*algorithms*

MDPI

*Article*

# A Performance Study of Some Approximation Algorithms for Computing a Small Dominating Set in a Graph

**Jonathan Li [1], Rohan Potru [1] and Farhad Shahrokhi [2,*]**

[1] Department of Computer Science, College of Natural Sciences, The University of Texas at Austin, Austin, TX 78712, USA; jonathansli@utexas.edu (J.L.); rohanpotru@utexas.edu (R.P.)

[2] Department of Computer Science and Engineering, College of Engineering, University of North Texas, Denton, TX 76203, USA

* Correspondence: farhad.shahrokhi@unt.edu

Received: 24 November 2020; Accepted: 8 December 2020; Published: 14 December 2020


**Abstract:** We implement and test the performances of several approximation algorithms for computing the minimum dominating set of a graph. These algorithms are the standard greedy algorithm, the recent Linear programming (LP) rounding algorithms and a hybrid algorithm that we design by combining the greedy and LP rounding algorithms. Over the range of test data, all algorithms perform better than anticipated in theory, and have small performance ratios, measured as the size of output divided by the LP objective lower bound. However, each have advantages over the others. For instance, LP rounding algorithm normally outperforms the other algorithms on sparse real-world graphs. On a graph with 400,000+ vertices, LP rounding took less than 15 s of CPU time to generate a solution with performance ratio 1.011, while the greedy and hybrid algorithms generated solutions of performance ratio 1.12 in similar time. For synthetic graphs, the hybrid algorithm normally outperforms the others, whereas for hypercubes and k-Queens graphs, greedy outperforms the rest. Another advantage of the hybrid algorithm is to solve very large problems that are suitable for application of LP rounding (sparse graphs) but LP formulations become formidable in practice and LP solvers crash, as we observed on a real-world graph with 7.7 million+ vertices and a planar graph on 1,000,000 vertices.

**Keywords:** minimum dominating set; linear programming; experimentation

## 1. Introduction

Domination theory has its roots in the $k$-Queens problem in the 18th century. Later, in 1957, Berge [1] formally introduced the domination number of a graph. A subset of vertices $S$ in a graph $G$ is a dominating set if every vertex not in $S$ is adjacent to some vertex in $S$. A dominating set of smallest cardinality is called a minimum dominating set. The cardinality of a minimum dominating set is called domination number of $G$ and is denoted by $\gamma(G)$. The vertices colored red in Figure 1 constitute a minimum dominating set in the graph $Q_3$, or the three dimensional cube.

For the remainder of the paper, we assume familiarity with general concepts of graph theory as in [2], the theory of algorithms as in [3], and linear and integer programming concepts as in [4], respectively. We refer the reader to the book by Haynes, Hedetniemi, and Slater [5] as a general reference in domination theory. The problem of computing the domination number of a graph is well studied, and has extensive applications, including the design of telecommunication networks, facility location, and social networks.

Computing $\gamma(G)$ is known to be an NP-hard problem, even in restricted cases, including unit disc graphs and grids [6], and hence the researchers have focused on approximation and finding a small

dominating set. A simple greedy algorithm is known to approximate $\gamma(G)$ to within a logarithmic factor from the optimal value. It is known that improving the logarithmic approximation factor is also NP-hard [7]. Hence, no algorithm for approximating $\gamma(G)$ can improve the asymptotic worst-case performance ratio of the greedy algorithm. Different variations of this algorithm are proposed and some are tested in practice. See the work of Chalupa [8], Campan et al. [9], Eubank et al. [10], Sanchis [11], and Siebertz [12].

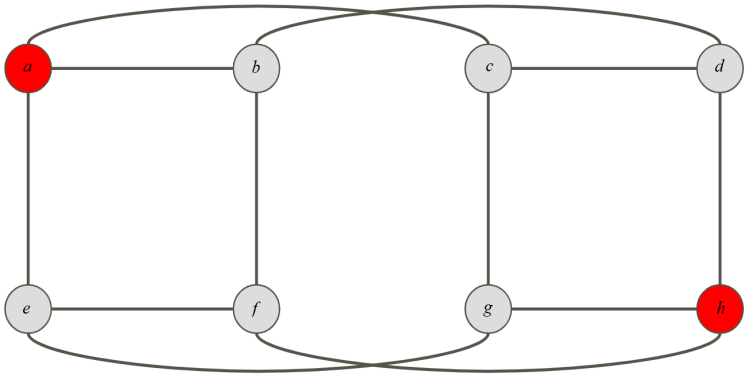


**Figure 1.** Minimum dominating set for $Q_3$.

There are other approximation algorithms for very specific classes of graphs, including planar graphs, which have better than constant performance ratio in the worst case but are more complex than the algorithms described here. See [12] for a brief reference to some related papers.

Very recently, Bansal and Umboh [13] and Dvořák [14] showed that an appropriate rounding of solutions to the linear programming (LP) formulations for computing $\gamma(G)$ provides dominating sets whose cardinalities are at most $3 \cdot a(G) \cdot L^*$ and $(2 \cdot a(G) + 1) \cdot L^*$, respectively, in polynomial time. Here, $a(G)$ is the arboricity of $G$, and $L^*$ is the value of the optimal solution to the linear programming, which is a lower bound on $\gamma(G)$. Hence, for graphs with bounded arboricity, one can improve the logarithmic performance ratio of the greedy algorithm to a constant.

