Linear Programming and the Worst-Case Analysis of Greedy Algorithms on Cubic Graphs ⋆

W. Duckworth [a,1], N.C. Wormald [b,2]

[a] *Department of Computing, Macquarie University, Sydney NSW 2109, Australia.*

[b] *Department of Combinatorics & Optimization, University of Waterloo, Waterloo ON, Canada N2L 3G1.*

Abstract

We introduce a technique using linear programming that may be used to analyse the worst-case performance of a class of greedy heuristics for certain optimisation problems on regular graphs. We demonstrate the use of this technique on heuristics for bounding the size of a minimum maximal matching (MMM), a minimum connected dominating set (MCDS) and a minimum independent dominating set (MIDS) in cubic graphs. As a result, we show that for n-vertex connected cubic graphs, the size of a MMM is at most $9n/20 + O(1)$, the size of a MCDS is at most $3n/4 + O(1)$ and the size of a MIDS is at most $29n/70 + O(1)$. We also consider n-vertex connected cubic graphs of girth at least 5 and for such graphs we show that the size of a MMM is at most $3n/7 + O(1)$, the size of a MCDS is at most $2n/3 + O(1)$ and the size of a MIDS is at most $3n/8 + O(1)$.

Key words: worst-case analysis, cubic, 3-regular, graphs, linear programming.

⋆ This research was carried out while the authors were in the Department of Mathematics and Statistics, The University of Melbourne, VIC 3010, Australia.

* Corresponding author.

Email addresses: billy@ics.mq.edu.au (W. Duckworth), nwormald@math.uwaterloo.ca (N.C. Wormald).

URLs: http://www.ics.mq.edu.au/~billy (W. Duckworth), http://www.math.uwaterloo.ca/~nwormald (N.C. Wormald).

[1] Research supported by Macquarie University while the author was supported by the Macquarie University Research Fellowships Grants Scheme.

[2] Research supported by the Australian Research Council while the author was affiliated with the Department of Mathematics and Statistics, The University of Melbourne; currently supported by the Canada Research Chairs program.

1 Introduction

Many NP-hard graph-theoretic optimisation problems remain NP-hard when the input is restricted to graphs of bounded or regular degree, even when the maximum degree of the graph is 3, for example, Maximum Independent Set [11, problem GT20] and Minimum Dominating Set [11, problem GT2] to name but two. (See, for example, [1] for recent results on the complexity and approximability of these problems.) In this paper, we introduce a technique that may be used to analyse the worst-case performance of greedy algorithms on cubic (*i.e.* 3-regular) graphs. The technique uses linear programming and may be applied to a variety of graph-theoretic optimisation problems. Suitable problems would include those problems where, given a graph, we are required to find a subset of the vertices (or edges) involving local conditions on the vertices and (or) edges. These include problems such as Minimum Vertex Cover [11, problem GT1], Maximum Induced Matching [6] and Maximum 2-Independent Set [19]. The technique could also be applied to regular graphs of higher degree, but with dubious benefit as the effort required would be much greater.

The technique we describe provides a method of comparing the performance of different greedy algorithms for a particular optimisation problem, in some cases determining the one with the best worst-case performance. In this way, we can also obtain lower or upper bounds on the cardinality of the sets of vertices (or edges) of interest. Using this technique, it is simple to modify the analysis in order to investigate the performance of an algorithm when the input is restricted to (for example) cubic graphs of bounded girth or cubic graphs with a forbidden subgraph.

Besides introducing a new general approach to giving bounds on the performance of greedy algorithms using linear programming, we demonstrate how the linear programming solution can sometimes lead to constructions that achieve the bounds obtained. In these cases, the worst case performance of these particular algorithms is determined quite precisely, even though the implied bound on the size of a the minimal or maximal subset of edges or vertices is not sharp.

Throughout this paper, when discussing any cubic graph on n vertices, we assume n to be even and we also assume the graph to contain no loops nor multiple edges. The cubic graphs are assumed to be connected; for disconnected graphs, for each particular problem under consideration, applying our algorithm for that problem in turn to each connected component would, of course, cause the constant terms in our results to be multiplied by the number of components.

In this paper, we present and analyse greedy algorithms for three problems related to domination in cubic graphs. A (vertex) dominating set of a graph $G = (V, E)$ is a set $D \subseteq V(G)$ such that for every vertex $v \in V(G)$, either $v \in D$ or v has a neighbour in D. An edge dominating set of a graph $G = (V, E)$ is a set $F \subseteq E(G)$ such that for every edge $e \in E(G)$, either $e \in F$ or e shares a common end-point with an edge of F. An independent set of a graph $G = (V, E)$ is a set $I \subseteq V(G)$ such that no two vertices of I are connected by an edge of $E(G)$. A matching of a graph $G = (V, E)$ is a set $M \subseteq E(G)$ such that no two edges of M share a common end-point.

We now formally define the problems that we consider in this paper.

Minimum Independent Dominating Set : An *independent dominating set* (IDS) of a graph $G = (V, E)$ is a set of vertices $\mathcal{I} \subseteq V(G)$ that is both an independent set and a dominating set. A minimum independent dominating set (MIDS) is therefore an IDS of minimum cardinality.

Minimum Maximal Matching : A *maximal matching* (MM) of a graph $G = (V, E)$ is a set of edges $\mathcal{E} \subseteq E(G)$ such that $\mathcal{E}$ is a matching and every edge in $E(G) \setminus \mathcal{E}$ shares at least one end-point with an edge of $\mathcal{E}$. A minimum maximal matching (MMM) is therefore a MM of minimum cardinality. Note that finding a MMM of a graph G is equivalent to finding a MIDS of the line graph of G. A MMM of G may also be considered as a minimum independent-edge dominating set of G.

Minimum Connected Dominating Set : A *connected dominating set* (CDS) of a graph $G = (V, E)$ is a set of vertices $\mathcal{C} \subseteq V(G)$ that is a dominating set with the additional property that the subgraph induced by the vertices of $\mathcal{C}$ is connected. A minimum connected dominating set (MCDS) is therefore a CDS of minimum cardinality.

Let $\mathcal{MCDS}$, $\mathcal{MIDS}$ and $\mathcal{MMM}$ denote the problems of finding a MCDS, a MIDS and a MMM of a graph, respectively. The algorithms we present in this paper are only heuristics for these problems; they find *small* sets when the problem asks for a *minimum* set.

Griggs, Kleitman and Shastri [12] showed that every n-vertex connected cubic graph has a spanning tree with at least $\lceil (n/4)+2 \rceil$ leaves, implying (by deleting the leaves) that such graphs have a CDS of size at most $3n/4$. Lam *et al* [17] showed that for $n \geq 10$, the size of a MIDS of n-vertex connected cubic graphs is at most $2n/5$. Both these results use rather complicated and elaborate arguments, so the extraction of an algorithm from them can be difficult. By contrast, our approach is an attempt to automate the proofs, greatly reducing the proportion of ad hoc arguments by using computer calculations.

Note that for n-vertex cubic graphs, it is simple to verify that, the size of a

MM is at least $3n/10$, the size of a CDS is at least $(n-2)/2$ and the size of an IDS is at least $n/4$. In this paper we prove that for n-vertex connected cubic graphs, the size of a MMM is at most $9n/20 + O(1)$, the size of a MCDS is at most $3n/4 + O(1)$ and the size of a MIDS is at most $29n/70 + O(1)$. For $\mathcal{MMM}$ (as far as the authors are aware) no other non-trivial approximation results were previously known for this problem when the input is restricted to cubic graphs.

Many optimisation problems have been considered when the input is restricted, see, for example, [3,23,24] for the study of the similar problem of maximum independent set size in graphs with restricted girth. Ever-increasing bounds were obtained as the girth increases. We note that for cubic graphs of girth 4 (in relation to all problems that we consider in this paper) our analysis gives no improved result than the unrestricted case. We therefore consider n-vertex connected cubic graphs of girth at least 5. For such graphs, we show that the size of a MMM is at most $3n/7+O(1)$, the size of a MCDS is at most $2n/3 + O(1)$ and the size of a MIDS is at most $3n/8 + O(1)$.

The following section describes the notion of analysing the worst-case performance of greedy algorithms using linear programming. Our algorithms (and their analysis) for $\mathcal{MMM}$, $\mathcal{MCDS}$ and $\mathcal{MIDS}$ of cubic graphs are given in Sections 3, 4 and 5 respectively. We conclude in Section 6 by mentioning some of the other problems to which we have applied this technique.

2 Worst-Case Analysis and Linear Programs

In each of the problems that we consider in this paper, we are given a graph and aim to find a subset of the vertices (or edges) of small cardinality that satisfies local conditions on the vertices and (or) edges. The algorithms we introduce in the following sections are greedy algorithms based on selecting vertices (that have particular properties) from an ever-shrinking subgraph of the input graph. After choosing such a vertex, which we call the *target*, a vertex (or edge) near the target is selected to be added to the set we wish to construct. Once this selection has been made, edges and vertices are deleted from the (sub)graph in order to guarantee that, after the next element is selected to be added to the set, the set constructed thus far satisfies the given requirements (domination, independence, *etc.*).

We describe the algorithms in terms of a series of *operations*; an operation being the process of selecting one or more elements to be added to the set and performing the necessary deletion of vertices and edges. The operations will be classified into types, according to their effects on the neighbourhood of the selected element.

The first operation of each algorithm involves selecting the first element to be added to the set and deleting the appropriate vertices and edges. For each subsequent operation (before the completion of the algorithm) we note that there must always exist a vertex of degree strictly less than 3 since the input graph is assumed to be connected.

