Vertex Cover Approximations: Experiments and Observations

Eyjolfur Asgeirsson* and Cliff Stein**

Department of IEOR,
Columbia University, New York, NY
{ea367, cliff}@ieor.columbia.edu

Abstract. The vertex cover problem is a classic NP-complete problem for which the best worst-case approximation ratio is roughly 2. In this paper, we use a collection of simple reductions, each of which guarantees an approximation ratio of $\frac{3}{2}$, to find approximate vertex covers for a large collection of test graphs from various sources. We explain these reductions and explore the interaction between them. These reductions are extremely fast and even though they, by themselves are not guaranteed to find a vertex cover, we manage to find a $3/2$-approximate vertex cover for every single graph in our large collection of test examples.

1 Introduction

The vertex cover problem is a classic problem in computer science and one of the first NP-complete problems, [17, 11]. 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.

A simple greedy algorithm gives a 2-approximation for this problem [9]. In spite of many attempts to design improved approximation algorithms for vertex cover [7, 18, 3, 13] the best known approximation ratio is $2 - \theta(\frac{\log\log n}{\log n})$ for a graph with n vertices [14]. Minimum vertex cover NP-hard to approximate within any factor smaller than 1.36 [8] and many people believe that there does not exist an algorithm with a fixed approximation ratio better than 2 [15, 10]. In recent years, much good work has been done on the fixed parameter version of vertex cover, where, given a fixed parameter k and a graph G with n vertices, we can find a vertex cover of size at most k in time $O(kn + 1.2832^k)$, if such a vertex cover exists [2, 5, 20].

In this paper, we look at the classic vertex cover problem and take a different approach. We use a collection of simple reductions and allow reductions that don't maintain optimality but only guarantee a worst case approximation ratio of $3/2$. 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 fundamentally different properties of the graph, we can get very beneficial interactions; these reductions create a scheme that is much more

* Research partially supported by NSF Grant DMI-9970063.
** Research partially supported by NSF Grant DMI-9970063.

S.E. Nikoletseas (Ed.): WEA 2005, LNCS 3503, pp. 545–557, 2005.

powerful than the sum of the individual methods. These interactions between different reductions create both possibilities and difficulties that we will explore further.

To test these reductions, we collected all graphs we could find on the internet and from previous works, and also generated graphs specially designed to be difficult for the vertex cover problem. In principle, our reductions do not guarantee that we will find an approximate vertex cover, however we managed to find a $3/2$-approximate vertex cover in several seconds for every single graph using only our reductions. For the graphs where we know the size of the minimum vertex cover, the actual approximation ratio was usually much lower than $3/2$; in many cases the vertex cover we found was either optimal or very close to optimal.

2 Graph Reductions

Our main approach is to use divide and conquer. The vertex cover problem is a hard problem, but instead of tackling the whole graph at once, we try to find chinks in its armor and solve it by breaking it into smaller and easier subproblems. For this 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 $V = V_1 \cup V_2 \cup \ldots \cup V_k$. 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 $VC^{approx} = \cup_i VC_i^{approx}$ 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 $\mathrm{VC}^{\mathrm{approx}} = \cup_i \mathrm{VC}_i^{\mathrm{approx}}$ and since the vertex partition is disjoint, $|\mathrm{VC}^{\mathrm{approx}}| = |\cup_i \mathrm{VC}_i^{\mathrm{approx}}| = \sum_i |\mathrm{VC}^{\mathrm{approx}}|$. Since $E_i \neq \emptyset$, $|\mathrm{VC}^{\mathrm{opt}} \cap V_i| \geq 1$ for all i. Then

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

■

This lemma 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 this lemma 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've found a vertex cover of the whole graph. The lemma contains the additional 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 for G' then we can find an optimal vertex cover for the original graph G.*

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 for* G' *then we can find a* ρ*-approximate vertex cover for* G*.*

We will use optimal graph reductions and ρ-approximate graph reductions with $\rho \leq \frac{3}{2}$. 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)*. Then set of vertices adjacent to* v *is the union of the vertices 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.

2.1 Optimal Graph Reductions

Almost all the optimal graph reductions that we present in this section are well known and have been used to simplify difficult graphs previously [1]. The exception to that rule is the Extended Network Flow method, which is a simple yet powerful idea that to our knowledge has not been used extensively before. For completeness we briefly explain these reductions.

