Approximation Algorithms

Nandini Mukherjee

Department of Computer Science and Engineering

Optimization Problems

- The goal is to find the best solution among a collection of possible solution
- When the problem is NP-complete, no polynomial time algorithm exists for it unless P=NP
- In practice, we may not need the absolute best or optimal solution
- A nearly optimal solution may be good enough and may be much easier to find
- An algorithm that returns a near-optimal solution is called an *approximation algorithm*

For an optimization problem we define a positive cost for each potential solution and we wish to either minimize or maximize the cost

Approximation Ratio

- An algorithm for a problem has an approximation ratio of $\rho(n)$ if for any input of size n, the cost C of the solution produced by the algorithm is within a factor of $\rho(n)$ of the cost C^* of an optimal solution

$max\ (C/C^*, C^*/C) \le \rho(n)$

The algorithm that achieves an approximation ratio of $\rho(n)$ is called a *$\rho(n)$-approximation algorithm*

For a maximization problem, $0 < C \le C^*$

For a minimization problem, $0 < C^* \le C$

Approximation ratio is never less than 1

A 1-approximation algorithm produces an optimal solution

Approximation Scheme

- Many problems have approximation algorithms with small constant approximation ratios
- For some problems, the best known polynomial-time approximation algorithms have approximation ratios that grow as functions of the input size *n*
- Some NP-complete problems allow polynomial time approximation algorithms that can achieve increasingly smaller approximation ratios by using more and more computation time
- For certain problems we can construct *(1+ε)-approximation algorithms* for any fixed value ε
- Running time of such algorithms depend on both, *n* (size of its input) and the fixed value ε

Approximation Scheme

- Collection of such algorithms is called *polynomial-time approximation scheme*

More formally

An approximation scheme for an optimization problem is an approximation algorithm that takes as input an instance of the problem and a value $\varepsilon > 0$ such that for any fixed ε, the scheme is a $(1+\varepsilon)$-approximation algorithm

The approximation scheme is a polynomial-time approximation scheme if for any fixed ε, the scheme runs in polynomial time in n (size of its input instance)

- When we have a *polynomial-time approximation scheme* for a given optimization problem, we can tune our performance guarantee based on how much time we can afford to spend
- Ideally, the running time is polynomial in both n and $1/\varepsilon$, in which case we have a *fully polynomial-time approximation algorithm*

Approximation Scheme

A polynomial time approximation scheme (PTAS) is an algorithm that takes as input not only an instance of the problem but also a value $\epsilon > 0$ and approximates the optimal solution to within a ratio bound $1 + \epsilon$. For any choice of ϵ, the algorithm has a running time that is polynomial in *n*, the size of the input.

Example: a PTAS may have a running time bound of $O(n^{2/\epsilon})$

A fully polynomial-time approximation scheme (FPTAS) is a PTAS with a running time that is polynomial not only in n, but also in $1/\epsilon$.

Example: a PTAS with a running time bound of $O((1/\epsilon)^2 n^3)$ is an FPTAS.

The vertex-cover problem

- *A vertex-cover of an undirected graph G=(V, E) is a subset V' ⊆ V, such that if (u,v) is an edge of G, then either u ∈ V' or v ∈ V' (or both). The size of a vertex cover is the number of vertices in it*
- The problem is to find a vertex-cover of *minimum size* in a given undirected graph – an NP-complete decision problem
- The approximation algorithm:

```
APPROX-VERTEX-COVER(G)
    C ← 0
    E' ← E[G]
    while E' ≠ 0
        do let (u,v) be an arbitrary edge of E'
        C ← C ∪ {u,v}
        remove from E' every edge that is incident on either u or v
    return C
```

The vertex-cover problem

- Running time of this algorithm is *O(V+E)* using adjacency lists to represent E'

Theorem: APPROX-VERTEX-COVER is polynomial-time **2-approximation algorithm**

Proof:

Step-1: Prove that APPROX-VERTEX-COVER runs in polynomial time

You can prove it by simply analysing the algorithm

Running time of this algorithm is *O(V+E)* using adjacency lists to represent E'

Step-2: Prove that Set C is a vertex cover

Set *C* is a vertex cover, since the algorithm loops until every edge in *E[G]* has been covered by some vertex in *C*

The vertex-cover problem

Step-3:

Let *A* denotes the set of edges that were picked in the first step of the while loop of APPROX-VERTEX-COVER

In order to cover the edges in *A,* any vertex cover must include at least one endpoint of each edge in *A*

No two edges in *A* share an endpoint - why?

Thus, $|C^*| \geq |A|$

Each execution of the first step of the while loop picks an edge for which neither of its endpoints is already in *C*, yielding an upper bound on the size of the vertex cover returned

Thus, $|C| = 2|A|$

Combining the two equations, $|C| = 2|A| \leq 2|C^*|$ -- Hence proved.

The Traveling-salesman problem

- Given a complete undirected graph $G=(V,E)$ that has a nonnegative integer cost $c(u,v)$ associated with each edge $(u,v) \in E$, goal is to find a Hamiltonian cycle of G with minimum cost.
- We shall use the concept of triangle inequality, i.e. for all vertices $u, v, w \in V$,

  $c(u,w) \leq c(u,v) + c(v,w)$
- This is satisfied if we assume that the vertices are points in the plane and the cost of traveling is the ordinary Euclidean distance between them
- The problem even with the triangle inequality remains NP-complete

The Traveling-salesman problem with the triangle inequality

A 2-approximation algorithm for traveling salesman

APPROX-TSP-TOUR(*G,c)*

select a vertex $r \in V[G]$ to be a "root" vertex

compute a minimum spanning tree *T* for *G* from root *r* using *MST-PRIM(G,c,r)*

let *L* be the list of vertices visited in a preorder tree walk of *T*

return the Hamiltonian cycle *H* that visits the vertices in the order *L*

- The running time of APPROX-TSP-TOUR is $\Theta(V^2)$

The Traveling-salesman problem with the triangle inequality

Theorem: APPROX-TSP-TOUR is a polynomial-time 2-approximation algorithm for the traveling salesman problem with triangle inequality

Proof: APPROX-TSP-TOUR runs in polynomial time

Let H^* denotes an optimal tour for the given set of vertices

If T is the minimum spanning tree, $c(T) \leq c(H^*)$

[$c(A)$ denotes the total cost of the edges in the subset $A \subseteq E$]

A full walk W of the tree lists the vertices when they are first visited and also whenever they are returned to after a visit to a subtree

Thus, full walk traverses every edge of T exactly twice

Therefore, $c(W) = 2c(T)$

From above we get, $c(W) \leq 2c(H^*)$

However, W is not a tour, since it visits some vertices more than once

The Traveling-salesman problem with the triangle inequality

By triangle inequality, we delete a visit to any vertex from *W* and the cost does not increase

(we directly move from one vertex to the other)

By repeatedly applying this operation, we can remove from W all but the first visit to each vertex

Ordering of these vertices is same as that obtained by a preorder walk of the tree *T.*

Let *H* be the cycle corresponding to this preorder walk – this is a Hamiltonian cycle

We have, $c(H) \leq c(W)$

Thus, $c(H) \leq 2c(H^*)$

Hence Proved.