The greedy algorithm is simple, fast, and is tested in practice. One anticipates that it outperforms the LP-based approach if CPU time is the criteria. Nonetheless, its performance ratio in the worst case is logarithmic even for planar graphs which have arboricity at most 3; see example 1 in the Appendix A. For sparse graphs, the recent LP rounding methods referenced above have a bounded performance ratio, which is better than greedy, but to our knowledge, and in contrast to the greedy algorithm, the performance of the LP-based approaches have not been tested in practice. Furthermore, one would expect that for large graphs, the LP formulations would become formidable. Can one hope that a combination of these methods would give a better result than each individually, and if so, in what scenarios?

In this paper, we compare and contrast the performance of the greedy algorithm, the LP rounding algorithm, and a hybrid algorithm that combines the greedy and LP approaches. Our hybrid algorithm first solves the problem using the greedy algorithm and finds a dominating set, then takes a portion of vertices in this set and forces their values to be 1 in the linear programming formulation, solves the resulting linear program, and finally properly rounds the solution.

*Our Findings*

Through experimentation, all algorithms perform better than anticipated in theory, particularly with respect to the performance ratios, measured as the value of solution divided by the computed LP objective lower bound. However, each may offer advantages over the others depending on the nature of the data.

LP rounding does well on sparse real-world graphs, consistent with theory, and normally outperforms the other algorithms. On a graph with 400,000+ vertices, LP rounding took less than 15 s of CPU time to generate a solution with performance ratio 1.011, while the greedy and hybrid algorithms generated solutions of performance ratio 1.12 in similar time. It is remarkable that the hybrid algorithm can solve the problem for very large sparse graphs, where the LP formulation becomes formidable in practice. For instance, it solved a real-world graph with 7.7 million+ vertices in 106 s of CPU time with a performance ratio of 2.0075. The LP solver crashed on this problem. We indicate that the simplex-based LP package used in our experiments performed very fast, although, in theory, simplex is not necessarily polynomially time bounded.

For hypercubes and k-Queens graphs (which are not sparse) greedy outperforms the rest, consistent with theory, both in terms of speed and performance ratio. In particular, on the 12-dimensional hypercube, greedy finds a solution with performance ratio 1.7 in 0.01 s. On the other hand, the LP rounding and hybrid algorithms produce solutions with performance ratio 13 and 3.3 using 7.5 and 0.08 s of CPU time, respectively. It is notable that greedy gives optimal results in some cases where the domination number is known. Specifically, the greedy algorithm produces an optimal solution on hypercubes with dimensions $d = 2^k - 1$ where $k = 1, 2, 3$, and 4. For synthetic graphs - generated k-trees ($G$ is a $k-$tree if it has tree width $k$ and the addition of any edge increases the tree width by one), and k-planar graphs ($G$ is $k-$planar if it can be drawn in the plane with no edge crossed by more than $k$ other edges) - the LP rounding is outperformed by the other two algorithms, and the hybrid algorithm outperforms greedy.

This paper is organized as follows. In Section 2, we set our notations and summarize related materials for the greedy algorithm. The LP rounding and hybrid algorithms are explained in sections three and four, respectively. Section 5 (Environment, implementations and datasets), contains our materials and methods. Section 6 contains the results for synthetic graphs (k-planar graphs and k-trees). Sections 7 and 8 contain the results for hypercubes and k-Queens graphs, and real-world graphs, respectively. In Section 9, we exclusively describe how the hybrid algorithm can be applied to very large sparse graphs, where the LP formulation becomes formidable. We present our conclusions in Section 10.

## 2. Preliminaries

Throughout this paper $G = (V, E)$ denotes an undirected graph on vertex set $V$ and edge set $E$ with $n = |V|$ and $m = |E|$. Two vertices $x, y \in V$ where $x \neq y$ are adjacent (or they are neighbors) if $xy \in E$. For any $x \in V$, the degree of $x$, denoted by $deg(x)$ is the number of vertices adjacent to $x$ in $G$. For any $x \in V$, let $N(x)$ denote the set of all vertices in $G$ that are adjacent to $x$. Let $N[x]$ denote $N(x) \cup \{x\}$. Arboricity of $G$, denoted by $a(G)$, is the minimum number of spanning acyclic subgraphs of $G$ that $E$ can be partitioned into. By a theorem of Nash-Williams, $a(G) = \max_S \lceil \frac{m_S}{n_S - 1} \rceil$, where $n_S$ and $m_S$ are the number of vertices and edges, respectively, of the induced subgraph on the vertex set $S$ [15]. Consequently, $m \leq a(G)(n-1)$, and thus $a(G)$ measures how dense $G$ is. It is known that $a(G)$ can be computed in polynomial time [16].

Let $D \subseteq V$. $D$ is a dominating set if for every $x \in V \setminus D$ there exists $y \in D$ such that $xy \in E$. The *domination number* of $G$, denoted by $\gamma(G)$, is the cardinality of a smallest dominating set of $G$. A dominating set of cardinality $\gamma(G)$ is called a *minimum dominating* set. Additional definitions will be introduced when required.

*Greedy Approximation Algorithm*

A simple greedy algorithm attributed to Chvátal [17] and Lovász [18] (for approximating the set cover problem) is known to approximate $\gamma(G)$ within a multiplicative factor of $H(\Delta(G))$ from its optimal value, where $\Delta(G)$ is the maximum degree of $G$ and $H(k) = \sum_{i=1}^{k}(1/i)$ is the $k-$th harmonic number. Note that $ln(k+1) \leq H(k) \leq ln(x) + 1$. The algorithm initially labels all vertices uncovered. At iteration one, the algorithm selects a vertex $v_1$ of maximum degree in $G$, places $v_1$ in a set $D$,

and labels all vertices adjacent to $v_1$ as covered. In general, at iteration $i \geq 2$, the algorithm selects a vertex $v_i \in V - \{v_1, v_2, \ldots, v_{i-1}\}$ with the largest number of uncovered vertices adjacent to it , adds $v_i$ to $D$, and labels all of the uncovered vertices adjacent to $v_i$ as covered. The algorithm stops when $D$ becomes a dominating set. It is easy to implement the algorithm in $O(n+m)$ time. It is known that approximating $\gamma(G)$ within a factor $(1-\epsilon)ln(\Delta)$ from the optimal is NP-hard [7]. Hence, no algorithm for approximating $\gamma(G)$ can improve the asymptotic worst-case performance ratio achieved by the greedy algorithm.

