CSC3160: Design and Analysis of Algorithms

Week 11: Approximation Algorithms

Instructor: Shengyu Zhang

Optimization

- Very often we need to solve an optimization problem.
  - Maximize the utility/payoff/gain/…
  - Minimize the cost/penalty/loss/…
- Many optimization problems are NP-complete
  - No polynomial algorithms are known, and most likely, they don’t exist.
  - More details in the previous lecture.
- Approximation: get an approximately good solution.

Example 1: A simple approximation algorithm for 3SAT

SAT

- 3SAT:
  - n variables: $x_1, \dots, x_n \in \{0,1\}$
  - m clauses: OR of 3 variables or their negations
    - e.g. $\overline{x_1} \vee x_2 \vee \overline{x_3}$
  - CNF formula: AND of these m clauses
    - E.g. $\phi = (\overline{x_1} \vee x_2 \vee \overline{x_3}) \wedge (\overline{x_2} \vee x_4 \vee \overline{x_5}) \wedge (x_1 \vee x_3 \vee \overline{x_5})$

$x = 10010$

- 3SAT Problem: Is there an assignment of variables x s.t. the formula ϕ evaluates to 1?
  - i.e. assign a 0/1 value to each x_i to satisfy all clauses.

Hard

- 3SAT is known as an NP-complete problem.
  - Very hard: no polynomial algorithm is known.
  - Conjecture: no polynomial algorithm exists.
  - If a polynomial algorithm exists for 3SAT, then polynomial algorithms exist for all NP problems.
- More details in last lecture.

7/8-approximation of 3SAT

- Since 3SAT appears too hard in its full generality, let’s aim lower.
- 3SAT asks whether there is an assignment satisfying all clauses.
- Can you find an assignment satisfying half of the clauses?
- Let’s run an example where
  - you give an input instance
  - you give a solution!

Observation

- What did we just do?
- How did we assign values to variables?
- For each variable x_i, we ___ choose a number from {0,1}.
- How good is this assignment?
  - Result: __ out 5; __ out 5.

Why?

- For each clause, there are 8 possible assignments for these three variables, and only 1 fails.
  - E.g. $x_1 \vee x_2 \vee x_3$: only $(x_1, x_2, x_3) = (0,0,0)$ fails.
  - E.g. $\overline{x_1} \vee x_2 \vee \overline{x_3}$: only $(x_1, x_2, x_3) = (1,0,1)$ fails.
- Thus if you assign randomly, then with each clause fails with probability only 1/8.
- Thus the expected number of satisfied clauses is $7m/8$.
  - m: number of clauses

Formally - algorithm

- Repeat

  Pick a random $a \in \{0,1\}^n$.

  See how many clauses the assignment $x = a$ satisfies.

  Return a if it satisfies $\geq 7m/8$ clauses.

- This is a Las Vegas algorithm:
  - The running time is not fixed. It's a random variable.
  - When the algorithm terminates, it always gives a correct output.
  - The complexity measure is the expected running time.

Formally - analysis

- Define a random variable Y_i for each clause i.
  - If clause i is satisfied, then $Y_i = 1$, otherwise $Y_i = 0$.
- Define another random variable $Y = \sum_i Y_i$
  - Y has a clear meaning: number of satisfied clauses
- What's expectation of Y?

$\mathbf{E}[Y]$

$$
\begin{aligned}
& \mathbf{E}[Y] && \text{// expected \# satisfied clauses} \\
&= \mathbf{E}[\textstyle\sum_i Y_i] && \text{// definition of } Y\text{: } Y = \textstyle\sum_i Y_i \\
&= \textstyle\sum_i \mathbf{E}[Y_i] && \text{// linearity of expectation} \\
&= \textstyle\sum_i \mathbf{Pr}[C_i \text{ satisfied}] && \text{// definition of } Y_i \\
&= \textstyle\sum_i 7/8 \\
&= \frac{7}{8} m.
\end{aligned}
$$

- This means that if we choose assignment randomly, then we can satisfy $\geq 7/8$ fraction of clauses *on average*.

Success probability of one assignment

- We've seen the average number of satisfied clauses on a random assignment.
- Now we translates this to the average running time of the algorithm?
- event "success": A random assignment satisfies $\geq 7/8$ fraction of clauses,
- We want to estimate the probability p of success.

