CS491I Approximation Algorithms

The Vertex-Cover Problem

Lecture Notes

Xiangwen Li

March 8th and March 13th, 2001

Absolute Approximation

Given an optimization problem P, an algorithm A is an approximation algorithm for P if, for any given instance $x \in I$, it returns an approximation solution, that is a feasible solution $A(x)$.

Definition 1 Given an optimization problem P, for any instance x of P and an arbitrary feasible solution y.

Denote $m^*(x)$ the optimum value of x and $m(x, y)$ the value of function at feasible point y for the instance x. The absolute error of y with respect to x is defined as

$$|m^*(x) - m(x, y)|.$$

Given an NP-hard optimization problem, it would be very satisfactory to have a polynomial-time approximation algorithm that is capable of providing a solution with a bounded absolute error for every instance x.

Definition 2 Given an optimization problem P and an approximation algorithm A for P, we say that A is an absolute approximation algorithm if there exists a constant k such that, for every instance x of P, $|m^*(x) - m(x, A(x))| \leq k.$.

Example Let us consider the problem of determining the minimum number of colors needed to color a planar graph. It is well-known that a 6-coloring of a planar graph can be found in polynomial

time[1]. It is also known that establishing whether the graph is 1-colorable (that is, the set of edges is empty) or 2-colorable(that is, the graph is bipartite) is decidable in polynomial time whereas the problem of deciding whether three colors are enough is NP-complete(see [2] for detail).

Algorithm D

---

begin

```
If the graph is 1- or 2-colorable,
    return either a 1- or a 2-coloring of the vertices of G.

else
    return a 6-coloring of the vertices of G.
```

end

Then we obtain an approximate solution with absolute error bounded by 3,

$$| m^* - m_D | \leq 3.$$

Consider the Knapsack Problem. This problem is characterized by n objects $O_1, O_2, ,, O_n$ where each O_i is associated with a profit P_i and a weight W_i. We want to choose a subset of objects to maximize the profit without violating the fact that the knapsack has a limit weight W. The Integer Programming Formulation is:

$$\max \sum_{i=1}^{n} p_i x_i$$

$$\sum_{i=1}^{n} w_i x_i \leq W$$

$$x_i \in \{0,1\}$$

Let g be an instance of the Knapsack problem n items with profits $P_1, P_2, ,, P_n$ and sizes $W_1, W_2, ,, , W_n$. Suppose that the problem has a polynomial time approximation algorithm A with absolute error k and suppose that m^* and m_A are the optimum solution and the approximation solution respectively. Then

$$| m^* - m_A | \leq k.$$

Let y be instance of n items with profits

$(k+1)P_1,(k+1)P_2,,,,(k+1)P_n$ and sizes $W_1,W_2,,,,W_n$. Suppose that m'^* and $m_{A'}$ are the optimum solution and the approximation solution respectively. Then

$$|m_{A'} - m'^*| > k \quad \text{for} \quad \forall k \geq 1.$$

(for more detail see[1]).
See the following Fig.

Thus we have the following theorem.

Theorem 1 Unless P=NP, no polynomial-time absolute approximation algorithm exist for Knapsack Problem.

Definition 3 Given an optimization problem P, for any instance x of P and for any feasible solution y of x, the relative error of y with respective to x is defined as

$$R(x,y) = \max\left(\frac{m(x,y)}{m^*(x)}, \frac{m^*(x)}{m(x,y)}\right)$$

By the definition, given any instance x of P and R, there is an algorithm A such that

$$m(x, A(x)) \leq R \quad m^*(x).$$

Weighted Vertex Cover

A vertex cover of a undirected graph $G=(V,E)$ is a subset $V' \subseteq V$ such that if (u, v) is an edge of G, then either $u \in V'$ or $v \in V'$. If a non-negative weight w_i is associated with each vertex v_i, we want to find a vertex cover having minimum total weight. We call this problem minimum weighted vertex cover. Given a weighted graph $G=(V,E)$, minimum weighted vertex cover can be formulated as the following integer linear program ILP

$$\min \sum_{v_i \in V} w_i x_i$$
$$x_i + x_j \geq 1 \qquad \forall (x_i, x_j) \in E,$$
$$x_i \in \{0,1\}.$$

Let LP be the linear program obtained by relaxing the integrality constraints to simple non-negativeness constraints (i.e., $x_i \geq 0$ for each $v_i \in V$).

