SIAM J. DISCRETE MATH.
Vol. 23, No. 3, pp. 1261–1280

DIVIDE-AND-CONQUER APPROXIMATION ALGORITHM FOR VERTEX COVER[*]

EYJÓLFUR INGI ÁSGEIRSSON[†] AND CLIFF STEIN[‡]

Abstract. The vertex cover problem is a classical NP-complete problem for which the best worst-case approximation ratio is $2-o(1)$. In this paper, we use a collection of simple graph transformations, each of which guarantees an approximation ratio of $\frac{3}{2}$, to find approximate vertex covers for a large collection of randomly generated graphs and test graphs from various sources. The graph reductions are extremely fast, and even though they by themselves are not guaranteed to find a vertex cover, we manage to find a $\frac{3}{2}$-approximate vertex cover for almost every single graph in our collection. The few graphs that we cannot solve have specific structure: they are triangle-free, with a minimum degree of at least 3, a lower bound of $\frac{n}{2}$ on the optimal vertex cover, and are unlikely to have a large bipartite subgraph.

Key words. vertex cover, approximation algorithm, heuristic, experiments

AMS subject classifications. Primary, 05C; Secondary, 68W, 68R

DOI. 10.1137/070710275

1. Introduction.

The vertex cover problem is a classical problem in computer science and one of the first problems that was proven to be NP-complete [22, 14]. A vertex cover of a graph $G = (V, E)$ is a subset of the vertices, $C \subseteq V$, such that each edge $e \in E$ has at least one endpoint in C. The objective is to minimize the size of the vertex cover, $|C|$.

A simple greedy algorithm gives a 2-approximation for the vertex cover problem [11]. Despite many attempts to design approximation algorithms for vertex cover [9, 25, 5, 16, 17], the best known approximation ratio is $2-\Theta(\frac{1}{\sqrt{\log n}})$ for a graph with n vertices [21]. It is NP-hard to approximate the minimum vertex cover within any factor smaller than 1.36 [10], and there is a conjecture that there does not exist an algorithm with a fixed approximation ratio better than 2 [18, 13].

In this paper, we look at the classical vertex cover problem. We use a collection of simple reductions and allow reductions that do not maintain optimality but only guarantee a worst-case approximation ratio. Our focus is mostly on reductions that guarantee an approximation ratio of no more than $\frac{3}{2}$, but we also introduce reductions with a higher approximation ratio. Each reduction has unique properties and utilizes various specific graph structures. We will look both at the performance of specific reductions and at how they work in combinations. By combining reductions that use different properties of the graph, we can get very beneficial interactions.

To test the graph reductions, we used two different methods. First we collected all graphs we could find on the Internet and from previous research projects, and also generated graphs specially designed to be difficult for the vertex cover problem. The first experiments focus on graphs that are not random, but are designed to be difficult

[*]Received by the editors December 6, 2007; accepted for publication (in revised form) May 14, 2009; published electronically August 14, 2009. A preliminary form of this paper appeared in the Proceedings of WEA 2007 [4]. Most of the graph reductions were introduced in the Proceedings of WEA 2005 [3].
http://www.siam.org/journals/sidma/23-3/71027.html

[†]Reykjavík University, 103 Reykjavík, Iceland (eyjo@ru.is).

[‡]Department of Industrial Engineering and Operations Research, Columbia University, New York, NY 10027 (cliff@ieor.columbia.edu).

for some graph-related problem. Some of these instances are designed to be difficult for the vertex cover problem, but many of these graphs focus on other graph problems such as the minimum coloring problem and maximum clique problem. Graphs that are designed for other graph problems might not be difficult for the vertex cover problem, but they can have some interesting structures and properties. The results from the experiments on these graphs were published in [3]. The second method focuses on random graphs. We use our reductions to find an approximate vertex cover for the randomly created graph and analyze the performance of each reduction with respect to various factors, such as the density of the graph and the number of vertices. The results on random graphs were published in [4].

In principle, our reductions do not guarantee that we will find an approximate vertex cover; however, we managed to find a $\frac{3}{2}$-approximate vertex cover in a few seconds for almost every single graph using only our reductions. We managed to solve both graphs that were specially designed for specific graph problems and also a large collection of randomly generated graphs. For the graphs where we know the minimum vertex cover, the actual approximation ratio was usually much lower than $\frac{3}{2}$; in many cases the vertex cover we found was either optimal or very close to optimal.

2. Graph reductions.

There have been many attempts to find an approximation algorithm for vertex cover. The most famous approximation algorithm for vertex cover is a simple greedy algorithm where we iteratively remove an edge from the graph, add both endpoints to the vertex cover, and remove all edges incident to either endpoint. Once we have removed all edges from the graph, we have a feasible vertex cover. Moreover, since every feasible vertex cover must include at least one endpoint of every edge, the size of the vertex cover we get from this greedy algorithm is at most twice the size of the optimal vertex cover, so we have an algorithm with an approximation ratio of 2. Many of the approximation algorithms for vertex cover are based on formulating the vertex cover problem as an integer or semidefinite program and then use rounding techniques to find an approximate solution [21, 5, 17]. The best approximation ratio is $2 - \Theta(\frac{1}{\sqrt{\log n}})$, by Karakostas [21], but the approximation ratio approaches 2 when the number of vertices increases. There is no known approximation algorithm for vertex cover with a fixed approximation ratio less than 2. Our approach is similar to divide and conquer; we use simple algorithms to solve the graphs step by step. The vertex cover problem is a hard problem, but instead of tackling the whole graph at once, we try to solve it by breaking it into smaller and easier subproblems. For our approach to work, we need the following lemma, which states that, by partitioning the graph and carefully finding an approximate vertex cover for each part, we can combine them into a feasible vertex cover for the whole graph with an approximation ratio equal to the largest approximation ratio of the vertex covers for the subgraphs.

LEMMA 1. *Assume we have a graph $G = (V, E)$ and a partition of the vertices $V = V_1 \cup V_2 \cup \cdots \cup V_k$. Let $G_i = (V_i, E_i)$ be the subgraph induced by V_i, and suppose that $\forall i$, $E_i \neq \emptyset$. Also assume that for each G_i we have a vertex cover VC_i^{approx} with the property that $\mathrm{VC}^{approx} = \cup_i \mathrm{VC}_i^{approx}$ is a vertex cover for G. Let VC^{opt} be an optimal vertex cover for G. Then*

$$\frac{VC^{approx}}{VC^{opt}} \leq \max_i \frac{|VC_i^{approx}|}{|VC^{opt} \cap V_i|}.$$

Proof. We have that $\mathrm{VC}^{approx} = \cup_i \mathrm{VC}_i^{approx}$ and since the vertex partition is disjoint, $|\mathrm{VC}^{approx}| = |\cup_i \mathrm{VC}_i^{approx}| = \sum_i |\mathrm{VC}^{approx}|$. Since $E_i \neq \emptyset$, $|\mathrm{VC}^{opt} \cap V_i| \geq 1 \ \forall\ i$. Then

$$\frac{|\mathrm{VC}^{approx}|}{|\mathrm{VC}^{opt}|} = \frac{|\cup_i \mathrm{VC}_i^{approx}|}{|\cup_i (\mathrm{VC}^{opt} \cap V_i)|} = \frac{\sum_i |\mathrm{VC}_i^{approx}|}{\sum_i |\mathrm{VC}^{opt} \cap V_i|} \leq \max_i \frac{|\mathrm{VC}_i^{approx}|}{|\mathrm{VC}^{opt} \cap V_i|}. \qquad \square$$

Lemma 1 implies that we can find an approximate vertex cover for a graph by iteratively finding an approximate vertex cover for small sections of the original graph, until hopefully we have an approximate vertex cover for the whole graph. The way we will use Lemma 1 is to iteratively break off small pieces of the graph. We are not claiming that finding a vertex cover for each subproblem implies that we have found a vertex cover of the whole graph. The lemma contains the restriction that the union of the vertex covers of the subgraphs is a vertex cover for the whole graph.

