Lecture 4 (20.03.2013)

Scribe: Daniel Malinowski
Lecturer: Marcin Mucha

Advanced approximation using linear programming

1 Linear programming

During the next two lectures we will apply the following strategy:

1. Take an easy (polynomially solvable) problem P_1.
2. Write a linear program LP_1 for P_1 and prove some properties of it.
3. Change program LP_1 into LP_2 whose (integer) solutions correspond to solutions of a hard problem P_2.
4. Create an algorithm for finding good approximate solutions to P_2 by using the structural results proved for P_1.

Now we will recall some basic properties of linear programming.

Let LP be a linear program with n variables. The set of feasible solution of LP is a polytope $W \in R^n$,

Fact 1. *For $x \in W$ the following conditions are equivalent:*

- *x is a vertex of W (x is not in an internal point of any segment contained in W).*
- *There exist n linearly independent constraints in LP which are tight (that means that the inequalities are in fact equalities).*

Such x is called a *vertex* or an *extreme solution*.

Corollary 2. *If LP contains constraints $x_i \geq 0$ for all variables x_i, and x is an extreme solution of LP with k non zero variables, then x satisfies k nontrivial (different than $x_i \geq 0$) constraints from LP with equality (tightly).*

Remark 3. These k constraints are linearly independent even after removing from them variables equal to zero.

Fact 4. *If W is bounded, then there exists an optimal solution which is a vertex.*

Fact 5. *Such solution can be found in polynomial time.*

Fact 6. *Even if number of constraints in LP is big (e.g. 2^n), then an optimal extreme solution can be found in $poly(n)$ time if we have an effective representation of solutions of LP i.e. an algorithm, which works in $poly(n)$ time and for a point $a \in R^n$ decides if $a \in W$, and if not, then it returns unsatisfied constraint. Such representation is called a* separation oracle. *The algorithm that can solve linear programs given only a separation oracle is the* ellipsoid algorithm.

2 A toy problem

Let us consider the maximum weighted bipartite matching problem.

Let $G = (V, U, E)$ is a bipartite graph, $E \neq \emptyset$ and (for simplicity) $|V| = |U|$. We also have a weight function $w\colon E \to R_{\geq 0}$.

The following linear program LP is a relaxation of the integer linear program for this problem:

$$\text{maximize} \sum_{e \in E} w(e) x_e$$

$$\forall_{e \in E} x_e \geq 0$$

$$\forall_{v \in V \cup U} \sum_{e \in \delta(v)} x_e \leq 1$$

where $\delta(v)$ is a set of edges incident to v.

Fact 7. *In optimal extreme solutions of LP each $x_i \in \{0, 1\}$.*

Proof. Let x^* be a optimal extreme solution of LP, $E^* = \text{supp}(x^*) = \{e \in E\colon w(e) > 0\}$, V^*, U^* – vertices incident with E^* from V, U respectively and $G^* = (V^*, U^*, E^*)$.

Without loss of generality we can assume that $G = G^*$, because any solution for G^* is a solution for G and any extreme solution for G is also an extreme solution for $G*$.

By corollary 2, there exist $|E|$ constraints of the form $x^*(\delta(v)) \stackrel{def}{=} \sum_{e \in \delta(v)} x^*_e \leq 1$ which are tight. Such vertices v are called tight.

Lemma 8. *There exist e s.t. $x^*_e = 1$.*

Proof. On the contrary assume that $\forall_{e \in E} 0 < x^*_e < 1$. So degree of each tight vertex is at least 2.

$$|E| \geq \frac{1}{2} \sum_{v \text{ tight}} \deg(v) \geq \frac{1}{2} \sum_{v \text{ tight}} 2 = \#\{\text{tight vertices}\} \geq |E|$$

That means that these inequalities are in fact equalities. So all vertices are tight and have degree 2. Therefore G consists of cycles of even length. But since G is bipartite, if we take vertices from the left side of G and sum their constraints, we get the same value as if we summed constraints corresponding to vertices from right side. That means that these constraints are not linearly independent. We get a contradiction, so there exist an edge e s.t. $x^*_e = 1$ □

Now we can prove by induction that $x^*(e) = 1$ on all edges (recall that there all edges with $x^*(e)$ have been removed in the preprocessing stage). If there are no edges then the claim is obviously true. Otherwise we take any $e = (uv)$ with $x^*_e = 1$. Constraints of the LP imply that e is the only edge in the support of x^* incident on u or v. Therefore, we can just remove u, v together with e from G. The resulting graph is smaller and x^* induces an optimal extreme solution for this graph, so by induction $x^*(e) = 1$ for all edges.

Finally, since we initially removed all edges with $x^*(e) = 0$ from the graph, we get the claim. □

3 Generalized Assignment Problem (GAP)

GENERALIZED ASSIGNMENT PROBLEM (GAP)
Input: set of jobs J, set of machines M, bipartite graph $G = (J, M, E)$, execution times p_{ji} for $(ji) \in E$, execution costs c_{ji} for $(ji) \in E$ and maximum total time t_i for $i \in M$
Question: assign each job j to a machine i s.t. $(ji) \in E$ and $\forall_{i \in M} \sum_{j \text{ assigned to } i} p_{ji} \leq t_i$ minimizing $\sum_{i \in M} \sum_{j \text{ assigned to } i} c_{ji}$

We will find a solution to this problem with cost $\leq OPT$, but possibly using as much as twice the allowed time on each machine.