Program A

---

begin

0. $V' = 0$;
1. Relax ILP to LP, by replacing the constraints $x_i \in \{0,1\}.$ with $x_i \in [0,1].$;i.e. x_i is a continuous variable between 0 and 1.
2. For each v_i such that $x_i \geq \frac{1}{2}$

$V' = V' \cup \{v_i\}$

return V'

end

Theorem 2 Given a graph G with non-negative vertex weights, Program A finds a feasible solution of minimum weighted vertex cover with value $m_{LP}(G)$ such that $m_{LP}(G) \leq 2m^*(G)$.

Proof. Let V' be the solution returned by the algorithm. The feasibility of V' can be easily proved by contradiction. Assume that V' does not cover edge (v_i, v_j). This implies that both $x_i^*(G)$ and $x_j^*(G)$ are less than 0.5 and hence $x_i^*(G) + x_j^*(G) < 1$, a contradiction.

Let $m^*(G)$ be the value of an optimal solution and let $m_{LP}^*(G)$ be the optimal value of the relaxed linear program.

Since the value of an optimal solution is always larger than the optimal value of the relaxed linear program, we have

$$m^*(G) \geq m_{LP}^*(G).$$

If $v_i \in V'$, then $x_i \geq \frac{1}{2}$ and

$$\sum_{v_i \in V'} w_i \leq 2 \sum_{v_i \in V'} w_i x_i^*(G).$$

Since $V' \subseteq V$,

$$2 \sum_{v_i \in V} w_i x_i^*(G) \leq 2 \sum w_i x_i^*(G).$$

By the linear program algorithm,

$$2 \sum_{v_i \in V'} w_i x_i^*(G) \leq 2 m_{LP}^*(G) \leq 2 m_{IP}^*(G).$$

Therefore the theorem follows.

Primal-dual algorithms

The implementation of program A requires the solution of a linear program with a possibly large number of constrains. Therefore it is computationally expensive. Another approach known as primal-dual allows us to obtain an approximate solution more efficiently. The chief idea is that any dual feasible solution is a good lower bound on the minimization primal problem. We start with a primal dual pair (x*, y*), where x* is a primal variable, which is not necessarily feasible, while y* is the dual variable, which is not necessarily optimal. At each step of the algorithm, we attempt to make y* "more optimal" and x* "more feasible"; the algorithm stops when x* becomes feasible. Given a weighted graph $G = (V, E)$, the dual of the previously defined LP is the following program DLP:

$$\min \sum_{(v_i, v_j) \in E} y_{ij}$$

$$\sum_{(v_i, v_j) \in E} y_{ij} \leq w_i, \forall v_i \in V$$

$$y_{ij} \geq 0, \forall (v_i, v_j) \in E$$

Note that the solution in which all y_{ij} are zero is a feasible solution with value 0 of DLP. Also note that there is no dual for an integer program; we are taking the dual of the linear programming relaxation of the primal integer program. The primal dual algorithm is described the following.

Program B

---

begin

for each dual variable y_{ij} ***do*** $y_{ij} = 0$

$V' = 0$

while V' is not a vertex cover ***do***

Select some edge (v_i, v_j) not covered by V';

Increase v_{ij} till one of the end-points is hit. i.e., $y_{ij} = w_i$ or $y_{ij} = w_j$

If $y_{ij} = w_i$ ***then***

$V' = V' \cup \{v_i\}$

else

$V' \cup \{v_j\}$

end while

return V'

end

Theorem 3 Given a graph G with non-negative weights, program B finds a feasible solution of minimum weighted vertex cover with value $m_{DLP}(G)$ such that $m_{DLP}(G) \leq 2m^*(G)$.