DEFINITION 1. *An* optimal graph reduction *is a mapping from a graph $G = (V, E)$ to a graph $G' = (V', E')$ with the property that if we have an optimal vertex cover VC'_{opt} for G', then we can create an optimal vertex cover for the original graph G from VC'_{opt} and from the graph reduction that we performed on the graphs.*

DEFINITION 2. *A* ρ-approximating graph reduction *is a mapping from a graph $G = (V, E)$ to a graph $G' = (V', E')$ such that if we have an optimal vertex cover VC' for G', then we can create a ρ-approximate vertex cover for G from VC' and from the graph reduction that we used.*

We will use both optimal graph reductions and approximate graph reductions. Most of the approximate reductions have approximation ratio $\rho \leq \frac{3}{2}$, but we will introduce reductions that have higher approximation ratios. An operation we will use extensively is a *vertex contraction*.

DEFINITION 3. *The* contraction of a set of vertices *$v_1, \ldots, v_k$ to a new vertex v is an operation where we replace the vertices $v_1, \ldots, v_k$ with a new vertex v, delete all edges between removed vertices, and replace each edge (v_i, u) with an edge (v, u). The set of vertices adjacent to v is the union of the vertices in $V \setminus \{v_1, \ldots, v_k\}$ that were adjacent to $v_1, \ldots, v_k$.*

When we perform a vertex contraction, we replace multiple edges that might appear with a single edge and encode information about the contracted vertices and adjacent edges so that we can recreate them later to get the original graph.

We will use at least ten different graph reductions, and each reduction focuses on particular properties of the graph. Most of these reductions try to find a subgraph of a special kind and then remove or contract the vertices in that subgraph. We will use the graph reductions one after the other on disjoint parts of the graph. By applying the graph reductions, we are effectively partitioning the graph according to Lemma 1 so that any vertex will be included in at most a single graph reduction. The only exception to this rule is a graph reduction called degree-two vertices, which will be explained in the next section. Once we determine whether a vertex should be included in the approximate vertex cover, we remove that vertex from the graph. Removing such a vertex from the graph is equivalent to marking whether it is included in the cover or not; by removing the vertex we are simply reducing the computational effort for later calculations.

Since we are interested in finding an approximate vertex cover, and each reduction we use has a particular approximation ratio, we will, as a first step, group the reductions into optimal graph reductions and approximate graph reductions, according to Definitions 1 and 2.

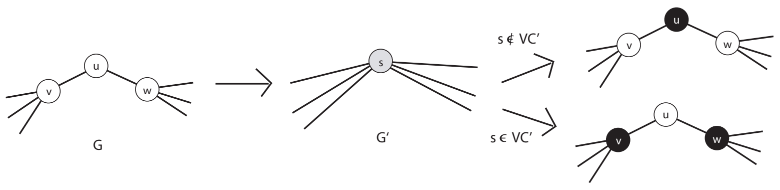

Fig. 1. *The degree-two vertex elimination. The black vertices are included in the vertex cover.*

2.1. Optimal graph reductions. We will now look at some optimal graph reductions. We start by looking at reductions that focus on vertices of low degree, namely, vertices that have no more than two neighbors. These reductions are well known and have been used previously [1]. For completeness we will explain these reductions. We will also introduce a new optimal graph reduction, the extended network flow method, which is a simple yet powerful idea that to our knowledge has not been used before.

Degree-zero and degree-one vertices. Any vertex of degree 0 or 1 can be instantly removed from any graph when we are trying to find a vertex cover. These reductions are the simplest reductions we have, but they are necessary and useful, especially when we are trying to clean up any vertices that remain after other reductions have removed vertices and edges from a graph.

Claim. *A vertex of degree* 0 *is not in any optimal vertex cover.*

Claim. *Let u be a vertex of degree* 1*, and w be its neighbor. Then there is an optimal vertex cover C such that $w \in C$ and $u \notin C$.*

A vertex with no adjacent edges is trivially not in an optimal vertex cover. If we have a vertex u of degree 1, then we need either u or its neighbor in any feasible vertex cover. If we select u to be in the cover, we might also have to include its neighbor, in order to cover some other edges. However, if we select the neighbor, we never have to include u in the cover.

Degree-two vertices. When we are removing a vertex of degree 2, there are two cases. If the neighbors of the degree-two vertex are adjacent, then we can easily remove the degree-two vertex and its neighbors, but if the neighbors are not adjacent, we can only contract the three vertices.

Claim. *If there is a degree-two vertex u whose neighbors v and w are adjacent, then there is an optimal vertex cover that includes both v and w and not u.*

Proof. Let u be a vertex of degree 2 with v and w as adjacent neighbors. To cover the edge (v, w), at least one of v or w must be in any vertex cover. By removing either the vertex v or w and all adjacent edges, u becomes a degree-one vertex. By the previous claim, we then remove the only neighbor of u. If we removed v, then w is the only neighbor of u, and vice versa. Since we will always remove both v and w, there is an optimal vertex cover that includes v and w but not u. ∎

Claim. *If there is a degree-two vertex u whose neighbors, v and w, are nonadjacent, we can find a new graph G' by contracting the vertices u, v, and w to a new vertex z. Given a vertex cover for G' with approximation ratio ρ, we can find a vertex cover for the original graph G with the approximation ratio ρ. Specifically, if the vertex cover for G' is optimal, then we can find an optimal vertex cover for G.*

This idea of eliminating degree-two vertices was proposed in [7] and is shown in Figure 1. Any optimal vertex cover of G will have at least one of the vertices and at most two. If there is just one of them in the vertex cover, then it must be vertex

u. Otherwise if there are two vertices in the vertex cover, then we can select v and w to be in the cover by the same argument as for vertices of degree 2 with adjacent neighbors. Let $\mathrm{VC}^{opt}(G)$ be the optimal vertex cover for G, and let G' be the graph we get by contracting the vertices u, v, and w. Then $|\mathrm{VC}^{opt}(G)| = |\mathrm{VC}^{opt}(G')| + 1$, since we need exactly one additional vertex in any feasible vertex cover for G' to cover the edges (v, u) and (w, u). If z is in the vertex cover for G', then v and w will be in the vertex cover for G, and if z is not in the vertex cover for G', then only u will be in the vertex cover for G. Since $|\mathrm{VC}^{opt}(G)| = |\mathrm{VC}^{opt}(G')| + 1$, any approximation ratio for a vertex cover of G' holds for the vertex cover of G. Since contracting a degree-two vertex and its neighbors is an optimal graph reduction, we can use it recursively on the graph. Of all the graph reductions that we have, the degree-two reduction is the only graph reduction that can be used in such a way.

Network flow. The vertex cover problem can be formulated as the following integer program:

$$\begin{aligned} \min \quad & \sum_i x_i \\ \text{s.t.} \quad & x_u + x_v \geq 1 \qquad \forall\, (u, v) \in E, \\ & x_i \in \{0, 1\} \qquad \forall\, i \in V. \end{aligned}$$

By solving the linear programming relaxation of the vertex cover problem we get a fractional solution. Nemhauser and Trotter [26] showed that the solution of the linear program can be used to find a partial solution to the vertex cover problem. Given an optimal solution x^* to the linear programming relaxation, define $P = \{u \in V | x_u > 0.5\}$, $Q = \{u \in V | x_u = 0.5\}$, and $R = \{u \in V | x_u < 0.5\}$. We can show that there is an optimal vertex cover that is a superset of P and disjoint from R. Hence we can solve the linear programming relaxation of the vertex cover and remove all vertices that correspond to solution variables with values not equal to 1/2. It is well known that this problem can be solved as a network flow problem [19]. If we solve the linear program as a network flow problem, we get the additional benefit that the solution is guaranteed to be $\frac{1}{2}$-integral. A half-integral solution means that the optimal solution will contain only the values 0, $\frac{1}{2}$, and 1.

Extended network flow. One of the problems with the network flow algorithm is that it tends to find solutions with many variables equal to $\frac{1}{2}$, even when other solutions exist with more variables either 0 or 1. In order to find as many non-half solution variables as possible, we first solve the network flow problem and remove all variables with values not equal to $\frac{1}{2}$. Then, using the optimal solution, we try to set each variable with value $\frac{1}{2}$ equal to 1 and resolve. If the objective value does not change, we keep the new value, remove all variables with values equal to 0 or 1, and then repeat the procedure. If the objective value increases when we set a variable equal to 1, we set the value back to $\frac{1}{2}$ and try the next variable.

