Approximation Algorithms

Suppose you just discovered a problem you'd like to solve is NPC...what should you do?

- ***hope/show the bad running time doesn't happen for the inputs you will use.***
- ***find heuristics to quickly find correct solutions in many cases.***
- ***develop an approximation algorithm that will find a solution that is close to optimal.***

An algorithm 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.

The vertex cover approximation algorithm

Let G be an undirected graph. A ***vertex cover*** of G is a subset COVER of V such that for every (u, v) E, at least one of u or v COVER.

The vertex cover optimization problem is to find a vertex cover of *minimum* size = *optimal vertex cover*.

While finding the optimal vertex cover is an NPC problem, there is a p-time approximation algorithm that returns a near-optimal vertex cover.

The vertex cover approximation algorithm

```
APPROX-VERTEX COVER (G)
1.  C =
2.  E' = E[G]
3.  while E' ≠
4.        let  (u,v) be an arbitrary edge of E'
5.        C = C     {u,v}
6.        remove from E' every edge incident on either u or v
7.  return C
```

Running time = ??

The vertex cover approximation algorithm

***Theorem*: APPROX-VERTEX-COVER is a p-time 2-approximation algorithm.**

***Proof*: The algorithm runs in O(V + E) time.**

The algorithm clearly returns a vertex cover, since it loops until all edges have been removed. Each edge removed in line 6 was covered by some vertex of an edge removed in line 4.

Let A denote the set of edges removed in line 4:

- **The optimal cover C* must contain at least one endpoint of each of the edges in A.**
- **No two edges in A share an endpoint because all edges that share an endpoint with an edge in A are removed in line 6.**

The vertex cover approximation algorithm

Since no two edges in A are covered by the same vertex in C*, we have that

$$|C^*| \geq |A| \quad (1)$$

is a lower bound on the size of C*.

Each execution of line 4 picks an edge, both of whose endpoints are added to the approximate cover C. This gives us an upper bound on the size of C:

$$|C| = 2\,|A| \quad (2)$$

Combining equations (1) and (2), we get

$$|C| \leq 2\,|C^*|$$

■

Approximation Algorithm for Restricted TSP

- **instance:** finite set of cities and distances between them
- **feasible solutions:** All possible permutations of cities
- **value of a solution:** length of the tour for that solution
- **approximation algorithm:** returns ***some*** feasible solution

Restrictive assumption: All inter-city distances satisfy the triangle inequality, i.e., shortest distance between 2 cities is a direct route. I.e., $d(c_1, c_2) \leq d(c_1, c_3) + d(c_3, c_2)$

Approximation Algorithm for Restricted TSP

APPROX-TSP (I)

1. convert TSP input (cities, distances) to complete weighted graph G = (V, E) with V = cities and edges weighted with distances (or ∞ if none given)
2. compute MST of G
3. go "twice around" MST to get a tour
4. use a preorder traversal of MST for the tour T
5. return T

7

Approximation Algorithm for Restricted TSP

Running time = $O(V^2 \log V)$.

Quality of the solution found:

$$|T^*| \geq MST$$

lower bound (because tour must contain a cycle)

$$|T| < 2\,|MST|$$

upper bound by triangle inequality.

So

$$|T| < 2\,|T^*|$$

Therefore, APPROX-TSP is a p-time 2-approximation scheme for the TSP (under assumption of triangle inequality).

8

TSP without Triangle Inequality

Theorem: If P ≠ NP, then no p-time approximation algorithm A for the general TSP can approximate the optimal tour within a constant factor of optimal.

Proof: Suppose, in contradiction, there is such an algorithm A that is a k-approximation algorithm for some positive constant k. We will use A to solve the HC problem in p-time (since HC is NPC and we assume P ≠ NP).

Algorithm for HC:

1. convert HC input G to an instance I of TSP with cities = V, and
   $d(u,v) = \begin{cases} 1 & \text{if } (u,v) \in E \\ kV & \text{otherwise} \end{cases}$
2. run A on I
3. if tour T returned has $|T| \leq kV$, then answer "yes" to HC, else "no"

9

TSP without Triangle Inequality

Algorithm for HC:

1. convert HC input G to an instance I of TSP with cities = V, and
   $d(u,v) = \begin{cases} 1 & \text{if } (u,v) \in E \\ kV & \text{otherwise} \end{cases}$
2. run A on I
3. if tour T returned has $|T| \leq kV$, then answer "yes" to HC, else "no"

Why is this algorithm correct?

if G has HC: then the optimal tour in I is that cycle and it has weight V (all edge weights = 1), so $|T^*| = |V|$, and $|T| \leq k|T^*| = kV$

if G has no HC: then an optimal tour in I must use some edge not in G, which has weight kV, so $|T^*| > kV$, and $|T| \geq |T^*| > kV$

10

Computing with DNA

DNA = deoxyribonucleic acid; the genetic material that encodes the characteristics of living things.

What does DNA have to do with computers???

In 1994, Len Adleman (the "A" in RSA encryption system), showed that an instance of the NP-complete problem HC could be solved using DNA.

Later, researchers applied these techniques to solving the 3-SAT and TSP problem.

Today, DNA computing is an active research area in computer science.

11

DNA Computing

DNA is composed of 4 nucleotides: adenine (A), cytosine (C), guanine (G), and thymine (T).

Through scientific breakthroughs, it is possible to synthesize any desired string of these nucleotides to represent data.

We can encode any information using this 4-letter alphabet, just as we can encode any data using the binary digits 0 and 1.

Using a few basic operations to cut and paste DNA strands, DNA computing has been shown to be a *universal model of computation*, i.e., anything that can be computed on a Turing machine can be computed using strands of DNA.

12

DNA Computing

Advantages of DNA computing:
(1) potentially faster,
(2) uses less energy,
(3) more capacity for data storage,
(4) all work can be done in parallel, making DNA computers 1000 times faster than modern supercomputers,
(5) randomized nature of output suits non-deterministic nature of NP-complete problems, results are generally reported "with high probability".

Disadvantages of DNA computing:
(1) input and output hard to obtain (e.g., to find a HC in a 7-node graph took 7 days).
(2) process not yet automated, requires lots of human intervention.

13

DNA Computing - why do we care?

In the words of a prominent DNA computing researcher:

"If you have a problem and solving it with an electronic computer will take ten thousand years, you will be able do it with DNA computing in three weeks."

DNA computing research is in its infancy...will it be the next computing revolution?

For more information, go to http://www.csd.uwo.ca/~lila/

14