Approximation Algorithms

Definitions and Examples

Abhijit Das

Department of Computer Science and Engineering
Indian Institute of Technology Kharagpur

October 11, 2020

Optimization Problems

- P is an optimization problem.
- $\mathscr{O}_I$ is the set of possible output instances on an input I.
- $f : \mathscr{O}_I \to \mathbb{R}$ is the **objective function**.
- Goal: To find an $O^* \in \mathscr{O}_I$ such that

  [Minimization problem] $f(O^*) \leqslant f(O)$

  [Maximization problem] $f(O^*) \geqslant f(O)$

  for all $O \in \mathscr{O}_I$.
- Ties may be broken arbitrarily.
- $f(O^*)$ is denoted by OPT_I or OPT.
- We say P is an optimization problem in NP if:
  - It is easy to test the membership $O \in \mathscr{O}_I$.
  - It is easy to compute $f(O)$ for every $O \in \mathscr{O}_I$.

Nondeterministic Polynomial-Time Optimization Algorithms

Nondeterministically generate candidates O.
Check whether $O \in \mathscr{O}_I$.
If yes, compute and return $f(O)$.

- There is a mechanism to take the minimum or maximum of all the returned values.
- This is similar to logically OR-ing all the returned values of nondeterministic algorithms for decision problems.
- If $p = |\mathscr{O}_I|$, then a common CRCW PRAM with p^2 processors can compute the minimum/maximum in $O(1)$ time.
- This algorithm must run in polynomial time. Therefore the candidate-generation stage should involve guessing only a polynomial number of bits.
- $|\mathscr{O}_I|$ should therefore be at most an exponential function of the input size.

Relation with Decision Problems

- Take an input I for P.
- Choose a bound B.
- The decision problem: Decide whether there exists an $O \in \mathscr{O}_I$ such that

$$\begin{array}{ll} \text{[Minimization problem]} & f(O) \leqslant B, \\ \text{[Maximization problem]} & f(O) \geqslant B. \end{array}$$

- For appropriate choices of B, the decision problem is solvable in polynomial time if and only if the optimization problem is solvable in polynomial time.
- The decision problem is in NP if and only if the optimization problem is in NP.
- Example: Let G be an undirected graph.
    - MIN_VERTEX_COVER: Find a smallest vertex cover of G.
    - VERTEX_COVER: Given k, decide whether G has a vertex cover of size $\leqslant k$.

Approximation Algorithms

- Let P be an optimization problem in NP.
- A is called an ρ-approximation algorithm for P if for all inputs I, A produces an output $O \in \mathscr{O}_I$ such that

$$\text{[Minimization problem]} \quad f(O) \leqslant \rho \times \text{OPT}_I,$$
$$\text{[Maximization problem]} \quad f(O) \geqslant \rho \times \text{OPT}_I.$$

- ρ is called the **approximation ratio** or the **approximation factor**.
- ρ is called **tight** if $f(O) = \rho \times \text{OPT}_I$ for some instances.
- For minimization problems, $\rho > 1$. For maximization problems, $0 < \rho < 1$.
- Values of ρ close to 1 are preferable.
- We require A to run in time polynomial in the size n of the input. The running time of A may also depend on ρ.

Note: Some authors define $\rho = \text{OPT}/f(O)$ for maximization problems, so $\rho > 1$ for all optimization problems.

Minimum Vertex Cover

- $G = (V, E)$ is an undirected graph.
- $|V| = n$ and $|E| = m$.
- A **vertex cover** for G is a subset $U \subseteq V$ such that every edge $e \in E$ has at least one endpoint in U.
- MIN_VERTEX_COVER: Find a vertex cover U with $|U|$ as small as possible.
- MIN_VERTEX_COVER is in NP:
  - It is easy to check whether U is a vertex cover.
  - It is easy to count the size of any vertex cover U.

A Logarithmic Approximation Algorithm for MIN_VERTEX_COVER

```
Initialize U = ∅.
while (E is not empty) {
        Find a vertex u ∈ V of largest (remaining) degree.
        Add u to U.
        Delete from E all the (remaining) edges with u as one endpoint.
}
Return U.
```

- This is a greedy algorithm.
- The running time is polynomial in $n+m$.