The extended network flow method works well in practice on sparse graphs where the size of the optimal vertex cover is close to half the number of vertices. The method is less successful when the size of the optimal vertex cover is closer to the number of vertices in the graph, since the optimal solution to the linear programming (LP) relaxation tends to be an all half solution. To our knowledge, the approach of fixing each variable and resolving the network flow has not been used previously. Our implementation is based on a bipartite matching and unit capacity flow algorithm from Andrew Goldberg's Network Optimization Library [15].

2.2. Approximating graph reductions. So far we have introduced graph reductions that maintain optimality. While the optimal reductions can find solutions to simple examples, they are usually not enough to solve difficult graphs. Optimal graph reductions are also sometimes used as a preprocessing step to decrease the size of the graph before finding an optimal vertex cover [1]. We will introduce graph reductions that do not maintain optimality but guarantee a worst-case approximation ratio. The approximate reductions in this section use Lemma 1 extensively. We try to find a subgraph with certain properties and reduce the subgraph while ensuring that all edges between the subgraph and the remaining graph are covered by our reduction. By Lemma 1 we can use the graph reductions repeatedly and get an approximate solution to the vertex cover problem with an approximation ratio equal to the largest approximation ratio of these reductions.

Triangles. The triangle elimination is the most powerful reduction that we have, but it is also one of the simplest.

Claim. *For any clique of size k, at least $k-1$ of the vertices are in any vertex cover.*

Claim. *If the vertices v, u, and w form a triangle, then we can include all three vertices in a vertex cover for a $\frac{3}{2}$-approximation.*

A triangle is a clique of size three, so at least two of the vertices must be in any vertex cover. Hence if we include all three vertices in the vertex cover, we get a $\frac{3}{2}$-approximation.

Five-cycles. Removing a cycle of length five gives us an approximation ratio of $\frac{5}{3}$, which is higher than our limit of $\frac{3}{2}$. We include the five-cycle reduction for completeness, but since our experiments focus on reductions that have an approximation ratio of $\frac{3}{2}$ or less, we do not use it in our experiments.

Claim. *If the vertices v_1, v_2, v_3, v_4, and v_5 form a chordless cycle of length five, then we can include all five vertices in a vertex cover for a $\frac{5}{3}$-approximation.*

Any chordless five-cycle has at least three of its vertices in any vertex cover, so by selecting all five vertices we get a $\frac{5}{3}$-approximation. This approximation ratio is the largest of all the reductions that we have introduced so far. If we want to get a $\frac{3}{2}$-approximate vertex cover, then we cannot use the five-cycle reduction. A similar argument gives us a reduction for chordless cycles of any odd length $2k-1$ for $k \geq 2$, with an approximation ratio of $\frac{2k-1}{k}$.

2.3. Almost-bipartite method. When we are trying to solve NP-hard problems it is often helpful to look for special cases. For many problems, while the general problem is NP-hard, there are special cases that yield an optimal solution in polynomial time. One such example is the vertex cover for bipartite graphs [24]. The vertex cover problem is well known to be NP-hard for general graphs, but for the special case of bipartite graphs, we can find an optimal vertex cover in polynomial time using maximum matching. It is also well known that the size of the minimum vertex cover for bipartite graphs is equal to the size of the maximum matching [8].

We can use the fact that it is easy to find an optimal vertex cover for a bipartite graph to try to find approximate vertex covers for general graphs. If we have a lower bound of k for the optimal vertex cover, any vertex cover of size no more than pk is a p-approximate vertex cover.

We partition the vertices of the graph into three sets, A, B, and C, such that the subgraph induced by the first two sets of vertices, A and B, will be bipartite. The last set, C, will include all vertices that violate the bipartite property if they are added to the subgraph, i.e., each vertex in the set C has neighbors in both sets A and B. The term *almost-bipartite* indicates that the size of the last group does not dominate the number of vertices in the graph. However, the success of the almost-bipartite method depends on the size of the last group compared to the other groups and also on the approximation ratio that we want to achieve and the lower bound on the optimal vertex cover. We will see that if the approximation ratio is $\frac{3}{2}$ and the lower bound on the size of the optimal vertex cover is $\frac{n}{2}$, then the set of vertices that violate the bipartite property can include half of the vertices of the graph, and the almost-bipartite method will still work.

CLAIM. *Let $G = (V, E)$ be a graph. Assume that we partition V into three sets, A, B, and C, with the property that the subgraph G_{AB} induced by the sets A and B is bipartite. Let $VC^*_{G_{AB}}$ be an optimal vertex cover for G_{AB}. Also let VC_{LB} be a lower bound on the size of the optimal vertex cover for the original graph, and let p be the approximation ratio that we want to achieve. Then, if $|VC^*_{G_{AB}}| + |C| \leq p\,VC_{LB}$, the set $VC^*_{G_{AB}} \cup C$ is a p-approximate vertex cover for G.*

Proof. The vertex cover $\mathrm{VC}^*_{G_{AB}}$ covers all edges in G_{AB}, and all other edges in G are covered by the set C, so $\mathrm{VC}^*_{G_{AB}} \cup C$ is a feasible vertex cover for G. If VC_{LB} is a lower bound on the optimal vertex cover for G, then any feasible vertex cover whose size is smaller than $p\mathrm{VC}_{LB}$ is a p-approximate vertex cover for G. We know that $\mathrm{VC}_{LB} \leq |\mathrm{VC}^*|$, where VC^* is an optimal vertex cover for G. If the sum of the size of the set C and the size of the optimal vertex cover on the bipartite subgraph G_{AB} is smaller than $p\mathrm{VC}_{LB}$, then

$$|\mathrm{VC}^*_{G_{AB}}| + |C| \leq p\mathrm{VC}_{LB} \leq p|\mathrm{VC}^*|,$$

and $\mathrm{VC}^*_{G_{AB}} \cup C$ is a p-approximate vertex cover for G. ◻

In our experiments, the approximation ratio p is usually equal to $\frac{3}{2}$. The almost-bipartite method is a success if $|\mathrm{VC}^*_{G_{AB}}| + |C| \leq p\mathrm{VC}_{LB}$.

The almost-bipartite method takes as input a graph $G = (V, E)$ and outputs three sets of vertices, sets A, B, and C, such that the subgraph induced by the sets A and B is bipartite. The algorithm is a simple greedy selection algorithm which processes the vertices in some order and partitions them into the three sets, with priority given to sets A and B. A vertex is included in set C only if it has neighbors in both A and B at the time it is processed. The method always returns these three sets, but the success of the method depends on the size of the set C and on the size of the optimal vertex cover on the subgraph induced by the sets A and B. Since the subgraph induced by the sets A and B is bipartite, we can find an optimal vertex cover in polynomial time.

The problem of finding a maximum sized bipartite subgraph in a graph is NP-hard [9]. The problem is a special case of the problem of finding a maximum induced subgraph with property P, where property P is hereditary and nontrivial. A property P of graph G' is hereditary if every subgraph of G' also satisfies P. In our case, P is the bipartite property. The problem of finding a maximum induced subgraph with property P is approximable within $O(\frac{n}{\log(n)})$, where n is the number of vertices in G. However, the problem is not approximable within n^ϵ for some $\epsilon > 0$ unless $P = NP$ [9]. In [20], Hüffner looked at exact algorithms for the GRAPH BIPARTIZATION problem, where the objective is to find a minimum set of vertices to delete from a graph to make it bipartite. However, the worst-case complexity is $O(3^k mn)$, where k is the number of vertices to delete. In our case, $k \geq \frac{n}{4}$, which makes the running time exponential in the number of vertices.

2.4. Other reductions. The crown reduction is an optimal reduction that was introduced by [1]. Here we try to find an independent set of vertices S such that there exists a matching on the edges connecting S and its neighborhood, $N(S)$, that matches all the vertices in $N(S)$. If we can find such a set, then we can include the neighborhood set in the vertex cover and exclude all the vertices in the independent set. Abu-Khzam et al. [1] proved that this reduction is an optimal reduction. However, it is easy to show that this reduction is subsumed by the extended network flow method.

