CS 598CSC: Combinatorial Optimization	Lecture date: 01/19/2010
Instructor: Chandra Chekuri	Scribe: Alina Ene

1 Introduction and Motivation

Roughly speaking, an optimization problem has the following outline: given an instance of the problem, find the "best" *solution* among all solutions to the given instance. We will be mostly interested in discrete optimization problems where the instances and the solution set for each instance is from a discrete set. This is in contrast to continuous optimization where the input instance and the solution set for an instance can come from a continuous domain. Some of these distinctions are not as clear-cut as linear programming shows.

We assume familiarity with the computational complexity classes **P** , **NP** , **coNP** . In this class we are mainly interested in polynomial time solvable "combinatorial" optimization problems. Combinatorial optimization problems are a subset of discrete optimization problems although there is no formal definition for them. A typical problem in combinatorial optimization has a ground set E of objects and solutions correspond to some subsets of 2^E (the power set of E) and a typical goal is to find either a maximum or minimum weight solution for some given weights on E. For example, in the spanning tree problem, the ground set E is the set of edges of a graph $G = (V, E)$ and the solutions are subsets of E that correspond to spanning trees in G.

We will be interested in **NP** optimization problems – **NPO** problems for short. Formally, a problem Q is a subset of Σ^*, where Σ is a finite alphabet such as binary. Each string I in Q is an *instance* of Q. For a string x we use $|x|$ to denote its length. We say that Q is an **NPO** problem if the following hold:

(i) for each $x \in \Sigma^*$ there is a polynomial time algorithm that can check if $x \in Q$, i.e., if x is a valid instance of Q

(ii) for each instance I there is a set $sol(I) \subset \Sigma^*$ such that

  (a) $\forall s \in sol(I)$, $|s| = poly(|I|)$

  (b) there exists a poly-time algorithm that on input I, s correctly outputs whether $s \in sol(I)$ or not

(iii) there is a function $val : \Sigma^* \times \Sigma^* \rightarrow \mathbb{Z}$ s.t. $val(I, s)$ assigns an integer to each instance I and $s \in sol(I)$ and moreover val can be computed by a poly-time algorithm.

Given an **NPO** problem, we say it is a minimization problem if the goal is to compute, given $I \in Q$, $\arg\min_{s \in sol(I)} val(I, s)$. It is a maximization problem if the goal is to compute $\arg\max_{s \in sol(I)} val(I, s)$. A natural *decision* problem associated with an **NPO** problem (say, maximization) is: given I and integer k, is there $s \in sol(I)$ s.t. $val(s, I) \geq k$.

Many problems we encounter are **NPO** problems. Some of them can be solved in polynomial time. It is widely believed and conjectured that **P** $\neq$ **NP** , which would mean that there are **NPO** problems (in particular, those whose decision versions are **NP** -complete) that do not have polynomial time algorithms. Assuming **P** $\neq$ **NP** , some important and useful questions are: What problems are in **P** ? What characterizes problems in **P** ?

These are not easy questions. One insight over the years is that computation and algorithms are difficult to understand and we are far from being able to characterize the complexity of problems. However, in limited settings we seek to study broad classes of problems and understand some unifying themes. One particular class of problems where this has been possible is the class of constraint satisfaction problems in the Boolean domain. A result of Schaefer completely characterizes which problems are in P and which are **NP** -complete. In fact, there is a nice dichotomy. However, the non-Boolean domain is much more complicated even in this limited setting.

In the field of combinatorial optimization some unified and elegant treatment can be given via polyhedra and the ellipsoid method. The purpose of this course is to expose you to some of these ideas as well as outline some general problems that are known to be solvable in polynomial time. The three ingredients in our study are

(i) polynomial time algorithms

(ii) structural results, especially via min-max characterizations of optimal solutions

(iii) polyhedral combinatorics

We will illustrate these ingredients as we go along with examples and general results. In this introductory lecture, we discuss some known examples to highlight the view point we will take.