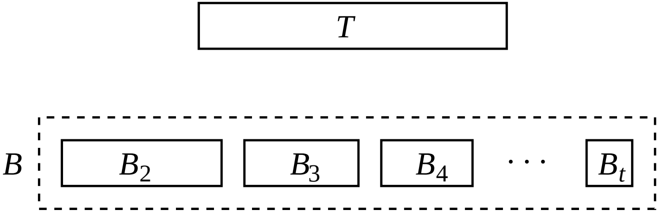
Derivation of the Approximation Ratio

- Let $|U| = k$.
- Vertices added to U are $u_1, u_2, \ldots, u_k$ in that order.
- Let $t = |U^*|$.
- $\rho = k/t$.
- $G_0 = G$.
- For $1 \leqslant i \leqslant k$, $G_i = (V, E_i)$ is the graph after the edges incident upon $u_1, u_2, \ldots, u_i$ are removed.
- $m_i = |E_i|$, so $m_0 = m$.

Passage from G_i to G_{i+1}

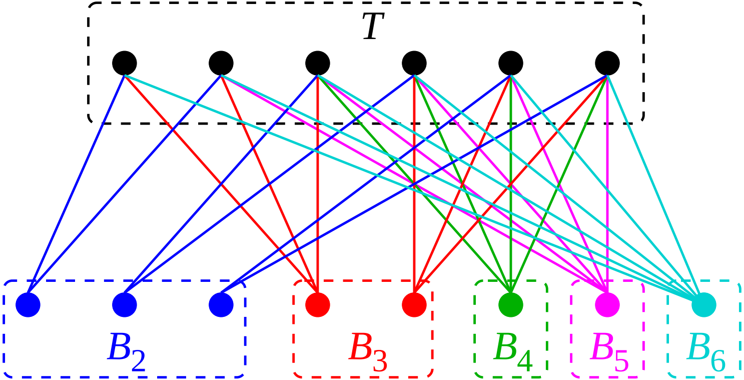
- $u_1, u_2, \ldots, u_i$ contain t_i of the t vertices of U^*.
- The remaining $t - t_i$ vertices of U^* constitute a vertex cover of G_i.
- There exists $v_{i+1} \in U^* \setminus \{u_1, u_2, \ldots, u_i\}$ whose degree in G_i is $\geqslant m_i/(t - t_i)$.
- $\deg(u_{i+1}) \geqslant \deg(v_{i+1})$ in G_i.
- $m_{i+1} \leqslant m_i \left(1 - \dfrac{1}{t - t_i}\right) \leqslant m_i \left(1 - \dfrac{1}{t}\right)$.
- $m_i \leqslant m \left(1 - \dfrac{1}{t}\right)^i$.
- For $i = t \ln m$, we have $m_i \leqslant m \left(1 - \dfrac{1}{t}\right)^{t \ln m} < m \left(e^{-1}\right)^{\ln m} = 1$.
- So $k \leqslant t \ln m$, that is, $\rho = k/t \leqslant \ln m = \Theta(\log n)$.

Tightness of ρ

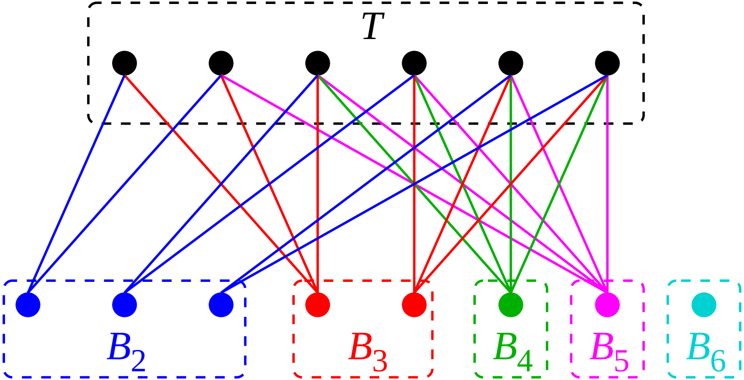


- Bipartite graph.
- $|T| = t$.
- $|B_i| = \lfloor t/i \rfloor$, so $|B| = \sum_{i=2}^{t} |B_i| = \sum_{i=2}^{t} \lfloor t/i \rfloor$.
- Each vertex in B_i is connected to i vertices in T.
- Vertices in B_i have mutually disjoint neighbor sets in T.

