University Nice Sophia-Antipolis — Resolution methods
Master 1 international + Data Science — October 2015

Exercises : Approximation algorithms

To be returned for November 27th 2015

The goal of the following problems is to analyze approximation algorithms for two problem : Maximum Cut Problem and Knapsack Problem. The two problems are independent.

Recall that, for any $c \geq 1$, a c-approximation algorithm for a maximization problem is an algorithm that computes **in polynomial-time** a feasible solution such that

$$OPT/c \leq value(solution) \leq OPT$$

where OPT is the optimal value.

1 Maximum Cut Problem

Notations : In this section, n will always denote the number of vertices of a graph and m will denote the number of its edges.

Let $G = (V, E)$ be a graph. A *cut* in G is a partition of V into two sets. Let $S \subseteq V$ be a subset of vertices. The *cost of the cut* $(S, V \setminus S)$, denoted by $cost(S)$, equals the number of edges between S and $V \setminus S$, i.e., the size of the set $\{\{x, y\} \in E \mid x \in S, y \in V \setminus S\}$.

The Maximum Cut Problem takes a graph $G = (V, E)$ as input and the objective is to find a cut with maximum cost.

Question 1 *Let $G = (A \cup B, E)$ be a bipartite graph (i.e., A and B are stable sets). Give a maximum cut of G. What is its cost ? (prove that the solution is optimal)*

Question 2 *Give an exponential-time algorithm that computes a maximum cut in arbitrary graphs. Prove that its time-complexity is $O(m \cdot 2^n)$.*

The Maximum Cut Problem is NP-hard, meaning that it does not admit a polynomial-time algorithm unless $P = NP$. The goal of next questions is to analyze an approximation algorithm for it.

Definition : Let $(S, V \setminus S)$ be a cut. A vertex v is *movable* if
- either $v \in S$ and $cost(S) < cost(S \setminus \{v\})$;
- or $v \in V \setminus S$ and $cost(S) < cost(S \cup \{v\})$.

That is, v is movable if moving v on the other side strictly increases the cost of the cut.

Algorithm 1 2-approximation algorithm for Maximum Cut

Require: A graph $G = (V, E)$
Ensure: A cut $(S, V \setminus S)$
$S = \emptyset$
while There is a movable vertex v **do**
  Move the vertex v on the other side, that is :
  – if $v \in S$ then replace S by $S \setminus \{v\}$
  – if $v \in V \setminus S$ then replace S by $S \cup \{v\}$
return S

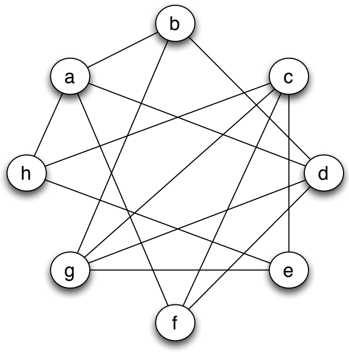

FIGURE 1 – A graph with 8 nodes and 14 edges

Question 3 *Apply Algorithm 1 on the example depicted in Figure 1*

Question 4 *What is the maximum number of iterations of the While loop of Algorithm 1 ?*

Let us assume that checking if a vertex is movable can be done in constant time. What is the order of magnitude of the time-complexity of Algorithm 1 ?

Notation : Let $G = (V, E)$ be a graph, $v \in V$ and $X \subseteq V$. Let $deg_X(v)$ denote the degree of v in X, that is the size of the set $\{w \in X \mid \{v, w\} \in E\}$. Let $d(v)$ denote the (classical) degree of v, i.e., $d(v) = deg_V(v)$.

Question 5 *Let $(S, V \setminus S)$ be a solution computed by the algorithm.*

1. *Let $v \in S$. Show that $deg_S(v) \leq \lfloor d(v)/2 \rfloor$.*
2. *Similarly, show that $deg_{V \setminus S}(v) \leq \lfloor d(v)/2 \rfloor$ for any $v \in V \setminus S$.*

Question 6 *Let $(S, V \setminus S)$ be a solution computed by the algorithm. Let $X = \{\{u, v\} \in E \mid u, v \in S\}$ be the set of edges between nodes in S. Let $Y = \{\{x, y\} \in E \mid x, y \notin S\}$ be the set of edges between nodes in $V \setminus S$. Let $Z = E \setminus (X \cup Y)$ be the set of edges between S and $V \setminus S$.*

1. *By summing the degree of the nodes in S, and using previous question, show that $2|X| \leq |Z|$.*
2. *Similarly, show that $2|Y| \leq |Z|$.*
3. *Deduce that $|Z| \geq |E|/2$.*

Question 7 *Prove that Algorithm 1 is a 2-approximation algorithm for the* MAXIMUM CUT *problem.*

2 KNAPSACK Problem

The SIMPLE KNAPSACK problem takes a set of integers $\mathcal{S} = \{w_1, \cdots, w_n\}$ and an integer b as inputs. The objective is to compute a subset $T \subseteq \{1, \cdots, n\}$ of items such that $\sum_{i \in T} w_i \leq b$ and $\sum_{i \in T} w_i$ is maximum. That is, we want to fill our knapsack without exceeding its capacity b and putting the maximum total weight in it.

2.1 Exact Algorithm via dynamic programming

Dynamic programming is a generic algorithmic method that consists in solving a problem by combining the solutions of sub-problems.

