The vertex cover problem is an NP–hard optimization problem for graphs. For both weighted and unweighted graphs, a simple probabilistic approximation algorithm is presented which always halts and produces a vertex cover. The expected value of the weight of the cover produced is at most twice that of an optimal cover.

A Simple Probabilistic Approximation Algorithm for Vertex Cover.

Leonard Pitt
YALEU/DCS/TR–404
June, 1985

This research was partially funded by the National Science Foundation under grant numbers MCS 8002447, MCS 8116678, MCS 8204246, and MCS 8404226.

1 Introduction

The vertex cover problem for graphs is a well known NP-hard optimization problem [5]. For the unweighted version, optimal solutions can be approximated to within a factor of two. Some approximation algorithms for the weighted version of the problem also achieve a factor of two, although these algorithms are generally more complicated and less intuitive. The first factor of two approximation for the unweighted problem appeared implicitly in Nemhauser and Trotter [9], and was based on linear programming relaxation. The first explicit factor of two approximation, also based on linear programming, appears in a paper of Hochbaum [7]. Subsequently, refinements of these algorithms obtained by exploiting the graphical structure of the problem were achieved by Clarkson [4], and Bar–Yehuda and Even [1]. These two latter algorithms are similar, and in both, the crucial part of the algorithm relies on operations which appear counter-intuitive. This has prompted the appearance of two recent papers which recast the counter-intuitive algorithm into a new framework, provide a more intuitive combinatorial proof, and extend the factor of two approximation to a larger class of problems (Gusfield and Pitt [6], and Bar–Yehuda and Even [2]).

In this paper we present a very simple and intuitive probabilistic expected factor of two approximation algorithm for both the unweighted and weighted vertex cover problem. The algorithm always halts and outputs a vertex cover in polynomial time, but the weight of the cover produced depends on the "coin flips" which occur during the run of the algorithm. In the unweighted case, the algorithm exemplifies the surprising notion that randomly picking vertices to be in a cover is an effective approximation heurisitic.

2 Definitions

$G = (V, E)$ is an undirected graph with vertex set V, and edge set E. Each vertex $v \in V$ has a weight $w(v) > 0$. $S \subseteq V$ is a *vertex cover* of G if and only if for all edges $e \in E$, e is incident to a vertex in S. The weight of the cover S is denoted by $w(S)$, and defined by $w(S) = \sum_{s \in S} w(s)$.

The weighted vertex cover problem is to find a vertex cover with minimum weight. A graph is *unweighted* if $w(v) = 1$ for all $v \in V$. Hence the unweighted vertex cover problem is to find a vertex cover of minimum cardinality.

3 Unweighted Vertex Cover Algorithm

Although there are simple, deterministic factor of two approximation algorithms for the unweighted vertex cover problem, we present the following probabilistic algorithm for two reasons: First, it motivates the introduction of our algorithm for the weighted version in the next section, and second, we find it surprising that such a simple minded algorithm actually performs adequately.

An extremely naïve attempt to obtain an approximate solution for a minimum vertex cover of a given graph would be as follows:

- While there are uncovered edges, pick a vertex adjacent to some uncovered edge and place it in the cover.

Surprisingly, a simple modification of this algorithm actually yields an expected factor of two approximation. The first modification is to interpret the word "pick" as "pick

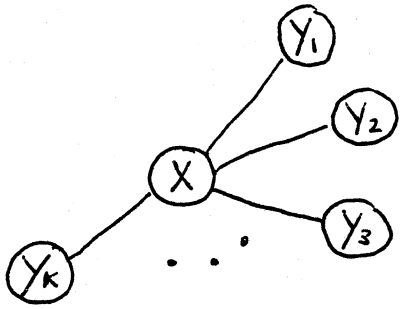

Figure 1: Star graph

randomly". A deterministic algorithm's performance is judged by its behavior on "worst case" examples. A random choice can prevent classes of bizarre graphs from forcing our algorithm to perform poorly. The other modification is that rather than picking randomly from among all unchosen vertices, we pick (deterministically) the next uncovered edge (according to some predetermined order), and then randomly choose one of its endpoints to be in the cover.