The appendix includes two examples of worst-case graphs (one sparse and one dense) for the greedy algorithm, which are derived from an instance of the set cover problem provided in [19]. For both instances, the $O(ln(\Delta))$ performance ratio is tight.

## 3. Linear Programming Approach

One can formulate the computation of $\gamma(G)$ as an integer programming problem IP1 stated below. However, since integer programming problems are known to be NP-hard [20], the direct application of the integer programming method would not be computationally fruitful. Next observe that by relaxing the integer program IP1, one obtains the linear program LP1 shown below.

**IP1:**
Minimize $I = \sum_{v \in V} x_v$
Subject to $\sum_{u \in N[v]} x_u \geq 1, \forall v \in V$
$x_v \in \{0,1\}, \ \forall v \in V$

**LP1:**
Minimize $L = \sum_{v \in V} x_v$
Subject to $\sum_{u \in N[v]} x_u \geq 1, \forall v \in V$
$0 \leq x_v \leq 1, \ \forall v \in V$

**Definition 1.** *Throughout rest of this paper we denote by $L^*$ and $I^*$ the values of $L$ and $I$ in LP1, and IP1, respectively, at optimality.*

Please note that $L^* \leq \gamma(G) = I^*$.

*Linear Programming Rounding*

Algorithm $R_1$ is due to Bansal and Umboh [13].

**Algorithm $R_1$ ([13])**

Solve LP1, and let $H$ be the set of all vertices that have weight at least $1/(3a(G))$, where $a(G)$ is the arboricity of graph $G$. Let $U$ be the set of all vertices not adjacent to any vertex in $H$ and return $H \cup U$.

Dvořák [14,21] studied the $d$-domination problem, i.e., when a vertex dominates all vertices at distance at most $d$ from it and its combinatorial dual, or a $2d$-independent set [22]. In [14] he employed the LP rounding approach of Bansal and Umboh as a part of his frame work and consequently, for $d = 1$, he improved the approximation ratio of algorithm $R_1$ by showing that the algorithm $R_2$ given below provides a $2a(G)+1$ approximation.

**Algorithm $R_2$ ([14])**

Solve LP1, and let $H$ be the set of all vertices that have weight at least $1/(2a(G)+1)$, where $a(G)$ is the arboricity of graph $G$. Let $U$ be the set of all vertices that are not adjacent to any vertex of $H$ and return $H \cup U$.

**Remark 1.** *Graph $G$ in Example 1 in the Appendix A is planar, so $a(G) \leq 3$. Thus, algorithms $R_1$ and $R_2$ have a worst-case performance ratio of nine and seven respectively, whereas greedy exhibits a worst-case $O(\log(n))$ performance ratio. Throughout our experiments, rounding algorithms returned an optimal solution of size two*

*for both examples, whereas greedy returned a set of size three for Example 1. Furthermore, in Example 2 in the appendix, it can be verified that $a(G) \geq (p+2)/2$ for graph G and hence in theory the worst-case performance ratios of the rounding algorithms are not constant either. Interestingly enough, in our experiments, $L^*$ was always two for graphs of type Example 2, and LP rounding algorithms also always found a solution of size two, which is the optimal value. Thus, the performance ratio was always one, and much smaller than the predicted worst case.*

## 4. Hybrid Approach

Next, we provide a description of the decomposition approach for approximating LP1 and our hybrid algorithm. Recall that a separation in $G = (V, E)$ is a partition $A \cup B \cup C$ of $V$ so that no vertex of $A$ is adjacent to any vertex of $C$. In this case, $B$ is called a vertex separator in $G$. Let $X = \{x_v | v \in V\}$ be a feasible solution to LP1, and let $V' \subseteq V$. Then $X(V')$ denotes $\sum_{v \in V'} x_v$.

**Lemma 1.** *Let $A \cup B \cup C$ be a separation in $G = (V, E)$ and consider the following linear programs:*

| ***LP2:*** | ***LP3:*** |
|---|---|
| *Minimize* $M = \sum_{v \in A \cup B} x_v$ | *Minimize* $N = \sum_{v \in B \cup C} x_v$ |
| *Subject to* $\sum_{u \in N[v]} x_u \geq 1, \forall v \in A$ | *Subject to* $\sum_{u \in N[v]} x_u \geq 1, \forall v \in C$ |
| $0 \leq x_v \leq 1, \ \forall v \in A \cup B$ | $0 \leq x_v \leq 1, \ \forall v \in B \cup C$ |

*Then* $\max\{M^*, N^*\} \leq L^*$.

**Proof.** Let $X = \{x_v | v \in V\}$ be an optimal solution to LP1. Please note that the restrictions of $X$ to $B \cup C$ and $A \cup B$ give feasible solutions for LP3 and LP2 of values $X(B \cup C)$ and $X(A \cup B)$, and hence the claim for the lower bound on $L^*$ follows. □

Please note that in LP2 and LP3 the constraints are not written for all variables, and the rounding method in [13] may not directly be applied.