Zero- and One-degree Vertices

Claim. A vertex of degree zero is not in an 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 $u \notin C$.

Degree-two 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.

Let u be a vertex of degree with v and w as adjacent neighbors. To cover the edge (u, v), at least one of v or w must be in any vertex cover. By removing that vertex and all adjacent edges, u becomes a degree-one vertex which means that there is an optimal vertex cover that includes u and w but does not include u.

Claim. If there is a degree-two vertex u whose neighbors, v and w are non-adjacent, 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 [5]. 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 before. Let $\text{VC}^{\text{opt}}(G)$ be the optimal vertex cover for G. Then $|\text{VC}^{\text{opt}}(G)| = |\text{VC}^{\text{opt}}(G')| + 1$. 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 $|\text{VC}^{\text{opt}}(G)| = |\text{VC}^{\text{opt}}(G')| + 1$, any approximation ratio for a vertex cover of G' holds for the vertex cover of G.

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

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

By solving the linear programming relaxation of this problem we get a fractional solution. Nemhauser and Trotter [19] showed that the solution of this linear program can be used 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 [16].

Extended Network Flow. One of the problems with the network flow algorithm is that it tends to find solutions with many variables equal to $1/2$, even when other solutions exists 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 $1/2$. Then, using the optimal solution, we try to set each variable with value $1/2$ equal to 1 and resolve. If the objective value doesn't change we keep this new value and repeat for the remaining variables. Otherwise we set the value back to $1/2$ 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. To our knowledge, this approach 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 [12, 6].

2.2 Approximating Graph Reductions

While the optimal reductions can find solutions to simple examples, they are usually not enough to solve difficult graphs. We use approximating graph reductions to make further progress. The approximate reductions in this section use Lemma 1 extensively. We try to find a subgraph with certain properties, reduce this subgraph while ensuring that all edges between this subgraph and the remaining graph are covered by our reduction. By Lemma 1 we can do this repeatedly and get an approximate solution to the vertex cover problem with approximation ratio equal to the largest approximation ratio of these reductions.

Definition 4. *Let $G = (V, E)$ be a graph and let G' be the graph obtained by contracting of a set of vertices $V_i \subset V$ to a new vertex z. If, given an optimal vertex cover of G', we can use the mapping of V_i to z to find a ρ-approximate vertex cover for G, then we call z a* **ρ-approximated vertex**. *(We will sometimes drop the ρ) A* **regular vertex** *is a vertex that is not an approximated vertex.*

Approximated vertices can be very useful in some situations but in others they can create difficulties. They are useful because they map a set of vertices to a single vertex and yield a simpler graph with fewer vertices and edges. The difficulty is that we cannot use them in further approximating reductions, although we can use approximated vertices in optimal reductions. Due to this, we try to use reductions that create approximated vertices only when necessary.

In Definition 4, we are using Lemma 1 when we find the ρ-approximate vertex cover of the original graph $G = (V, E)$ from the vertex cover of G'. Let z be the ρ-approximated vertex in G' and let V_1 be the set of vertices in G that we contract to z, and $V_2 = V \setminus V_1$. Let VC' be the vertex cover of G' and $VC_2 = VC' \cap V_2$. Let VC_1 be the cover of V_1 that we can find by using the knowledge whether z is in VC' or not, while making sure that VC_1 covers all edges with one or more endpoint in V_1 that are not covered by VC_2. Then by Lemma 1 and Definition 4, the vertex cover $VC = VC_1 \cup VC_2$ is a ρ-approximate vertex cover of G.

Degree-three Vertices. The idea for degree-three vertices is similar to the degree-two vertices reduction. We find a degree-three vertex, u, with non-adjacent neighbors, v, w and z, and contract them to a new vertex, q to get a new graph G'. This new vertex, q, is a $\frac{3}{2}$-approximated vertex, so if we find an optimal vertex cover for this new graph, we can determine which of u, v, w and z are in an $\frac{3}{2}$-approximate vertex cover for our original graph.