Greedily decreasing the vertex cover. After we find an approximate vertex cover, we run a simple greedy algorithm to eliminate vertices from the vertex cover. We look at each vertex in the cover and check whether all its neighbors are also in the vertex cover. If all the neighbors are in the cover, we remove the vertex from the cover to decrease the size of the vertex cover.

3. Order of reductions. The reductions introduced in section 2 are all simple and straightforward. We can use them in any order, but since the reductions focus on different characteristics of the graph and have different effects, applying them in different orders will give different results. When we automate the reductions, we must take the interaction between the reductions into account, since the wrong choices can leave us in a dead end, while using the reductions in a different order might have solved the problem. We can group the reductions, based on whether they are optimal or not:

- *Optimal reductions:* degree-zero, degree-one, degree-two with adjacent neighbors, degree-two, extended network flow.
- *Approximate reductions:* triangle elimination, five-cycles.
- *Special case:* almost bipartite.

Since the five-cycles reduction has a higher approximation ratio than $\frac{3}{2}$, we will not use it unless the other reductions cannot solve a graph, but we keep it in the list for completeness. The almost-bipartite reduction is a special case, since in order to use it, the graph must fulfill certain conditions. We will focus on the almost-bipartite reduction in section 3.1.

We did many experiments with different orderings and different approaches to the reductions and measured how they affected the final result. In our experiments, we particularly focused on the use of triangle elimination. The triangle elimination proved to be very effective on almost all of the graphs, but it is also responsible for the largest approximation ratios. After trying different approaches to the triangle elimination, such as sorting the triangles we found on the combined degrees of the vertices in the triangle and prioritizing the triangles with high total degree or removing a single triangle at a time while emphasizing the optimal reductions, our experiments showed that the most efficient use of the triangle elimination is to simply remove all triangles greedily in the order in which they are found.

An example of beneficial interactions between the reductions is that between the triangle elimination and the degree-two vertex contraction. If there exists a cycle of length five where one of the vertices has degree 2, then by contracting that vertex, a triangle is created. The removal of triangles also decreases the degrees of adjacent vertices, increasing the chances of the low degree method's being applicable.

There were some graphs where the triangle elimination, the low degree reductions, and the extended network flow were not enough to solve the graph. For these graphs, we turned to the almost-bipartite method to find an approximate vertex cover.

3.1. The special case of almost bipartite. The almost-bipartite method is somewhat difficult to combine with the other reductions since it needs a good lower bound on the size of the optimal vertex cover in order to work. Also, if the graph does not include a large enough bipartite subgraph, this reduction does not remove anything from the graph. However, if we can overcome these obstacles, then the almost-bipartite reduction is very powerful because if it works, then it will completely solve the graph. In our case, if we first run the extended network flow method, it will give us a lower bound of $\frac{n}{2}$ on the minimum vertex cover. The triangle elimination removes all triangles from the graph, which results in a graph that is closer to being bipartite. Hence, by running the almost-bipartite method only after we have used the extended network flow method and triangle elimination, we can guarantee that there is a good lower bound on the optimal vertex cover. Since we focus on the approximation ratio of $\frac{3}{2}$ and the extended network flow method gives us a lower bound of $\frac{n}{2}$ on the optimal vertex cover, we can use any vertex cover of size no more than $\frac{3n}{4}$ as a $\frac{3}{2}$-approximate vertex cover. This means that if the size of the set of vertices that violate the bipartite property of the original graph is no more than $\frac{n}{2}$, then we can easily find a $\frac{3}{2}$-approximate vertex cover.

3.2. The automated order of reductions. After much experimentation, we settled on using the following order of reductions to automate the approximation process. We run the extended network flow method, triangle elimination, and low degree in a loop until we find a solution or no improvements are made during an iteration. If the graph is not empty, we try the almost-bipartite reduction. The stopping criteria is either having processed all the vertices from the graph or after running the almost-bipartite method. If the almost-bipartite method is successful, then we have an approximate vertex cover. However, if the almost-bipartite method is not successful, we stop and must use some other methods, such as branch-and-bound, to get a solution. If we find a solution, then the final step in the algorithm is to use a simple greedy algorithm to eliminate unnecessary vertices from the cover. In our experiments we almost never had to resort to branch-and-bound; our algorithm managed to solve almost every single graph we found.

4. Experiments on designed graphs.

In our first experiments, we focused on graphs that were specially designed to be difficult for specific graph problems. The graph problems for which the problems were designed were not only the vertex cover problem and the closely related independent set problem, but also minimum coloring, maximum clique, and problems from biology. The idea behind this set of experiments was to use instances that are designed to be difficult for some specific problem and might therefore have specific properties that might be hard for our heuristics to solve. To gather these problem instances, we did an extensive search on the Internet for datasets for vertex cover and other problems, and also gathered all datasets we could find from previous works. We also used the complement graph of every single instance that we could find.

1. From the DIMACS website we took 78 graphs that were used as a challenge for the MaxClique problem [6]. The complement graphs for these graphs are of special interest: Let VC^{opt} be an optimal vertex cover for the graph $G = (V, E)$. Then the set $V \setminus \mathrm{VC}^{opt}$ is a maximum clique in the complement graph $\bar{G} = (V, \bar{E})$.
2. From the DIMACS challenges we obtained 60 benchmark graphs for the Min-Color problem.
3. From the DIMACS challenges, we found five benchmark graphs for the vertex cover problem.

4. sh2-3.dim and sh2-10.dim are graphs used in [1]. These graphs were obtained from the biological data repositories NCBI and SWISS-PROT.
5. From [28] we found four small graphs used as a benchmark for vertex cover algorithms. These graphs are of special interest because the optimal vertex cover is known.
6. We generated 32 graphs using Sanchis' graph generator [27]. These graphs have 500 vertices each, with numbers of edges ranging from 2000 to 110,000. We split these graphs into three groups, with maximum clique sizes of 2, 4, and 10. These groups were called group A, B, and C, respectively. The reason we focused more on graphs with small cliques is that the triangle elimination is just too powerful on graphs with large cliques, leaving at most two vertices from a clique of size greater than two. Other parameters for the graph generator were $r = 10$, 120, 49 for the three groups, A, B, and C and a random seed value of 12,345 for all graphs.

5. Results for designed graphs. We solved every one of the 362 graphs in our collection of graph problems, finding a $\frac{3}{2}$-approximate vertex cover in under 5 minutes for each one. The running time for most of the graphs was less than a second. Moreover, in only one case did we need to use any reduction other than the extended network flow method, triangle elimination, or low degree reduction. In that case, either the almost-bipartite method or a simple 3-degree reduction finished off the graph. The triangle elimination is very powerful; on average it removed 88.78% of the vertices and 93.34% of the edges from each graph. The extended network flow method removed on average 5.09% of the vertices and 5.56% of the edges from each graph, while low degree reductions averaged 6.12% of the vertices and 1.10% of the edges from each graph.

These graphs have a total of 194,126 vertices and 50,216,078 edges. In total, the triangle elimination removed 183,577 vertices and 49,909,113 edges from these graphs, which is about 94.57% of the total number of vertices and 99.39% of the total number of edges. The extended network flow method removed 8,203 vertices and 292,510 edges, while low degree reductions were responsible for 7,575 vertices and 8,640 edges. Other methods were hardly used in these experiments; we needed them on only a single graph. The almost-bipartite reduction solved that graph easily by removing 35 vertices and 74 edges. The average numbers for the fraction of edges and vertices that the triangle elimination removed are higher when we look at the total number of edges and vertices than when we look at the average fraction for each graph. This is because there are a few very large and dense graphs where the triangle elimination removes almost all of the edges and the vertices. The number of vertices and edges removed from those large and dense graphs dominates the sum when we look at the total number of vertices and edges removed.

The experiments were run on a machine with a 1.6GHz Intel Pentium 4 and 512MB RAM. The largest running time we saw for the extended network flow method was just under 4 minutes on the graph MANN_a81.clq with 3,321 vertices and 5,506,380 edges, even though it did not manage to remove anything from that graph. The largest running time for the triangle elimination was 56 seconds on the same graph, removing every single vertex and all the edges. These running times are, however, largely related to paging; for smaller graphs with less than 500,000 edges, the running time for each reduction was just a fraction of a second.

