CS 630 – Fall 2024 – Lab 3
Sep 25, 2024

1. Vertex Cover Formulations

A vertex cover is a subset C of vertices of a graph $G = (V, E)$ such that $C \subseteq V$ and $\forall (u, v) in E, u \in C \vee v \in C$. Vertex cover problems are generally concerned with finding minimum sized covers, but the exact problem can be formulated as an optimization problem, decision problem, or construction problem.

- **VC-OPTIMIZE.** What is the minimum size $|C|$ of a vertex cover C for G?
- **VC-DECIDE.** Does there exist a vertex cover C for G with $|C| \leq k$?
- **VC-CONSTRUCT.** Construct a minimum size $|C|$ vertex cover C for G.

For each of the following questions, express your answer with $O()$ notation for both the time and number of API calls used.

1. If you are given an API for **VC-OPTIMIZE**, how fast can you implement **VC-DECIDE**?
2. If you are given an API for **VC-DECIDE**, how fast can you implement **VC-OPTIMIZE**?
3. If you are given an API for **VC-OPTIMIZE**, how fast can implement **VC-CONSTRUCT**?

Solution:

1. **VC-DECIDE** can be implemented in $O(1)$ time and one API call (so $O(1)$) to **VC-OPTIMIZE**. Simply call **VC-OPTIMIZE** on the same input, compare the returned minimum clique size to the requested k, and respond yes or no accordingly.
2. In the degenerate case of no edges, the minimum vertex cover size is zero and your algorithm can immediately return.

   Otherwise, the minimum vertex cover size is between 1 and $n = |V|$. This range can be reduced to a single integer value by using binary search to repeatedly halve the size of the range. Once the range just has one integer value, return that value. The binary search takes $O(\log n)$ rounds with $O(1)$ time and one API call per round, so the total is $O(\log n)$ time and $O(\log n)$ API calls. That time bound covers checking the degenerate case too.
3. The critical element of reducing a construction problem to an optimization or decision problem is figuring out how to use the latter problems to decide "what to keep" or "what to lock in" as part of your solution.

   Suppose that calling **VC-OPTIMIZE** on the whole graph returns k as the minimum size. If we pick any $v \in V$ and remove v and all edges using v from the graph to get G', what is the minimum vertex cover size of the remaining graph G'? It must be either k or $k-1$. A number higher than k would imply more than k vertices were needed to cover the original graph which has a superset of the reduced edges which would be a contradiction of k being the minimum vertex cover size for G. A number $k-2$ or lower would mean that the a minimum vertex cover for G' could add v to get a vertex cover for G of size at most $k-1$ which would again be a contradiction.

If the minimum vertex cover of G' is $k-1$, then there must be a minimum vertex cover including v. This can be shown by recursively calling **VC-CONSTRUCT** on G' and adding v to get a vertex cover for G. This case will always come up if v is in any minimum vertex cover C, since by definition of a vertex cover, C covers all the edges and $C - \{v\}$ must cover all edges not including v, and $|C - \{v\}| = k - 1$. If all v are tested without finding a G' with minimum vertex cover $k-1$, then none of them are included in minimum vertex covers. In this case, there will also be no more edges in the graph since any edge would require at least one vertex to be included in any vertex cover.

VC-CONSTRUCT can use this recursive structure to construct a minimum cover in $O(knm) = O(n^2m)$ time and $O(kn)$ API calls where $n = |V|$ and $m = |E|$. This is because the recursion will take $k+1$ levels to identify all the vertices in the output, there are $O(n)$ vertices to test in each recursive call, and updating the graph may take $O(m)$ time.

An iterative implementation can find the minimum vertex cover faster by updating G with G' after each vertex in a minimum vertex cover is found and continuing to check each vertex instead of recursing. That is, it checks $v_1, v_2, \ldots, v_n$ in one pass. Updating G has the effect of conditioning later analysis to assume the choice of v. This reduces the time number of API calls to n. The total time is $O(nm)$ time for n checks costing $O(m)$ time and $O(n)$ API calls.

2. Understanding Approximation Algorithms

1. If your 2-approximate algorithm for vertex cover returns 20, what are the minimum and maximum sizes of the actual minimum vertex cover?

2. The maximum clique problem looks for the largest subset of vertices that are fully connected. The independent set problem looks for the largest subset of vertices that have no edges at all. In lecture, we saw a one-to-one correspondence between these problems. If someone claims an algorithm with an $f(n)$-approximation for independent set, what does that mean for approximating the maximum clique problem?

3. The independent set problem and the vertex cover problems are also closely related. In lecture, we saw another one-to-one correspondence where $S \subseteq V$ was an independent set if

and only if $(V - S)$ was a vertex cover of the same graph. If someone claims an algorithm with an α-approximation for vertex cover for some constant α, what does that imply for approximation bounds for independent set?

4. Suppose the best known polynomial time approximation algorithms for vertex cover and set cover are respectively α-approximate and β-approximate. What is the relationship between α and β? Are they equal? Must one be larger?

Solution:

1. Since the algorithm is a 2-approximation, the minimum size of the minimum vertex cover is $20/2 = 10$. The maximum size is 20; it is possible that the approximation algorithm gets lucky and returns the optimum solution.
2. Since the reduction from maximum clique to independent set preserves the set of vertices which is the solution, the same approximation factor applies. So there is also a $f(n)$-approximate algorithm for maximum clique.
3. The α-approximation for vertex cover does not carry over easily to independent set. That is because the sizes of S and $V - S$ might be very different from each other.
4. It should be the case that $\alpha \leq \beta$. The reduction from vertex cover to set cover preserves the size of the minimum cover, so any β-approximate algorithm immediately applies to vertex cover too.