Lemma 2. *Let $G = (V, E)$ be a graph and $u \in V$ be a degree-three vertex with v, w and z as neighbors, v, w and z are non-adjacent. Let G' be a the graph we get by contracting u, v, w and z to a new vertex q. Then, if VC^* is an optimal vertex cover for G' then we can find a vertex cover VC^{approx} for G that is a $\frac{3}{2}$-approximation.*

Proof. Let VC^* be an optimal vertex cover for G' and set $\text{VC}^{\text{approx}}$ be the same as VC^* for all vertices $v \in V \setminus \{u, v, w, z\}$. Now if q is not in VC^* then every vertex in the neighborhood of q must be in the vertex cover. Hence, every vertex in the neighborhoods of v, w and z are in the vertex cover $\text{VC}^{\text{approx}}$ and none of v, w or z need to be in the vertex cover. However, we know that at least one of the vertices u, v, w or z must be in the vertex cover, hence we can select u to be in the vertex cover $\text{VC}^{\text{approx}}$ and by Lemma 1 this is an optimal vertex cover for G. If q is in VC^* then at least one of q's neighbors is not in VC^* so at least one of v, w or u must be in the vertex cover $\text{VC}^{\text{approx}}$. In that case we would need at least two of u, v, w and z to be in $\text{VC}^{\text{approx}}$ to cover the subgraph induced by u, v, w and z. We can select v, w and z in the vertex cover $\text{VC}^{\text{approx}}$ and get a $\frac{3}{2}$-approximation for this subgraph. Hence by Lemma 1, if VC^* is an optimal vertex cover for G' then $\text{VC}^{\text{approx}}$ is a $\frac{3}{2}$-approximation of the optimal vertex cover for G. ■

This reduction removes four regular vertices and at least three edges from our graph while creating one $\frac{3}{2}$-approximated vertex.

Triangles

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. Removing a triangle removes three vertices from the graph along with all adjacent edges. This has the nice property that it does not create any approximated vertices.

Four-cycles. Assume we have a cordless cycle of length four with the vertices v_1, v_2, v_3 and v_4. For any cycle of length four, at least two of the vertices must be in a vertex cover and the only way to get exactly two of the vertices in the vertex cover is to select the opposite corners, i.e. either v_1 and v_3 or v_2 and v_4. Any other choice will give us at least three of the vertices in the vertex cover. We use this property to get a $\frac{4}{3}$-approximate reduction that removes four regular vertices and at least three edges from the graph, but at the same time creates two new $\frac{4}{3}$-approximated vertices.

Lemma 3. *Let $G = (V, E)$ be a graph with v_1, v_2, v_3 and v_4 as a cordless cycle of length four. Let G' be a new graph where we contract v_1 and v_3 to a new vertex z_1 and contract v_2 and v_4 to a new vertex z_2. Then if VC^* is an optimal vertex cover for G' then we can find a $\frac{4}{3}$-approximate vertex cover for G.*

Proof. For any vertex cover of G, at least two of the vertices $v_1, \ldots, v_4$ must be in the cover. The only way to get exactly two of the vertices is to select either v_1 and v_3 or v_2 and v_4. Any other selection will have at least three of the vertices in the vertex cover. In the graph G', there is an edge between z_1 and z_2 so at least one of these vertices must be in any vertex cover. If $z_1 \in \text{VC}^*$ and $z_2 \notin \text{VC}^*$ then that corresponds to v_1 and v_3 being in the vertex cover for G. Since the remaining graphs of G' and G are the same, this vertex cover is optimal for G. Similarly if $z_2 \in \text{VC}^*$ and $z_1 \notin \text{VC}^*$. Then v_2 and v_4 will be in the vertex cover for G and we have an optimal vertex cover. However, if both z_1 and z_2 are in the optimal vertex cover for G' then at least three of the vertices $v_1, \ldots, v_4$ must be in the optimal vertex cover for G. In that case, we take all four vertices in the cover and get a $\frac{4}{3}$-approximation. ∎

Six-cycles. The idea for cycles of length six is the same as for cycles of length four. If we have a cordless cycle of length six, we can replace it with only two vertices and get a $\frac{3}{2}$-approximation. This reduction removes six regular vertices and at least five edges from the graph, but creates two $\frac{3}{2}$-approximated vertices.