2 Network Flow

Given directed graph $D = (V, A)$, two distinct nodes $s, t \in V$, and arc capacities $c : A \to \mathbb{R}^+$, a flow is a function $f : A \to \mathbb{R}^+$ s.t.

(i) $\sum_{a \in \delta^-(v)} f(a) = \sum_{a \in \delta^+(v)} f(a)$ for all $v \in V - \{s, t\}$

(ii) $0 \leq f(a) \leq c(a)$ for all $a \in A$

The value of f is

$$val(f) = \sum_{a \in \delta^+(s)} f(a) - \sum_{a \in \delta^-(s)} f(a) = \sum_{a \in \delta^-(t)} f(a) - \sum_{a \in \delta^+(t)} f(a)$$

The optimization problem is to maximize the flow from s to t.
An s-t cut is a partition of V into (A, B) s.t. $s \in A$, $t \in B$, and the capacity of this cut is

$$c(A, B) = \sum_{a \in \delta^+(A)} c(a)$$

Clearly, for any s-t flow f and any s-t cut (A, B)

$$val(f) \leq c(A, B) \Rightarrow \max\ s\text{-}t \text{ flow} \leq \min\ s\text{-}t \text{ cut capacity}$$

The well-known maxflow-mincut theorem of Menger and Ford-Fulkerson states that

Theorem 1 *In any directed graph, the max s-t flow value is equal to the min s-t cut capacity.*

Ford and Fulkerson proved the above theorem algorithmically. Although their augmenting path algorithm is not a polynomial-time algorithm, it can be made to run in polynomial time with very slight modifications (for example, the Edmonds-Karp modification to use the shortest augmenting path). Moreover, the algorithm shows that there is an *integer* valued maximum flow whenever the capacities are integer valued. Thus we have

Theorem 2 *In any directed graph, the max s-t flow value is equal to the min s-t cut capacity. Moreover, if c is integer valued then there is an integer valued maximum flow.*

This is an example of a polynomial time (good) algorithm revealing structural properties of the problem in terms of a min-max result and integrality of flows. Conversely, suppose we knew the maxflow-mincut theorem but did not know any algorithmic result. Could we gain some insight? We claim that the answer is yes. Consider the decision problem: given G, s, t, is the s-t max flow value at least some given number k? It is easy to see that this problem is in **NP** since one can give a feasible flow f of value at least k as an **NP** certificate[1]. However, using the maxflow-mincut theorem we can also see that it is in **coNP** . To show that the flow value is smaller than k, all we need to exhibit is a cut of capacity smaller than k. Therefore the min-max result shows that the problem is in **NP** $\cap$ **coNP** . Most "natural" decision problems in **NP** $\cap$ **coNP** have eventually been shown to have polynomial time algorithms (there are a few well-known exceptions). Moreover, a problem in **NP** $\cap$ **coNP** being **NP** -complete or **coNP** -complete would imply that **NP** = **coNP** , some thing that most believe is unlikely. Thus a min-max result implies that the decision version is in **NP** $\cap$ **coNP** , strong evidence for the existence of a poly-time algorithm. That does not imply that such an algorithm will come by easily. Examples include matchings in general graphs and linear programming.

Finally, let us consider network flow as a special case of a linear programming problem. We can write it as

$$\begin{array}{lll} \max & \sum_{a \in \delta^+(s)} f(a) - \sum_{a \in \delta^-(s)} f(a) & \\ s.t. & & \\ \sum_{a \in \delta^+(v)} f(a) - \sum_{a \in \delta^-(v)} f(a) = 0 & & \forall v \in V, v \neq s, t \\ f(a) \leq c(a) & & \forall a \in A \\ f(a) \geq 0 & & \forall a \in A \end{array}$$

From the polynomial time solvability of LP, one can conclude a poly-time algorithm for network flow. However, one can say much more. It is known that the matrix defining the above system of inequalities is *totally unimodular* (TUM). From this, we have that the vertices of the polytope are integral whenever the capacities are integral! Not only that, the dual also has integer vertices since the objective function has integer coefficients. In fact, one can derive the maxflow-mincut theorem from these facts about the polyhedra in question. In addition, one can derive more quite easily. For example, we could add lower bounds on the flow

$$\ell(a) \leq f(a) \leq c(a)$$

[1]We are ignoring a technicality here that the flow specification be of polynomial size in the input.

and one still obtains the fact that if there exists a flow that respects the lower/upper bounds and ℓ, c are integer valued, then there is a corresponding integer valued flow. Same for min-cost flow and several other variants such as circulations and transshipments.

