CMPSCI 611: Advanced Algorithms

Lecture 21: Reductions and Finishing Approximation Algorithms

Andrew McGregor

Last Compiled: February 1, 2024

Outline

Finishing Approximation Algorithms

Polynomial Time Reductions

PTAS for Knapsack Problem

General Knapsack Problem:

1. Input: A set of items numbered $1, 2, \ldots, n$, where each the i-th item has weight w_i and value v_i. C is the capacity of your knapsack. (Assume each $w_i \leq C$.)

PTAS for Knapsack Problem

General Knapsack Problem:

1. Input: A set of items numbered $1, 2, \ldots, n$, where each the i-th item has weight w_i and value v_i. C is the capacity of your knapsack. (Assume each $w_i \leq C$.)
2. Goal: Find a subset B of the items with maximum total value subject to $\sum_{i \in B} w_i \leq C$.

PTAS for Knapsack Problem

General Knapsack Problem:

1. Input: A set of items numbered $1, 2, \ldots, n$, where each the i-th item has weight w_i and value v_i. C is the capacity of your knapsack. (Assume each $w_i \leq C$.)
2. Goal: Find a subset B of the items with maximum total value subject to $\sum_{i \in B} w_i \leq C$.

Last time we showed:

PTAS for Knapsack Problem

General Knapsack Problem:

1. Input: A set of items numbered $1, 2, \ldots, n$, where each the i-th item has weight w_i and value v_i. C is the capacity of your knapsack. (Assume each $w_i \leq C$.)
2. Goal: Find a subset B of the items with maximum total value subject to $\sum_{i \in B} w_i \leq C$.

Last time we showed:

1. Dynamic programming algorithm taking $O(n^2 V)$ time where $V = \max_i v_i$.

PTAS for Knapsack Problem

General Knapsack Problem:

1. Input: A set of items numbered $1, 2, \ldots, n$, where each the i-th item has weight w_i and value v_i. C is the capacity of your knapsack. (Assume each $w_i \leq C$.)
2. Goal: Find a subset B of the items with maximum total value subject to $\sum_{i \in B} w_i \leq C$.

Last time we showed:

1. Dynamic programming algorithm taking $O(n^2 V)$ time where $V = \max_i v_i$.
2. Define v_i' by setting k lowest order bits of v_i to zero. Use the dynamic program to find the best set B' with respect to the values v_i'. Then,
$$\frac{\sum_{i \in B} v_i}{\sum_{i \in B'} v_i} \leq 1 + \frac{n2^k}{V - n2^k}$$
where B be the optimal set.

Finishing off Analysis

Claim

If $k \leq \log(\epsilon V/(2n))$ *and* $\epsilon \leq 1$ *then* $1 + \frac{2^k n}{V - 2^k n} \leq 1 + \epsilon$.

Finishing off Analysis

Claim

If $k \leq \log(\epsilon V/(2n))$ *and* $\epsilon \leq 1$ *then* $1 + \frac{2^k n}{V - 2^k n} \leq 1 + \epsilon$.

1. Let $k = \lfloor \log(\epsilon V/(2n)) \rfloor$

Finishing off Analysis

Claim

If $k \leq \log(\epsilon V/(2n))$ *and* $\epsilon \leq 1$ *then* $1 + \frac{2^k n}{V - 2^k n} \leq 1 + \epsilon$*.*

1. Let $k = \lfloor \log(\epsilon V/(2n)) \rfloor$
2. Solve for v' by solving for another set of values v'' where

$$v_i'' = v_i'/2^k$$

Finishing off Analysis

Claim

If $k \leq \log(\epsilon V/(2n))$ *and* $\epsilon \leq 1$ *then* $1 + \frac{2^k n}{V - 2^k n} \leq 1 + \epsilon$.