Getting a Las Vegas algorithm

- $\frac{7m}{8} = \mathbf{E}[Y] = \sum_{k=1}^{m} k \cdot \mathbf{Pr}[Y = k]$
  $\leq pm + (1-p)\left(\left\lceil \frac{7m}{8} \right\rceil - 1\right)$
  $\leq pm + (1-p)\left(\frac{7m}{8} - \frac{1}{8}\right)$
- Rearranging, we get $p \geq \frac{1}{8m}$.
- If we repeatedly take random assignments, it needs $\leq 8m$ times (on average) to see a "success" happening.
  - i.e. the complexity of this Las Vegas algorithm is $\leq 8m$.

Example 2: Approximation algorithm for Vertex Cover

Vertex Cover: Use vertex to cover edges

- Vertex Cover: "Use vertices to cover edges". For an undirected graph $G = (V, E)$, a vertex set $S \subseteq V$ is a vertex cover if all edges are touched by S.
  - i.e. each edge is incident to at least one vertex in S.
- Vertex Cover: Given an undirected graph, find a vertex cover with the minimum size.

- NP-complete
  - So it’s (almost) impossible to find the minimum vertex cover in polynomial time.

- But there is a polynomial time algorithm that can find a vertex cover of size at most twice of that of minimum vertex cover.

IP formulation

- Formulate the problem as an integer programming.
- Suppose S is a min vertex cover. How to find S?
- Associate a variable $x(v) \in \{0,1\}$ with each vertex $v \in V$.
  - Interpretation: $x(v) = 1$ iff $v \in S$.
- The constraint that each edge (u, v) is covered?
  - $x(u) + x(v) \geq 1$.
- The objective?
  - $\min|\{v: x(v) = 1\}| = \min \sum_{v \in V} x(v)$

IP formulation, continued.

- Thus the problem is now
  - min $\quad \sum_{v \in V} x(v)$

    s.t. $\quad x(u) + x(v) \geq 1, \ \forall (u, v) \in E$

    $\quad x(v) \in \{0,1\}, \ \forall v \in V$
- Integer Programming. NP-hard in general.
  - For this problem: even the feasibility problem, i.e. to decide whether the feasible region is empty or not, is NP-hard.
- What should we do?

LP relaxation

min $\sum_{v \in V} x(v)$

s.t. $x(u) + x(v) \geq 1, \forall (u, v) \in E$

$x(v) \in \{0,1\}, \forall v \in V$

- Note that all problems are caused by the integer constraint.
- Let's change it to: $0 \leq x(v) \leq 1, \forall v \in V$.
- Now all constraints are linear, so is the objective function.
- So it's an LP problem, for which polynomial-time algorithms exist.

Relaxation

- Original IP

  min $\sum_{v \in V} x(v)$
  s.t. $x(u) + x(v) \geq 1,$
  $x(v) \in \{0,1\}$,

  Relaxed LP

  min $\sum_{v \in V} x(v)$
  s.t. $x(u) + x(v) \geq 1,$
  $0 \leq x(v) \leq 1$

- This is called the linear programming relaxation.

Two key issues

- The solution to the LP is not integer valued. So it doesn't give an interpretation of vertex cover any more.
  - Originally, solution $(1,0,0,1,1,0,1)$ means $S = (v_1, v_4, v_5, v_7)$.
  - Now, solution $(0.3, 0.8, 0.2, 1, 0.5, 0.7, 0, 0.9)$ means what?
- What can we say about the relation of the solutions (to the LP and that to the original IP)?

Issue 1: Construct a vertex cover from a solution of LP

- Recall:
  - In IP: solution $(1{,}0{,}0{,}1{,}1{,}0{,}1)$ means $S = (v_1, v_4, v_5, v_7)$.
  - In LP: solution $(0.3, 0.8, 0.2, 1, 0.5, 0.7, 0, 0.9)$ means …?

- Naturally, let's try the following:
  - If $x(v) \geq 1/2$, then pick the vertex v.
  - In other words, we get an integer value solution by rounding a real-value solution.