At any stage of a given algorithm, we may characterise the vertices of the graph based on their degree (for $\mathcal{MCDS}$ we also consider their "colour" and this will be defined in the relevant section). Let V_i denote such a set and let Y_i denote $|V_i|$, where $1 \leq i \leq t$ for some t. Operations performed by a given algorithm will cause a change in the values of some of the variables Y_i.

As an example, we consider $\mathcal{MIDS}$ where we are required to find an IDS, $\mathcal{I}$, of small cardinality. Each operation may be represented by equations showing the change in the variables Y_i and the change in the size of $\mathcal{I}$.

Consider Figure 1 in which vertex 1 has degree 1, vertex 5 has degree 2 (all other vertices have degree 3) and assume that vertex 2 has been selected to be added to $\mathcal{I}$. Vertex 2 and its neighbours are deleted along with all their incident edges ensuring that, after the next selection of a vertex to be added to the IDS, the set $\mathcal{I}$ constructed thus far is indeed independent and dominating. Black vertices indicate those which are selected to be added to $\mathcal{I}$ and dotted lines indicate edges that are deleted. The deletion of these edges may cause additional vertices to be added to $\mathcal{I}$ (vertices that are isolated as a consequence of the deletion of these edges) and vertex 5 in the figure is such a vertex.

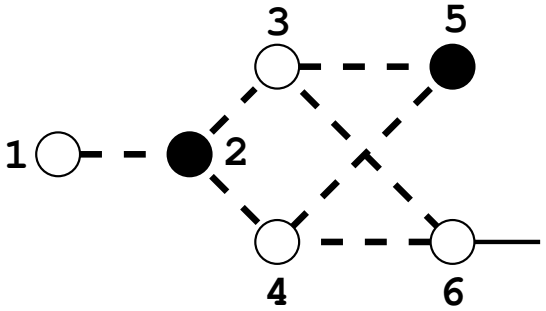


Fig. 1. An example operation

For an operation Op, we let $\Delta Y_i(Op)$ denote the *net* change in Y_i due to operation Op. We use $m(Op)$ to denote the increase in the size of $\mathcal{I}$ due to operation Op and its value will depend on the individual operation being performed. We represent an operation by a set of equations. The equations representing the operation given in Figure 1 are $\Delta Y_3 = -4$, $\Delta Y_2 = -1$, $\Delta Y_1 = 0$ and $m(Op) = 2$ (since the edges incident with the vertices $1, \ldots, 5$ are deleted, vertex 6 is changed from a vertex of degree 3 to a vertex of degree 1 and the size of $\mathcal{I}$ increases by 2 as a result of this operation).

At the start of each of our algorithms, all vertices are of degree 3 (for $\mathcal{MCDS}$ all vertices are also of the same "colour"), and consequently belong to the same set V_i, which we may assume is V_t. So, initially $Y_t = n$ and $Y_i = 0$ for all $1 \leq i < t$, and at the end of the algorithm $Y_i = 0$ for all $1 \leq i \leq t$. This implies that the total change in Y_t over the execution of the algorithm is $-n$ (since vertices of V_t are destroyed but never created). The *net* change in Y_i for $1 \leq i < t$ will be zero over the execution of the algorithm.

We construct the set of all possible operations performed by an algorithm for a given problem by considering each possible subgraph that may be encountered. Denote the set of all operations using OPS and for each operation in this set, there is a set of corresponding equations of the above form.

For operation $Op \in OPS$, we use $r(Op)$ to denote the number of times operation Op is performed by an algorithm. Then, the solution to the linear program LP_0 given in Figure 2 gives an upper bound on the size of the set returned by the algorithm. Note that $\Delta Y_i(Op)$ for $1 \leq i < t$ can be considered

$$\textbf{MAXIMISE}: \quad \sum_{Op \in OPS} m(Op) r(Op)$$

$$\begin{aligned}
\textbf{SUBJECT TO}: \mathbf{C_t}: & \sum_{Op \in OPS} \Delta Y_t(Op) r(Op) = -n \\
\mathbf{C_i}: & \sum_{Op \in OPS} \Delta Y_i(Op) r(Op) = 0 \qquad 1 \leq i < t \\
& r(Op) \geq 0 \qquad Op \in OPS
\end{aligned}$$

Fig. 2. The linear program LP_0

to be composed of two parts since an operation may destroy vertices of V_i and simultaneously create new vertices of V_i. We denote the negation of the number of vertices of V_i destroyed by operation Op using $Y_i^-(Op)$ and similarly the number created by $Y_i^+(Op)$, so that, $Y_i^+(Op) + Y_i^-(Op) = \Delta Y_i(Op)$.

To eliminate operations that would otherwise increase the maximum value of the solution to the linear program, and similarly to add further constraints, we will prioritise the operations to some extent. In two of our examples we find it convenient to use the following idea. Before each operation, we have an ordered list of sets S_i of vertices, called a *priority list*. The sets in this list are in decreasing order of priority. The priority of a vertex is defined to be the priority of the highest priority set it appears in (or, if it appears in none, the vertex has "arbitrarily low" priority). Vertices of lower priority can be chosen as the target only when the sets of higher priority are empty. For example, the priority list for our algorithm for $\mathcal{MIDS}$ is as follows.

S_1: vertices that have at least one neighbour of degree 1,

S_2: vertices of degree 2 (and their neighbours) that have precisely one vertex at distance 2,

S_2: vertices of degree 2 (and their neighbours) that have precisely two vertices at distance 2,

S_3: vertices of degree 2 (and their neighbours) that have precisely three vertices at distance 2,

S_4: vertices of degree 2 (and their neighbours) that have precisely four vertices at distance 2.

For each algorithm, we will define rules which further prioritise operations of otherwise equal priority.

Since the input graph is assumed to be connected, the first operation of an algorithm is unique in the sense that it is the only operation where the minimum degree of the vertices is 3. Let $OPS_0 \subseteq OPS$ denote the set of operations for which $Y_i^- = 0$ for all $1 \leq i < t$. OPS_0 therefore consists of all the possible first operations. Due to the priorities of the algorithm, we may exclude certain operations. We denote the set of operations that are excluded on this basis by OPS_1. To analyse the algorithm, we include only those operations that are permitted (by the priorities of the algorithm) to occur after the first operation. Denote this set of operations by $OPS_2 = OPS \setminus \{OPS_0 \cup OPS_1\}$. For our $\mathcal{MIDS}$ example, consider the operations given in Figure 3 and assume that vertex v has been selected to be added to $\mathcal{I}$. The operation in Figure 3(a) is excluded as it is in OPS_0. As the algorithm prioritises the selection of a vertex with a neighbour of degree 1 over that of any other vertex, operations such as that given in Figure 3(b) are in OPS_1 and are also excluded. When we restrict the input to cubic graphs of girth at least 5, further operations are also excluded such as the example given in Figure 3(c).

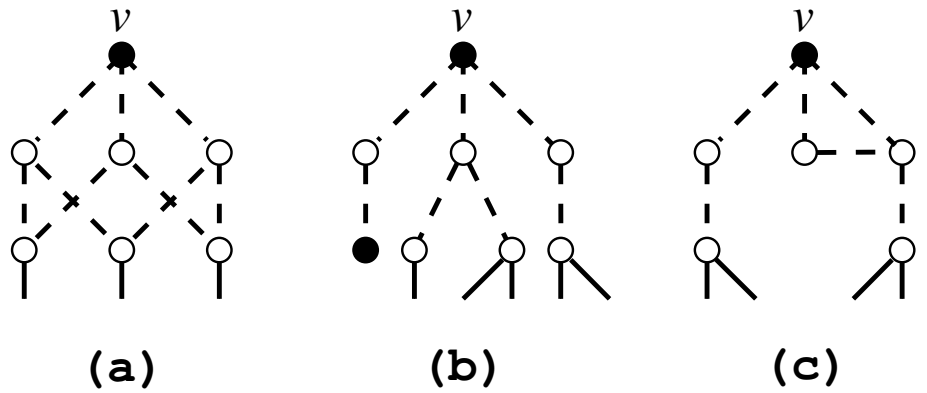


Fig. 3. Excluded operations

In each of our algorithms, there will be a γ (actually 1) such that all operations destroying at least one vertex in V_γ to have priority over all others. It follows that

(A) when $Y_\gamma > 0$, any operation Op being performed must have $Y_\gamma^-(Op) < 0$.

Let K denote the possible range of values for $Y_\gamma^-(Op)$ where $Op \in OPS_2$ (note that all such values are negative). For $-k \in K$, let $s_k = \max(0, Y_\gamma - k + 1)$, *i.e.* the number of vertices in V_γ over and above $k-1$ (if any). Consider the possible contributions to s_k from the various operations.

From property (A), if $Y_\gamma^-(Op) = 0$, then Op cannot be performed due to the priority constraints unless $Y_\gamma = 0$. Thus, such an operation contributes $\max(0, Y_\gamma^+(Op) - k + 1)$ to s_k. If $Y_\gamma^-(Op) < 0$ and $\Delta Y_\gamma(Op) \geq 0$, then Op contributes at most $\Delta Y_\gamma(Op)$ to s_k. No other operation can increase s_k. On the other hand, if $Y_\gamma^-(Op) \leq -k$ and $\Delta Y_\gamma(Op) < 0$, then Op must *subtract* at least $m_{\gamma,k} := \min(-\Delta Y_\gamma(Op), -Y_\gamma^- - k + 1)$ from s_k. It follows that the net increase in s_k throughout the algorithm, after the first step, is bounded above by the left hand side of the following constraint, which we call $\mathbf{C_{P_k}(s)}$:

$$\sum_{\substack{Y_\gamma^-(Op)=0 \\ Y_\gamma^+(Op)\geq k \\ Op\in OPS_2}} (Y_\gamma^+(Op) - k + 1)r(Op) + \sum_{\substack{Y_\gamma^-(Op)<0 \\ \Delta Y_\gamma(Op)>0 \\ Op\in OPS_2}} \Delta Y_\gamma(Op)r(Op) - \sum_{\substack{Y_\gamma^-(Op)\leq -k \\ \Delta Y_\gamma(Op)<0 \\ Op\in OPS_2}} m_{\gamma,k}r(Op) \qquad \geq -s.$$

If s_k has value s after the first operation then, since $s_k = 0$ at the end of the algorithm, $\mathbf{C_{P_k}(s)}$ must hold (noting that the first operation cannot be in OPS_2). We refer to these constraints, for each $-k \in K$, as *priority constraints*.

