Greedy algorithms
Or Do the right thing

September 22, 2003

1 Greedy Algorithm

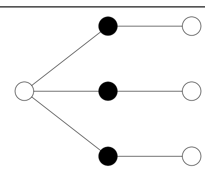
Basic idea: When solving a problem do locally the right thing.

Problem: Usually does not work.

> **VertexCover** (Optimization Version)
> Instance: A graph G, and integer k.
> Q: Return the smallest subset $S \subseteq V(G)$, s.t. S touches all the edges of G.

Example: VERTEXCOVER problem: Greedy algorithm always takes the vertex with the highest degree, add it to the cover set, remove it from the graph, and repeats.

Counter Example:



Optimal solution is the black vertices, but greedy would pick the four white vertices.

Well, maybe we do not get the optimal vertex cover, but we still get some kind of vertex cover which is good?

Q: What is good?

Definition 1.1 A minimization problem is an optimization problem, where we look for a valid solution that minimizes a certain target function.

Example 1.2 VertexCover, the target function is the size of the cover. Formally

$$Opt(G) = \min_{S \subseteq V(G), S \text{ cover of } G} |S|$$

The $VertexCover(G)$ is just the set S realizing this minimum.

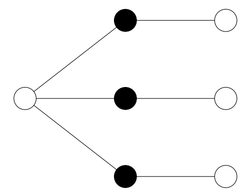
Definition 1.3 Let $Opt(G)$ denote the value of the target function for the optimal solution.

Good = vertex cover of size "close" to the optimal solution.

Definition 1.4 Algorithm A for a minimization problem achieves an approximation factor α if for all inputs, we have: $\frac{A(G)}{Opt(G)} \leq \alpha$.

Example 1.5 An algorithm is a 2-approximation for VertexCover,. if it outputs a vertex cover which is at most twice the size of the optimal solution for vertex cover.

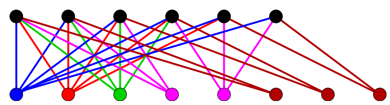
Q: How good is the Greedy VertexCover?



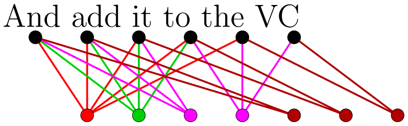
Optimal solution is the black vertices, but greedy would pick the four white vertices.
Thus, approximation factor is *at least* 4/3.

.

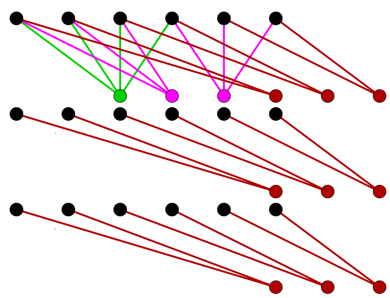
Example 1.6 Does the greedy VertexCover algorithm is a 2-approximation?
Ansser: No...



Remove the blue vertex... And add it to the VC



Remove red vertex



So, approximation algorith removes 8 vertices
Optimal vertex cover uses 6 vertices.

This is just shows approx up to 8/6=4/3. However, extending this example to larger n (homework exercise), shows that the approximation algorithm in the worst case is a $\Omega(\log(n))$ approximation.

Theorem 1.7 *The greedy algorithm for VertexCover achieves $\Theta(\log n)$ approximation, where n is the number of vertices in the graph.*

Proof: Lower bound follows from the example indicated above. Upper bound will follow from similar proofs we will do shortly, and is omitted.
■

1.1 Two for the price of one

(Pay more, get less)

Q: Any better approximation algorithm for vertex cover?

Algorithm: **Approx-Vertex-Cover**

Choose an edge from G, add both endpoints to the vertex cover, and remove the two vertices from G and repeat.

Theorem 1.8 ***Approx-Vertex-Cover*** *achieves approximation factor 2.*

Proof: Every edge removed contains at least one vertex of the optimal solution. As such, the cover generated is at most twice larger than the optimal. ■

2 Traveling Salesman Person

Theorem 2.1 *TSP can not be approximated within* ***any*** *factor unless $P = NP$.*

Proof: Consider the reduction from Hamiltonian cycle into TSP. We set the weight of every edge to 1 if it was present in the instance of the hamiltonian cycle, and 2 otherwise. In the resulting complete graph, if there is a tour price n then there is a HC in the original graph. If on the other hand, there was no cycle in G then the cheapest TSP is of price $n + 1$.

Instead of 2, use cn, for c an arbitrary constant. Clearly, if G does not contain any Hamiltonian cycle in then the price of the TSP is at least $cn + 1$.

If one can do a c-approximation in polynomial time then using it on the TSP graph, would yield a tour of price $\leq cn$ if a tour of price n exists. But a tour of price $\leq cn$ exists iff G has a himtilontian cycle. ■

2.1 Traveling salesman problem with the triangle inequality

$G = (V, E)$ - graph

$c(e)$ - The cost of traveling on the edge e

Definition 2.2 (Triangle inequality)

For any u, v, w in $V(G)$ we have:

$$c(u, v) \leq c(u, w) + c(w, v)$$

Purpose: Develope a 2-approximation algorithm.