1. Let $k = \lfloor \log(\epsilon V/(2n)) \rfloor$
2. Solve for v' by solving for another set of values v'' where
$$v_i'' = v_i'/2^k$$
3. The maximum value for v'' satisfies:
$$\max v_i'' \leq V/2^k \leq 2V/(\epsilon V/(2n)) = 4n/\epsilon$$
so the run time is $O(n^3/\epsilon)$

Summary of Approximation Algorithms

- Algorithms:
  - 2-approximation for vertex cover
  - 2-approximation for max-cut
  - 3/2-approximation for metric traveling salesperson
  - $O(\log n)$-approximation for weighted set-cover
  - FPTAS for knapsack

Summary of Approximation Algorithms

- Algorithms:
  - 2-approximation for vertex cover
  - 2-approximation for max-cut
  - 3/2-approximation for metric traveling salesperson
  - $O(\log n)$-approximation for weighted set-cover
  - FPTAS for knapsack
- For a reference of what approximation factors are known check out:

  `http://www.csc.kth.se/~viggo/wwwcompendium/`

Alternative Approaches to “hard” problems

- Restrict the input:
  - Assuming input graph is acyclic, has bounded degree, is planar
  - Solving metric TSP where the points are in Euclidean space
- Assume a probability distribution over input: *Average case analysis*
- Assume all integers in the input are polynomial in the input size…

Alternative Approaches to “hard” problems

- Restrict the input:
  - Assuming input graph is acyclic, has bounded degree, is planar
  - Solving metric TSP where the points are in Euclidean space
- Assume a probability distribution over input: *Average case analysis*
- Assume all integers in the input are polynomial in the input size. . .

Definition

An algorithm runs in *pseudo-polynomial time* if the running time is polynomial in the input size and any integer in the input.

Outline

Problem 1: Clique

Definition

A clique of size k in a graph G is a completely connected subgraph of G with k vertices.

Problem 1: Clique

Definition

A clique of size k in a graph G is a completely connected subgraph of G with k vertices.

- Input: Given graph $G = (V, E)$ and integer k.

Problem 1: Clique

Definition

A clique of size k in a graph G is a completely connected subgraph of G with k vertices.

- Input: Given graph $G = (V, E)$ and integer k.
- Question: Does G contain a clique of size k?

Problem 2: 3-SAT

- Input: A boolean formula $\phi(x_1, \ldots, x_n)$ in *conjunctive normal form* with m clauses and 3 literals per clause, e.g.,

$$(x_1 \vee \bar{x_2} \vee x_3) \wedge (\bar{x_1} \vee x_2 \vee \bar{x_3})$$

where $\bar{x}_i$ is "not x_i", $\wedge$ is "and", $\vee$ is "or." We call x_i and $\bar{x}_i$ *literals*.

Problem 2: 3-SAT

- Input: A boolean formula $\phi(x_1, \ldots, x_n)$ in *conjunctive normal form* with m clauses and 3 literals per clause, e.g.,

$$(x_1 \vee \bar{x}_2 \vee x_3) \wedge (\bar{x}_1 \vee x_2 \vee \bar{x}_3)$$

  where $\bar{x}_i$ is "not x_i", $\wedge$ is "and", $\vee$ is "or." We call x_i and $\bar{x}_i$ *literals*.
- Question: Is there a setting of each x_i to TRUE or FALSE such that the formula is satisfied.

A Polynomial Time Reduction for 3-SAT to Clique

We'll show that if you have a polynomial time algorithm for Clique, then you also have a polynomial time algorithm for 3-SAT.

A Polynomial Time Reduction for 3-SAT to Clique

We'll show that if you have a polynomial time algorithm for Clique, then you also have a polynomial time algorithm for 3-SAT.