Let us consider a relaxation LP of a ILP program for GAP:

$$\text{minimize} \sum_{(ji) \in E} c_{ji} x_{ji}$$

$$\forall_{j \in J} \sum_{i \in M} x_{ji} = 1$$

$$\forall_{i \in M'} \sum_{j \in J} p_{ji} x_{ji} \leq T_i$$

$$\forall_{j \in J, i \in M} x_{ji} \geq 0$$

where at the beginning $M' = M$ (but with time we will remove some machines from M) and $T_i = t_i$.

As before we can remove variables x_{ji} s.t. x^*_{ji} is equal to 0. So from corollary 2 we get

Fact 9. *Let x^* be an optimal extreme solution of LP. Then there exist $\tilde{J} \subset J$ and $\tilde{M} \subset M'$ s.t.*

- $\forall_{j \in \tilde{J}} \sum_{i \in M} x_{ji} = 1$ *(note that this holds for all $j \in J$)*
- $\forall_{i \in \tilde{M}} \sum_{j \in J} p_{ji} x_{ji} = T_i$
- *constraints corresponding to $\tilde{J}$ and $\tilde{M}$ are linearly independent.*
- $|\tilde{J}| + |\tilde{M}| = |E|$

□

Algorithm 1: (due to David B. Shmoys and Éva Tardos)

Remove from G edges (ji) s.t. $p_{ji} > t_i$ (opt does not use them),
so we can assume that always $p_{ji} \leq t_i$;
while *there exist unassigned jobs* **do**
  [2.1] Find a new optimal and extreme solution x^* of LP;
  [2.2] Remove edges (ji) s.t. $x^*_{ji} = 0$;
  [2.3] If there exists edge (ji) s.t. $x^*_{ji} = 1$ then assign j to i, remove vertex j from G and decrease T_i by p_{ji};
  [2.4] Otherwise if there exists $i \in M'$ s.t. $\deg(i) = 1$ then remove i from M';
  [2.5] Otherwise if there exists $i \in M'$ s.t. $\deg(i) = 2$ and $\sum_{(ji) \in E} x_{ji} \geq 1$ then remove i from M';

Theorem 10. *This algorithm works in polynomial time, finds a solution with cost $\leq OPT$ and every machine $i \in M$ uses at most $2t_i$ time.*

Proof. To show, that the algorithm is polynomial it is sufficient to show that during every iteration of the loop one of the conditions 2.3, 2.4 or 2.5 is satisfied. This is because operation 2.3 decreases the size of G, while 2.4 and 2.5 decrease the size of M'.

Assume that there are no jobs with degree 1 (otherwise x^*_{ji} would be equal to 1) and no machines in M' with degree 1. Counting the edges and using Fact 9 we get

$$|E| \geq \frac{2|J| + 2|M'|}{2} \geq |\tilde{J}| + |\tilde{M}| \geq |E|$$

Each inequality must be an equality so $J = \tilde{J}$, $\tilde{M} = M'$ (which means that all constraints are tight) and degrees of all vertices are equal to 2. Like before, G consists of cycles of even length. This time however, we cannot use the simple left side vs right side dependence argument, because the constraint coefficients are no longer 0/1. Instead, note that for each job in any of the cycles in G, the sum of x^* is equal to 1. Therefore one of the machines in this cycle has sum of x^* at least 1. Condition 2.5 is satisfied for this machine.

Therefore, the algorithm stops after a polynomial number of steps. It is clear that the solution returned it is not worse than OPT. This is because whenever $x^*_{ji} = 1$ and we assign job j to machine i, we pay p_{ji}, but we also decrease the cost of the optimum LP solution by p_{ji}. Therefore the total cost paid in this way is at most OPT. The only other thing that we change in the linear program is we remove some constraints – but this can only make the cost of the optimum solution smaller.

To show that every machine $i \in M$ uses at most $2t_i$ time we have to look at the moment in which it is removed from M'. Before that moment we have used $t_i - T_i$ time on it.

If we delete it (from M') in 2.4 then in the future we can only use it for the single remaining job, or not use it at all. But even if we use it for this job, we will use at most t_i additional time, because we know, that $p_{ji} \leq t_i$ – so it will be used for at most $t_i + t_i = 2t_i$ time.

And if we delete it in 2.5 then we know, that it is connected to exactly two jobs – j_1 and j_2. In this case, we know that $x^*_{j_1 i} + x^*_{j_2 i} \geq 1$ and the solution is feasible, so $p_{j_1 i} x^*_{j_1 i} + p_{j_2 i} x^*_{j_2 i} \leq T_i$. By easy calculation we get that $\min(p_{j_1 i}, p_{j_2 i}) \leq T_i$, so even if we assign both of these jobs to machine i, the final solution will use machine i for at most $\max(p_{j_1 i}, p_{j_2 i}) \leq t_i$ additional time. □