Lemma 4. *Let $v_1, v_2, \ldots, v_6$ be a cycle of length six in G. Let G' be a graph where v_1, v_3 and v_5 in G are contracted to a new vertex z_1 and v_2, v_4 and v_6 are contracted to z_2 while keeping all other vertices and edges the same. If VC^* is an optimal vertex cover for G' then we can find a vertex cover VC^{approx} for G that is a $\frac{3}{2}$-approximation of the optimal vertex cover.*

The proof for this is almost identical to the proof to Lemma 3..

2.3 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 take the neighborhood set in the vertex cover and none of the vertices in the independent set. Abu Khzam et. al. [1] proved that this is an optimal reduction. However, it is easy to show that this reduction is captured 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 if all its neighbors are also in the vertex cover. If that is the case then we remove this vertex from the cover.

3 Order of Reductions

The reductions in previous section are all simple and straightforward. We can use them in any order and given one input graph, applying them in different orders will give different results. This creates some difficulties when we try to automate the reduction, since the wrong choices can leave us in a dead end without any means of removing approximated vertices, while a different approach might have solved the problem. The greatest danger is in using 3-degree, 4-cycle and 6-cycle reductions since they leave approximated vertices that cannot be used in further approximations. The extended network flow and low degree reductions are optimal while the triangle elimination completely removes the vertices from the graph, so these methods are safe in the sense that they do not leave any approximated vertices.

The triangle elimination proved to be very effective on almost all of the graphs but it is also responsible for the largest approximation ratios. In many cases, just running triangle elimination followed by the extended network flow method was enough to find an approximate vertex cover, but often it was necessary to use multiple iterations of several reductions to get a solution. One example of this is shown in Table 1. This table shows the results of each iteration for the complement graph of the graph 's20.vc' which we created using Laura Sanchis' graph generator (see Section 4). We show the number of vertices and edges at the start of each iteration and how many vertices and edges each reduction removes from the graph. In this case, the extended network flow method and triangle elimination remove many edges which help create low degree vertices, which are eliminated with low-degree methods. During the low-degree methods, the 2-degree reduction creates more triangles, thus creating a cycle where we slowly but surely clear all the vertices.

For a few graphs we had to use the 3-degree reduction or 4-cycle and 6-cycle reduction to create a way into the graph for the optimal methods or the triangle elimination. One example of this is shown in Table 2. Here we are trying to find a vertex cover for the complement graph of the graph 'san400_0.9_1.clq' [4]. The table shows how many vertices and edges are at the start of each iteration, which reduction we used and how many vertices and edges this reduction removed from the graph. In this example, the

Table 1. Interaction between triangle elimination and low-degree reductions on the graph 's20.vc'. $|V|$ and $|E|$ are the number of vertices and edges at the start of each iteration while ΔV and ΔE show how many vertices and edges are removed from the graph

Method	$\|V\|$	$\|E\|$	ΔV	ΔE
Network flow	500	2000	3	17
Triangles	497	1983	147	1076
Low Degree	350	907	59	96
Triangles	291	811	24	205
Low Degree	267	606	54	67
Triangles	213	539	15	108
Low Degree	198	431	46	49
Triangles	152	382	21	139
Low Degree	131	243	50	57
Triangles	81	186	6	41
Low Degree	75	145	26	60
Triangles	49	85	3	11
Low Degree	46	74	33	50
Triangles	13	24	3	10
Low Degree	10	14	10	14
Finished	0	0	-	-

Table 2. The complement graph of 'san400_0.9_1.clq'. On the left we find a $3/2$-approximate vertex cover by using 3-degree reduction. On the right we use 4-cycle reduction instead. There we get stuck and cannot finish

Method	$\|V\|$	$\|E\|$	ΔV	ΔE
Triangles	400	7980	348	7864
Low Degree	52	116	12	21
Triangles	40	95	3	18
Low Degree	37	77	2	3
3-Degree	35	74	15	27
Network Flow	20	47	20	47
Finished	0	0	-	-

Method	$\|V\|$	$\|E\|$	ΔV	ΔE
Triangles	400	7980	348	7864
Low Degree	52	116	12	21
Triangles	40	95	3	18
Low Degree	37	77	2	3
4-cycle	35	74	10	17
6-cycle	25	57	4	5
Stuck	21	52	-	-

three major reductions are not enough and we must use the 3-degree reduction to find a way into the graph, if we use 4-cycle reduction instead we get stuck.