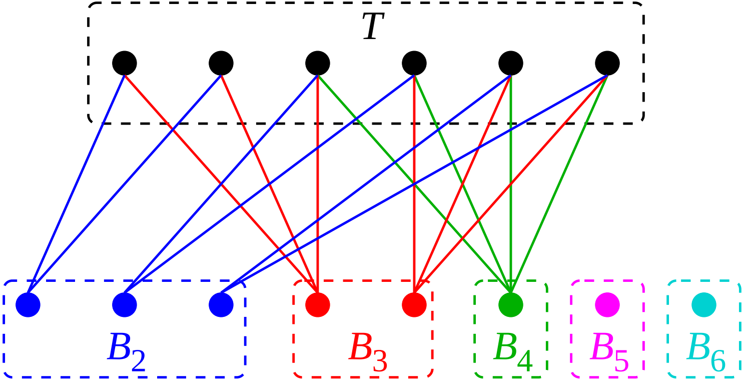
Tightness of ρ



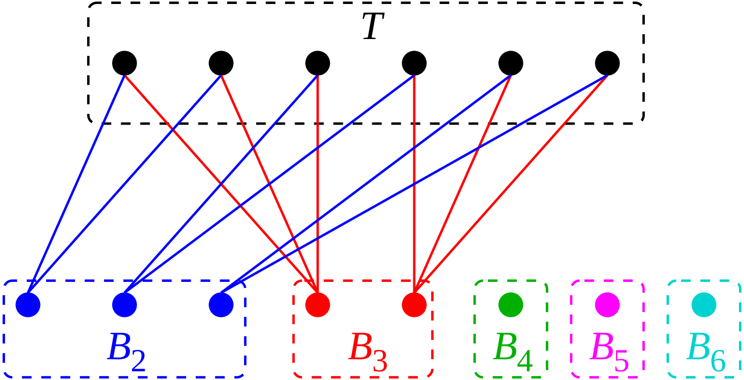
Tightness of ρ



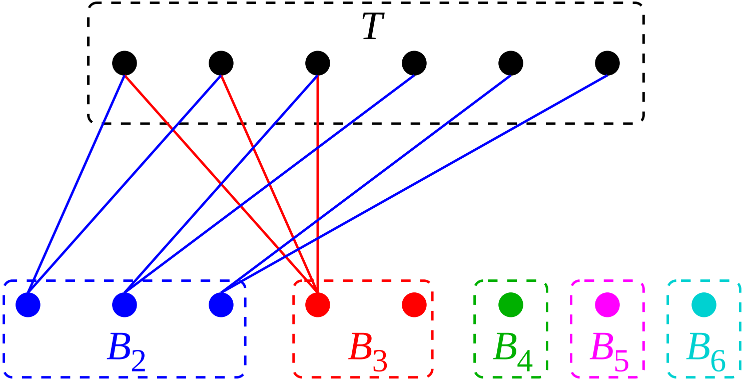
Tightness of ρ



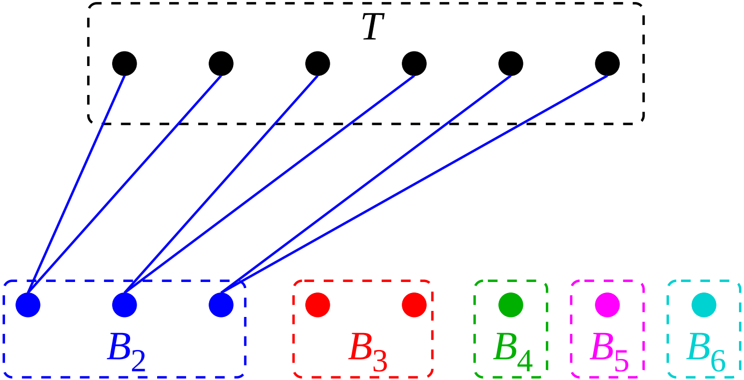
Tightness of ρ



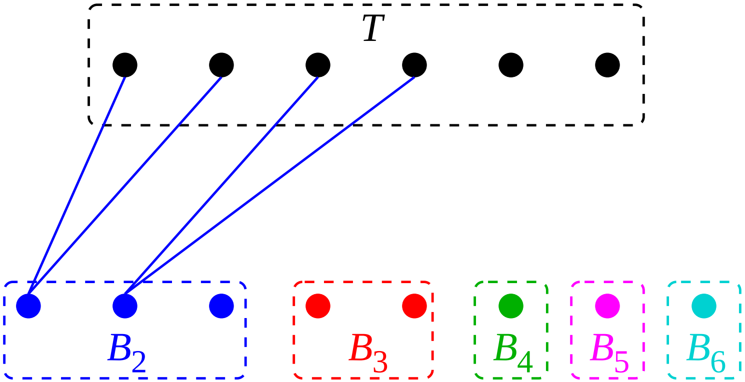
Tightness of ρ



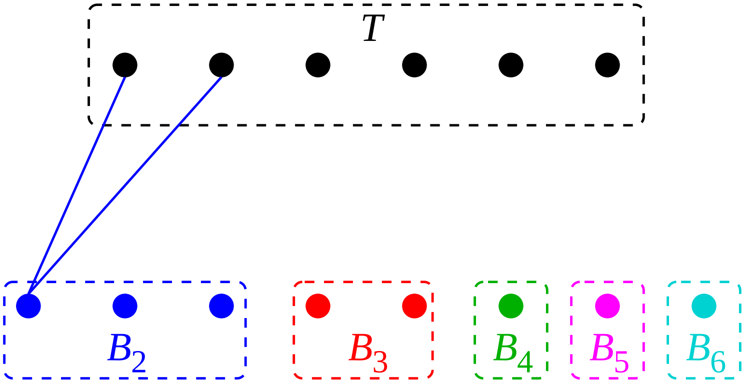
Tightness of ρ



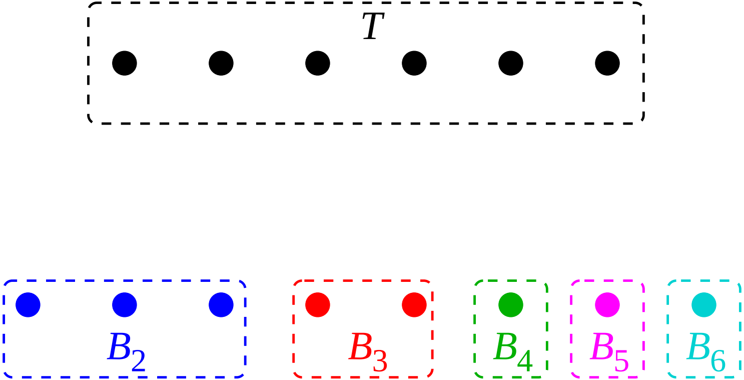
Tightness of ρ



Tightness of ρ



Tightness of ρ



Tightness of ρ

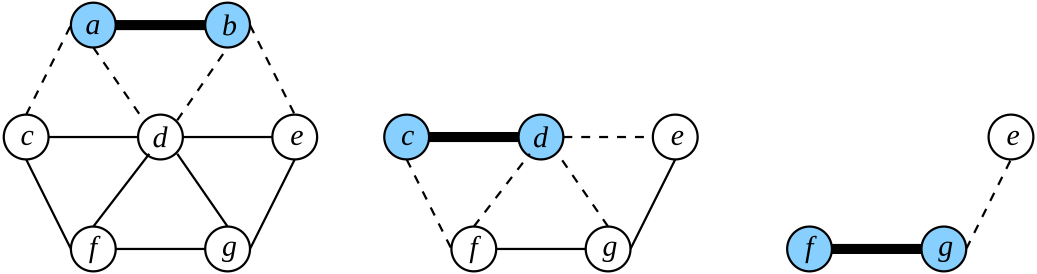
- $|B| = \sum_{i=2}^{t} \left\lfloor \frac{t}{i} \right\rfloor \leqslant \sum_{i=2}^{t} \frac{t}{i} = t(H_t - 1) \leqslant t \ln t.$
- $|B| = \sum_{i=2}^{t} \left\lfloor \frac{t}{i} \right\rfloor \geqslant \sum_{i=2}^{t} \frac{t-(i-1)}{i} = (t+1)\left(\sum_{i=2}^{t} \frac{1}{i}\right) - (t-1) \geqslant (t-1)(H_t - 2) \geqslant (t-1)(\ln(t+1) - 2).$
- $|U| = |B| = \Theta(t \log t).$
- T is a vertex cover, so $|U^*| \leqslant |T| = \frac{1}{\Theta(\log t)} |U|.$
- $n = |V| = |B| + |T| = \Theta(t \log t) \;\Rightarrow \log t = \Theta(\log n) \;\Rightarrow \rho = \frac{|U|}{|U^*|} \geqslant \Theta(\log n).$