**Theorem 1.** *Let $G = (V, E)$, let $A \subset V$, let $B = N(A)$ and let $C = V - (A \cup B)$. Let $X$ be an optimal solution for LP3, and let $X(C)$ denote the sum of the weights assigned to all vertices in $C$. Then there is a dominating set in $G$ of size at most $|A| + 3a(G)X(C) \leq |A| + 3a(G)N^*$.*

**Proof.** Let $H$ be the set of all vertices $v$ in $C$ with $x(v) \geq \frac{1}{3a}$, and let $U = C - (H \cup N(H))$. Now apply the method in [13] to $C$ to obtain a rounded solution, or a dominating set $D$, of at most $|U| + |H| \leq 3a(G)X(C)$ vertices in $C$. Finally, note that $A \cup D$ is a dominating set in $G$ with cardinality at most $|A| + 3a(G)X(C) \leq |A| + 3a(G)N^*$. □

**The Hybrid Algorithm:**

Fix $0 < \alpha < 1$. Apply the greedy algorithm to $G$ to obtain a dominating set $D = \{x_1, x_2, \ldots, x_d\}$, and let $S = \{x_1, x_2, \ldots, x_{\alpha \cdot d}\}$ be the first $\alpha \cdot d$ vertices in $D$. Now, solve the following linear program on the induced subgraph of $G$ with the vertex set $V - \{S\}$.

$$\text{Minimize } J = \sum_{v \in V - \{S\}} x_v \tag{1}$$

$$\text{Subject to } \sum_{u \in N[v]} x_u \geq 1, \forall v \in V - \{S \cup N[S]\} \tag{2}$$

$$0 \leq x_v \leq 1, \ \forall v \in V - S \tag{3}$$

Next, let $A = S, B = N(S)$ and $C = V - (A \cup B)$, and apply the rounding scheme in algorithms $R_1$ or $R_2$ to $C$. Let $H$ and $U$ be the corresponding sets, and output the set $S \cup H \cup U$.

**Remark 2.** *Please note that with Theorem 1, the hybrid algorithm can be implemented in polynomial time. Furthermore,* $|S \cup H \cup U| \leq \alpha.d + 3a(G)N^* \leq \alpha.(ln(\Delta) + 1) + 3a(G)).\gamma(G)$*, and thus the hybrid algorithm has a bounded performance ratio.*

**Remark 3.** *We choose the value of* $\alpha$ *by trial and error, normally starting at* $\alpha = 1/2$*.*

## 5. Environment, Implementation, and Datasets

We used a laptop with modest computational power—8th generation Intel i5 (1.6 GHz) and 8 GB RAM—to perform the experiments. We implemented the $O(n + m)$ time version of the greedy algorithm in C++. At the time of writing this paper we did not have access to packages that offer the polynomially time bounded versions of linear programming. We used IBM Decision Optimization CPLEX Modeling (DOCPLEX) for Python to solve the LP relaxation of the problem for the LP rounding and hybrid algorithms.

Our data sources are listed below.

https://github.com/joklawitter/GraphGenerators †

http://www.inf.udec.cl/~jfuentess/datasets/graphs.php ††

http://davidchalupa.github.io/research/data/social.html [8]

https://www.cc.gatech.edu/dimacs10/downloads.shtml [23]

The graph generator at † was used to create the k-planar graphs (graphs embedded in the plane with at most $k$ crossings per edge) and k-trees (graphs with tree width $k$ with largest number of edges) up to 20,000 vertices. We also used publicly available Google+ and Pokec social-network graphs, a publicly available 1,000,000-vertex planar graph ††, and real-world DIMACS Graphs with up to more than 7,700,000 vertices. Furthermore, we generated the k-Queens graphs, hypercubes (up to 12 dimensions) and graphs in Example A.1 (Figure A1) and Example A.2.

**Remark 4.** *Throughout our experiments, the value of the solution computed by rounding algorithm* $R_2$ *was always better than the value of the solution computed by rounding algorithm* $R_1$*, as predicted in theory. Likewise, the value of the solution computed by the hybrid algorithm using* $R_2$ *was always better compared to when* $R_1$ *was used. Throughout sections six through nine (tables),* $r$ *denotes the value of the solution computed by* $R_2$*, and* $h$ *denotes the value of the solution computed by the hybrid algorithm associated with* $R_2$*. We denote by* $g$ *the value of the solution computed by the greedy algorithm. Throughout the tables in sections six through nine, the best computed values are bolded.*

## 6. Performance on k-Planar Graphs and k-Trees

In this section, we compare the performance ratios of the greedy, LP rounding and hybrid algorithms on k-planar graphs and k-trees. In all cases, the hybrid algorithm performed better than the others. Greedy performed close to hybrid and LP rounding performed the worst.

The arboricity of each of the planar graphs is at most 3. For k-trees, we use $\lceil k - \frac{(k/2)(k-1)}{n-1} \rceil$ for arboricity. For k-planar graphs, $k \geq 1$, we use the upper bound of $\lceil 8\sqrt{k} \rceil$ on arboricity. The k-planar graphs and k-trees were all made using † described in Section 5. The typical value of $\alpha$ was $1/2$, but increasing it to $3/4$ resulted in better performance in some cases. All algorithms were able to compute dominating sets in less than 2 s in all cases.

### *6.1. Performance on Sparse k-Planar Graphs and k-Trees*

In Tables 1–3, we present the performance of the algorithms on sparse k-planar graphs and sparse k-trees.