Algorithm U

1. $S \leftarrow \emptyset$. Order the edges of E arbitrarily.
2. Get the next edge $e = (u, v) \in E$ which is not yet covered (neither u nor v are in S). If there is no such edge, then let $S_G = S$, stop and output S_G.
3. Flip a 2 - sided coin. If the result is "heads" then place u in S, if "tails" then place v in S.
4. Go to 2

The vertex cover S_G produced by running algorithm U with input graph G depends on the results of the coin flips. Let $E(w(S_G))$ denote the expected value (taken over all possible runs of U) of the weight of the cover S_G produced by running U with input G. Then we have

Theorem 1 *For all undirected, unweighted graphs G, if C is any optimal cover of G, then $E(w(S_G)) \leq 2w(C)$.*

We've already explained that the introduction of randomization to our "naïve" algorithm was to foil a worst-case adversary ... what is the explanation for picking edges, and then randomly picking one of its endpoints to be in the cover, rather than simply making a random choice of a vertex from among the vertices yet to be chosen? This is best illustrated by the following example. Consider the "star" graph of Figure 1 consisting of a vertex x with neighbors $y_1, y_2, \ldots y_k$.

Clearly the optimal vertex cover consists of the single vertex x. Were we to randomly choose vertices to be in a cover, we'd pick roughly $\frac{k}{2}$ vertices on average before we picked the vertex x. An approximation algorithm for vertex cover has been suggested which first

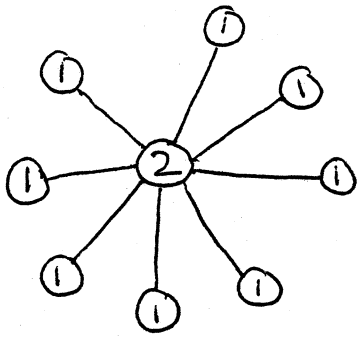

Figure 2: "Least weight" heuristic ineffective

picks the vertex of maximum degree, reduces the graph by eliminating covered edges, and then repeats this process. It has been shown that this algorithm does not approximate the optimal vertex cover to within any constant factor [8]. Step 2 of algorithm U incorporates the advantages of this heuristic. Since vertices with high degree will occur in the list of edges many times, it is likely that once two or three edges incident to such a vertex is examined, the vertex will have been placed in the cover S.

Although Theorem 1 is a special case of Theorem 2 proved in the next section, we offer a simpler intuitive proof here:

Let G be a graph and let C be an optimal vertex cover of G. Suppose algorithm U on input G produces cover S_G. Then it has flipped $|S_G|$ coins while considering $|S_G|$ edges and placed $|S_G|$ vertices into the cover S_G. Call a flip "good" if it caused a vertex in C to be placed in S_G, and call the flip "bad" otherwise. Certainly the algorithm will terminate by the time $|C|$ good flips have been made. But each edge considered has at least one of its vertices in C; thus each flip is a "good" flip with probability $\geq \frac{1}{2}$, so the expected number of flips required before obtaining $|C|$ good ones (and hence terminating) is at most $2|C|$. □

4 Weighted Vertex Cover Algorithm

One heuristic for the weighted vertex cover problem is to first pick vertices which have the least weight. The graph in Figure 2 clearly demonstrates why this heuristic is ineffective.

A modified heuristic would be to first pick the vertices for which the ratio weight/degree was the least. It has been shown that an algorithm based on this heuristic does not approximate the optimal cover to within any constant factor [3]. However, using a probabilistic strategy, we can capture the advantages of this heuristic while avoiding the "worst cases" which force deterministic algorithms to perform poorly. As in algorithm U, we will consider edges one at a time according to some arbitrary predetermined order. For each edge considered, if the edge has not yet been covered, a biased coin will be flipped and one of the two vertices incident to the edge will be placed in the cover being built, depending on the result of the coin flip. Thus vertices with high degree will participate in many coin flips, and have a greater chance of being included in the cover. The biased flip is used to increase the likelihood that vertices of small weight are placed in the cover. Consider the graph in Figure 3.