2-Approximation Algorithm for MIN_VERTEX_COVER

- Based on matching.
- $D \subseteq E$ is called a matching if no two edges of D share an endpoint.
- Let D be any matching, and U any vertex cover.
- U must contain one endpoint of each edge in D.
- $|D| \leqslant |U|$.

```
Initialize U = ∅.
while (E is not empty) {
    Pick any edge e = (u, v) from E.
    Add u and v to U.
    Remove u and v from V.
    Remove from E all edges incident on u or v.
}
Return U.
```

Example



$U = \{a,b\}$ $\quad$ $U = \{a,b,c,d\}$ $\quad$ $U = \{a,b,c,d,f,g\}$

Approximation Ratio

- Let D be the set of edges chosen in the loop.
- D is a matching in G.
- $|U| = 2|D|$.
- $|D| \leqslant |U^*|$.
- $|U| \leqslant 2|U^*|$.
- $\rho = \dfrac{|U|}{|U^*|} \leqslant 2$.
- Tightness:
  - Take $G = K_{n,n}$ (complete bipartite graph).
  - $|U^*| = n$.
  - $|U| = 2n$.

Approximation Algorithms

More Examples

Abhijit Das

Department of Computer Science and Engineering
Indian Institute of Technology Kharagpur

October 11, 2020

Minimum Set Cover

- $X = \{x_1, x_2, x_3, \ldots, x_m\}$.
- $S_1, S_2, S_3, \ldots, S_n \subseteq X$ with $\bigcup_{i=1}^{n} S_i = X$.
- Take $1 \leqslant i_1 < i_2 < \cdots < i_k \leqslant n$.
- $S_{i_1}, S_{i_2}, \ldots, S_{i_k}$ is a **cover** of X if $\bigcup_{j=1}^{k} S_{i_j} = X$.
- To find a cover of X with k as small as possible.
- Vertex cover is a special case of set cover.

Logarithmic Approximation Algorithm for MIN_SET_COVER

```
Set U = ∅.
While (X ≠ ∅) {
    Find a subset S of maximum (current) size.
    Add S to U.
    Set X = X \ S.
    For all remaining subsets S_i (including S itself) {
        Set S_i = S_i \ S.
        If S_i is empty, remove S_i from the collection.
    }
}
Return U.
```

- Similar to the greedy algorithm for MIN_VERTEX_COVER.
- Analysis is similar. $\rho = \Theta(\log n)$.

Traveling Salesperson Problem (TSP)

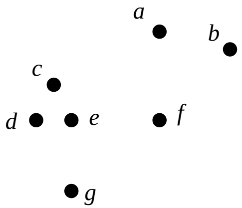
- $G = (V, E)$ is a complete undirected graph.
- Cost function $c : E \to \mathbb{R}^+$.
- $c(u, v) = c(v, u)$ for all $u, v \in V$.
- To find a Hamiltonian cycle Z in G for which the sum $c(Z)$ of all the edge costs on Z is as small as possible.
- TSP is in NP:
  - It is easy to check whether a vertex sequence is a Hamiltonian cycle.
  - It is easy to compute the cost of a Hamiltonian cycle.

- EUCLIDEAN_TSP:
  - Vertices are points in the 2-dimensional plane.
  - $c(u, v) = d(u, v)$ (Euclidean distance).

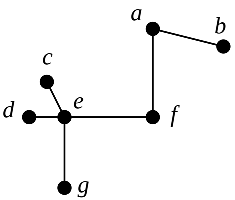
2-Approximation Algorithm for EUCLIDEAN_TSP

Compute a minimum spanning tree T of G.
Choose an arbitrary vertex u_1 of T.
Make a preorder traversal of T starting from u_1.
Let $W = (u_1, u_2, u_3, \ldots, u_{2n-1})$ be the list of visited nodes.
Remove duplicates from this list.
Append u_1 at the end to obtain the Hamiltonian cycle Z.
Return Z.