Our algorithms will also be such that

(B) vertices in V_γ have degree less than 3.

This leads to extra priority constraints $\mathbf{C'_{P_k}}(s)$ for each positive k:

$$\sum_{\substack{Y_\gamma^-(Op)=0 \\ Op\in OPS_2}} \lfloor Y_\gamma^+(Op)/k \rfloor r(Op) + \sum_{\substack{Y_\gamma^-(Op)<0 \\ \Delta Y_\gamma(Op)>0 \\ Op\in OPS_2}} \lceil \Delta Y_\gamma(Op)/k \rceil r(Op) - \sum_{\substack{\Delta Y_\gamma(Op)\leq -k \\ Op\in OPS_2}} \lfloor -\Delta Y_\gamma(Op)/k \rfloor r(Op) \geq -s,$$

where $-s$ is determined by the first operation.

The justification for this constraint is as follows. Let $Y_{\gamma,k} = \lfloor Y_\gamma/k \rfloor$. By property (B), the net change in $Y_{\gamma,k}$ over the course of the whole algorithm is 0. The third summation, being subtracted, is a lower bound on the net decrease in $Y_{\gamma,k}$ due to operations which decrease Y_γ. The first two summations provide an upper bound on the net increase in $Y_{\gamma,k}$ due to all other operations, apart from the increase s due to the first operation. The operations in the first summation can only be performed, in view of condition (A), when $Y_\gamma = 0$, and so $\lfloor Y_\gamma^+(Op)/k \rfloor$ is the actual increase in $Y_{\gamma,k}$ due to such an operation. Any other operation Op which can increase $Y_{\gamma,k}$ must have $Y_\gamma^-(Op) < 0$ and $\Delta Y_\gamma(Op) > 0$, and $\lceil \Delta Y_\gamma(Op)/k \rceil$ is the maximum possible increase in $Y_{\gamma,k}$ in such a step.

For any possible initial operation Op_{init}, consider the linear program obtained from LP_0 by adding any prescribed set of the priority constraints, excluding the operations in $\{OPS_0 \cup OPS_1\}$, and altering to represent the part of the algorithm remaining after the first step. Denote this linear program by LP_1. In comparison with LP_0, the right hand sides of the constraints $\mathbf{C_i}$ will have changed by $O(1)$ (representing the changes in the variables due to Op_{init}). Then create the linear program LP_2 from LP_1 by setting the right hand side of all constraints (except $\mathbf{C_t}$) to 0. This basically ignores the effect if the initial operation Op_{init}. LP_2 is then independent of Op_{init} and differs from LP_0 in that all operations in $\{OPS_0 \cup OPS_1\}$ are excluded, and a set of priority constraints have been added with $s = 0$ in all cases. We scale all linear programs by $1/n$ (so that the right hand side of the constraint $\mathbf{C_t}$ becomes -1) and, using a linear program solver, solve LP_2. Post-optimal analysis of this solution will indicate the degree to which the solution to LP_1 can differ from this.

Lemma 1 *For any one of the problems under consideration, and for any valid initial operation, the solutions of LP_1 and LP_2 differ by at most c/n for some constant c.*

Proof: We analyse the effect of each possible operation in OPS_0 by considering how such an operation may affect the right hand sides of the constraints. (We refer the reader to [2,9,22], for example, for a background in the theory of linear programming.)

After scaling, we represent the column vector of the right hand sides of the constraints of LP_2 by

$$\mathbf{b} = [-1, 0, \ldots, 0]^{\mathbf{T}}.$$

Let $\Delta\mathbf{b}_i$ for $1 \leq i \leq t$ represent the change in the right hand side of constraint $\mathbf{C_i}$: in passing from LP_2 to LP_1. Thus, $-n\Delta\mathbf{b}_i$ is the change in Y_i due to the initial operation. We represent the column vector of changes to the right hand sides of the constraints in passing from LP_2 to LP_1 by

$$\Delta\mathbf{b} = [\Delta\mathbf{b}_t, \Delta\mathbf{b}_{t-1}, \ldots, \Delta\mathbf{b}_1, 0, \ldots, 0]^{\mathbf{T}}.$$

Let κ_i denote the optimum value of the objective function of the linear program LP_i and let y^* be an optimum dual solution. If we replace $\mathbf{b}$ by $\mathbf{b} - \Delta\mathbf{b}$, we know [22, equation (20) page 126] that $\kappa_1 \leq \kappa_2 - y^*\Delta\mathbf{b}$ and as $\Delta\mathbf{b}_i = c_i/n$ for some constant c_i depending on i, the solutions to LP_1 and LP_2 differ by at most c/n for some constant c. □

One of the themes of this work is that instead of developing ad hoc arguments for each problem of this type, the same general argument can be used and to some extent automated. For the present work, our elimination of operations that cannot occur in LP_2 due to prioritisation is simply by inspection, but this too could presumably be automated. (We did use a degree of automation in generating the possible operations and the LP constraints.) One could possibly construct a program for which the input is a list of priorities in some form, and the output is an upper or lower bound from LP_2.

Apart from giving an upper bound on the size of the set of interest, often, the solution to the linear program may also be used to construct a subgraph of a cubic graph for which the given algorithm has a worst case indicated by the solution to the linear program. From this subgraph we are able to construct an infinite family of cubic graphs for which the given algorithm has a worst case indicated by the solution to within a constant number of vertices of the input graph.

In general, there may be many different combinations of operations that, for a given problem, give the same numerical solution. In particular, two or more sets of operations may have equivalent sets of equations for the overall changes in the variables. As a consequence of this, the subgraph constructed from the solution may not be of the simplest form.

For a given algorithm, additional priorities may be present in order to decide which element is selected to be added to the set under construction. No extra constraints are added to ensure these priorities are adhered to (although we do exclude certain operations on this basis). Due to these priorities, the subgraph constructed from the solution may be infeasible for the given algorithm. Should this occur, we can attempt to simplify the subgraph indicated by the solution. Replacing operations by other operations, that have the same sets of equations, may remove the conflict of the solution with the non-constrained priorities of the algorithm. By replacing sets of operations by ones with equivalent sets of equations, the number of operations in the solution may possibly be reduced.

Given a solution to the linear program, we may try to create a cubic graph which "corresponds" to the solution by first finding a "piece" of a cubic graph with one vertex of degree 2 say, on which the algorithm would perform the desired operations with the desired relative frequencies, up to the point that there is again only one vertex of degree 2.

We may then take multiple copies of the subgraph formed by the deleted vertices, and identify a vertex in one copy with a vertex in the next copy to form one or more *chains*. Chains of subgraphs may be formed into a cubic graph by identifying vertices in the first operation with vertices in either ends of the chains.

Figure 4 represents one of the many possible ways of how these example graphs may be formed. The shaded region (and an incident edge) represents the repeating subgraph and the black vertex represents the vertex that is selected to be added to the set by the initial operation of the algorithm. Each repeated subgraph in the chains is then processed in turn until the last operation is performed.

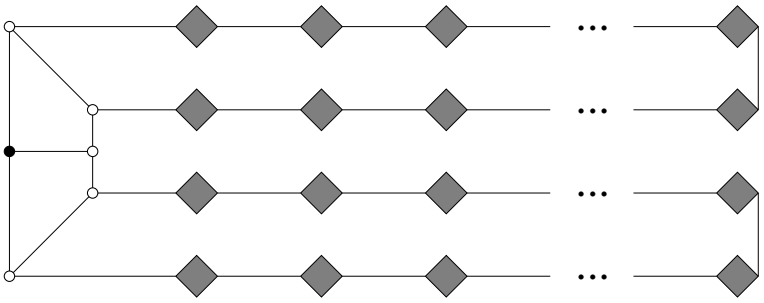


Fig. 4. Forming a cubic graph

In order to find families of cubic graphs for which a given algorithm has a worst-case performance indicated by the solution to the corresponding linear program, to within a constant number of vertices, it suffices to find the corresponding repeating subgraph.

3 Small Maximal Matchings

Yannakakis and Gavril [26] showed that the size of a smallest edge dominating set (EDS) of a graph is the same as the size of a smallest maximal matching (MM) and that the problem of finding a *minimum* edge dominating set (MEDS) is NP-hard even when restricted to planar or bipartite graphs of maximum degree 3. In the same paper they also gave a polynomial time algorithm that finds a MEDS of trees. Horton and Kilakos [15] showed that the problem of finding a MEDS remains NP-hard for planar bipartite graphs and planar cubic graphs. They also gave a polynomial time algorithm that finds a MEDS for various classes of chordal graphs. More recently, Zito [27], extended these NP-hardness results to include bipartite $(ks, 3s)$-graphs for every integer $s > 0$ and for $k \in \{1, 2\}$. (A (Δ, δ)-graph is a graph with maximum degree Δ and minimum degree δ.) It is simple to verify that, for cubic graphs, the problem of finding a minimum maximal matching (MMM) is approximable within 5/3.