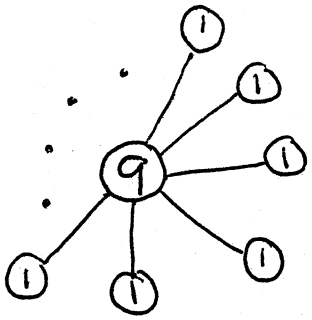

Figure 3: Weighted star graph

Depending on how many perimeter vertices of weight 1 there are, the optimal cover consists either of the center vertex, or all of the perimeter vertices. The "break-even" point is when there are exactly nine perimeter vertices. If on considering an edge to be covered, the center vertex is chosen with probability 1/10, and the outside vertex with probability 9/10, then if there are nine perimeter vertices, the cover will probably include them all. If there are greater than nine perimeter vertices, then we expect that after roughly nine edges are examined (and nine perimeter vertices placed in the cover), the center vertex will be chosen, thus forming a cover. In both cases, the size of the cover produced is less than twice optimal. This observation motivates the following algorithm.

Algorithm W

1. $S \leftarrow \emptyset$. Order the edges of E arbitrarily.
2. Get the next edge $e = (u, v) \in E$ which is not yet covered (neither u nor v are in S). If there is no such edge, then let $S_G = S$, stop and output S_G.
3. Flip a coin with $w(u) + w(v)$ equiprobable outcomes. Depending on the flip, with probability $\frac{w(u)}{w(u)+w(v)}$ place v in S, and with probability $\frac{w(v)}{w(u)+w(v)}$ place u in S.
4. Go to 2

To clarify step 3, consider the case that an edge $e = (u, v)$ is considered with $w(u) = 5$ and $w(v) = 10$. Then u is placed in S with probability $\frac{10}{15}$, and v is placed in S with probability $\frac{5}{15}$. (In the unweighted case $\frac{w(v)}{w(u)+w(v)} = \frac{w(u)}{w(u)+w(v)} = \frac{1}{2}$.) We now have

Theorem 2 *For all undirected, weighted graphs G, if C is any optimal vertex cover of G, then $E(w(S_G)) \leq 2w(C)$.*

Again, $w(S_G)$ is the weight of the cover produced by algorithm W on input graph G, and the expectation is taken over all possible sequences of coin flips.

We note that Theorem 1 follows from Theorem 2, and that the order in which edges are examined may be any arbitrary fixed order.

We now prove Theorem 2. Let C be some optimal vertex cover of $G = (V, E)$. Our "experiment" is the running of algorithm W with input graph G. We define for each

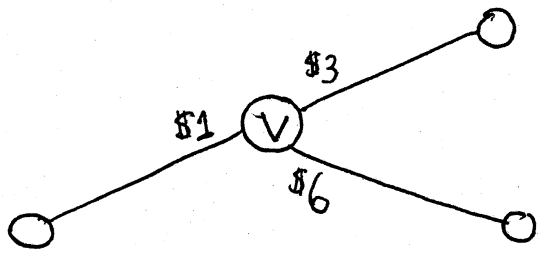

Figure 4: Distributing money on incident edges

vertex $v \in V$, the random variable X_v, whose value depends on the outcome of the experiment:

$$X_v = \begin{cases} w(v), & \text{if } v \in S_G; \\ 0, & \text{otherwise.} \end{cases}$$

