

329E - February 19, 2008

Types of Problems

- Decision problems:
 - Input: Graph G and integer k
 - Output: Yes if G has a k -clique, and No otherwise
- Optimization problems:
 - Input: Graph G
 - Output: the largest k so that G has a k -clique
- Construction problems:
 - Input: Graph G
 - Output: a largest clique in G

Errors in heuristics

- Heuristics can say “Yes” when the correct answer is “No”, and vice-versa.
- Heuristics can give a suboptimal value for an optimization problem (e.g., answer “5” when a 6-clique is present)
- Heuristics can give a suboptimal object (e.g. clique that is not the largest in the graph)

Types of errors

- Saying “Yes” when the correct answer is “No”: this is a False Positive
- Saying “No” when the correct answer is “Yes”: this is a False Negative
- Depending upon the context, some kinds of errors are more problematic than others!

Max clique heuristics

- To answer “does graph G have a clique of size k ” using exhaustive search takes $O(n^k)$ time.
- A greedy heuristic -- putting vertices together into a clique -- will often make mistakes. But it only makes False Negative errors, and makes no False Positive errors.

Optimization heuristics

- If you want to find the largest clique, and use the greedy heuristic to find a clique, you can return a clique which is smaller than the largest possible.
- The error in your output is
 - Optimum-size / output-size
- Example: you return a 6-clique, but the max clique in G has 15 vertices. Your error is $15/6 = 2.5$

Bounding errors

- You may wish to design a heuristic which has no False Positives, but can have False Negatives (or vice-versa). This may or may not be easy to ensure.
- Example: answering “does G have a clique of size k ?” is easy to do without false positives, but is not easy to do without false negatives.
- Bounding the error ratio (for optimization problems) is harder.

Approximation algorithms

- The error bound for a heuristic for an maximization problem is stated as
 - $\text{Max} \{ \text{Optimum}(I) / \text{Output}(I) : \text{inputs } I \}$

For minimization problems, you reverse the denominator and numerator, so that the larger number is always on top. That way, the error bound is always at least 1.

Vertex Cover

- Input: graph $G = (V, E)$
- Output: subset $V_0 \subseteq V$ such that for all edges e in E , at least one endpoint of e is in V_0 .
- Vertex-Cover is NP-hard, so we use heuristics.

Greedy algorithm for VC

- Let $V_0 = \emptyset$
- While G has any edge, DO
 - let v be a node with at least one edge incident with v , and add v to V_0 .
 - delete all edges incident to v from G .
- Return V_0 .

- What is the error bound for this greedy algorithm?
- Can you bound the error if you always pick the node of largest degree?

A 2-approximation algorithm for Vertex Cover

- Let $V_0 = \emptyset$
- While G has any edge DO
 - Pick any edge in G and add both its endpoints (v and w) to V_0
 - Delete all edges incident with either v or w
- Return V_0

Minimum spanning tree

- Input: graph G with weights on the edges
- Output: subgraph G' of G that includes all the vertices of G , of minimum total weight

Minimum spanning tree

- It is not hard to see that if G is connected, then the minimum cost subgraph is a tree (connected graph without any cycles)
- If G has n nodes and is connected, then its minimum spanning tree has $n-1$ edges.
- Exhaustive search strategy for solving MST would take $O(m^{n-1})$, where G has m edges and n nodes.
- What about a greedy strategy?

Greedy algorithm for MST

- Sort the edges of $G=(V,E)$ from smallest weight to largest.
Let $G'=(V, \emptyset)$ (same vertices, no edges)
- While G' is not connected, DO:
 - Let e be the smallest weight edge in E .
 - If the graph formed by adding e to G' is acyclic, then add e to G' .
 - Delete e from E .

Error bound for this greedy algorithm?

- Try some examples
- Make a conjecture
- Prove your conjecture

Maximum parsimony

- Input: set S of DNA sequences, each of the same length
- Output: tree T which has each element of S labelling a leaf, and other DNA sequences at the internal nodes (all of the same length), which minimizes the sum of the Hamming distances on the edges of T .

Maximum parsimony

- Example:
 - X = ACTG,
 - Y = ACGT,
 - Z = AGGA,
 - W = CGGC
- What is the best tree?

Maximum parsimony

- MP is NP-hard
- Exhaustive search??? Way too expensive.
- Greedy algorithm?
- Other heuristics?

Constrained version of MP

- Don't allow any new sequences for the internal nodes -- use only what you have for all the labels of all nodes!

Solving this constrained version

- Compute the Hamming distances between every pair of sequences in S
- Consider the complete graph with weight $w(v_1, v_2) = \text{Hamming}(s_1, s_2)$
- Compute the MST (minimum spanning tree) of this weighted graph.
- Error bound?