Zito [28] showed that for a *random* n-vertex cubic graph G, the size of a MMM of G, $\beta(G)$, asymptotically almost surely (a.a.s. *i.e.* with probability tending to 1 as n goes to infinity) satisfies $0.3158n < \beta(G) < 0.47653n$. This upper bound has since been improved to $0.34622n$ [4].

In this section, we present an algorithm that finds a small MM of cubic graphs. We analyse the worst-case performance of this algorithm using the linear programming technique outlined in Section 2 and show that for n-vertex connected cubic graphs, the algorithm returns a MM of size at most $9n/20+O(1)$. We also show that there exists infinitely many n-vertex cubic graphs that have no MM of size less than $3n/8$. When we restrict the input to be n-vertex connected cubic graphs of girth at least 5, the algorithm returns a MM of size at most $3n/7 + O(1)$.

3.1 Algorithm Edge_Greedy

We describe a greedy algorithm, Edge_Greedy, that is based on selecting edges which have an end-point that has a neighbour of minimum degree and finds a small MM, $\mathcal{E}$, of an n-vertex cubic graph G. In order to guarantee that the matching chosen is indeed a matching and maximal, once an edge e is chosen to be added to the matching, all edges incident with the end-points of e are deleted and any isolated edges created due to the deletion of these edges are added to the matching. We categorise the vertices of the graph at any stage of the algorithm by their current degree so that for $1 \leq i \leq 3$, V_i denotes the set of vertices of degree i. Define $\tau(e)$ to be the ratio of the increase in the size of the matching to the number of edges deleted when an operation is performed after selecting the edge e to be added to the matching.

Figure 5 shows an example of an operation for this algorithm. The edge e has been chosen to be added to the matching and deleted edges are indicated by dotted lines. The edge e' is isolated as a consequence of deleting these edges and is also added to the matching.

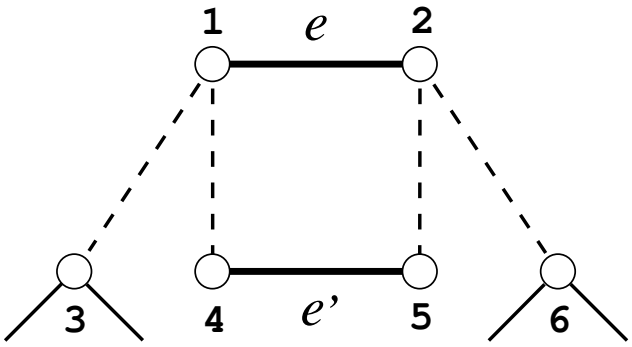


Fig. 5. An example operation for Edge_Greedy

The set of equations associated with the example of Figure 5 is $\Delta Y_3 = -4$, $\Delta Y_2 = 0$, $\Delta Y_1 = 0$ and $m(Op) = 2$ (as the edges incident with the vertices 1, 2, 4 and 5 are deleted, vertices 3 and 6 are changed from vertices of degree 3 to vertices of degree 2 and the size of the matching is increased by 2 due to this operation). For this operation $\tau(e) = 1/3$. Note that in this operation, it is assumed that the current minimum degree in G is 2, therefore no other edges may be isolated by this operation.

For a given set of vertices S, let $(S, *)$ denote the set of all edges incident with the vertices of S. The algorithm Edge_Greedy is given in Figure 6. The function **MinN**(T) operates on the given set of vertices T and returns an edge e for which $\tau(e)$ is the minimum of all edges incident with the vertices of T. The function **Add_Isolates()** involves the process of adding any isolated edges to the matching and deleting them from G.

$e = (u, v) \leftarrow$ **MinN** (V);
$\mathcal{E} \leftarrow \{e\}$;
delete $(\{u, v\}, *)$;
Add_Isolates();

while $(\{V_1 \cup V_2\} \neq \emptyset)$
do
    $S \leftarrow \{v \mid \{\{N(v) \cap V_1\} \neq \emptyset\}\}$;
    if $(S = \emptyset)$ $S \leftarrow \{v \mid \{\{N(v) \cap V_2\} \neq \emptyset\}\}$;
    $e = (u, v) \leftarrow$ **MinN** (S);
    $\mathcal{E} \leftarrow \mathcal{E} \cup \{e\}$;
    delete $(\{u, v\}, *)$;
    Add_Isolates();
od

Fig. 6. Algorithm Edge_Greedy

3.2 Edge_Greedy Analysis

The initial operation of the algorithm selects the first edge to be added to the matching and deletes the necessary edges. Subsequently, edges are repeatedly selected to be added to the matching based on the minimum degree of the vertices available. At each iteration, we use the following priority list to choose a target vertex.

S_1: vertices that have at least one neighbour of degree 1,

S_2: vertices that have at least one neighbour of degree 2.

As a further restriction, we choose an edge e to add to the matching for which $\tau(e)$ is the minimum of all edges incident with the vertices of S_i. Should there exist two edges in T, say e and e', for which $\tau(e) = \tau(e')$, the function returns the edge with the fewest vertices neighbouring its endpoints. Any further ties are broken arbitrarily. We now analyse the worst-case performance of Edge_Greedy and in this way prove the following theorem.

Theorem 1 *Given a connected, n-vertex, cubic graph, algorithm Edge_Greedy returns a maximal matching of size at most $9n/20 + O(1)$.*

Proof: We form the linear program LP_2 as outlined in Section 2. From the set OPS_1 of all operations that may occur after the initial operation, we exclude those that may not be performed due to the priorities of the algorithm. As we prioritise the selection of a vertex with a neighbour of degree 1 over the selection of any other vertex when $Y_1 = 0$, we have $\gamma = 1$. So for each k such that $V_1^-(Op) = k$, we add the constraints $C_{P_k}(0)$ and $C'_{P_k}(0)$. (In the case of $C'_{P_k}(0)$, the choice of which k to use is rather arbitrary; all choices produce valid results.)

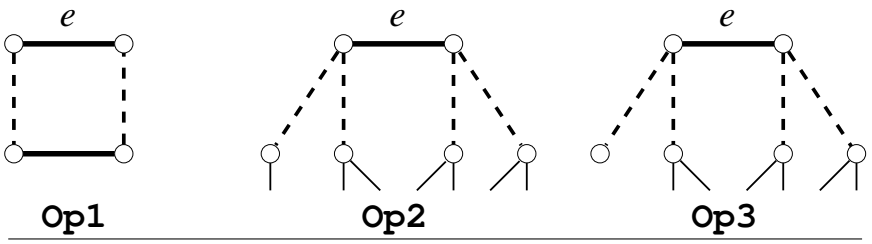
Using an exact linear program solver (in Maple), we solve LP_2. The solution is shown in Figure 7 and by Lemma 1 this shows that for n-vertex cubic graphs, algorithm Edge_Greedy returns a MM of size at most $9n/20 + O(1)$. The operations Op_i (for $i \in \{1, 2, 3\}$) are shown in Figure 8 and their corresponding equations may be derived from the table in the figure. For each operation, the edge e is selected by the algorithm to be added to the matching. Edges added to the matching are indicated by heavier lines and deleted edges are indicated by dotted lines. □

Op_1	Op_2	Op_3	Solution
$\frac{1}{8}$	$\frac{1}{10}$	$\frac{1}{10}$	$\frac{9}{20}$

Fig. 7. A solution to the LP for Edge_Greedy

A subgraph that forms part of a cubic graph that realises the solution of the linear program is shown in Figure 9. For each component (which has forty vertices), eighteen of the sixty edges are chosen to be added to the matching in the numbered order. Edges labelled 1,4,7 and 10 are each added by an Op_2, those labelled 2,5,8 and 12 are each added by an Op_3 and the remaining edges are added in pairs, each pair by an Op_1.

An edge i^* denotes that this edge was isolated and added to the matching in the same operation that the edge i was chosen for addition. Connecting a number of these subgraphs by identifying vertices in adjacent components and adding a subgraph to represent the first and last operations of the algorithm gives a family of cubic graphs for which the algorithm returns a MM of size at most $9n/20 + O(1)$.



Op	$\Delta {Y_3}^-$	$\Delta {Y_2}^-$	$\Delta {Y_1}^-$	$\Delta {Y_3}^+$	$\Delta {Y_2}^+$	$\Delta {Y_1}^+$	$m(Op)$
Op_1	0	-4	0	0	0	0	2
Op_2	-5	-1	0	0	3	1	1
Op_3	-5	0	-1	0	3	0	1

Fig. 8. Operations in the LP solution for Edge_Greedy

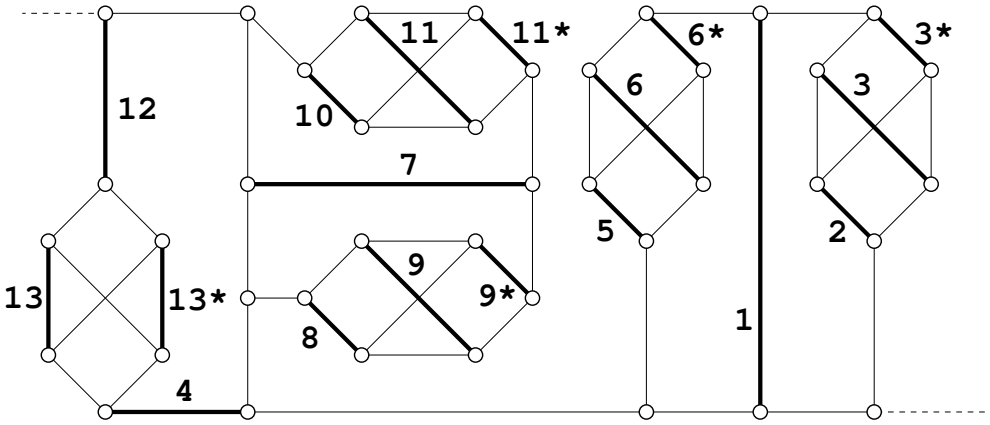


Fig. 9. Repeating component

Having considered an upper bound on the size of a MMM of a cubic graph, we now consider the maximum, over all n-vertex cubic graphs, of the the size of a MMM. The graph of Figure 10 represents a family of cubic graphs. As each component of eight vertices must contribute at least three edges to any MM, this shows that there exists infinitely many n-vertex cubic graphs with no MM of size less than $3n/8$.