Then the random variable $w(S_G)$ represents the weight of the vertex cover produced, and is defined by

$$w(S_G) = \sum_{v \in V} X_v.$$

Now in the cover S_G, some vertices may also be in the cover C. Let C_W be this collection of vertices which are in $S_G \cap C$. Then the random variable $w(C_W)$ is defined by

$$w(C_W) = \sum_{v \in C} X_v.$$

Thus $w(C_W)$ is the total weight of vertices in C which are chosen to be in the cover S_G obtained by running algorithm W. Clearly $w(C_W) \leq w(C)$ for every possible run, since at most every vertex in C is placed in S_G. Therefore, $E(w(C_W)) \leq w(C)$. We will show that $E(w(S_G)) \leq 2E(w(C_W)) \leq 2w(C)$. The proof is based on the following idea. Imagine that the value of $E(X_v)$ dollars is given to each node v. Then the value $E(w(S_G))$ is the total amount of dollars on the graph, since $E(w(S_G)) = \sum_v E(X_v)$. We will show that it is possible to redistribute the money in such a way that each vertex $c \in C$ has at most $2E(w(X_c))$ dollars, and no vertex not in C has any dollars. This demonstrates that the total amount of dollars $E(w(S_G))$ satisfies

$$E(w(S_G)) \leq \sum_{c \in C} 2E(w(X_c)) = 2E(w(C_W)) \leq 2w(C).$$

Each vertex v takes its $E(X_v)$ dollars, and distributes them at the ends of the edges which are incident to v. For example if $E(X_v) = 10$, and there are three edges incident to v, then vertex v might choose to place its money as illustrated in Figure 4.

Suppose there exists a way for the vertices to place their dollars at the ends of the edges incident to them such that the sum of the dollars surrounding each vertex v is exactly $E(X_v)$, and each edge has the same dollar amount at each end. Then we claim that the total dollars on the graph, $E(w(S_G))$, is at most $2E(w(C_W))$.

To see this, note that if $v \notin C$ then all of the neighbors of v *are* in C. Thus v may then slide its dollars along the incident edges on which they have been placed to its neighbors, which are in C. After all vertices not in C slide their dollars in this manner, no vertex not in C has any money, and each vertex $c \in C$ has at most $2E(X_c)$ dollars, since it started with $E(X_c)$ dollars, and at most received another $E(X_c)$ dollars along its incident edges. Thus the total amount of money on the graph must be $E(w(S_G)) \leq \sum_{c \in C} 2E(X_c) \leq 2E(w(C_W))$.

We now need only show that such a distribution of dollars around each vertex is possible, and Theorem 2 follows. During the run of algorithm W, each edge (u, v) is examined. If (u, v) is already "covered", *i.e.* either u or v is already in the cover S being built, then the edge is discarded. If it is not yet covered, then a biased coin is flipped and either u or v is placed in S. We call any edge which causes such a flip a *chosen* edge, and if (u, v) is a *chosen* edge, and as a result of the associated coin flip, vertex u is placed in S, we'll say that edge (u, v) *causes* u (to be placed in the set S).

Let the random variable $X_{u,v}$ be defined as follows:

$$X_{u,v} = \begin{cases} w(u), & \text{if } (u, v) \textit{ causes } u, \\ 0, & \text{otherwise.} \end{cases}$$

Then clearly, for all u,

$$X_u = \sum_{v \textit{ incident to } u} X_{u,v}$$

since at most one of the terms in the sum is nonzero, and at most one edge incident to u causes u to be placed in S. Then

$$E(X_u) = \sum_{v \textit{ incident to } u} E(X_{u,v}).$$

We show that for all nodes u, v for which an edge (u, v) exists,

$$E(X_{u,v}) = E(X_{v,u}). \tag{1}$$

If Equation 1 holds, then each vertex u may distribute its $E(X_u)$ dollars at the ends of the edges incident to it by placing $E(X_{u,v})$ dollars on the edge leading to vertex v. In this manner, all dollars will be placed on the edges, and each edge will have the same amount at each end. By the reasoning above, this proves Theorem 2.