**Table 1.** Results for 0-Planar Graphs.

| $n$ | $m$ | $L^*$ | $g/L^*$ | $r/L^*$ | $h/L^*$ |
|---|---|---|---|---|---|
| 2000 | 5980 | 316.93 | 1.12 | 1.39 | **1.11** |
| 4000 | 11,972 | 620.72 | 1.16 | 1.34 | **1.14** |
| 6000 | 17,978 | 942.59 | **1.13** | 1.29 | **1.13** |
| 8000 | 23,974 | 1239.16 | 1.14 | 1.40 | **1.13** |
| 10,000 | 29,972 | 1579.06 | **1.13** | 1.27 | **1.13** |
| 12,000 | 35,973 | 1874.66 | 1.13 | 1.35 | **1.12** |
| 14,000 | 41,974 | 2185.35 | **1.14** | 1.32 | **1.14** |
| 16,000 | 47,975 | 2514.62 | 1.14 | 1.33 | **1.13** |
| 18,000 | 53,971 | 2811.98 | 1.15 | 1.35 | **1.14** |
| 20,000 | 59,971 | 3127.20 | 1.14 | 1.31 | **1.13** |

**Table 2.** Results for 5-Planar Graphs.

| $n$ | $m$ | $L^*$ | $g/L^*$ | $r/L^*$ | $h/L^*$ |
|---|---|---|---|---|---|
| 2000 | 11,465 | 171.42 | **1.19** | 1.65 | **1.19** |
| 4000 | 23,033 | 336.57 | **1.21** | 1.63 | **1.21** |
| 6000 | 34,577 | 510.02 | **1.24** | 2.19 | **1.24** |
| 8000 | 46,130 | 680.88 | **1.25** | 1.91 | **1.25** |
| 10,000 | 57,786 | 840.92 | **1.23** | 2.10 | **1.23** |
| 12,000 | 69,220 | 1019.54 | 1.23 | 2.02 | **1.22** |
| 14,000 | 80,680 | 1181.05 | **1.22** | 1.90 | **1.22** |
| 16,000 | 92,300 | 1355.13 | **1.23** | 2.03 | **1.23** |
| 18,000 | 103,862 | 1516.14 | **1.24** | 1.99 | **1.24** |
| 20,000 | 115,354 | 1689.35 | 1.22 | 2.08 | **1.21** |

**Table 3.** Results for 5-Trees.

| $n$ | $m$ | $L^*$ | $g/L^*$ | $r/L^*$ | $h/L^*$ |
|---|---|---|---|---|---|
| 2000 | 9985 | 39.00 | **1.05** | 1.08 | **1.05** |
| 4000 | 19,985 | 70.50 | **1.04** | 1.06 | **1.04** |
| 6000 | 29,985 | 90.83 | **1.03** | 1.17 | **1.03** |
| 8000 | 39,985 | 132.25 | **1.03** | 1.07 | **1.03** |
| 10,000 | 49,985 | 158.00 | **1.03** | **1.03** | **1.03** |
| 12,000 | 59,985 | 209.67 | **1.02** | 1.08 | **1.02** |
| 14,000 | 69,985 | 225.58 | **1.04** | 1.09 | **1.04** |
| 16,000 | 79,985 | 270.25 | **1.02** | 1.09 | **1.02** |
| 18,000 | 89,985 | 291.83 | **1.02** | 1.06 | **1.02** |
| 20,000 | 99,985 | 339.58 | **1.04** | 1.08 | **1.04** |

### *6.2. Performance on Dense k-Planar Graphs and k-Trees*

In Tables 4 and 5, we present the algorithms' performance on k-planar graphs where $k = \lfloor \ln(|V|) \rfloor$, and k-trees where $k = \lfloor |V|^{0.25} \rfloor$, respectively. These graphs are dense.

**Table 4.** Results for k-Planar Graphs where $k = \lfloor \ln(n) \rfloor$.

| $n$ | $m$ | $L^*$ | $g/L^*$ | $r/L^*$ | $h/L^*$ |
|---|---|---|---|---|---|
| 2000 | 12,986 | 151.97 | 1.26 | 2.11 | **1.24** |
| 4000 | 27,254 | 289.69 | **1.27** | 2.64 | **1.27** |
| 6000 | 40,885 | 431.77 | **1.26** | 2.50 | **1.26** |
| 8000 | 54,568 | 568.01 | **1.24** | 2.57 | **1.24** |
| 10,000 | 71,414 | 684.20 | 1.27 | 2.56 | **1.26** |
| 12,000 | 85,580 | 821.65 | **1.26** | 2.62 | **1.26** |
| 14,000 | 100,241 | 957.77 | **1.25** | 2.46 | **1.25** |
| 16,000 | 114,270 | 1098.18 | **1.27** | 2.21 | **1.27** |
| 18,000 | 128,725 | 1238.09 | **1.27** | 2.22 | **1.27** |
| 20,000 | 142,891 | 1368.44 | 1.26 | 2.23 | **1.25** |

**Table 5.** Results for k-Trees where $k = \lfloor n^{0.25} \rfloor$.

| $n$ | $m$ | $L^*$ | $g/L^*$ | $r/L^*$ | $h/L^*$ |
|---|---|---|---|---|---|
| 2000 | 13,972 | 15.00 | 1.07 | 1.20 | **1.00** |
| 4000 | 31,964 | 10.00 | **1.00** | **1.00** | **1.00** |
| 6000 | 53,955 | 11.00 | **1.00** | **1.00** | **1.00** |
| 8000 | 71,955 | 13.00 | **1.00** | **1.00** | **1.00** |
| 10,000 | 99,945 | 11.19 | **1.07** | 2.23 | **1.07** |
| 12,000 | 119,945 | 12.00 | **1.00** | **1.00** | **1.00** |
| 14,000 | 139,945 | 18.50 | **1.08** | 1.89 | **1.08** |
| 16,000 | 175,934 | 11.25 | **1.16** | 1.60 | **1.16** |
| 18,000 | 197,934 | 11.00 | **1.18** | 2.00 | **1.18** |
| 20,000 | 219,934 | 10.50 | **1.14** | 1.43 | **1.14** |

## 7. Performance on Hypercubes and k-Queen Graphs

In this section, we present the performance of the greedy, rounding, and hybrid algorithms on hypercubes from $d = 5$ to 12 dimensions and k-Queens graphs.