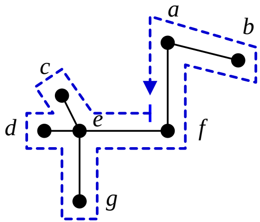
Example



(a) Location of the cities

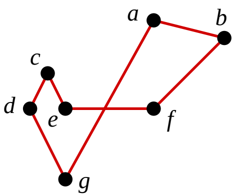


(b) Computation of an MST



(c) Preorder traversal of MST

$f,e,c,e,d,e,g,e,f,a,b,a,f$



(d) The TSP cycle

f,e,c,d,g,a,b,f

Approximation ratio

- Z is a Hamiltonian cycle returned by the algorithm.
- Z^* is an optimal Hamiltonian cycle.
- Removal of an edge from Z^* gives a spanning tree of G.
- $c(T) \leqslant c(Z^*)$.
- $c(W) = 2c(T)$.
- Duplicate removal:
  - Change u, v, w to u, w.
  - By the triangle inequality, $c(u,v) + c(v,w) \geqslant c(u,w)$.
  - The cost of W does not increase by duplicate removals.
- $c(Z) \leqslant c(W) = 2c(T) \leqslant 2c(Z^*)$.
- $\rho = \dfrac{c(Z)}{c(Z^*)} \leqslant 2$.

Inapproximability

Claim: For any constant $\rho > 1$, the existence of a polynomial-time ρ-approximation algorithm for (the general) TSP implies P = NP.

Proof

- Let A be a (hypothetical) polynomial-time ρ-approximation algorithm for TSP.
- Let $G = (V, E)$ be an instance of HAM-CYCLE with $|V| = n$.
- Consider the complete graph $G' = (V, E')$ with costs $c(e) = \begin{cases} \frac{1}{n} & \text{if } e \in E, \\ 2\rho & \text{otherwise.} \end{cases}$
- Run A on G'.
- If G contains a Hamiltonian cycle, the optimal TSP tour has cost 1, so A returns a tour of cost $\leqslant \rho$. This tour cannot contain an edge of cost 2ρ. Therefore A returns an optimal TSP tour.
- If G does not contain a Hamiltonian cycle, any TSP tour must use at least one edge of cost $2\rho > 2$.

Linear Programming (LP)

- Let $x_1, x_2, \ldots, x_n \geqslant 0$ be real-valued variables.
- The objective is to minimize/maximize a linear function

$$a_1x_1 + a_2x_2 + \cdots + a_nx_n$$

  subject to a set of linear constraints of the form

$$u_1x_1 + u_2x_2 + \cdots + u_nx_n \lesseqqgtr b,$$

  where $\lesseqqgtr$ is $=$, $\leqslant$ or $\geqslant$.
- Algorithms for solving LP:
  - Simplex method
  - Interior-point method

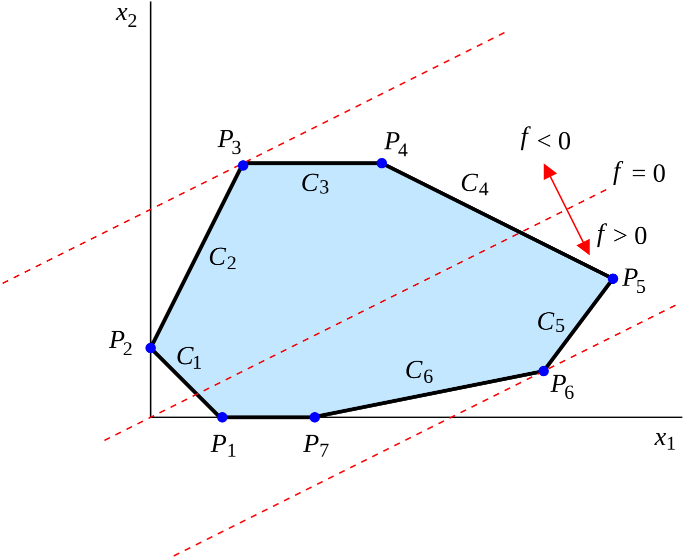
Example

The objective function is $f(x_1, x_2) = x_1 - 2x_2$ with $x_1, x_2 \geqslant 0$.