We now show that Equation 1 holds:

$$\begin{aligned} E(X_{u,v}) &= w(u) \times \Pr[\text{edge } (u, v) \textit{ causes } u] \\ &= w(u) \times \Pr[\text{edge } (u, v) \textit{ chosen}] \times \frac{w(v)}{w(u) + w(v)} \\ &= w(v) \times \Pr[\text{edge } (u, v) \textit{ chosen}] \times \frac{w(u)}{w(u) + w(v)} \\ &= w(v) \times \Pr[\text{edge } (u, v) \textit{ causes } v] \\ &= E(X_{v,u}) \end{aligned}$$

□

5 Remarks and Conclusion

We've seen how an extremely simple heuristic may be used to achieve an expected factor of two approximation for the weighted vertex cover problem. The heuristic is actually very similar to heuristics which fail miserably in the deterministic case (take the vertex of highest degree, or least ratio weight/degree), prompting the conclusion that introducing randomization may foil "worst case adversaries" which drag down the performance of deterministic algorithms based on the same heuristic.

The expected performance of algorithm W is (in general) no better than twice optimal; Let $G_{m,n}$ be a graph consisting of n disjoint copies of a star graph with m leaves. Let $OPT(G_{m,n})$ = the size of the smallest vertex cover of $G_{m,n} (= n.)$ Then the expected value of the *absolute performance ratio* R_W^∞ (as defined in [5]) is easily shown to be equal to 2 by examining the behavior of W on the class of graphs $G_{m,n}$. That is, as m grows, the ratio $E(W(G_{m,n}))/OPT(G_{m,n})$ approaches 2 for any fixed n. A simple analysis shows that the algorithm runs in linear time.

In practice, one might iterate algorithm W a number of times, and choose as a cover the least weight cover produced. Naïve assumptions about the variance of the weight of the cover produced suggest that only by repeating the experiment "exponentially many" times can one expect to obtain a cover with size at most $(2-\epsilon) \times$ *optimal* for any fixed ϵ. Alternatively, one might hope that by iteration, a factor of $2 - f(n)$ might be achieved, where f is a decreasing function of n, the number of vertices of the graph. We conjecture that this is possible, although it is unlikely that the function f would decrease slower than that provided by the algorithm of [2], where a factor of $2 - \frac{\log\log n}{2\log n}$ is obtained.

Admittedly, the algorithms U and W are not guaranteed to perform well as often as the known deterministic factor of two approximation algorithms. We find these algorithms appealing nonetheless, for the simplicity of the underlying ideas, and the interesting proof of the expected performance of algorithm W.

Finally, we note that similar algorithms obtain a factor of d approximation for the set cover problem, as well the "subset selection" problems of [6].

Acknowledgements

The author thanks D. Angluin, M. Fischer, and D. Gusfield for helpful discussions regarding this work.

References

[1] R. Bar-Yehuda and S. Even, *A linear time approximation algorithm for the weighted vertex cover problem*, Journal of Algorithms, vol. 2, 198-203, 1981.

[2] R. Bar-Yehuda and S. Even, *A local-ratio theorem for approximating the weighted cover problem*, (preprint), 1983.

[3] V. Chvatal, *A greedy heuristic for the set-covering problem*, Math. of Operations Research, vol. 4, no. 3, 233-235, 1979.

[4] K. Clarkson, *A modification of the greedy algorithm for the vertex cover*, Information Processing Letters, no. 16, 23-25, 1983.

[5] M. Garey and D. Johnson, *Computers and Intractability; A Guide to the Theory of NP–Completeness*, Freeman, San Francisco, CA, 1979.

[6] D. Gusfield and L. Pitt, *Understanding approximations for node cover and other subset selection problems*, Technical Report YaleU/DCS/TR-308, Yale University, 1984

[7] D.S. Hochbaum, *Approximation algorithms for the set covering and vertex cover problems*, Siam J. Computing, vol. 11, no. 3, 555-556, 1982.

[8] D. Johnson, *Approximation algorithms for combinatorial problems*, J. Comput. Sys. Sci., no. 9, 256-278, 1974.

[9] G.L. Nemhauser and R.E. Trotter, *Vertex packing structural properties and algorithms*, Math. Programming, no. 8, 232-248, 1975.