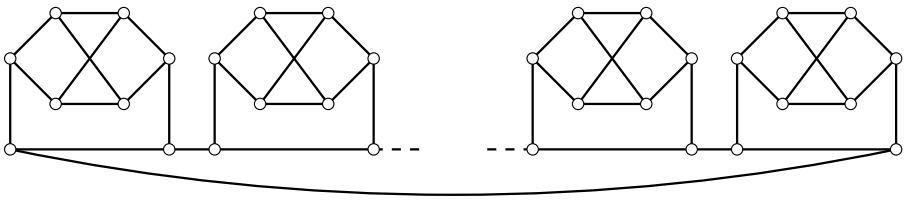


Fig. 10. No MM of size less than $3n/8$

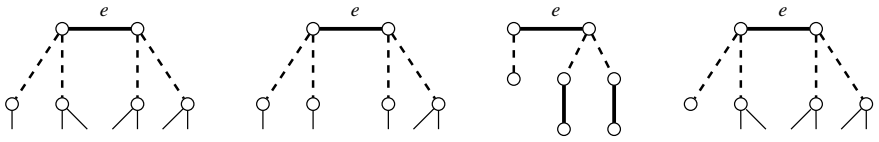
For graphs with girth 4, the introduction of more priorities of selection into the algorithm gives no better result than the unrestricted case. We now restrict the input to graphs of girth at least 5. Algorithm Edge_Greedy5 takes as input an n-vertex cubic graph of girth at least 5, G, and returns a MM, $\mathcal{E}$, of G.

Theorem 2 *Given a connected, n-vertex, cubic graph of girth at least 5, algorithm Edge_Greedy5 returns a maximal matching of size at most* $3n/7 + O(1)$.

Proof: This is the same as the proof of Theorem 1 except that there are less operations to consider as we may remove those involving any cycles of length less than 5. The solution is shown in Figure 11. The operations Op_i (for $i \in \{1, 2, 3, 4, 5\}$) are shown in Figure 12. The corresponding equations may be derived from the table in the figure. □

Op_1	Op_2	Op_3	Op_4	Solution
$\frac{1}{14}$	$\frac{1}{14}$	$\frac{1}{14}$	$\frac{1}{14}$	$\frac{3}{7}$

Fig. 11. Edge_Greedy5 solution



	Op1		Op2		Op3		Op4
Op	$\Delta {Y_3}^-$	$\Delta {Y_2}^-$	$\Delta {Y_1}^-$	$\Delta {Y_3}^+$	$\Delta {Y_2}^+$	$\Delta {Y_1}^+$	$m(Op)$
Op_1	-5	-1	0	0	3	1	1
Op_2	-3	-3	0	0	1	3	1
Op_3	-1	-3	-3	0	0	0	3
Op_4	-5	0	-1	0	3	0	1

Fig. 12. Equations for the operations in the LP solution for Edge_Greedy5

4 Small Connected Dominating Sets

The problem of finding a MCDS is a well known NP-hard optimisation problem [11] and is *polynomially equivalent* to the Maximum Leaf Spanning Tree problem ($\mathcal{MLST}$). A spanning tree of a graph $G = (V, E)$ is a connected spanning subgraph $T = (V, E')$ which does not contain a cycle. Vertices of degree 1 in T are called leaves and we are interested in finding a spanning tree with a large number of leaves. The non-leaf vertices of T form a CDS.

Solis-Oba [25] showed that $\mathcal{MLST}$ is approximable with approximation ratio 2, improving the previous best known approximation ratio of 3 by Lu and Ravi [18]. Galbiati, Maffioli and Morzenti [10] showed that when restricted to cubic graphs, this problem is APX-Complete (*i.e.* there exists a constant c such that it is NP-hard to approximate $\mathcal{MLST}$ within c). It is simple to verify that, for cubic graphs, $\mathcal{MCDS}$ is approximable with asymptotic approximation ratio 2. Griggs, Kleitman and Shastri [12] showed that every n-vertex connected cubic graph has a spanning tree with at least $\lceil (n/4)+2 \rceil$ leaves. Duckworth [5] showed that the size of the smallest CDS of a *random* n-vertex cubic graph is a.a.s. less than $0.5854n$.

In this section we present a greedy version of the algorithm introduced by Guha and Khuller [13] that "*grows a tree*". The idea behind the algorithm of [13] is given in Figure 13.

Start out with all vertices marked "white"
Select an initial vertex v to add to T (colour it "black")
Add all edges incident with v to T
Colour all neighbours of v "grey"
While there are still "white" vertices
    Add a "grey" vertex v to T (colour it "black")
    Add all edges incident with v and a "white" vertex to T
    Colour all "white" neighbours of v "grey"

Fig. 13. Guha and Khuller's Algorithm

At the end of this algorithm, the set of "black" vertices forms a CDS. We analyse this algorithm using the linear programming technique and show that n-vertex connected cubic graphs have a MCDS of size at most $3n/4 + O(1)$ which gives a new derivation, to within a constant number of vertices, of the main result of [12]. When the input is restricted to n-vertex connected cubic graphs of girth at least 5, a modified algorithm returns a CDS of size at most $2n/3 + O(1)$. We also show that there exist infinitely many n-vertex cubic graphs of girth 5 that have no CDS of size less than $4n/7$.

4.1 Algorithm Build_Tree

In our greedy version of the algorithm of [13], Build_Tree, after each operation we delete any edges that connect two "grey" vertices. Note that all "white" vertices have degree 3 and all "grey" vertices have degree 1 or 2. Each "grey" vertex then has one or two "white" neighbours and we assign a priority to selecting "grey" vertices of degree 2 over selecting "grey" vertices of degree 1. The input graph is assumed to be connected and so after the initial operation and before the completion of the algorithm there always exists a "grey" vertex with at least one "white" neighbour.

We distinguish vertices by means of their colour and their number of "white" neighbours so that the cardinalities of the sets of vertices in Figure 14 may characterise the graph at any stage of the algorithm.

Set	Colour	N^o white neighbours	Set	Colour	N^o white neighbours
V_0	grey	1	V_3	white	1
V_1	grey	2	V_4	white	2
V_2	white	0	V_5	white	3

Fig. 14. CDS categories

An example operation for Build_Tree is given in Figure 15. Vertex 1 is selected to be added to the CDS and deleted along with its incident edges. The neighbours of vertex 1 are coloured "grey" and the edges (2,4) and (3,4) are deleted since all their end-points are now "grey". The set of equations associated with the example operation of Figure 15 is $\Delta Y_0 = 2$, $\Delta Y_1 = -2$, $\Delta Y_2 = 0$, $\Delta Y_3 = -1$, $\Delta Y_4 = 0$, $\Delta Y_5 = -1$ and $m(Op) = 1$.

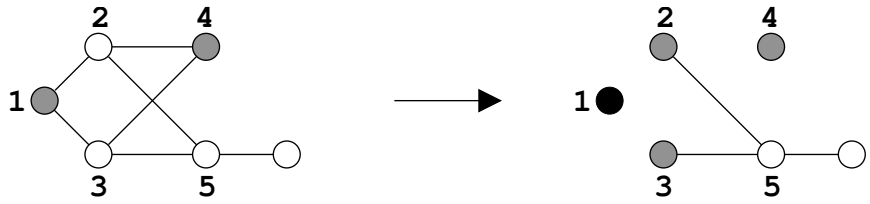


Fig. 15. An example operation for Build_Tree

Recall that for a set of vertices S, $(S, *)$ denotes all edges incident with the vertices of S. Algorithm Build_Tree in Figure 16 takes an n-vertex cubic graph G as input and returns a connected dominating set $\mathcal{C}$ of G.

Select v from $V(G)$
$\mathcal{C} \leftarrow \{v\}$;
colour $N(v)$ "grey";
delete $(v, *)$;
while $(\{V_2 \cup V_3 \cup V_4 \cup V_5\} \neq \emptyset)$
do
    delete all edges incident with two "grey" vertices;
    if $(V_1 \neq \emptyset)$ select v from V_1;
    else select v from V_0;
    $\mathcal{C} \leftarrow \mathcal{C} \cup \{v\}$;
    colour $N(v)$ "grey";
    delete $(v, *)$;
od

Fig. 16. Algorithm Build_Tree

For each operation we select a "grey" vertex to add to $\mathcal{C}$ and vertices are repeatedly selected until no "white" vertices remain. After each operation we delete all edges that are incident with two "grey" vertices.

4.2 Build_Tree Analysis

We now analyse the worst-case performance of Build_Tree and in this way prove the following theorem.

Theorem 3 *Given a connected, n-vertex, cubic graph, algorithm Build_Tree returns a connected dominating set of size at most* $3n/4 + O(1)$.