3 Bipartite Matchings

Many of you have seen bipartite matchings reduced to flow. We can also treat them independently. Let $G = (X \cup Y, E)$ be a bipartite graph with bipartition given by X, Y. Recall that $M \subseteq E$ is a *matching* in a graph if no vertex in G is incident to more than one edge in M.

A vertex cover in $G = (V, E)$ is a subset of vertices $S \subseteq V$ such that for each edge $uv \in E$, u or v is in S. In other words, S covers all edges. We use $\nu(G)$ to indicate the cardinality of a maximum matching in G and $\tau(G)$ for the cardinality of a minimum vertex cover in G.

Proposition 3 *For every G, $\nu(G) \leq \tau(G)$.*

In bipartite graphs, one has the following theorem.

Theorem 4 *(König's Theorem) If G is bipartite then $\nu(G) = \tau(G)$.*

The above proves that the max matching and min vertex problems (decision versions) in bipartite graphs are both in **NP** $\cap$ **coNP** . (Note that the vertex cover problem is **NP** -hard in general graphs.) We therefore expect a polynomial time algorithm for $\nu(G)$ and $\tau(G)$ in bipartite graphs. As you know, one can reduce matching in bipartite graphs to maxflow and König's theorem follows from the maxflow-mincut theorem. One can also obtain a polynomial time augmenting path algorithm for matching in bipartite graphs (implicitly a maxflow algorithm) that proves König's theorem algorithmically.

We now look at the polyhedral aspect. We can write a simple LP as a relaxation for the maximum matching in a graph G.

$$\begin{aligned} \max \quad & \sum_{e \in E} x(e) \\ \sum_{e \in \delta(u)} x(e) \quad & \leq \quad 1 \qquad \forall u \in V \\ x(e) \quad & \geq \quad 0 \qquad \forall e \in E \end{aligned}$$

For bipartite graphs the above LP and its dual have integral solutions since the constraint matrix is TUM. One can easily derive König's theorem and a polynomial time algorithm from this. One also obtains a polynomial time algorithm for weighted matching (assignment problem).

4 General Graph Matchings

The constraint matrix of the basic LP for matchings given above is not integral for general graphs, as the following simple graph shows. Let $G = K_3$ be the complete graph on 3 vertices. The solution $x(e) = 1/2$ for each of the 3 edges in G is an optimum solution to the LP of value $3/2$ while the maximum matching in G has size 1.

The algorithmic study of general graph matchings and the polyhedral theory that was developed by Jack Edmonds in the 1960's, and his many foundational results are the start of the field of polyhedral combinatorics. Prior to the work of Edmonds, there was a min-max result for $\nu(G)$ due to Berge which is based on Tutte's necessary and sufficient condition for the existence of a perfect matching. To explain this, for a set $U \subseteq V$, let $o(G-U)$ be the number of odd cardinality components in the graph obtained from G by removing the vertices in U.

Tutte-Berge formula

$$\nu(G) = \min_{U \subseteq V} \frac{1}{2}(|V| + |U| - o(G-U))$$

We will prove the easy direction for now, i.e.,

$$\nu(G) \leq \frac{1}{2}(|V| + |U| - o(G-U)) \qquad \forall U \subseteq V$$

To see this, the number of unmatched vertices is at least $o(G-U) - |U|$, since each odd component in $G-U$ needs a vertex in U. Hence

$$\nu(G) \leq \frac{|V|}{2} - \frac{o(G-U) - |U|}{2} \leq \frac{1}{2}(|V| + |U| - o(G-U))$$

Corollary 5 *(Tutte's 1-factor theorem) G has a perfect matching iff $\forall U \subseteq V$ $o(G-U) \leq |U|$.*

The formula shows that the decision version of matching is in **NP** $\cap$ **coNP** . Edmonds gave the first polynomial time algorithms for finding a maximum cardinality matching and also more difficult maximum weight matching problem. As a consequence of his algorithmic work, Edmonds showed the following results on matching polytopes.