Issue 1, continued

- Question: Is this a vertex cover?
- Answer: Yes.
- For any edge (u, v), since $x(u) + x(v) \geq 1$, at least one of $x(u)$, $x(v)$ is $\geq$ ½, which will be picked to join the set.
- In other words, all edges are covered.

Issue 2: What can we say about the newly constructed vertex cover?

- [Claim] This vertex cover is at most twice as large as the optimal one.
- Denote:
  - S^*: an optimal vertex cover.
  - x^*: an solution of the LP
  - $R(x^*)$: the rounding solution from x^*
- Last slide: $|S^*| \leq |R(x^*)|$
  - min vertex cover $|S^*| \leq$ one vertex cover $|R(x^*)|$
- Now this claim says: $|R(x^*)| \leq 2|S^*|$

$|R(x^*)| \le 2|S^*|$

- Proof. We're gonna show that
  $$|R(x^*)| \le 2\sum_v x^*(v) \le 2|S^*|$$

- $\sum_v x^*(v) \le |S^*|$:
  - The feasible region of the LP is larger than that of the IP.
  - Thus the minimization of LP is smaller.

- $|R(x^*)| \le 2\sum_v x^*(v)$:
  - $\sum_v x^*(v) \ge \sum_{v:x^*(v)\ge 1/2} x^*(v)$ // we throw some part away
    $\ge \sum_{v:x^*(v)\ge 1/2} 1/2$ // $x^*(v) \ge 1/2$
    $= \frac{1}{2}|R(x^*)|$

Example 3: st-Min-Cut by randomized rounding

Obtaining an exact algorithm!

st-Min-Cut

- st-Min-Cut: "min-cut that cuts s and t" Given a weighted graph G and two vertices s and t, find a minimum cut $(S, V-S)$ s.t. $s \in S$ and $t \in V-S$.
  - Minimum: the total weight of crossing edges.
- Max-flow min-cut theorem gives one polynomial-time algorithm.
- We now give a new polynomial-time algorithm.

IP formulation

- Form as an IP:
  - Weight function: $c(u,v)$
  - $x_i = 0$ if vertex $i \in S$, 1 otherwise.
  - How about objective function?
- Objective function is

$$\sum_{\substack{(i,j)\in E:\ x_i=0,\ x_j=1, \\ or\ x_i=1,\ x_j=0}} c(i,j)$$

- But this is not a linear function of $\{x_i\}$.

Modification

- Introduce new variables $z_{ij} = |x_i - x_j|$
  - $z_{ij} = 1$ if (i, j) is a crossing edge, 0 otherwise
- Now the objective function is

$$\sum_{(i,j)\in E} c(i,j) z_{ij}$$

- But $z_{ij} = |x_i - x_j|$ is not a linear function either.

- Let's change $z_{ij} = |x_i - x_j|$ to $z_{ij} \geq |x_i - x_j|$,
  - It is ok since we are minimizing $\sum_{(i,j)\in E} c(i,j) z_{ij}$,
  - Since $c(i,j) \geq 0$, the minimization is always achieved by the smallest possible z_{ij}.
  - Thus the equality is always achieved in $z_{ij} \geq |x_i - x_j|$.
- What's good about the change?
- $z_{ij} \geq |x_i - x_j|$ is equivalent to

$$z_{ij} \geq x_i - x_j \text{ and } z_{ij} \geq x_j - x_i.$$

IP

- Now the IP is as follows.

min $\sum_{(i,j)\in E} c(i,j) z_{ij}$

s.t. $z_{ij} \geq x_i - x_j$ and $z_{ij} \geq x_j - x_i$

$x_s = 0, x_t = 1$

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

- As before, we relax it to an LP by changing the last constraint to

  $x_i \in [0,1]$.

- Solve it and get a solution (to LP) (x^*, z^*) with objective function value y^*.
- Since it's a LP relaxation of a minimization problem, it holds that

  $$y^* \leq OPT$$

  - OPT: the optimum value of the original IP, i.e. the cost of the best cut.

- [Thm] $y^* = OPT$

We prove this by randomized rounding

- Recall that rounding is a process to map the opt value of LP back to a feasible solution of IP.

- Randomized rounding: use randomization in this process.

- Our job: get an IP solution (x, z) from an opt solution (x^*, z^*) to LP.