Proof: As we prioritise the selection of a vertex from V_1 over the selection of a vertex from V_0, we have $\gamma = 1$. The rest of the proof is as for Theorem 1, again using both priority constraints for each k such that $V_1^-(Op) = k$. The solution to LP_2 and the non-zero variables in the solution are shown in Figure 17. The operations Op_i (for $i \in \{1, 2, 3\}$) are shown in Figure 18. For each operation, the grey vertex v is selected by the algorithm to be added to the CDS. Deleted edges are indicated by dotted lines. The equations associated with these operations may be derived from Figure 19. □

Op_1	Op_2	Op_3	Solution
$\frac{1}{4}$	$\frac{1}{4}$	$\frac{1}{4}$	$\frac{3}{4}$

Fig. 17. Build_Tree LP solution

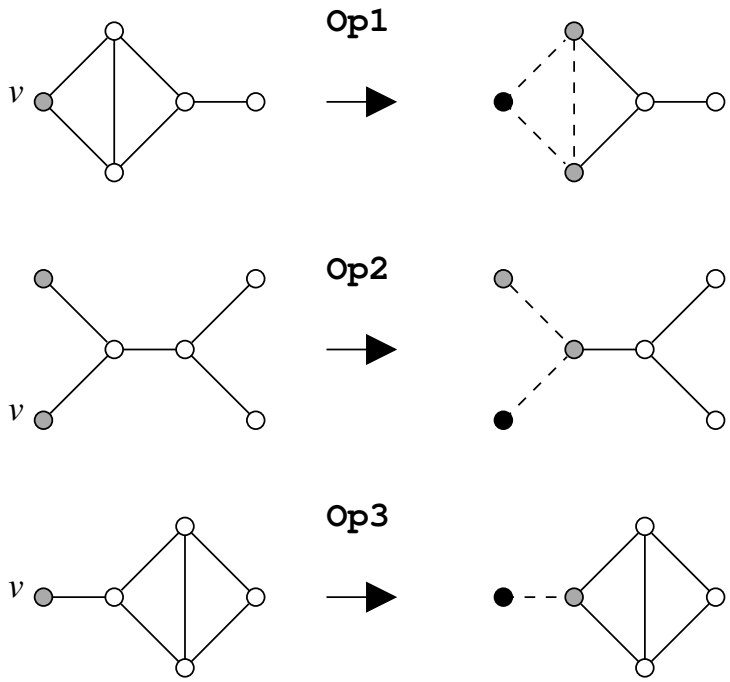


Fig. 18. Operations in the LP solution for Build_Tree

Op	$\Delta {Y_5}^-$	$\Delta {Y_4}^-$	$\Delta {Y_3}^-$	$\Delta {Y_2}^-$	$\Delta {Y_1}^-$	$\Delta {Y_0}^-$
Op_1	-1	-2	0	0	-1	0
Op_2	-1	0	-1	0	0	-2
Op_3	-2	-1	0	0	0	-1

Op	$\Delta {Y_4}^+$	$\Delta {Y_3}^+$	$\Delta {Y_2}^+$	$\Delta {Y_1}^+$	$\Delta {Y_0}^+$	$m(Op)$
Op_1	0	1	0	0	2	1
Op_2	1	0	0	0	1	1
Op_3	2	0	0	1	0	1

Fig. 19. Operations in the LP solution for Build_Tree

The subgraph that forms part of a cubic graph that realises the solution of the linear program is shown in Figure 20.

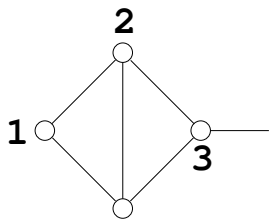


Fig. 20. Repeating component

For each component, the algorithm adds three of the four vertices to the CDS in the numbered order. Connecting a number of these subgraphs by identifying vertices in consecutive subgraphs and adding a subgraph to represent the initial operation of the algorithm gives a family of cubic graphs for which the algorithm returns a CDS of size at most $3n/4 + O(1)$.

The tightness of this bound was demonstrated in [12] using the example given in Figure 21. The graph consists of multiple copies of "K_4 minus an edge". Adjacent copies are connected together in a chain by an edge and the final copy in the chain is connected back to the first as indicated.

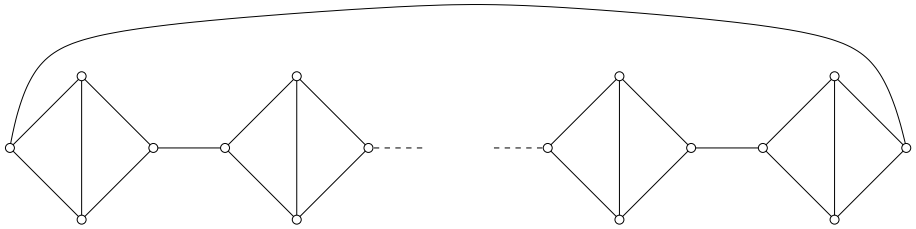


Fig. 21. No CDS of size less than $3n/4$

Since each component of four vertices must contribute at least three vertices to any CDS, this shows that there exist infinitely many n-vertex cubic graphs with no CDS of size less than $3n/4$.

4.3 Cubic Graphs with Girth at least 5

When we restrict the input to cubic graphs of girth at least 5, we add more priorities to the selection of "grey" vertices of degree 1 and, in some instances, add more than one vertex to the CDS per operation. The modified algorithm, Build_Tree5, given in Figure 22.

Select v from $V(G)$;
$\mathcal{C} \leftarrow \{v\}$;
colour $N(v)$ "grey"; delete $(v, *)$;
while $(\{V_2 \cup V_3 \cup V_4 \cup V_5\} \neq \emptyset)$
do
delete all edges incident with two "grey" vertices;
if $(V_1 \neq \emptyset)$ select v from V_1;
else $S \leftarrow \{v \mid \{\{v \in V_0\} \wedge \{N(v) \cap V_4\} \neq \emptyset\}\}$;
if $(S = \emptyset) S \leftarrow \{v \mid \{\{v \in V_0\} \wedge \{N(v) \cap V_3\} \neq \emptyset\}\}$;
if $(S = \emptyset) S \leftarrow \{v \mid \{\{v \in V_0\} \wedge \{N(v) \cap V_2\} \neq \emptyset\}\}$;
select u from S;
if $(N(u) \in V_2)$ $v \leftarrow u$;
else if $(N(u) \in V_4)$ $v \leftarrow N(u)$;
$\mathcal{C} \leftarrow \mathcal{C} \cup \{u\}$;
delete $(u, *)$;
else $w \leftarrow N(u)$;
$\mathcal{C} \leftarrow \mathcal{C} \cup \{u\}$;
delete $(u, *)$;
$v \leftarrow \{N(w) \cap \{V_2 \cup V_3 \cup V_4 \cup V_5\}\}$;
$\mathcal{C} \leftarrow \mathcal{C} \cup \{w\}$
delete $(w, *)$;
$\mathcal{C} \leftarrow \mathcal{C} \cup \{v\}$; colour $N(v)$ "grey"; delete $(v, *)$;
od

Fig. 22. Algorithm Build_Tree5

The algorithm takes as input an n-vertex cubic graph G of girth at least 5 and returns a CDS $\mathcal{C}$ of G. As before, the selection of a "grey" vertex of degree 2 has priority over the selection of a "grey" vertex of degree 1.

The priority list to describe the priorities of selection of a vertex from V_0 is as follows:

S_1: vertices in V_0 that have a neighbour u in V_4

S_2: vertices in V_0 that have a neighbour u in V_3

S_3: vertices in V_0 that have a neighbour u in V_2.

In the instance where S_3 is the highest priority non-empty set, we add the vertex v to the CDS, colour all neighbours of v "grey" and delete all edges incident with v. In the instance where S_1 is the highest priority non-empty set, we add u and v to the CDS, colour the neighbours of u "grey" and delete all edges incident with u and v. In the instance where S_2 is the highest priority non-empty set we add u, v and the "white" neighbour w of u to the CDS, colour all neighbours of w "grey" and delete all edges incident with u, v and w.

Theorem 4 *Given a connected, n-vertex, cubic graph of girth at least 5, algorithm Build_Tree5 returns a connected dominating set of size at most* $2n/3 + O(1)$.

Proof: This is as for the proof of Theorem 3, but excluding operations based on the condition that the input graph has girth at least 5, and taking note of operations that cannot occur due to the prioritisation. The solution to LP_2 and the non-zero variables in the solution are shown in Figure 23. The equations associated with the operations Op_i (for $i \in \{1, 2, 3\}$) may be derived from Figure 24 and the operations are shown in Figure 25. For each operation, black vertices are added to the CDS and deleted edges are indicated by dotted lines.

□

Op_1	Op_2	Op_3	Solution
$\frac{1}{24}$	$\frac{1}{8}$	$\frac{1}{6}$	$\frac{2}{3}$

Fig. 23. Build_Tree5 LP solution

Now consider the maximum, over all n-vertex cubic graphs of girth 5, of the the size of a MCDS. The graph of Figure 26 represents a family of cubic graphs which contain a chain of k repeating components.

Each component has fourteen vertices indicating that the entire graph has $n = 14k$ vertices. Adjacent components are chained together by an edge and the last component in the chain is connected back to the first as indicated. This graph has girth 5. As each component must contribute at least eight vertices to any CDS, this shows the existence of a family of n-vertex cubic graphs of girth 5 with no CDS of size less than $4n/7$.