Six additional constraints:

$$
\begin{aligned}
C_1 &: \quad x_1 + x_2 \geqslant 3, \\
C_2 &: \quad 2x_1 - x_2 \leqslant 3, \\
C_3 &: \quad x_2 \leqslant 11, \\
C_4 &: \quad x_1 + 2x_2 \leqslant 32, \\
C_5 &: \quad 4x_1 - 3x_2 \leqslant 62, \\
C_6 &: \quad x_1 - 5x_2 \leqslant 3.
\end{aligned}
$$

Example



Minimum Vertex Cover

- To find a minimum vertex cover U in $G = (V, E)$.
- Introduce variables x_u for all $u \in V$.

$$x_u = \begin{cases} 1 & \text{if } u \text{ is included in the cover } U, \\ 0 & \text{otherwise.} \end{cases}$$

- Objective: Minimize $\sum_{u \in V} x_u$.
- For each $(u, v) \in E$, add the constraint

$$x_u + x_v \geqslant 1.$$

- Note that x_u are integer/Boolean-valued variables.

Relaxation and Rounding

- Treat x_u as real-valued variable.
- Let $(\bar{x}_u)_{u \in V}$ be a solution of the relaxed LP.
- Take $x_u = \begin{cases} 0 & \text{if } 0 \leqslant \bar{x}_u < 0.5, \\ 1 & \text{if } 0.5 \leqslant \bar{x}_u \leqslant 1. \end{cases}$
- Let $(u, v) \in E$. The constraint $\bar{x}_u + \bar{x}_v \geqslant 1$ implies that either $x_u = 1$ or $x_v = 1$ (or both).
- If $\bar{x}_u < 0.5$, we have $0 = x_u \leqslant 2\bar{x}_u$. If $\bar{x}_u \geqslant 0.5$, we have $1 = x_u \leqslant 2\bar{x}_u$.
- $\sum\limits_{u \in V} x_u \leqslant 2 \sum\limits_{u \in V} \bar{x}_u$.
- Variables x_u^* corresponding to a minimum vertex cover satisfy all the constraints.
- $\sum\limits_{u \in V} \bar{x}_u \leqslant \sum\limits_{u \in V} x_u^*$.
- $\sum\limits_{u \in V} x_u \leqslant 2 \sum\limits_{u \in V} \bar{x}_u \leqslant 2 \sum\limits_{u \in V} x_u^*$, so $\rho \leqslant 2$.

Approximation Algorithms

Polynomial-Time Approximation Schemes

Abhijit Das

Department of Computer Science and Engineering
Indian Institute of Technology Kharagpur

October 11, 2020

Good Approximation Ratios

- Can we achieve $\rho = 1 \pm \varepsilon$ with ε as small as we like?
- In certain cases, we can.
- Running time becomes a function of n and $1/\varepsilon$.
- $\mathrm{O}(n^{1/\varepsilon})$ is polynomial in n if ε is constant, but not so if ε is $1/\log n$ or $1/n$.
- $\mathrm{O}(n^3/\varepsilon^2)$ is polynomial in both n and $1/\varepsilon$.

Definition: Let A be a $(1 \pm \varepsilon)$-approximation algorithm.

- A is called a **polynomial-time approximation scheme** (**PTAS**) if its running time is polynomial in n.
- A is called a **fully polynomial-time approximation scheme** (**FPTAS**) if its running time is polynomial in n and $1/\varepsilon$.

Knapsack Problem

- We have n objects $O_1, O_2, \ldots, O_n$.
- O_i has weight w_i and value (profit) p_i.
- Assume that w_i and p_i are positive integers.
- There is a knapsack of capacity C.
- Goal: To pack a subcollection $O_{i_1}, O_{i_2}, \ldots, O_{i_m}$ of the given objects in the knapsack such that:
  1. the profit $p_{i_1} + p_{i_2} + \cdots + p_{i_m}$ of the packed objects is maximized, and
  2. $w_{i_1} + w_{i_2} + \cdots + w_{i_m} \leqslant C$.

- We may assume that each $w_i \leqslant C$ (discard objects that do not fit individually in the knapsack).
- Obvious greedy strategies "most profitable first" and "maximum profit/weight first" lead to arbitrarily bad solutions.