Given formula 3-SAT

$$\phi = (l_{1,1} \vee l_{1,2} \vee l_{1,3}) \wedge (l_{2,1} \vee l_{2,2} \vee l_{2,3}) \wedge \ldots \wedge (l_{m,1} \vee l_{m,2} \vee l_{m,3})$$

in poly-time, we can construct $G_\phi = (V_\phi, E_\phi)$:

$$V_\phi = \{l_{i,j} : i \in [m], j \in [3]\}$$

$$E_\phi = \{(l_{i,j}, l_{k,l}) : i, k \in [m], j \in [3], i \neq k, l_{i,j} \neq \bar{l}_{k,l}\}$$

A Polynomial Time Reduction for 3-SAT to Clique

We'll show that if you have a polynomial time algorithm for Clique, then you also have a polynomial time algorithm for 3-SAT.

Given formula 3-SAT

$$\phi = (l_{1,1} \vee l_{1,2} \vee l_{1,3}) \wedge (l_{2,1} \vee l_{2,2} \vee l_{2,3}) \wedge \ldots \wedge (l_{m,1} \vee l_{m,2} \vee l_{m,3})$$

in poly-time, we can construct $G_\phi = (V_\phi, E_\phi)$:

$$V_\phi = \{l_{i,j} : i \in [m], j \in [3]\}$$

$$E_\phi = \{(l_{i,j}, l_{k,l}) : i, k \in [m], j \in [3], i \neq k, l_{i,j} \neq \bar{l}_{k,l}\}$$

We'll show ϕ is satisfiable iff G_ϕ has a clique of size m

ϕ is satisfiable iff G_ϕ has a clique of size m

Suppose ϕ is satisfiable:

ϕ is satisfiable iff G_ϕ has a clique of size m

Suppose ϕ is satisfiable:

1. In a satisfying assignment, at least one literal is true in each clause

ϕ is satisfiable iff G_ϕ has a clique of size m

Suppose ϕ is satisfiable:

1. In a satisfying assignment, at least one literal is true in each clause
2. Pick one true literal per clause: let Y be set of corresponding nodes

ϕ is satisfiable iff G_ϕ has a clique of size m

Suppose ϕ is satisfiable:

1. In a satisfying assignment, at least one literal is true in each clause
2. Pick one true literal per clause: let Y be set of corresponding nodes
3. $G_\phi[Y]$ is a clique because x_k and $\bar{x}_k$ can't both be in Y for any k

ϕ is satisfiable iff G_ϕ has a clique of size m

Suppose ϕ is satisfiable:

1. In a satisfying assignment, at least one literal is true in each clause
2. Pick one true literal per clause: let Y be set of corresponding nodes
3. $G_\phi[Y]$ is a clique because x_k and $\bar{x}_k$ can't both be in Y for any k

Suppose G_ϕ has a clique of size m:

1. Let Y be the clique of size m

ϕ is satisfiable iff G_ϕ has a clique of size m

Suppose ϕ is satisfiable:

1. In a satisfying assignment, at least one literal is true in each clause
2. Pick one true literal per clause: let Y be set of corresponding nodes
3. $G_\phi[Y]$ is a clique because x_k and $\bar{x}_k$ can't both be in Y for any k

Suppose G_ϕ has a clique of size m:

1. Let Y be the clique of size m
2. For each clause:
   - Exactly one node l from i-th clause is in Y

ϕ is satisfiable iff G_ϕ has a clique of size m

Suppose ϕ is satisfiable:

1. In a satisfying assignment, at least one literal is true in each clause
2. Pick one true literal per clause: let Y be set of corresponding nodes
3. $G_\phi[Y]$ is a clique because x_k and $\bar{x}_k$ can't both be in Y for any k

Suppose G_ϕ has a clique of size m:

1. Let Y be the clique of size m
2. For each clause:
   - Exactly one node l from i-th clause is in Y
   - Set $x_k = \text{TRUE}$ if $l = x_k$ and set $x_k = \text{FALSE}$ if $l = \bar{x}_k$
3. We can't set x_k to be true and false because literals x_k and $\bar{x}_k$ can't both be in Y