Op	ΔY_5^-	ΔY_4^-	ΔY_3^-	ΔY_2^-	ΔY_1^-	ΔY_0^-	
Op_1	-3	-4	-1	0	0	-2	
Op_2	-7	0	-1	0	0	-2	
Op_3	0	-2	-2	0	-2	-1	
Op		ΔY_4^+	ΔY_3^+	ΔY_2^+	ΔY_1^+	ΔY_0^+	$m(Op)$
Op_1		0	4	0	2	0	3
Op_2		4	0	0	2	0	3
Op_3		0	2	0	0	3	1

Fig. 24. Operations in the LP solution for Build_Tree5

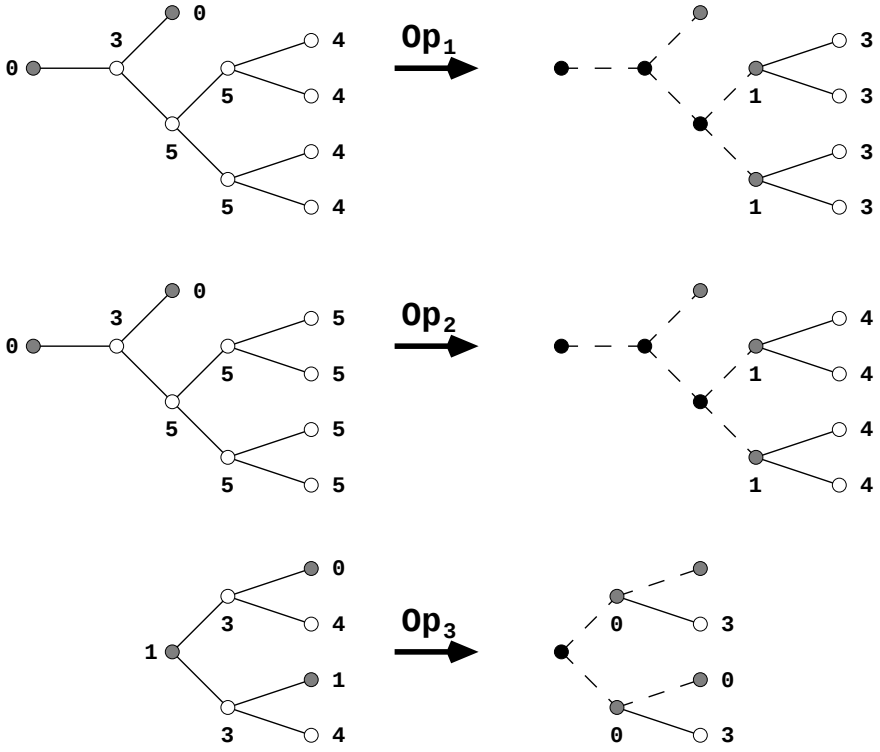


Fig. 25. Operations in the LP solution for Build_Tree5

5 Small Independent Dominating Sets

The problem of finding a MIDS is one of the core NP-hard optimisation problems in graph theory [11]. Halldórsson [14] showed that for general graphs, this problem is not approximable within $n^{1-\epsilon}$ for any $\epsilon > 0$. Kann [16] showed that when restricted to graphs of maximum degree at least 3, the problem is APX-Complete. Lam, Shiu and Sun [17] showed that for $n \geq 10$, the size of an IDS of n-vertex connected cubic graphs is at most $2n/5$. They also give an example of a cubic graph on ten vertices with no IDS of size less than four.

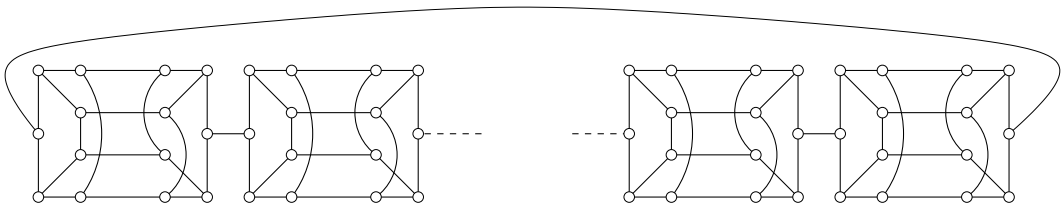


Fig. 26. No CDS of size less than $4n/7$

The algorithm we present is simple to implement and, for n-vertex connected cubic graphs, ensures that the size of the IDS returned is at most $29n/70 + O(1)$. We also show that there exist infinitely many n-vertex cubic graphs which have no IDS of size less than $3n/8$. This obviously does not rule out the possibility that there exists an alternative algorithm that always returns an IDS of size at most $3n/8$. Restricting the input to n-vertex connected cubic graphs of girth at least 5, a modified algorithm returns an IDS of size at most $3n/8 + O(1)$.

Reed [21] showed that the size of a minimum *dominating set* of n-vertex connected cubic graphs is at most $3n/8$. He also gave an example of a cubic graph on eight vertices with no dominating set of size less than three, demonstrating the tightness of this bound. Molloy and Reed [20] showed that the the size of the smallest dominating set $D(G)$ of a *random* cubic graph G on n vertices, a.a.s. satisfies $0.2636n \leq |D(G)| \leq 0.3126n$. Duckworth and Wormald [7] tightened these bounds by showing that the size of a MIDS, $\mathcal{D}$, of a random cubic graph, a.a.s. satisfies $0.2641n \leq |\mathcal{D}| \leq 0.2794n$.

5.1 Algorithm Min_Ratio

We describe a greedy algorithm based on selecting vertices of minimum degree that finds a small independent dominating set $\mathcal{I}$ of an n-vertex cubic graph G. In order to guarantee that $\mathcal{I}$ is indeed independent and dominating, once a vertex is chosen to be added to $\mathcal{I}$, it is deleted along with all its neighbours and the edges incident with each of those neighbours. The only other vertices that are added to $\mathcal{I}$ are those which are isolated by such an operation of the algorithm. At any stage of our algorithm we characterise the vertices of the input graph based on their current degree so that V_i for $1 \leq i \leq 3$ denotes the set of vertices of degree i.

For any particular operation performed by the algorithm, a number of vertices are deleted and a subset of these are added to $\mathcal{I}$. We define $\rho(v)$ to be the ratio of the increase in size of $\mathcal{I}$ to the number of vertices deleted when an operation is performed by selecting vertex v to be added to $\mathcal{I}$. The algorithm, Min_Ratio, is given in Figure 27.

$v \leftarrow$ **MinR**(V);
$\mathcal{I} \leftarrow \{v\}$;
delete $(N(v), *)$;
Add_isolates();
while $(\{V_1 \cup V_2\} \neq \emptyset)$
do
    $S \leftarrow \{v \mid \{\{N(v) \cap V_1\} \neq \emptyset\}\}$;
    if $(S = \emptyset)$ $S \leftarrow \{\{\{v\} \cup N(v)\} \mid \{v \in V_2 \wedge Q(v) = 1\}\}$;
    if $(S = \emptyset)$ $S \leftarrow \{\{\{v\} \cup N(v)\} \mid \{v \in V_2 \wedge Q(v) = 2\}\}$;
    if $(S = \emptyset)$ $S \leftarrow \{\{\{v\} \cup N(v)\} \mid \{v \in V_2 \wedge Q(v) = 3\}\}$;
    if $(S = \emptyset)$ $S \leftarrow \{\{\{v\} \cup N(v)\} \mid \{v \in V_2 \wedge Q(v) = 4\}\}$;
    $u \leftarrow$ **MinR**(S);
    $\mathcal{I} \leftarrow \mathcal{I} \cup \{u\}$;
    delete $(N(u), *)$;
    Add_isolates ();
od

Fig. 27. Algorithm Min_Ratio

In the algorithm, $Q(v)$ denotes the number of vertices at minimum distance 2 from v. The function **MinR**(T) operates on the given set of vertices T and returns an element $u \in T$ for which $\rho(u)$ is the minimum of all vertices in T. Should there exist two vertices $\{v, v'\} \in T$ such that $\rho(v) = \rho(v')$ (and $\rho(v)$ is the minimum of all vertices in T) a vertex is selected arbitrarily from $\{v, v'\}$. The function **Add_isolates**() adds any isolated vertices created in $V(G)$ to $\mathcal{I}$.

After the initial operation of the algorithm, vertices are repeatedly selected to be added to $\mathcal{I}$ based upon the minimum degree of the vertices available and the number of vertices at minimum distance 2 from these vertices, until no vertices remain.

The priority list for this algorithm is the one given in Section 2.

5.2 Min_Ratio Analysis

Theorem 5 *Given a connected, n-vertex, cubic graph, algorithm Min_Ratio returns an independent dominating set of size at most* $29n/70 + O(1)$.

Proof: As we prioritise the selection of a vertex of degree 1 over the selection of a vertex of degree 2, we have $\gamma = 1$. The rest of the proof of Theorem 1 is then followed. The solution to the LP is shown in Figure 28. The operations Op_i (for $i \in \{1, 2, 3\}$) are as shown in Figure 29 and their associated equations may be derived from Figure 30.

Op_1	Op_2	Op_3	Solution
$\frac{2}{35}$	$\frac{1}{14}$	$\frac{1}{10}$	$\frac{29}{70}$

Fig. 28. Min_Ratio solution

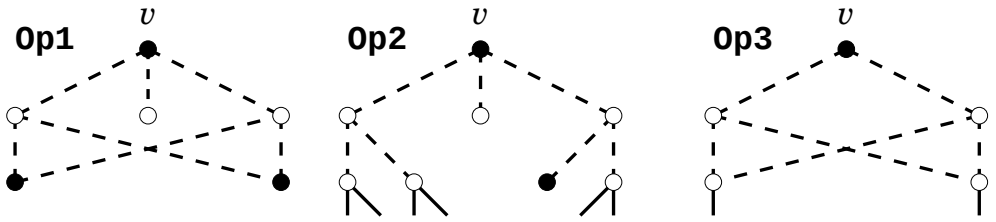


Fig. 29. Operations performed in the worst case

Op	$\Delta {Y_3}^-$	$\Delta {Y_2}^-$	$\Delta {Y_1}^-$	$\Delta {Y_3}^+$	$\Delta {Y_2}^+$	$\Delta {Y_1}^+$	$m(Op)$
Op_1	-3	-2	-1	0	0	0	3
Op_2	-6	0	-2	0	3	0	2
Op_3	-4	-1	0	0	0	2	1

Fig. 30. Operations performed in the worst case

For each operation, the black vertex v is selected by the algorithm to be added to the IDS. In each case, vertices may be isolated and these are also coloured black to indicate their addition to the IDS. Deleted edges are indicated by dotted lines. □

The subgraph that forms part of a cubic graph that realises the solution of the linear program is shown in Figure 31.