Table 6 contains the results for hypercubes. We use the arboricity for hypercubes $a = \lfloor d/2 + 1 \rfloor$ for LP rounding and hybrid [24]. For k-Queens graphs, arboricity is unknown, so we use the upper bound $3(k-1)$, where $k$ is the length of the chessboard.

For both Table 6 and 7, greedy performs the best, followed by hybrid. Rounding algorithms perform the worst by far. This is not surprising as LP rounding approaches are known to in general perform worse on dense graphs than sparse graphs. The chosen value of $\alpha$ was always $1/2$. Solutions were computed in under 8 s for all graphs and algorithms.

**Table 6.** Results for Hypercubes.

| Dimension | $n$ | $L^*$ | $g/L^*$ | $r/L^*$ | $h/L^*$ |
|---|---|---|---|---|---|
| 5 | 32 | 5.33 | **1.50** | 3.00 | **1.50** |
| 6 | 64 | 9.14 | **1.75** | 7.00 | **1.75** |
| 7 | 128 | 16.00 | **1.00** | **1.00** | **1.00** |
| 8 | 256 | 28.44 | **1.13** | 9.00 | **1.13** |
| 9 | 512 | 51.20 | **1.25** | 7.07 | 2.99 |
| 10 | 1024 | 93.09 | **1.38** | 11.00 | 2.70 |
| 11 | 2048 | 170.67 | **1.50** | 6.59 | 2.85 |
| 12 | 4096 | 315.08 | **1.63** | 13.00 | 3.14 |

**Table 7.** Results for k-Queens.

| k | $m$ | $L^*$ | $g/L^*$ | $r/L^*$ | $h/L^*$ |
|---|---|---|---|---|---|
| 15 | 5180 | 4.89 | **2.05** | 36.40 | 6.75 |
| 17 | 7616 | 5.50 | **1.82** | 44.03 | 8.91 |
| 19 | 10716 | 6.10 | **1.97** | 50.78 | 9.67 |
| 21 | 14560 | 6.71 | **1.94** | 56.91 | 11.17 |
| 23 | 19228 | 7.32 | **1.91** | 62.83 | 10.11 |
| 25 | 24800 | 7.93 | **2.02** | 69.61 | 10.34 |
| 27 | 31356 | 8.54 | **1.87** | 74.48 | 11.13 |
| 29 | 38976 | 9.15 | **1.97** | 78.81 | 11.70 |

## 8. Performance on Real-World Graphs

In this section, we present the performance of LP rounding, greedy, and hybrid on the real-world social network graphs from DIMACS [23], Google+ [8], and Pokec [8]. All of these graphs are sparse, but their arboricity is unknown. Since arboricity is unknown, we experiment with the threshold applied during LP rounding. Through experimentation, the best threshold which we found for rounding was $2/a'$, where $a' = \lceil m/(n-1) \rceil$. We denote the value of the solution computed by the LP rounding

algorithm with this threshold as $r'$ and the value of the solution computed by the hybrid algorithm with this threshold as $h'$. The chosen value of $\alpha$ was $1/2$ in all cases.

Table 8 contains the results for three sparse social network graphs from DIMACS. LP rounding performs better than the greedy and hybrid approaches, with greedy ranking last out of the algorithms tested.

**Table 8.** Results for DIMACS Graphs.

| Graph | $n$ | $m$ | $L^*$ | $g/L^*$ | $r'/L^*$ | $h'/L^*$ |
|---|---|---|---|---|---|---|
| coAuthorsDBLP | 299,067 | 977,676 | 43,969.00 | 1.02 | **1.00** | 1.02 |
| coPapersCiteseer | 434,102 | 16,036,720 | 26,040.92 | 1.12 | **1.01** | 1.12 |
| citationCiteseer | 268,495 | 1,156,647 | 43,318.85 | 1.04 | **1.02** | 1.04 |

In Tables 9 and 10, we compare $r'$, $h'$, and $g$ on the Google+ and Pokec graphs. The performance ratios, although different, happen to be very close. Thus, we list the actual sizes of the dominating sets returned by the algorithms.

Compared to the best results from [8], which used a randomized local search algorithm that is run for up to one hour, LP rounding approaches generally produced, with the exception of a few cases, a smaller or as good solution using significantly less cpu-time at less than 0.5 s for each graph.