A Dynamic-Programming Algorithm for KNAPSACK

- Let $P = p_1 + p_2 + \cdots + p_n$. We populate an $n \times P$ table T.
- For $1 \leqslant i \leqslant n$ and $1 \leqslant p \leqslant P$, the entry $T(i,p)$ stores the weight of a lightest subcollection of $O_1, O_2, \ldots, O_i$, whose profit is exactly p.
- If the profit p is not achievable by any subcollection, we store $T(i,p) = \infty$.
- Initialize the first row: $T(1,p) = \begin{cases} w_1 & \text{if } p = p_1, \\ \infty & \text{otherwise.} \end{cases}$
- For $i > 1$, we have $T(i,p) = \begin{cases} T(i-1,p) & \text{if } p_i > p, \\ \min\left(w_i, T(i-1,p)\right) & \text{if } p_i = p, \\ \min\left(w_i + T(i-1,p-p_i), T(i-1,p)\right) & \text{if } p_i < p. \end{cases}$
- The maximum profit is $\max\limits_{1 \leqslant p \leqslant P} \left\{ p \mid T(n,p) \leqslant C \right\}$.

Running Time

- First suppose that the weights and profits are single-precision integers.
- Let $p_{max} = \max(p_1, p_2, \ldots, p_n)$, so $P \leqslant np_{max}$.
- Each entry $T(i, p)$ can be stored $\mathrm{O}(\log n)$ bits/words.
- There are $nP \leqslant n^2 p_{max}$ entries in T.
- The total running time is therefore $\mathrm{O}(n^2 p_{max} \log n)$.

- Now allow p_i to be arbitrarily large.
- If $2^{l-1} \leqslant p_{max} < 2^l$, each profit can be stored using l bits.
- The input size is $\mathrm{O}(nl)$.
- The running time is polynomial in n but exponential in l.

An FPTAS for KNAPSACK

- Take a scaling-down factor σ.
- Consider the scaled-down profits $p'_i = \left\lfloor \frac{p_i}{\sigma} \right\rfloor$.
- Run the dynamic-programming algorithm with the original weights and the scaled-down profits.
- Since the weights are not changed, the capacity constraint is satisfied.
- Suppose that the algorithm returns the scaled-down total profit SOPT'. This is optimal with respect to the scaled-down item profits p'_i.
- We pack the same objects that achieve SOPT' but consider the original profit values of the objects. Call this total profit SOPT.
- Let OPT be the optimal total profit with the original p_i.
- Let OPT' be the scaled-down total profit of the objects that achieve OPT.
- We want $\boxed{\mathrm{SOPT} \geqslant (1-\varepsilon)\mathrm{OPT}.}$

Determination of σ

- $p_i' = \left\lfloor \frac{p_i}{\sigma} \right\rfloor \Rightarrow p_i' \geqslant \frac{p_i}{\sigma} - 1 \Rightarrow \sigma p_i' \geqslant p_i - \sigma \Rightarrow p_i - \sigma p_i' \leqslant \sigma.$
- Sum over all (say, k) objects corresponding to OPT: $\boxed{\mathrm{OPT} - \sigma \mathrm{OPT}' \leqslant k\sigma \leqslant n\sigma.}$
- $p_i' = \left\lfloor \frac{p_i}{\sigma} \right\rfloor \leqslant \frac{p_i}{\sigma} \Rightarrow \sigma p_i' \leqslant p_i.$
- Sum over all objects corresponding to SOPT$'$: $\boxed{\sigma \mathrm{SOPT}' \leqslant \mathrm{SOPT}.}$
- SOPT$'$ is optimal for the scaled-sown profits: $\boxed{\mathrm{SOPT}' \geqslant \mathrm{OPT}'.}$
- We have: $\boxed{\mathrm{SOPT} \geqslant \sigma \mathrm{SOPT}' \geqslant \sigma \mathrm{OPT}' \geqslant \mathrm{OPT} - n\sigma.}$
- We want: $\boxed{\mathrm{SOPT} \geqslant (1-\varepsilon)\mathrm{OPT}.}$
- This is fulfilled by any σ satisfying $\boxed{\sigma \leqslant \dfrac{\varepsilon \times \mathrm{OPT}}{n}.}$
- Since $p_{max} \leqslant \mathrm{OPT}$, we take $\boxed{\sigma = \dfrac{\varepsilon \times p_{max}}{n}.}$

Running Time

- The dynamic-programming algorithm with scaled-down profits runs in $O(n^2 p'_{max} \log n)$ time.
- $p'_{max} = \left\lfloor \frac{p_{max}}{\sigma} \right\rfloor \leqslant \frac{p_{max}}{\sigma} = \frac{n}{\varepsilon}$.
- So the running time is $O\left(\frac{n^3 \log n}{\varepsilon}\right)$.
- This is polynomial in both n and $1/\varepsilon$.
- So this is an FPTAS for the knapsack problem.