Consider the following polytope:

$$\begin{aligned}
\sum_{e \in \delta(v)} x(e) &\leq 1 \qquad \forall v \in V \\
\sum_{e \in E[U]} x(e) &\leq \lfloor \frac{|U|}{2} \rfloor \qquad \forall U, |U| \text{ odd} \\
0 &\leq x(e) \qquad \forall e \in E
\end{aligned}$$

Edmonds showed that the vertices of the above polytope are exactly the characteristic vectors of the matchings of G. Observe that the polytope has an exponential number of constraints. One can ask whether this description of the matching polytope useful. Clearly if one takes the convex hull of the matchings of G, one obtains a polytope in $\mathbb{R}^E$; one can do this for any combinatorial optimization problem and in general one gets an exponential number of inequalities. Edmonds argued that the matching polytope is different since

(i) the inequalities are described implicitly in an uniform way

(ii) his algorithms gave a way to optimize over this polytope

At that time, no poly-time algorithm was known for solving LPs, although LP was known to be in **NP** $\cap$**coNP** . In 1978, Khachiyan used the ellipsoid algorithm to show that linear programming is in **P** . Very soon, Padberg-Rao, Karp-Papadimitriou, and Grötschel-Lövasz-Schrijver independently realized that the ellipsoid algorithm has some very important features, and can be used to show the polynomial-time equivalence of optimization and separation for polyhedra.

Separation Problem for Polyhedron Q
Given n (the dimension of Q), and an upper bound L on the size of the numbers defining the inequalities of Q, and a rational vector $x_0 \in \mathbb{R}^n$, output correctly either that $Q\ x \in Q$ or a hyperplane $ax = b$ s.t. $ax \leq b\ \forall x \in Q$ and $ax_0 > b$.

Optimization Problem for Polyhedron Q
Given n (the dimension of Q), and an upper bound L on the size of the numbers defining the inequalities of Q, and a vector $c \in \mathbb{R}^n$, output correctly one of the following: (i) Q is empty (ii) $\max_{x \in Q} cx$ has no finite solution (iii) a vector x^* s.t. $cx^* = \max_{x \in Q} cx$.

Theorem 6 *(Grötschel-Lövasz-Schrijver) There is a polynomial time algorithm for the separation problem over Q iff there is a polynomial time algorithm for the optimization problem over Q.*

The above consequence of the ellipsoid method had/has a substantial theoretical impact on combinatorial optimization. In effect, it shows that an algorithm for a combinatorial optimization problem implies an understanding of the polytope associated with the underlying problem and vice-versa. For example, the weighted matching algorithm of Edmonds implies that one can separate over the matching polytope. Interesting, it took until 1982 for Padberg and Rao to find an *explicit* separation algorithm for the matching polytope although one is implied by the above theorem.

References

[1] J. Edmonds Paths, trees, and flowers. *Canadian J. of Mathematics*, 17, 449–467, 1965.

[2] J. Edmonds Maximum matching and a polyhedron with 0,1 vertices. *J. of Research National Bureau of Standards Section B*, 69, 73–77, 1965.

[3] M. Grötschel, L. Lövasz and A. Schrijver. *Goemetric Algorithms and Combinatorial Optimization.* Springer, 1988.

[4] L.G. Khachiyan. Polynomial algorithms in linear programming. (English translation of original Russian article). *Doklady Akademii Nauk SSSR* 244, 1093–1096, 1979.

[5] A. Schrijver. *Theory of Linear and Integer Programming (Paperback).* Wiley, 1998.

[6] A. Schrijver. *Combinatorial Optimization: Efficiency and Polyhedra.* Springer, 2003.