**Table 9.** Results for *Google+* Graphs.

| $n$ | $m$ | $L^*$ | $g$ | $r'$ | $h'$ |
|---|---|---|---|---|---|
| 500 | 1006 | 42 | **42** | **42** | **42** |
| 2000 | 5343 | 170 | 176 | **170** | 176 |
| 10,000 | 33,954 | 860 | 900 | **864** | 893 |
| 20,000 | 81,352 | 1715 | 1817 | **1716** | 1800 |
| 50,000 | 231,583 | 4565 | 4849 | **4585** | 4790 |

**Table 10.** Results for *Pokec* Graphs.

| $n$ | $m$ | $L^*$ | $g$ | $r'$ | $h'$ |
|---|---|---|---|---|---|
| 500 | 993 | 16 | **16** | **16** | **16** |
| 2000 | 5893 | 75 | **75** | **75** | **75** |
| 10,000 | 44,745 | 413 | **413** | **413** | **413** |
| 20,000 | 102,826 | 921 | 928 | **921** | 923 |
| 50,000 | 281,726 | 2706 | 2773 | **2712** | 2743 |

## 9. Very Large Sparse Graphs

Table 11 contains the results on a 7 million+ vertices graph (Great Britain street network) and a 1 million vertices planar graph, where we used Lemma 1 to compute $Z = max\{M^*, N^*\}$. The LP solver crashed when it was directly applied to the Great Britain street network and took over 3.5 h of CPU time to compute an answer on the 1 million vertices planar graph. Through a search for $\alpha$, we came up with $\alpha = 3/4$ for hybrid. The hybrid algorithm's performance ratio to $Z$ was better than greedy's. Greedy took 14 s to produce a solution while hybrid took 107 s on the street network graph. On the 1 million vertices planar graph, greedy took 4 s and hybrid took 19 s.

**Table 11.** Results for Large Graphs.

| Graph | $n$ | $m$ | $Z$ | $g$ | $h'$ | $g/Z$ | $h'/Z$ |
|---|---|---|---|---|---|---|---|
| Great-Britain Street Network | 7,733,822 | 1,357,189 | 1,357,189 | 2,732,935 | **2,724,608** | 2.014 | 2.008 |
| Large Planar Graph | 1,000,000 | 2,999,978 | 84,616 | 180,686 | **176,295** | 2.135 | 2.083 |

## 10. Conclusions

Our findings indicate that all of these algorithms perform better than anticipated in theory, particularly with respect to the performance ratio. The LP rounding does well on sparse real-world graphs, consistent with theory, and normally outperforms the other algorithms. For hypercubes and k-Queens graphs (which are not sparse) greedy outperforms the rest, consistent with theory, both in terms of speed and performance ratio. For synthetic graphs (generated k-trees and k-planar graphs), LP rounding is outperformed by the other two algorithms, and the hybrid algorithm outperforms greedy.

Throughout our experimentation, the hybrid algorithm was never the worst. The hybrid algorithm's success in solving large sparse problems suggests that more research in this area will be fruitful with respect to characterizations of parameter $\alpha$. In particular, a theoretical research direction would be attempting to tighten the upper bound on the value of the solution computed by the hybrid algorithm, as stated in Remark 2. Can the upper bound on the value of the solution be shown to be better than $(2a+1) * \gamma(G)$ with the appropriate choice of $\alpha$?

**Author Contributions:** J.L. and R.P. wrote the code, collected and generated data, ran experiments and collected the results, and participated in analysis and writing the paper. F.S. conceptualized, wrote a draft of the paper, and supervised and provided guidance for the work. All authors have read and agreed to the published version of the manuscript.

**Funding:** This research received no external funding.

**Acknowledgments:** We thank the University of North Texas and the Texas Academy of Mathematics and Sciences for supporting our research. This work was initiated and completed at the University of North Texas and was supported in part by a TAMS summer scholarship.

**Conflicts of Interest:** The authors declare no conflict of interest.

## Abbreviations

The following abbreviations are used in this manuscript:

LP Linear Programming
IP Integer Programming

## Appendix A

**Example A1.** *Let $p \geq 2$ be an integer and for $i = 1, 2, \ldots, p$, let $S_i$ be a star on $2^i$ vertices. Consider a graph $G$ on $n = 2^{p+1}$ vertices whose vertices are the disjoint union of the vertices of the $S_i$'s ($i = 1, 2, \ldots, p$) plus two additional vertices $t_1$ and $t_2$. Now, place edges from $t_1$ and $t_2$ to the first half of the vertices in each $S_i$ (including the root), and the second half of the vertices in each $S_i$, respectively. Please note that the root of each $S_i$ has degree $2^i$ and the degree of both $t_1$ and $t_2$ is $2^p - 1$. Initially, greedy chooses the root of $S_p$ which can cover $2^p + 1$ vertices (including itself). Generally, at iteration $i \geq 2$, there is a tie between the root of $S_{p+1-i}$ and $t_2$ since each can cover $2^{p-2}$ uncovered vertices. If tie breaking does not result in selecting $t_2$, there will be a tie in every iteration until the algorithm returns the set of $S_i$'s ($i = 1, 2, \ldots, p$). This dominating set has cardinality $p = log(\Delta) - 1$, but $\gamma(G) = 2$, since $\{t_1, t_2\}$ is a minimum dominating set. Please note that $G$ is a planar graph.*

