced by D'.

Theorem 7 (An additional result from the Lucchesi-Younger theorem) Let $D = (V, A)$ and $D' = (V, A')$ be directed graphs, such that for each arrow $a = (v, w)$ of D there are vertices v' and w' and directed paths in D' from v to v', from w' to v', and from w' to w. Let $l : A \rightarrow \mathbf{Z}_+$ be a length function. Then the minimum length of a strong connector for D' is equal to the maximum number of strong cuts induced by D' such that no arrow a is in more than $l(a)$ of these strong cuts.

3 The Proof

We will use these smaller theorems to prove Schrijver's result. By using their proofs as examples, we can extend and generalize them to arrive at the full min/max theorem. At first, it seems circular, since we consider the smaller theorems to be special cases or corollaries of Theorem 1. In actuality, the proof works as a breakdown into cases. It so happens that the individual cases (i.e., the smaller theorems) have been proven, and further that the cases are useful in themselves.

Let us assume the Lucchesi-Younger theorem (Theorem 7). It has been proven before, so this assumtion is safe. Two questions come to mind. First, why must the graph be acyclic? Second, why must there be a path from every source to every sink?

Consider this observation from Schrijver's paper. Given v and w in X (and thus nodes in both D and D'), such that there is an edge $a \in A$. Let us also take v' and w', such that v' is a sink, and w' is a source. We will have paths in D' as follows: $v \rightarrow v'$, $w' \rightarrow v'$, $w' \rightarrow w$. (See figure 1) Note especially that $(v, w) \in D$ but $(w, v) \notin D'$. The number of edges a will be the basis for induction used later in the proof.

We want to assume the existence of v' and w', hence we need the existance of a path from from every source to every sink. Note that when $v' \neq v$ and $w' \neq w$, we have a generalization of the Lucchesi-Younger theorem. In particular, we would have one direction of the theorem (the other being given by Edmonds-Giles, Theorem 5). If the number of edges a is zero, we have a condition covered either by Lucchesi-Younger or by Edmonds-Giles (depending on which direction we are going).

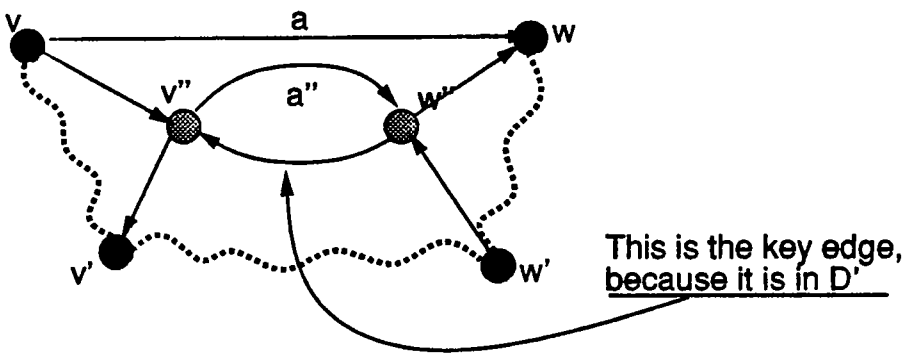


Figure 3: The arrangement of the edges connecting the new nodes with the nodes in the old graph.

One of the first questions that comes up when an algorithm for finding a strong connector is shown is how does one find a minimum weight strong connector. In the general case, finding minimum strong cuts is polynomial-time solvable, but finding minimum strong connectors is not. By using blocking theory, the $\mathbb{Z}$ property, and (indirectly) the Ellipsoid method, it is possible to show that there is a polynomial-time solution for finding integer solutions. In other words, some minimum strong connectors are easy to find, but all strong connectors are easy.

As a final note, it is possible to draw many parallels between this theorem and the T-Join theorem. Indeed, this theorem can be seen as an analogy for directed graphs to the T-Join theorem (which operates on undirected graphs).