The MaxClique-complement graphs are of special interest since many of them have a known optimal solution. The vertices not in the vertex cover form an independent

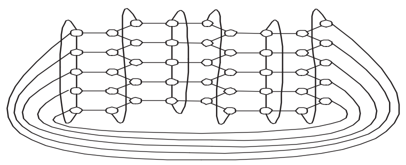

FIG. 2. *A simple graph that highlights the need for the almost-bipartite method.*

set, which is a clique in the complement graph. Since we are trying to minimize the size of the vertex cover, it is equivalent to maximizing the clique size. Of the 75 MaxClique-complement graphs, we had optimal solutions to 48 of them. The average approximation ratio was 1.043, and the largest ratio was 1.459.

Since our reductions perform so well and we manage to find an approximate vertex cover for every graph, we will go into details only about the most difficult and interesting problems.

Worst approximation ratio. The graphs on which our algorithms got the worst approximation ratios were the complement graphs of the MANN series. These graphs are made by Carlo Mannino and are a part of the MaxClique challenge on the DIMACS web page. These graphs are a clique formulation of the Steiner triple problem, and they show how easy it is to create graphs where our algorithms find vertex covers with approximation ratios close to $\frac{3}{2}$. They include a large set of independent triangles, and the optimal vertex cover includes only two vertices from each triangle, while the triangle elimination includes all three vertices from each triangle.

The method of greedily decreasing the vertex cover after we have found a feasible cover is very inconsistent. In the worst cases, it does not help at all, while in one of the best cases we manage to decrease the size of the vertex cover on a graph with 200 vertices from 198 vertices down to the optimal vertex cover of 142 vertices.

Most challenging problems. Some complement graphs of the "san200" and "san400" series from the DIMACS challenges were the most challenging in the sense that we could not solve them in only a couple of iterations through the reduction, forcing us to iterate through low degree, triangle elimination, and extended network flow to get a result while still having approximation ratio over 1.2.

The almost-bipartite method is necessary. Even though we could have found an approximate vertex cover for every graph in our collection of graphs without using the almost-bipartite method, it is easy to construct graphs where we would get stuck if we did not have the almost-bipartite method. One example of such a graph, consisting of connected five-cycles, is shown in Figure 2.

The triangle elimination, extended network flow method, and the low degree reductions all have no effect on this graph, so if we do not use the almost-bipartite reduction, the only thing we can do in this case is to use 3-degree reduction to decrease the size of the graph, and then we must turn to branch-and-bound or some other method to get a solution. However, using the almost-bipartite method, this instance is easily solvable. Section 6 will show how vital the almost-bipartite method is for finding approximate vertex covers on random graphs.

Table 1 shows how the reductions are performed on a selected subset of the graphs. The first column is the name of the graph, and then we have the number of vertices and number of edges. The next column shows by how much a simple greedy approach managed to decrease the size of our vertex cover. Next is the size of the vertex cover we found. Then we have the performance of the extended network flow

TABLE 1
Detailed performance of the extended network flow method, low degree reductions, and the triangle elimination on a small subset of the graphs.

Name	n	m	VCdec	$\|VC\|$	ENF n	ENF m	ENF time	LD n	LD m	LD time	T n	T m	T time
MaxClique													
brock800_3.clq	800	207333	4	794	0	0	0.161	2	1	0	798	207332	0.171
C4000.5.clq	4000	4000268	8	3989	0	0	13.734	4	3	0	3996	4000265	10.392
c-fat500-2.clq	500	9139	0	481	0	0	0.016	44	59	0	456	9080	0.005
johnson32-2-4.clq	496	107880	10	465	30	190	0.056	1	0	0	465	107690	0.101
MANN_a81.clq	3321	5506380	2	3318	0	0	229.387	0	0	0	3321	5506380	55.991
p_hat1500-2.clq	1500	568960	25	1473	0	0	0.639	3	1	0	1497	568959	0.722
MaxClique-comp.													
C1000.9.clq(comp)	1000	49421	27	952	12	16	0.165	28	49	0	960	49356	0.045
hamming10-2.clq(comp)	1024	5120	0	512	1024	5120	0.004	0	0	0	0	0	0
keller6.clq(comp)	3361	1026582	29	3330	0	0	1.536	1	0	0	3360	1026582	0.318
MANN_a45.clq(comp)	1035	1980	0	990	0	0	0.04	45	0	0	990	1980	0.002
MinColor													
DSJC1000.1.col	1000	49629	26	956	0	0	0.172	28	28	0	972	49601	0.081
flat1000_50_0.col	1000	245000	6	980	0	0	0.272	22	56	0	978	244944	0.294
le450_25a.col	450	8260	12	370	0	0	0.034	123	332	0	327	7928	0.01
R250.1.col	250	867	8	190	10	41	0.007	90	125	0	150	701	0
zeroin.i.2.col	211	3541	0	84	108	3342	0.003	79	1	0	24	198	0
MinColor-comp.													
DSJR500.1c.col(comp)	500	3475	10	438	0	0	0.014	98	134	0	402	3341	0.001
R1000.5.col(comp)	1000	261233	186	808	0	0	0.343	10	12	0	990	261221	0.216
Sanchis' generator													
s17.vc	500	70000	124	375	0	0	0.059	2	1	0	498	69999	0.025
s20.vc	500	2000	18	337	3	17	0.07	278	393	0	219	1590	0.008
s22.vc	500	10000	16	454	0	0	0.04	56	82	0	444	9918	0.009
sh2 problems													
sh2-3.dim.sh	839	5860	0	246	839	5860	0.006	0	0	0	0	0	0
sh2-10.dim.sh	839	129697	92	644	44	167	0.244	78	135	0	717	129395	0.149
sh2-10.dim.sh.pp	726	69982	54	529	101	11222	0.213	124	384	0.001	501	58376	0.091
VC benchmarks													
vtx_cov_3.gph	100	200	3	56	8	40	0.002	74	98	0.001	18	62	0

method (ENF), low degree (LD), and triangle elimination (T) reductions. We show how many vertices and edges each method removed from each graph, and the total time it took in seconds. Due to space constraints we show only a small selection of the results we have.

Looking at these results, it is clear that for dense graphs (C4000.5.clq, MANN_a81.clq), the triangle elimination is very powerful, while the extended network flow method works well on sparser graphs (hamming10-2.clq(comp), zeroin.i.2.col, sh2-3.dim.sh). The three largest graphs show an extreme case of how the running time increases when the size of the graphs increases; the triangle elimination takes about 0.3 seconds on a graph with just over 1,000,000 edges, while for a graph with 4,000,000 edges this takes over 10 seconds, and then up to 56 seconds on a graph with around 5,500,000 edges. Most of this time increase is due to the fact that these largest graphs do not fit into memory, so the performance of the algorithms takes a hit. The time it takes to do low degree elimination is so small that it is hardly measurable, even though this method is very helpful in some cases (s20.vc).

6. Experiments on random graphs.

The graphs that we have solved so far are graphs that are specially designed and created with some property in mind, usually to be difficult to solve for a specific problem. Graphs that are designed and created for a specific purpose are more likely to include a specific structure or have certain properties that could make it easier for our reductions to find an approximate vertex cover. As we saw in the previous section, we managed to solve every single graph that we looked at, by using only reductions that have approximation ratio no more than $\frac{3}{2}$. Since our reductions worked so well on graphs that were designed for a specific purpose, we wanted to see how the reductions perform on graphs that are completely random.

The random graphs we generated were based on a simple random graph model by Erdős and Rényi [12], where we have a fixed set of vertices V with size $|V| = n$ and each possible edge appears in the graph with probability ρ, where ρ was randomly selected for each graph.

We used four differently sized sets of vertices. The first three sets had $n = 200$, 500, and 1000, respectively. For the set of 200 vertices, we created 20,000,000 random graphs. For the set of 500 vertices, we created 1,000,000 random graphs, and for the largest set of 1000 vertices we generated 100,000 random graphs. The last set consists of 5000 graphs with 10,000 vertices each. These large graphs were specially designed to be difficult; the graphs are sparse with edge density parameter less than 0.1. The only graphs that we could not solve using our reductions came from this last set.