Rounding algorithm

- Pick a number $u \in [0,1]$ uniformly at random.
- For each i, $x_i = 0$ if $x_i^* < u$ and $x_i = 1$ if $x_i^* \geq u$.
- For each edge (i,j), define $z_{ij} = |x_i - x_j|$
- Easy to verify that this is a feasible solution of IP.

min $\quad \sum_{(i,j)\in E} c(i,j) z_{ij}$

s.t. $\quad z_{ij} \geq x_i - x_j$ and $z_{ij} \geq x_j - x_i$

$\quad x_s = 0, x_t = 1$

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

- We now show that it's also an optimal solution.

- For each edge (i, j), what's the prob that it's a crossing edge? (i.e. $\mathbf{E}[z_{ij}]$.)
- Suppose $x_i^* < x_j^*$. Then

$$\mathbf{Pr}[(i,j) \text{ is crossing}] = \mathbf{Pr}\left[u \in \left[x_i^*, x_j^*\right]\right] = x_j^* - x_i^*.$$

- The other case $x_i^* \geq x_j^*$ is similar and

$$\mathbf{Pr}[(i,j) \text{ is crossing}] = x_i^* - x_j^*.$$

- Thus in any case,

$$\mathbf{Pr}[(i,j) \text{ is crossing}] = \left|x_i^* - x_j^*\right| = z_{ij}^*.$$

- We showed that $\mathbf{E}\left[z_{ij}\right] = z_{ij}^*$

- Thus by linearity of expectation,
  $\mathbf{E}\left[\sum_{(i,j)\in E} c(i,j) z_{ij}\right]$
  $= \sum_{(i,j)\in E} c(i,j)\mathbf{E}\left[z_{ij}\right]$
  $= \sum_{(i,j)\in E} c(i,j) z_{ij}^*$
  $= y^*$

- $\mathbf{E}\left[\sum_{(i,j)\in E} c(i,j) z_{ij}\right] = y^*$
- So the LP opt value y^*
  = average of some IP solution values
- Recall: $y^* \leq$ the best IP solutions values.
- Thus there must exist IP solutions values achieving the optimal LP solution value y^*.
- i.e. $y^* = OPT$.

Summary

- Many optimization problems are NP-complete.
- Approximation algorithms aim to find almost optimal solution.
- An important tool to design approximation algorithms is LP.

Appendix: Las Vegas → Monte Carlo

- Las Vegas algorithm:
  - The running time is not fixed. It's a random variable.
  - When the algorithm terminates, it always gives a correct output.
  - The complexity measure is the expected running time.
- Monte Carlo algorithm:
  - The running time is at most some fixed number $T(n)$.
  - When algorithm terminates, it gives an output which is correct with high probability, say 0.99.
  - The complexity is the running time $T(n)$.

Las Vegas → Monte Carlo

- A general way to change a Las Vegas algorithm $\mathcal{A}$ with cost $T(n)$ to a Monte Carlo algorithm $\mathcal{B}$ with cost $O\big(T(n)\big)$ is as follows.
- $\mathcal{B}$:

  for $i = 1$ **to** k

  run $\mathcal{A}$ for up to $2T(n)$ time

  if $\mathcal{A}$ outputs O, **then return** O

  return "Fail"

- Recall: $\mathbf{E}[\mathcal{A}$'s running time$] = T(n)$.
- By Markov's inequality, with prob. at least ½, $\mathcal{A}$ outputs an answer within $2T(n)$ time.
  - Recall Markov: $\mathbf{Pr}[X > a] \leq \mathbf{E}[X]/a$.
- As long as $\mathcal{A}$ outputs, the answer is correct.
- $\Pr[\mathcal{A}$ doesn't terminate within $2T(n)$ time in all k iterations$] \leq 2^{-k}$.
- So in $20 \cdot T(n)$ time, $\mathcal{B}$ outputs a correct answer with probability $1 - 2^{-10} \geq 0.999$.

Make-up class

- Topic: online algorithms.
  - The input is given as a sequence.

- Venue: ERB 713.
- Time: 2:30-5:15pm, April 17 (this Friday).
- Tutorial follows at 5:30pm, same classroom.

- Not required in exam.