Approximation Algorithms

CSCI 3100

Introduction

In general, computer cannot solve NPC problem efficiently

But, many NPC problems are too important to abandon

If a problem is an NPC problem, you may try to

- find a pseudo polynomial time algorithm if it is not NPC in the strong sense
  - Pseudo-polynomial: polynomial in the numeric value of the input
  - Subset sum problem: O(N*K)
- solve restricted problems
- find approximation algorithms
- Use heuristic based methods

Approximation Algorithms

Let's consider optimization problems only.

An algorithm A is an approximation for a problem L : if given any valid instance I, it finds a solution A(I) for L and A(I) is "close" to optimal solution OPT(I).

3

Approximation Ratio

Approximation ration (or bound) β of approximation algorithm A for problem L.

$A(I)/OPT(I) \leq \beta$; if L is a minimization problem and $A(I) \geq OPT(I) > 0$

$OPT(I)/A(I) \leq \beta$; if L is a maximization problem and $OPT(I) \geq A(I) > 0$

Example:

- Suppose we have an "approximation algorithm for the max-clique" problem.
- Given a graph G, the algorithm finds a clique of size M.
- If our approximation algorithm has approximation ratio β, we know that the optimal solution is no bigger than β *M
- Suppose β is 2 and M is 10.
- Then we know that the optimal solution is no bigger than 20.

4

Maximum Programs Stored (PS) Problem

Optimization PS Problem: Given a set of n program and two storage devices. Let s_i be the amount of storage needed to store the i^{th} program. Let L be the storage capacity of each disk. Determine the maximum number of these n programs that can be stores on the two disks (without splitting a program over the disks).

Example : L = 10 S =(2, 4, 5, 6) How many programs (from the set S) can we store on two disks of size 10?

A. 1

B. 2

C. 3

D. 4

The decision PS problem is NPC

Approximation PS Algorithm

// assume programs are sorted in nondecreasing order of program size,

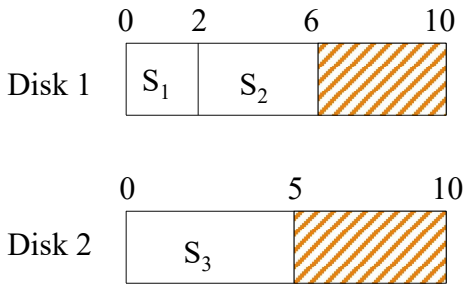
// i.e. $s_1 \le s_2 \le \ldots \le s_n$.

// Put as many programs as you can in the 1st disk, then go to the 2nd disk.

```
i = 0; c=0; // c count the number of stored program
for j = 1 to 2  {
      sum = 0
      while ( sum + s_i ≤ L ) {
         store i^th program into j^th device
         sum += s_i
         i++;  c++
         if i > n return
      }
}
```

Run time:

Example : L = 10 S_i =(2, 4, 5, 6)



Approximation Ratio

Let C^* be the optimal (maximum) number of programs that can be stores on the two disks.

The above approximation PS algorithm gives very good approximation ratio.

$C^* <= (C + 1)$ OR $C^* / C <= 1 + 1/C$

i.e. the given program stores

at most 1 program less than

the optimal solution

The above approximation algorithm returns value C such that $C^* \leq C+1$, where C^* is the optimal solution.

Our example showed that there exists a set S and value L such that $C^* = C+1$.

Need to show that $\forall\{s_1, s_2, \ldots, s_n\}$ and L, $C^* \leq (C + 1)$

Let's consider only one disk with capacity 2L.

Greedy proof: We can store maximum number of programs into the disk by considering programs in the order of

$s_1 \leq s_2 \leq \ldots \leq s_n$

Let α be the maximum number of programs that are stored in the disk

Clearly $\alpha \geq C^*$ and $s_1 + s_2 + \ldots + s_\alpha \leq 2L$ (i)

Let j be an index such that

$(s_1 + s_2 + \ldots + s_j) \leq L$ and

$(s_1 + s_2 + \ldots + s_{j+1}) > L$ (ii)

$j \leq \alpha$ and j programs are stored in the 1st disk by the above approximation algorithm

By (i) & (ii), $(s_{j+2} + s_{j+3} + \ldots + s_\alpha) \leq L$

→ → $(s_{j+1} + s_{j+2} + \ldots + s_{\alpha-1}) \leq L$

→ at least $(j+1)^{th}$ program, $(j+2)^{th}$ program, …, $(\alpha-1)^{th}$ program are stored in 2nd disk by the above approximation alg. **Done!**

11

Travelling Salesman Problem

Given

- a set of cities/clients and travel duration between cities/clients

Find

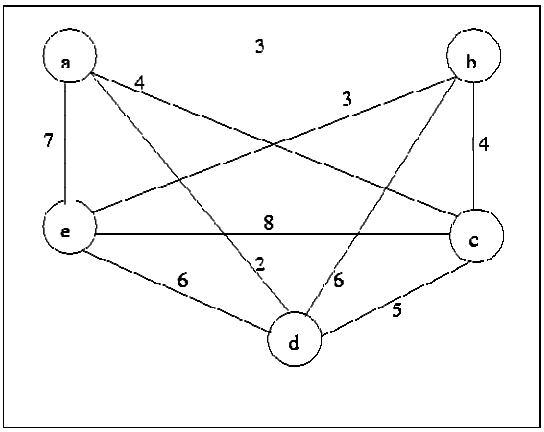
- A route that visits all cities/clients as quickly as possible, returning to the starting point

Translate this to a graph problem:

- Given a graph G = (V, E), where V is a set of cities/clients and E is a weighted set of edges/roads connecting cities/clients and a vertex s in V,
- Find a cycle that starts at s and visits each vertex such that the edge weights of this cycle are minimized
- Edges can be repeated

12

Example: start at vertex a



Approximation Algorithm

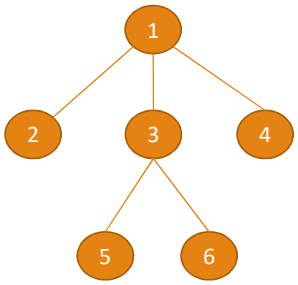
Let s be the starting point

Construct MST starting at s

List vertices of the MST in the order visited by "pre-order" walk:

```
pre-order-walk(T, root)
visit root
for (each vertex v adjacent to root)
   pre-order-walk(Subtree of T starting at v, v)
   visit v
```

Pre-order walk starting at 1



The above approximation algorithm for TSP has an approximation factor 2

Let O be the cost of the optimal solution

We know that O ≥ weight (MST). Which of the following statements is true?

A. weight (path produced by algorithm) ≤ 2 * weight (MST) ≤ 2 * O

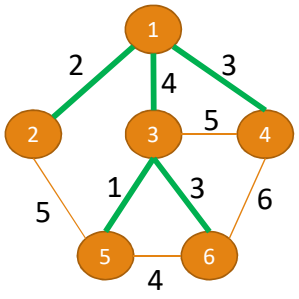
B. O < 2 * weight(MST)

C. weight(path produced by algorithm) < O

D. 2*O < weight(path produced by algorithm)

E. None of the above

Example: starting at vertex 1, what is the weight of the solution produced by our TSP approximation algorithm.



A. 26

B. 13

C. 5

D. 20