6.1. Using the almost-bipartite reduction.

The almost-bipartite method is a special case. The almost-bipartite method is the only reduction that is guaranteed to finish the graph if it works, and for which the graph needs to fulfill specific conditions in order for us to use it. Since the almost-bipartite method is different from every other reduction we have, we wanted to see how the almost-bipartite method fits with the other reductions and how crucial it is when we are trying to find approximate vertex covers. In Figure 2 we saw an example of a graph that we cannot solve without using the almost-bipartite reduction. In section 4, where we used graphs that were designed for specific graph problems, there was only a single graph that needed the almost-bipartite method. However, when we started looking at random graphs, we encountered many graphs where we had to use the almost-bipartite reduction to find a solution, even though the majority of the graphs were solved by using only the triangle elimination, the extended network flow method, and the low degree reductions.

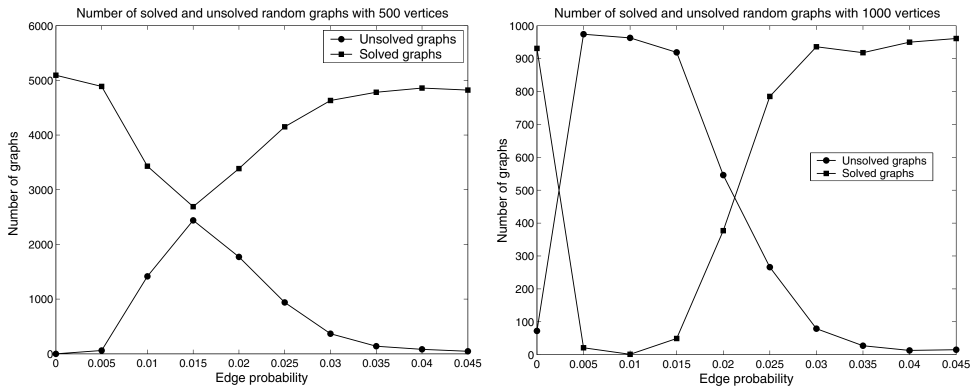

FIG. 3. *The number of random graphs that were solved without using the almost-bipartite method and the number of graphs that could not be solved without the almost-bipartite method. The left graph shows the results for randomly generated graphs on* 500 *vertices, and the right graph shows the results for random graphs with* 1000 *vertices.*

Figure 3 shows the fraction of graphs that were solved as a function of the edge probability. We see that the graphs that were not solved without the almost-bipartite method all have very similar edge probabilities. For a random graph with 500 vertices, if the edge probability is between 0.005 and 0.03, then it is likely that the graph will include a subgraph that has properties similar to the bad case in Figure 2, and will be difficult to solve without using the almost-bipartite method. For random graphs on 1000 vertices, using an edge probability between 0.005 and 0.015 almost guarantees that the resulting random graph cannot be solved without the almost-bipartite method. It is more difficult to generate random graphs with 200 vertices that cannot be solved without the almost-bipartite method, since only about one out of every thirty graphs with edge probability between 0.03 and 0.045 cannot be solved without using the almost-bipartite method. The graphs in Figure 3 indicate that there are some subgraphs that are very difficult to solve by using only the triangle elimination, the extended network flow method, and the low degree reductions, and the probability that the graph will include such a subgraph increases as the number of vertices grows. The graphs that could not be solved without the almost-bipartite method are very sparse, and since we have removed all triangles and have a good lower bound on the optimal vertex cover from the extended network flow method, the graphs are well suited for the almost-bipartite method. Our experiments show that almost every single instance that could not be solved using only the triangle elimination, the extended network flow, and the low degree reductions was solved easily when we used the bipartite reduction.

There are some similarities between the behavior shown in Figure 3 and a greedy algorithm proposed by Karp and Sipser [23] for finding maximum matchings in random graphs. In 1998, Aronson, Frieze, and Pittel [2] analyzed the performance of the Karp and Sipser algorithm when used on sparse random graphs, $G_{n,M}$, where the number of edges, M, is given by $M = \lfloor \frac{cn}{2} \rfloor$, where c is a constant. They showed that the performance of the algorithm is dependent on the constant c, and $c = e$ is a critical value. When $c < e$, they showed that the algorithm finds a matching with high probability, while if $c > e$, then with high probability the algorithm finds a matching that is within $n^{\frac{1}{5}+o(n)}$ of maximum size. The threshold at $c = e$ is often referred to as the e-phenomenon. Our reductions show behavior that resembles this e-phenomenon.

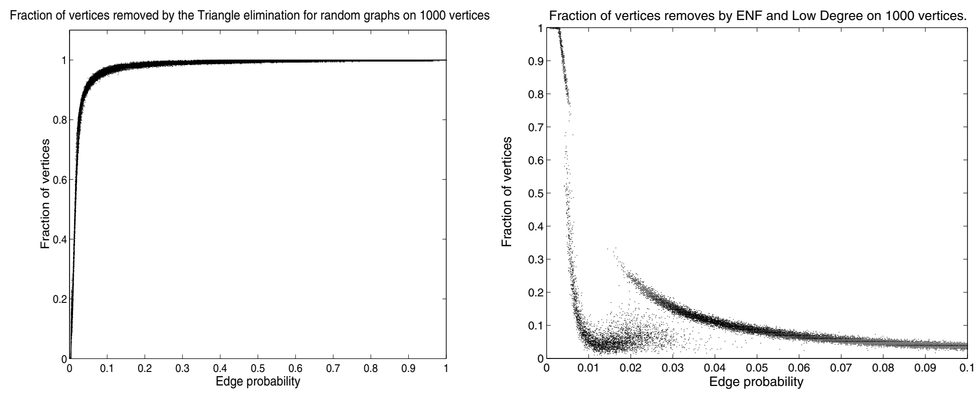

FIG. 4. *The fraction of vertices removed by the triangle elimination and the total fraction of vertices removed by low degree methods and the extended network flow method. The left graph shows the results for the triangle elimination on random graphs with* 1000 *vertices. The right graph shows the combined fraction of vertices removed by the low degree method and extended network flow on random graphs with* 1000 *vertices. The right graph is zoomed in to better illustrate the performance of the methods.*

Since we are using a different model of random graphs and the graphs are not very large, the threshold is not as sharp in our case. For random graphs on 200, 500, and 1000 vertices, the threshold $c = e$ corresponds to edge probabilities of 0.0137, 0.0054, and 0.0027, respectively, and it is around these values that our reductions first start to show erratic behavior.

7. Results for random graphs.

By using our reductions, we managed to find a $\frac{3}{2}$-approximate vertex cover for every randomly generated graph with 1000 vertices or fewer, so we will focus on the performance of each reduction and show how effective each reduction is for the smaller random graphs. The large graphs, with 10,000 vertices, are a special case since the graphs are all sparse and we only have a relatively small collection. We will analyze the large graphs specially, once we have looked at the performance of the reductions on the smaller graphs.

7.1. Triangle elimination.

As in the nonrandom case, the triangle elimination is by far the most powerful reduction in our arsenal. It is responsible for eliminating the largest number of both vertices and edges. The triangle elimination works extremely well when the graphs are dense. When the edge probability is more than 0.1, the triangle elimination usually removes more than 95 percent of both vertices and edges. The result is that, for finding a $\frac{3}{2}$-approximate vertex cover, the only interesting random graphs are the graphs that are very sparse. For the dense random graphs, the triangle elimination is simply too effective for those instances to be interesting. The left-hand graph in Figure 4 shows the fraction of vertices that the triangle elimination removes from the graphs with 1000 vertices. The graph uses a single point for every problem solved, and it is interesting to see that the fraction of vertices that the triangle elimination removes is very consistent, with almost no outliers.

7.2. Extended network flow and low degree.

The extended network flow method and the low degree eliminations combine to find approximate vertex cover when the graphs are very sparse. When we look at the success of these methods we focus on the combined effort since the order in which they are used is critical for their relative performance.