Proof Let V' be the solution obtained by the algorithm. By construction V' is a feasible solution. We observe that for every $v_i \in V'$ we have $\sum_{(v_i, v_j) \in E} y_{ij} = w_i$. Therefore

$$m_{DLP}(G) = \sum_{v_i \in V'} w_i = \sum_{v \in V'} \sum_{(v_i, v_j) \in E} y_{ij} .$$

Since $V' \subseteq V$,

$$\sum_{v \in V'} \sum_{(v_i, v_j) \in E} y_{ij} \leq \sum_{v_i \in V} \sum_{(v_i, v_j) \in E} y_{ij} .$$

Since every edge of E counts two times in $\sum_{v_i \in V} \sum_{(v_i, v_j) \in E} y_{ij}$,

$$\sum_{v_i \in V} \sum_{(v_i, v_j) \in E} y_{ij} = 2 \sum_{(v_i, v_j) \in E} y_{ij} .$$

Since $\sum_{(v_i, v_j) \in E} y_{ij} \leq m^*(G)$, the theorem follows.

Randomization

Let X be a discrete random variable and take the values $a_1, a_2, , , , a_n$, with probabilities $p(a_i), p(a_2), .. p(a_n)$. The expectation of X is defined as

$$E(X) = \sum_{i=1}^{n} a_i p(a_i) .$$

Linearity properties of Expectation

1. $E(X_1 + X_2) = E(X_1) + E(X_2).$

2. $E(aX) = aE(X),$ where a is a positive, real number.

Example 1 Suppose that there are 40 sailors and 40 keys one of which is for one room. Suppose that each sailor takes one key randomly and opens his room with this key. What is the Expected number of sailors in their own rooms?

Let $x_i = 1$ if sailor i enters his room, $x_i = 0$ otherwise.

By using linearity of expectation, we have

$$E(\sum_{i=1}^{40} X_i) = \sum_{i=1}^{40} E(X_i) = 40(\frac{1}{40}(1) + \frac{39}{40}(0)) = 1.$$

We have used the fact that $E(X_i) = 1.p_i + 0.(1 - p_i)$, since this is a binary event. Thus, $E(X_i) = (1/40)$, for all i

Example 2

A Boolean expression is said to be in k-conjunctive normal form (k-SAT) if it is the conjunction of clauses such that each clause contains exactly k variables. Suppose that there are *m* clauses and *n* variables in the Boolean expression. The goal is to find an

assignment of {True/False} to each of the variables, so that the given Boolean expression is satisfied. The corresponding optimization problem is to *maximize* the number of clauses that are satisfied (The optimization problem is called MAX-SAT and is also NP-complete.). Consider a random assignment of {True/False} to the variables i.e. each variable x_i is set to True or False, depending upon the random outcome of flipping a fair coin. It is easy to see that the probability that

A single clause is not satisfied is at most $\frac{1}{2^k}$, since in order to falsify a given clause, all its variables must be set to False, in the random assignment. Hence, with probability $(1-\frac{1}{2^k})$, each clause is satisfied. We want to know the expected number of clauses that are satisfied by our random assignment.
By using linear properties of expectation, we have

$$E(\sum_{i=1}^{m} X_i) = \sum_{i=1}^{m} E(X_i) = \sum_{i=1}^{m} (1-\frac{1}{2^k}) = m(1-\frac{1}{2^k}).$$

Note that the maximum number of clauses that can be satisfied under any assignment is m. Hence, our random assignment is an approximation algorithm with approximation factor $(1-\frac{1}{2^k})$.

(Exercise: Substitute various values of k and see how good the approximation gets.)

Reference:

[1] B.korte and J.Vygen, Combinatorial Optimization, Springer-Verlag, 2000.
[2] M. R. Garey and D. S. Johnson, Computers and intractability—a guide to the theory of NP-completeness, 1979.