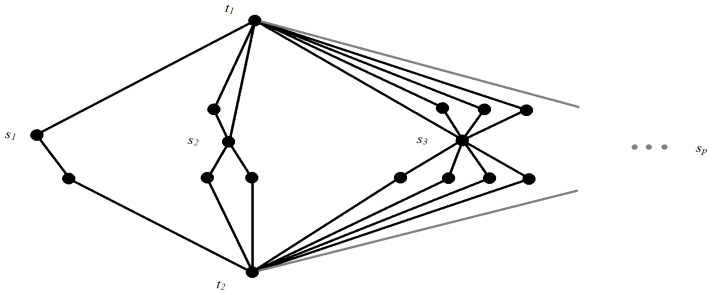


**Figure A1.** Construction of Example 1.

**Example A2.** *Let $p \geq 2$ be an integer, and let $G$ be a graph with vertices $V_1 \cup V_2$, where $V_1 = \{s_1, s_2, \ldots, s_p, t_1, t_2\}$ and $V_2 = \{v_1, v_2, \ldots, v_{2^{p+1}-2}\}$. Now make $V_1$ a clique and $V_2$ an independent set of vertices, respectively. Next, consider a linear ordering $L$ on $V_2$: for $i = 1, 2, \ldots, p$, the set of neighbors of $s_i$ in $V_2$, denoted by $W_i$, has cardinality $2^i$ and is disjoint from $W_k$, for any $k \leq i$. Finally, for $i = 1, 2, \ldots, p$ place edges between $t_1$ and the first half of the vertices in each $W_i$, and place edges between $t_2$ and the second half of the vertices in each $W_i$. Now note that the greedy algorithm will be forced to pick the vertices $s_p, s_{p-1}, \ldots, s_1$, in that order, but the minimum dominating set in $G$ is $\{t_1, t_2\}$ and $\Delta = 2^p + p + 1$.*

## References


1. Berge, C. Two Theorems in Graph Theory. *Proc. Natl. Acad. Sci. USA* **1957**, *43*, 842–844. [CrossRef] [PubMed]
2. Chartrand, G.; Lesniak, L.; Zhang, P. *Graphs and Digraphs*; CRC Press: Boca Raton, FL, USA, 2010.
3. Cormen, T.H.; Leiserson, C.E.; Rivest, R.L.; Stein, C. *Introduction to Algorithms*; MIT Press: Cambridge, MA, USA, 1990.
4. Schrijver, A. *Theory of Linear and Integer Programming*; John Wiley & Sons: Hoboken, NJ, USA, 1998.
5. Haynes, T.W.; Hedetniemi, S.; Slater, P. *Fundamentals of Domination in Graphs*; CRC Press: Boca Raton, FL, USA, 1988.
6. Clark, B.N.; Colbourn, C.J.; Johnson, D.S. Unit disk graphs. *Discret. Math.* **1990**, *86*, 165–177. [CrossRef]
7. Dinur, I.; Steurer, D. Analytical approach to parallel repetition. In *Symposium on Theory of Computing*; 2014; pp. 624–633. Available online: https://www.dsteurer.org/talk/productgames.pdf (accessed on 9 December 2020).
8. Chalupa, D. An Order-based Algorithm for Minimum Dominating Set with Application in Graph Mining. *Inf. Sci.* **2018**, *426*, 101–116. [CrossRef]
9. Campan, A.; Truta, T.M.; Beckerich, M. Fast Dominating Set Algorithms for Social Networks. In *MAICS*; 2015. Available online: http://ceur-ws.org/Vol-1353/paper_16.pdf (accessed on 9 December 2020).
10. Eubank, S.; Anil Kumar, V.S.; Marathe, M.V.; Srinivasan, A.; Wang, N. Structural and Algorithmic Aspects of Massive Social Networks. 2004; pp. 718–727. Available online: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.214.2097&rep=rep1&type=pdf (accessed on 9 December 2020).
11. Sanchis, L.A. Experimental Analysis of Heuristic Algorithms for the Dominating Set Problem. *Algorithmica* **2002**, *33*, 3–18. [CrossRef]
12. Siebertz, S. Greedy domination on biclique-free graphs. *Inf. Process. Lett.* **2019**, *145*, 64–67 [CrossRef]
13. Bansal, N.; Umboh, S.W. Tight approximation bounds for dominating set on graphs of bounded arboricity. *Inf. Process. Lett.* **2017**, *122*, 21–24. [CrossRef]
14. Dvořák, Z. On distance r-dominating and 2r-independent sets in sparse graphs. *J. Graph Theory* **2017**. Available online: https://arxiv.org/pdf/1710.10010.pdf (accessed on 9 December 2020).
15. Nash-Williams, C.S.J. Decomposition of finite graphs into forests. *J. Lond. Math. Soc.* **1964**, *1–39*, 12. [CrossRef]
16. Gabow, H.N.; Westermann, H.H. Forests, frames, and games: Algorithms for matroid sums and applications. *Algorithmica* **1992**, *7*, 465–497. [CrossRef]
17. Chvatal, V. A greedy heuristic for the set-covering problem. *Math. Oper. Res.* **1979**, *4*, 233–235. [CrossRef]
18. Lovász, L. On the Ratio of Optimal Integral and Fractional Covers. *Discret. Math.* **1975**, *13*, 383–390. [CrossRef]

19. Brönnimann, H.; Goodrich, M.T. Almost Optimal Set Covers in Finite VC-Dimension. *Discret. Comput. Geom.* **1995**, *14*, 463–479. [CrossRef]
20. Karp, R.M. Reducibility among Combinatorial Problems. In *Complexity of Computer Computations*; Miller, R.E., Thatcher, J.W., Bohlinger, J.D., Eds.; Plenum: New York, NY, USA, 1972; pp. 85–103.
21. Dvořák, Z. Constant-factor approximation of domination number in sparse graphs. *Eur. J. Comb.* **2013**, *34*, 833–840. [CrossRef]
22. Bohme, T.; Mohar, B. Domination, packing and excluded minors. *Electron. J. Comb.* **2003**, *10*, N9. [CrossRef]
23. Bader, D.A.; Kappes, A.; Meyerhenke, H.; Sanders, P.; Schulz, C.; Wagner, D. Benchmarking for Graph Clustering and Partitioning. In *Encyclopedia of Social Network Analysis and Mining*; Springer: Berlin/Heidelberg, Germany, 2014; pp. 73–82.
24. Karisani, N.; Mahmoodian, E.S. On the Construction of Tree Decompositions of Hypercubes. *arXiv* **2013**, arXiv:1311.4852.