As an example, the SIMPLE KNAPSACK Problem consists in computing an optimal solution for an instance $\mathcal{S} = \{w_1, \cdots, w_n\}$ and an integer b. Let $OPT(\mathcal{S}, b)$ denote such a solution. We will compute it using solutions for sub-problems with inputs $\mathcal{S}_i = \{w_1, \cdots, w_i\}$ and $b' \in \mathbb{N}$, for any $i \leq n$ and $b' < b$. That is, we will compute $OPT(\mathcal{S}, b)$ from all solutions $OPT(\mathcal{S}_i, b')$ for $i \leq n$ and $b' < b$.

Algorithm 2 Dynamic programming algorithm for SIMPLE KNAPSACK

Require: A set of integers $\mathcal{S} = \{w_1, \cdots, w_n\}$ and $b \in \mathbb{N}$.
Ensure: A subset $OPT \subseteq \{1, \cdots, n\}$ of items such that $\sum_{i \in T} w_i \leq b$

For any $0 \leq i \leq n$ and any $0 \leq b' \leq b$, let $OPT[i, b'] = \emptyset$;
For any $0 \leq i \leq n$ and any $0 \leq b' \leq b$, let $opt\_cost[i, b'] = 0$;
for $i = 1$ to n **do**
    for $b' = 1$ to b **do**
        if $w_i \leq b'$ and $opt\_cost[i-1, b'-w_i] + w_i > opt\_cost[i-1, b']$ **then**
            $OPT[i, b'] = OPT[i-1, b'-w_i] \cup \{i\}$
            $opt\_cost[i, b'] = opt\_cost[i-1, b'-w_i] + w_i$
        else
            $OPT[i, b'] = OPT[i-1, b']$
            $opt\_cost[i, b'] = opt\_cost[i-1, b']$
return $OPT = OPT[n, b]$

Question 8 *Prove that Algorithm 2 has time-complexity $O(n \cdot b)$.*

Question 9 *Prove that Algorithm 2 proceed in polynomial-time if $\max_i w_i$ is polynomial in n but exponential if $\max_i w_i$ is exponential in n.*

Actually, the KNAPSACK Problem is an example of *Weakly NP-hard* (roughly, it can be solved in polynomial-time if the weights are polynomial).

Question 10 *Prove by induction on i and b' that the solution $OPT = OPT[n, b]$ returned by Algorithm 2 is optimal.*

2.2 Approximation Algorithm and PTAS

Algorithm 3 Greedy algorithm for SIMPLE KNAPSACK

Require: A set of integers $\mathcal{S} = \{w_1, \cdots, w_n\}$ and $b \in \mathbb{N}$.
Ensure: A subset $T \subseteq \{1, \cdots, n\}$ of items such that $\sum_{i \in T} w_i \leq b$

$T = \emptyset$
$total\_weight = 0$
Sort $\mathcal{S}$. Let us assume that $w_1 \geq w_2 \geq \cdots \geq w_n$.
for $i = 1$ to n **do**
    if $total\_weight + w_i \leq b$ **then**
        Add i to T
        Add w_i to $total\_weight$
return T

Question 11 *What is the time-complexity of Algorithm 3 ?*

Question 12 *Prove that Algorithm 3 is a* 2*-approximation algorithm for the* Simple Knapsack *problem.*

hint : let T be the computed solution and assume it is not optimal. Let $j \geq 1$ be the smallest integer such that $i+1$ is NOT in T. Show that $w_{j+1} \leq b/j$.

A **polynomial-time approximation scheme (PTAS)** is an algorithm which takes an instance of an optimization problem and a parameter $\epsilon > 0$ and, in polynomial time in the size of the instance (not necessarily in ϵ), produces a solution that is within a factor $1+\epsilon$ of being optimal.

That is, when ϵ tends to 0, the solution tends to an optimal one, while the complexity increases (generally, the complexity is of the form $O(n^{1/\epsilon})$).

Algorithm 4 PTAS for Simple Knapsack

Require: A set of integers $\mathcal{S} = \{w_1, \cdots, w_n\}$, $b \in \mathbb{N}$ and a real $\epsilon > 0$.
Ensure: A subset $T \subseteq \{1, \cdots, n\}$ of items such that $\sum_{i \in T} w_i \leq b$

$best = \emptyset$
$best\_cost = 0$
$k = \lceil 1/\epsilon \rceil$
for Any subset $X \subseteq \mathcal{S}$ of size k **do**
  Complete X using the Greedy Algorithm. That is :
  $T = X$
  $total\_weight = \sum_{i \in X} w_i$
  Sort $\mathcal{S} \setminus X$. Let us assume that $\mathcal{S} \setminus X = \{w_1, \cdots, w_{n-k}\}$ and $w_1 \geq w_2 \geq \cdots \geq w_{n-k}$.
  for $i = 1$ to $n-k$ **do**
    if $total\_weight + w_i \leq b$ **then**
      Add i to T
      Add w_i to $total\_weight$
  if $total\_weight > best\_cost$ **then**
    Replace $best$ by T
return T

Question 13 *Prove that Algorithm 4 has time-complexity $O(n^{\lceil 1/\epsilon \rceil + 1})$.*

Question 14 *Prove that Algorithm 4 is a* $(1+\epsilon)$*-approximation algorithm for the* Simple Knapsack *problem.*

hint : Consider an optimal solution M and let $X^* = \{i_1, \cdots, i_k\}$ be the k items with largest weight in M. Consider the iteration of Algorithm 4 when it considers X^*.

Actually, we can do better. Indeed, the Knapsack Problem admits a **fully polynomial-time approximation scheme (FPTAS)** algorithm, that is an algorithm that computes a solution that is within a factor $1+\epsilon$ of being optimal **in time polynomial both in the size of the instance AND in** $1/\epsilon$.