After much experimentation, we settled on using the following order of reductions to automate the approximation process. We ran the extended network flow method, triangle elimination and low-degree in a loop until we had found a solution or no improvements were made during an iteration. Then we ran 3-degree, 4-cycle and 6-cycle reductions, stopping as soon as any one of them made some progress and returning to the original loop. The stopping criteria is either having processed all the vertices from the graph which gives us a vertex cover, or running 3-degree, 4-cycle and 6-cycle without any improvements. In that case 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 al-

gorithm is to use a simple greedy algorithm to eliminate unnecessary vertices from the cover. In our experiments we never had to resort to branch-and-bound, our algorithm managed to solve every single graph we found.

4 Experiments and Results

We did an extensive search for datasets we could use for vertex cover and also gathered all datasets we could find from previous works.

1. From the DIMACS website we took 78 graphs that were used as a challenge for the MaxClique problem [4]. We also used the complement graphs for these graphs, where we find the vertex cover for the complement graphs. These graphs are of special interest since there is a direct connection between the optimal vertex cover in a graph and the maximum clique in the complement graph.
2. From the DIMACS challenges we also obtained 60 graphs used as a benchmark for the MinColor problem, along with the complement graphs.
3. Also from the DIMACS challenges, we found 5 additional benchmark graphs. We also used the complement graphs.
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. We also used the complement graphs.
5. From [22] we found 4 small graphs used as a benchmark for vertex cover algorithms. These graphs are of special interest because the optimal vertex cover is known. We also tried the complement graphs.
6. We generated 32 graphs using Laura Sanchis' graph generator [21]. These graphs have 500 vertices each, with number of edges ranging from 2000 to $110,000$. We split these graphs into three groups, with maximum clique sizes of 2,4 and 10. 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. And not surprisingly, we also tried the complement graphs of these generated graphs for additional 32 graphs.

We solved every one of these 362 graphs, finding a $3/2$-approximate vertex cover in under 5 minutes for each one. The running time for most of these 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, 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.

The experiments were run on a machine with 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 didn't 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 set, which is a clique in the complement graph. Since we are trying to minimize the size of the vertex cover, it's 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 only go into details about the most difficult and interesting problems.

Worst approximation ratio: The graphs that had the worst approximation ratio were the complement graphs of the MANN series. These graphs are made by Carlo Mannino and they are a part of the Maximum Clique challenge on the DIMACS webpage. These graphs are a clique formulation of the Steiner Triple Problem and they show how easy it is to create graphs with approximation ratios close to 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, forcing us to use low degree, triangle elimination and extended network flow to get a result while still having approximation ratio over 1.2. The only graph where we had to use 3-degree reduction is the complement graph of 'san400_0.9_1.clq'. This is shown in Table 2.

Some graphs we generated using Laura Sanches' graph generator showed very interesting behavior. The triangle elimination and the extended network flow method removed about half the vertices and then low degree elimination removed a few more. We seemed to be stuck, the graph had a few 4 and 6-cycles, but if we reduced them we couldn't finish the graph completely since we then couldn't eliminate the approximated vertices that these reductions created. However, if we went back and forth between 2-degree elimination and triangle elimination then slowly but surely we finished off the whole graph. This is shown in Table 1.

Special bad case: Even though we managed to find an approximate vertex cover for every graph we tested using only these simple methods, it's easy to construct graphs where our algorithm gets stuck. One example of such a graph, consisting of connected 5-cycles is shown in Figure 1. The only thing we can do in this case is to use 3-degree reduction to decrease the size of this graph down from 45 vertices and 75 edges to a reduced graph of 18 vertices and 48 edges. After that, we are stuck and must turn to branch-and-bound or some other methods to get a solution. The graph in Figure 1 is a

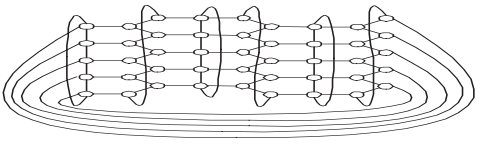

Fig. 1. A simple bad case

Table 3. Detailed performance on a small subset of the graphs

Name	n	m	VCdec	$\lvert VC \rvert$	ENF n	ENF m	ENF time	LD n	LD m	LD time	T n	T m	T time
Max-Clique													
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
Max-Clique Complement													
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
Min-Color													
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
Min-Color Complement													
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

simple bad case, by combining graphs similar to this and making them larger we can easily construct large graphs where the methods used here have little or no effect.