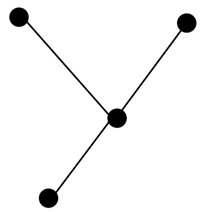
Observations:

1. C_{opt}- optiamal TSP Cycle in G.

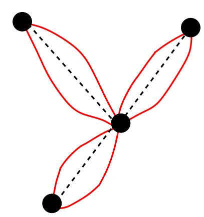
2. C_{opt} is a spanning graph.
3. $w(C_{opt}) \geq w(\textit{cheapest spanning graph of } V)$
4. Cheapest spanning graph of G, is just the minimum spanning tree.

$$w(C_{opt}) \geq w(MST(G))$$

5. MST can be computed in $O(n \log n + m)$ time.
6. Convert the MST into a tour.



   (a) Convert each edge of $T = MST(G)$ by duplicating each edge. Let T denote the new graph.

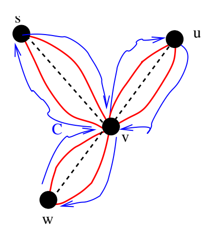


   $w(T) = 2w(MST(G))$

   (b) For every vertex $v \in V(T)$ we have $d(v)$ is an even number.

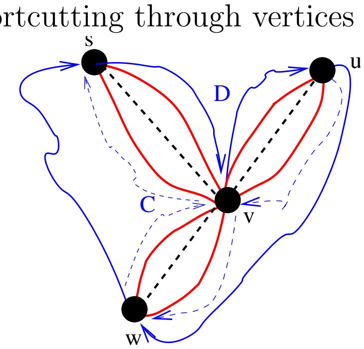
   (c) The graph T is Eulerian.

   (d) Let C' denote the Eulerian cycle in T



$$w(C) = w(T) = 2w(MST(G))$$

We next normalize C by shortcutting through vertices we already visited:



By the triangle ineqouality:$w(uw) \leq w(uv) + w(vw)$.
Thus, the resulting cycle D is not longer.

$$w(MST) \leq w(TSP) \leq w(D) \leq w(C) = 2 * w(MST) \leq 2 * w(TSP)$$

Theorem 2.3 *TSP with the triangle inequality, can be approximated up to a factor of* 2 *in* $O(n \log n + m)$ *time.*

3 Max Exact 3SAT

Instance of the 3SAT problem:

$$F = (x_1 + x_2 + x_3)(x_4 + \overline{x_1} + x_2)$$

Problem: MAX 3SAT

Instance: A collection of clauses: $C_1, \ldots, C_m$.
Question: Find the assignment to $x_1, ..., x_n$ that satisfies the maximum number of clauses.

Max 3SAT is NP-Hard.
For example, F becomes:

$$x_1 + x_2 + x_3$$
$$x_4 + \overline{x_1} + x_2$$

Note, that this is a maximization problem.

Definition 3.1 Algorithm A for a maximization problem achieves an approximation factor α if for all inputs, we have:
$\frac{A(G)}{Opt(G)} \geq \alpha$.

Theorem 3.2 *There is an algorithm which "achieves"* $(7/8)$*-approximation in polynomial time. Namely, if the instance has* m *clauses it satisfies* $(7/8)m$*.*

Algorithm: RandMax3SAT
$x_i \leftarrow$1 with probability 1/2, and 0 otherwise.
Return $x_1, ..., x_n$

Y_i- the i-th clause is satisfied by the random assignment
Y_i is an indicator variable - get 1 if the i-th clause is satisfied, else 0.

$$Y_i = \begin{cases} 1 & C_i \text{ is satisfied by rand assignment} \\ 0 & \text{Otherwise.} \end{cases}$$

Clearly, the number of clauses satified by the given assignment is:

$$Y = \sum_{i=1}^{m} Y_i$$

Lemma 3.3 $E[Y] = (7/8)m$, *where* m *is the number of clauses in the input.*

Proof: We have

$$E[Y] = E\left[\sum_{i=1}^{m} Y_i\right] = \sum_{i=1}^{m} E\left[Y_i\right]$$

by linearity of expectation. Now, what is the probability that $Y_i = 0$?

$$P\left[Y_i = 0\right] = \frac{1}{2} * \frac{1}{2} * \frac{1}{2} = \frac{1}{8}$$

This is the probability that C_i is not satisisified. C_i is made out of exactly three literals, and as such....?

$$P\left[Y_i = 1\right] = 1 - P\left[Y_i = 0\right] = \frac{7}{8}.$$

Thus,

$$E\left[Y_i\right] = P\left[Y_i = 0\right] * 0 + P\left[Y_i = 1\right] * 1 = \frac{7}{8}.$$

Namely,

$$E\left[\text{\# of clauses sat}\right] = E[Y] = \sum_{i=1}^{m} E\left[Y_i\right] = \frac{7}{8}m.$$

■