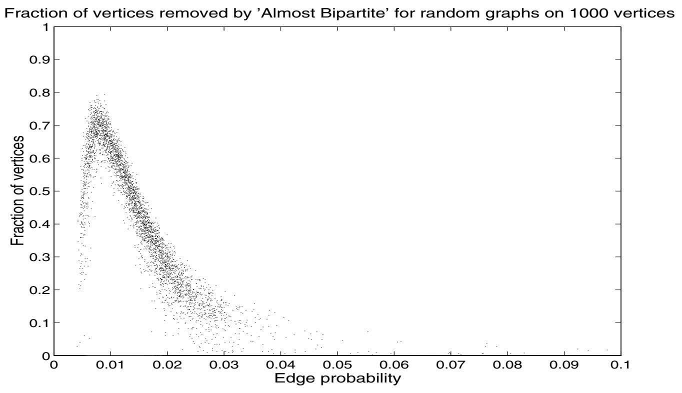

FIG. 5. *The fraction of vertices removed by the almost-bipartite method. The graphs are random graphs on* 1000 *vertices. We focus on edge probabilities less than* 0.1 *since the almost-bipartite method is seldom used on graphs where the edge probability is larger than* 0.1*.*

Figure 4 shows that the performance of the reductions is consistent for every value of edge probability, except when the edge probability is between 0.005 and 0.02 for random graphs with 1000 vertices. This gap is also obvious in Figure 3, since these are the graphs that we could not solve without using the almost-bipartite reduction.

7.3. Almost-bipartite reduction. The left-hand graph of Figure 4 shows that the results for the triangle elimination are consistent, even for the values of edge probability where the extended network flow and the low degree reductions get into trouble. The fraction of eliminated vertices drops sharply once the edge probability falls below a certain threshold, but the graph does not show any jumps or erratic behavior. On the other hand, the right-hand graph of Figure 4 shows how the extended network flow method and the low degree reductions become very inconsistent and erratic for certain values of the edge probability, and this is where the almost-bipartite method comes into play, as shown in Figure 5. We use the almost-bipartite method only once the triangle elimination, extended network flow, and the low degree reductions have all tried and failed to remove vertices from the graph. When we had to use the almost-bipartite method, it worked admirably and found a $\frac{3}{2}$-approximate vertex cover for every single instance of the graphs with 1000 vertices or less.

7.4. Comparison with the optimal vertex cover. We solved 88 of the random graphs with 200 vertices optimally by using two machines running the LINDO API solver. The first machine had 4GB of memory and used a 2.4GHz dual Intel CPU, while the second machine had 512MB of memory and used an AMD Sempron 3000+ CPU. We then compared the size of the optimal vertex cover to the approximate vertex cover that we got from our reductions. The approximation ratio follows a similar pattern as the performance of each reduction, where we find that the edge probabilities of the most interesting graphs are found in a small interval. When the edge probability is too high, the graphs are very dense and the optimal vertex covers are large, which means that any feasible cover is likely to have a good approximation ratio. However, if the edge probability is too small, we can easily solve the graphs optimally using the extended network reduction.

For random graphs on 200 vertices, when the edge probability is greater than 0.30, the size of the optimal vertex cover is usually greater than 180. This means that even if we take every single vertex in our cover, we still have an approximation ratio

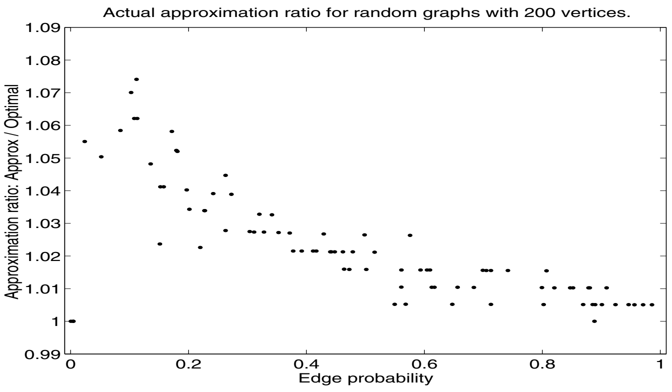

FIG. 6. *The ratio of the size of the approximate vertex cover that we get from our reductions over the size of an optimal vertex cover. The graphs are random graphs on* 200 *vertices.*

around 1.1. The vertex covers that we get by using our reductions have approximation ratios that are much better than that; the average approximation ratio over graphs with edge probability ≥ 0.3 is 1.015, and the largest approximation ratio over these dense graphs is 1.033. When the graphs are very sparse, where the size of the optimal vertex cover is less than 100, the extended network flow method usually manages to find an optimal cover, so the approximation ratio for these graphs is equal to 1. There is, however, an interval in the edge probability where the graph is not too dense and not too sparse, and in that interval there is more variation in the approximation ratio. The highest approximation ratio is 1.074, and we get that when the edge probability is equal to 0.112. The average approximation ratio for graphs with edge probability between 0.05 and 0.2 is equal to 1.052. We see that the largest approximation ratio, 1.074, is still very far from our upper limit of 1.5. The average approximation ratio is equal to 1.023, which shows that our reductions perform well on small random graphs. The comparison with the optimal results is shown in Figure 6.

We have optimal results only for small graphs, since even random graphs on 200 vertices were often very difficult to solve optimally. The most difficult problem we solved had 200 vertices and 1664 edges. The edge probability for this random graph was 0.0846. We found an optimal vertex cover of size 154 for this graph, but finding that optimal cover by using the branch-and-bound method required 3,711,654 branches with 6,835,431 different LPs, and we needed 1,577,840,155 iterations of the Simplex method. The calculations used to solve the graph optimally took more than 149 hours on the faster machine. We also found an approximate vertex cover for this graph. The approximate vertex cover had size 163, and we found that cover in less than a second. Of the 36 problems solved on the slower machine, 8 problems required more than 24 hours to be solved optimally, while 7 out of 52 problems needed more than 24 hours to be solved optimally on the faster machine. For comparison, even the largest graphs with 10,000 vertices took only a few minutes to solve approximately on the slower machine, and all graphs with 200 vertices were solved approximately within a second. The calculations needed for optimal solutions to small random graphs with 200 vertices indicate that finding optimal vertex covers for the random graphs on 500 and 1000 vertices is not computationally feasible.

7.5. Large sparse graphs. The reductions that we have introduced do not guarantee that we will find an approximate vertex cover for every single graph. How-

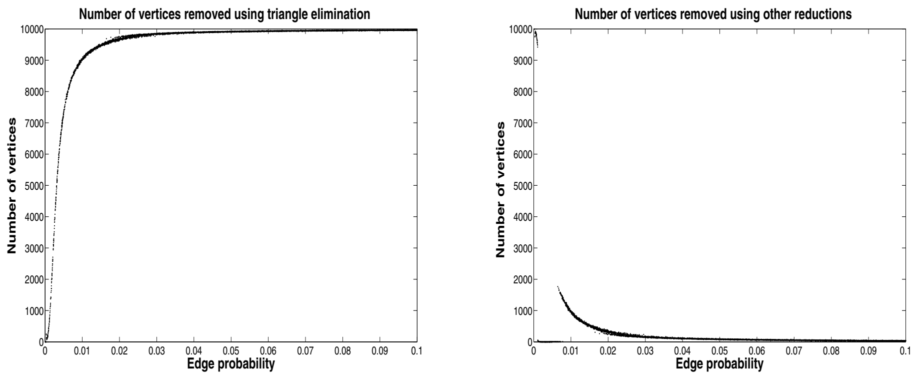

FIG. 7. *The graph on the left shows how many vertices were removed from the graphs using the triangle elimination, while the graph on the right shows the combined total of vertices removed by using the low degree methods, the extended network flow, and the almost-bipartite method. The gap in the graph on the right, for graphs with edge probability between* 0.001 *and* 0.008*, corresponds to the* 295 *graphs for which we could not find a* $\frac{3}{2}$*-approximate vertex cover.*

ever, we have managed to find approximate vertex covers for every single graph we have introduced so far. When we looked at sparse random graphs with 10,000 vertices, we finally found some graphs for which we couldn't find a $\frac{3}{2}$-approximate vertex cover. By using our graph reductions, we reduced the size of the graphs, but the reductions were not able to finish off the graphs completely.