Table 3 shows how the reductions performed on a selected subset of the graphs. The first column is the name of the graph, 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 method, triangle elimination and low degree reductions. We show how many vertices and edges each method removed from each graph, and the total time it took in seconds. This is only a small selection of the results we have, due to space constraints we cannot show all our results here.

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 it is very helpful in some cases ('s20.vc').

5 Conclusions

simple reductions where we allowed reductions that have a worst case approximation ratio of $3/2$. 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. 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. *Proceedings, ACM-SIAM Workshop on Algorithm Engineering and Experiments*, 2004.
2. R. Balasubramanian, M. R. Fellows, and V. Raman. An improved fixed-parameter algorithm for vertex cover. *Information Processing Letters*, 65:163–168, 1998.
3. R. Bar-Yehuda and S. Even. A local-ratio theorem for approximating the weighted vertex cover problem. *Annals of Disc. Math.*, 25:27–46, 1985.
4. DIMACS Implementation Challenges. Center for Discrete Mathematics & Theoretical Computer Science. ftp://dimacs.rutgers.edu/pub/challenge/graph/benchmarks/.
5. J. Chen, I. Kanj, and W. Jia. Vertex cover: further observations and further improvements. *Journal of Algorithms*, 41(2):280–301, 2001.
6. B.V. Cherkassky, A.V. Goldberg, P. Martin, J. C. Setubal, and J. Stolfi. Augment or Push? A computational study of Bipartite Matching and Unit Capacity Flow Algorithms. Technical Report 98-036R, NEC Research Institute, Inc., 1998.
7. P. Crescenzi and V. Kann. A compendium of NP optimization problems. http://www.nada.kth.se/theory/problemlist.html.
8. I. Dinur and S. Safra. The importance of being biased. *Proc. 34th Ann. ACM Symp. on Theory of Comp.*, pages 33–42, 2002.
9. P. Erdös and T. Gallai. On the minimal number of vertices representing the edges of a graph. *Publ. Math. Inst. Hungar. Acad. Sci.*, 6:181–202, 1961.
10. 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, 2003.
11. M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. Freeman, San Fransisco, 1979.
12. A. Goldberg. Andrew Goldberg's Network Optimization Library. http://www.avglab.com/andrew/soft.html.

13. M. M. Halldórsson. Approximating discrete collections via local improvements. *Proc. 6th Ann. ACM-SIAM Symp. on Discrete Algorithms*, pages 160–169, 1995.
14. E. Halperin. Improved approximation algorithms for the vertex cover problem in graphs and hypergraphs. *Proc. 11th Ann. ACM-SIAM Symp. on Discrete Algorithms*, pages 329–337, 2000.
15. D. S. Hochbaum. Efficient bounds for the stable set, vertex cover and set packing problems. *Discrete Applied Mathematics*, 6:243–254, 1983.
16. D. S. Hochbaum. *Approximation Algorithms for NP-hard Problems*. PWS, 1997.
17. R. M. Karp. Reducibility among combinatorial problems. In R. E. Miller and J. W. Thatcher, editors, *Complexity of Computer Computations*, pages 85–103. Plenum Press, New York, 1972.
18. B. Monien and E. Speckenmeyer. Ramsey Numbers and an Approximation Algorithm for the Vertex Cover Problem. *Acta Inf.*, 22:115–123, 1985.
19. G. L. Nemhauser and L. E. Trotter. Vertex packing: Structural properties and algorithms. *Mathematical Programming*, 8:232–248, 1975.
20. R. Niedermeier and P. Rossmanith. On Efficient Fixed Parameter Algorithms for Weighted Vertex Cover. *Journal of Algorithms*, 47:63–77, 2003.
21. L.A Sanchis. Test Case Construction for the Vertex Cover Problem. *DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, 15, 1994.
22. M. Truszczynski, N. Leone, and I. Niemela. Benchmark problems for answer set programming systems. http://www.cs.uky.edu/ai/benchmarks.html.