We generated 5000 graphs with 10,000 vertices and edge probability less than 0.1. We could solve only 4705 of these 5000 graphs using the reductions with approximation ratio less than $\frac{3}{2}$. The remaining 295 graphs were solved by adding the five-cycle reductions to the set of reductions, which gives us an approximation ratio of $\frac{5}{3}$.

The 295 graphs that remain once we have used all the graph reductions have similar structural properties. The graphs are triangle-free, so the maximum clique size is 2, while the minimum degree is at least 3. Moreover, the graphs have a lower bound of $\frac{n}{2}$ on the optimal vertex cover. The unsolved graphs are also unlikely to have a large bipartite subgraph. In this case, a large bipartite subgraph means that there is a bipartite subgraph that includes the majority of the vertices. It is interesting to note the gap in the right-hand graph in Figure 7. Of the 5000 random graphs we generated, there were 1239 graphs where we could not find a $\frac{3}{2}$-approximate vertex cover using only the low degree methods, the extended network flow, and the triangle elimination. The almost-bipartite method managed to find $\frac{3}{2}$-approximate vertex covers for 944 of these 1239 graphs but failed on 295 graphs. Every graph that the almost-bipartite method could not solve had edge probability between 0.001 and 0.008. Moreover, the almost-bipartite method failed on every single graph where the edge probability was between 0.0012 and 0.0065, while succeeding on every graph with edge probability less than 0.001 or higher than 0.008.

The actual graph density of the graphs that remain after we have removed all the vertices that we can is between 0.001 and 0.007, where the smallest graph, with 1374 vertices, has the highest density and the graphs get sparser as the graphs get larger. The largest graph that remains after we have run our graph reductions has 9384 of the 10,000 vertices it started with.

8. Conclusions.

We used a collection of simple reductions where we allowed reductions that have a worst-case approximation ratio of $\frac{3}{2}$. Even though these reductions do not guarantee that we will find a solution, we ran these reductions on a wide collection of test problems from every source we could find, and by combining them, we managed to find an approximate vertex cover for every single graph in our collection of graph problems. We also tried using our reductions on over 21.5 million randomly generated graphs. The results were similar to the results for the collection of test problems; we managed to find a $\frac{3}{2}$-approximate vertex cover for almost every single random graph that we generated. Moreover, the reductions are extremely fast and easily applied, and since the bad examples have a very restrictive structure, these reductions should work well in practice.

To our knowledge, applying reductions that maintain a worst-case guarantee has not been widely studied. This approach should be applicable to other problems.

REFERENCES

[1] F. N. Abu-Khzam, R. L. Collins, M. R. Fellows, M. A. Langston, W. H. Suters, and C. T. Symons, *Kernelization algorithms for the vertex cover problem: Theory and experiments*, in Proceedings of the ACM Workshop on Algorithm Engineering and Experiments, New Orleans, LA, 2004, ACM, New York, 2004, pp. 62–69.

[2] J. Aronson, A. M. Frieze, and B. Pittel, *Maximum matchings in sparse random graphs: Karp-Sipser revisited*, Random Struct. Algorithms, 12 (1998), pp. 111–177.

[3] E. Asgeirsson and C. Stein, *Vertex cover approximations: Experiments and observations*, in Proceedings of the 4th International Workshop on Experimental and Efficient Algorithms, Lecture Notes in Comput. Sci. 3503, Springer, New York, 2005, pp. 545–557.

[4] E. Asgeirsson and C. Stein, *Vertex cover approximations on random graphs*, in Proceedings of the 6th International Workshop on Experimental and Efficient Algorithms, Lecture Notes in Comput. Sci. 4525, Springer, New York, 2007, pp. 285–296.

[5] R. Bar-Yehuda and S. Even, *A local-ratio theorem for approximating the weighted vertex cover problem*, Ann. Discrete Math., 25 (1985), pp. 27–46.

[6] D. Johnson and M. Trick, eds., *Cliques, Coloring and Satisfiability—Second DIMACS Implementation Challenge, October* 11–13, 1993, AMS, Providence, RI, 1996.

[7] J. Chen, I. Kanj, and W. Jia, *Vertex cover: Further observations and further improvements.*, J. Algorithms, 41 (2001), pp. 280–301.

[8] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, *Introduction to Algorithms*, 2nd ed., MIT Press, Cambridge, MA, 2001.

[9] P. Crescenzi and V. Kann, *A Compendium of NP Optimization Problems*, available online at http://www.nada.kth.se/theory/problemlist.html.

[10] I. Dinur and S. Safra, *The importance of being biased*, in Proceedings of the 34th Annual ACM Symposium on Theory of Computing, Montreal, QC, 2002, ACM, New York, 2002, pp. 33–42.

[11] P. Erdős and T. Gallai, *On the minimal number of vertices representing the edges of a graph*, Publ. Math. Inst. Hungar. Acad. Sci., 6 (1961), pp. 181–202.

[12] P. Erdős and A. Rényi, *On random graphs*, Publ. Math. Debrecen, 6 (1959), pp. 290–297.

[13] U. Feige, *Vertex Cover Is Hardest to Approximate on Regular Graphs*, Technical Report MCS03-15, Computer Science and Applied Mathematics, The Weizmann Institute of Science, Rehovot, Israel, 2003.

[14] M. Garey and D. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness*, W. H. Freeman, New York, 1979.

[15] A. Goldberg, *Andrew Goldberg's Network Optimization Library*, online at http://www.avglab.com/andrew/soft.html.

[16] M. M. Halldórsson, *Approximating discrete collections via local improvements*, in Proceedings of the 6th Annual ACM-SIAM Symposium on Discrete Algorithms, San Francisco, CA, 1995, SIAM, Philadelphia, ACM, New York, 1995, pp. 160–169.

[17] E. Halperin, *Improved approximation algorithms for the vertex cover problem in graphs and hypergraphs*, in Proceedings of the 11th Annual ACM-SIAM Symposium on Discrete Algorithms, San Francisco, CA, 2000, SIAM, Philadelphia, ACM, New York, 2000, pp. 329–337.

[18] D. S. Hochbaum, *Efficient bounds for the stable set, vertex cover and set packing problems*, Discrete Appl. Math., 6 (1983), pp. 243–254.

[19] D. S. Hochbaum, *Approximation Algorithms for NP-hard Problems*, PWS Publishing, Boston, 1997.

[20] F. Hüffner, *Algorithm engineering for optimal graph bipartization*, in Proceedings of the 4th International Workshop on Experimental and Efficient Algorithms, Lecture Notes in Comput. Sci. 3503, Springer, New York, 2005, pp. 240–252.

[21] G. Karakostas, *A better approximation ratio for the vertex cover problem*, in Proceedings of the 2005 ICALP Conference, Lecture Notes in Comput. Sci. 3580, Springer, New York, 2005, pp. 1043–1050.

[22] R. M. Karp, *Reducibility among combinatorial problems*, in Complexity of Computer Computations, R. E. Miller and J. W. Thatcher, eds., Plenum Press, New York, 1972, pp. 85–103.

[23] R. M. Karp and M. Sipser, *Maximum matchings in sparse random graphs*, in Proceedings of the 22nd Annual Symposium on Foundations of Computer Science, Nashville, TN, 1981, IEEE Computer Society Press, Piscataway, NJ, 1981, pp. 364–375.

[24] E. L. Lawler, *Combinatorial Optimization: Networks and Matroids*, Holt, Rinehart and Winston, New York, 1976.

[25] B. Monien and E. Speckenmeyer, *Ramsey Numbers and an Approximation Algorithm for the Vertex Cover Problem*, Acta Inform., 22 (1985), pp. 115–123.

[26] G. L. Nemhauser and L. E. Trotter, *Vertex packing: Structural properties and algorithms*, Math. Program., 8 (1975), pp. 232–248.

[27] L. Sanchis, *Test case construction for the vertex cover problem*, in Computational Support for Discrete Mathematics, N. Dean and G. E. Shannon, eds., Ser. Discrete Math. Theoret. Comput. Sci. 15, AMS, Providence, RI, 1994, pp. 315–326.

[28] M. Truszczynski, N. Leone, and I. Niemela, *Benchmark Problems for Answer Set Programming Systems*, online at http://www.cs.uky.edu/ai/benchmarks.html.