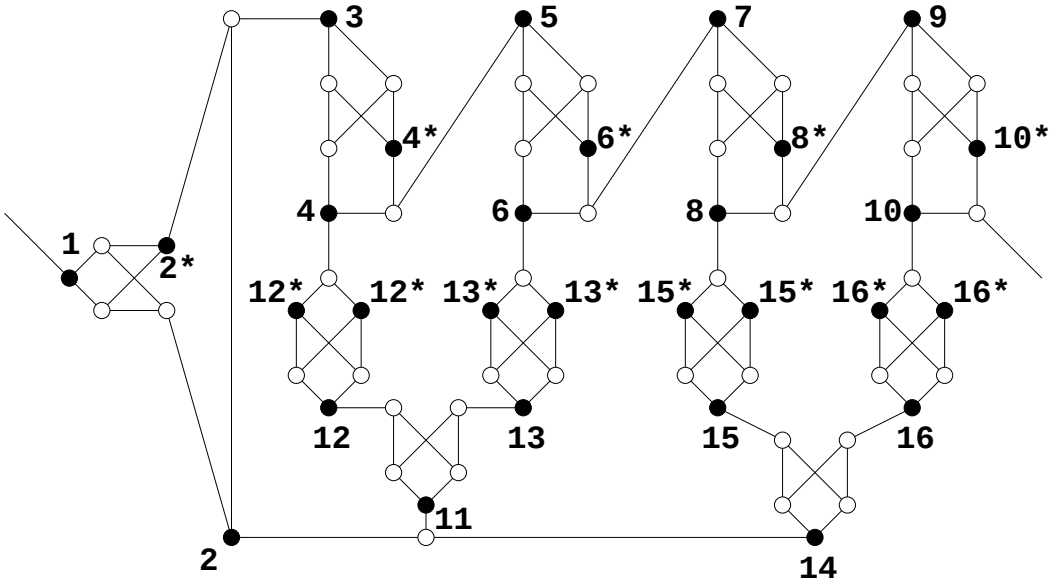


Fig. 31. Repeating component

For each component, the algorithm selects 29 of the 70 vertices to be added to $\mathcal{I}$ in the numbered order. A vertex labeled i^* indicates that this vertex was isolated after the vertex labeled i was added to $\mathcal{I}$ and was subsequently added to $\mathcal{I}$. Connecting a number of these subgraphs by identifying vertices in consecutive components and adding a subgraph to represent the first and last operations of the algorithm gives a family of cubic graphs for which the algorithm returns an IDS of size at most $29n/70 + O(1)$.

Now consider sharpness of the result. The graph of Figure 32 represents a family of cubic graphs which contain a chain of k repeating components. Each component has eight vertices indicating that the entire graph has $n = 8k$ vertices.

A component is connected to the next component in the chain by one edge and the final component in the chain is connected back to the first as indicated. As each component must contribute at least three vertices to any IDS, this shows the existence of a family of n-vertex cubic graphs with no IDS of size less than $3n/8$.

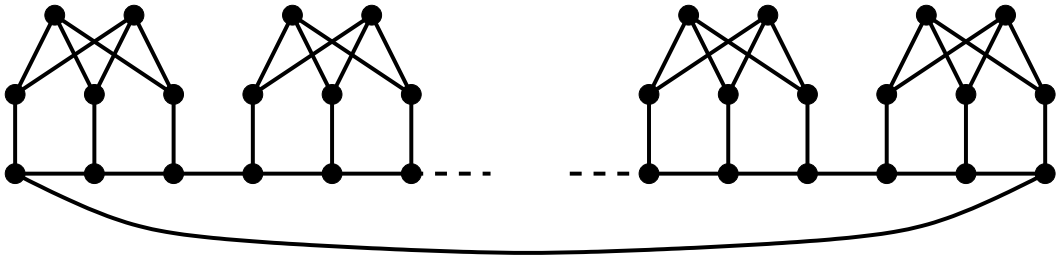


Fig. 32. No independent dominating set of size less than $3n/8$

5.3 Cubic graphs with Girth at least 5

Restricting the input to n-vertex cubic graphs of girth at least 5, we apply a modified algorithm that is based on selecting vertices of minimum degree, using operations that remove the fewest edges and combining this with selecting vertices that give a minimal ratio.

The algorithm, which we call Min_Deg_One, is essentially the same as the algorithm Min_Ratio except that for each of the sets in the priority list, should there exist two vertices v and v' for which $\rho(v) = \rho(v')$ (and $\rho(v)$ is the minimum of all vertices in the set), we choose the operation that deletes the fewest edges.

Theorem 6 *Given a connected, n-vertex, cubic graph of girth at least 5, algorithm Min_Deg_One returns an independent dominating set of size at most $3n/8 + O(1)$.*

Proof: The linear program LP_2 is formed in the same way as that for Min_Ratio. Due to the restriction in girth for the input, this linear program has approximately one third the number of variables. The solution is of the form shown in Figure 33. The operations Op_i (for $i \in \{1, 2, 3, 4, 5, 6\}$) represent the operations shown in Figure 34 and their associated equations may be derived from Figure 35. □

Op_1	Op_2	Op_3	Op_4	Op_5	Op_6	Solution
$\frac{1}{70}$	$\frac{1}{70}$	$\frac{1}{28}$	$\frac{3}{56}$	$\frac{1}{42}$	$\frac{1}{14}$	$\frac{3}{8}$

Fig. 33. Min_Deg_One solution

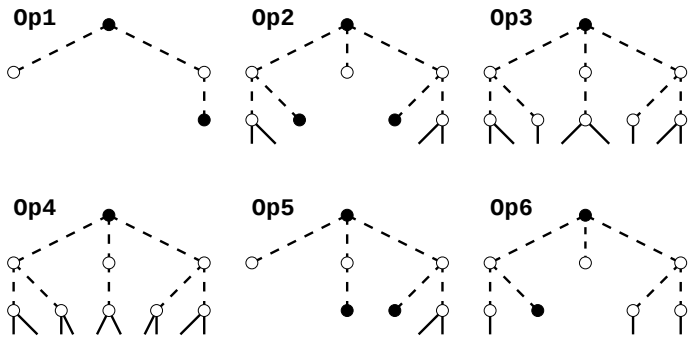


Fig. 34. Operations performed in the worst case

Op	$\Delta {Y_3}^-$	$\Delta {Y_2}^-$	$\Delta {Y_1}^-$	$\Delta {Y_3}^+$	$\Delta {Y_2}^+$	$\Delta {Y_1}^+$	$m(Op)$
Op_1	0	-2	-2	0	0	0	2
Op_2	-5	0	-3	0	2	0	3
Op_3	-6	-3	0	0	3	2	1
Op_4	-8	-1	0	0	5	0	1
Op_5	-3	-1	-3	0	1	0	3
Op_6	-3	-3	-2	0	0	3	2

Fig. 35. Operations performed in the worst case

Regarding sharpness, the graph of Figure 36 represents a family of cubic graphs which contain a chain of k repeating components.

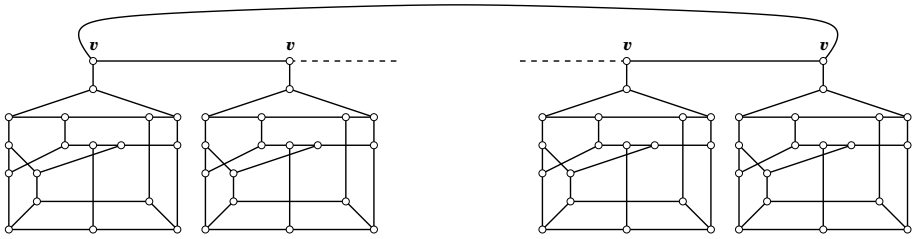


Fig. 36. No IDS of size less than $n/3$

Each component has eighteen vertices indicating that the entire graph has $n = 18k$ vertices. Components are connected by a cycle of k edges passing through the vertex labeled v in each component as indicated. This graph has girth 5. As each component must contribute at least six vertices to any IDS, this shows the existence of a family of n-vertex cubic graphs of girth 5 with no IDS of size less than $n/3$.

6 Remarks

In joint work with Zito [8], we used our linear programming technique to find a large induced matching of a $(2, 3)$-regular bipartite graph as a means of approximating the sparsest 2-spanner problem on 4-connected planar triangulations.

The technique has also been used to find a large induced matching of a cubic graph but a simpler analysis may be used to achieve the same result. The technique was used in [6] to find families of cubic graphs for which an algorithm has its worst case performance.

We have also used this technique to find a large 2-independent set and a small vertex cover of a cubic graph. In both cases, while a result is achieved, the same result is relatively easily achieved by means of a simpler analysis.

Acknowledgement The authors would like to thank an anonymous referee whose thoughtful comments led us to find an error in an earlier version of this paper.

References

[1] G. Ausiello, P. Crescenzi, G. Gambosi, V. Kann, A. Marchetti-Spaccamela and M. Protasi, Complexity and Approximation: Combinatorial Optimization Problems and their Approximability Properties, Springer-Verlag, 1999.

[2] J.E. Calvert and W.L. Voxman, Linear Programming, Harcourt Brace Jovanovich, Florida, U.S.A., 1989.

[3] T. Denley, The independence number of graphs with large odd girth, The Electronic Journal of Combinatorics, 1 (1994) #R9 12 pages.

[4] W. Duckworth, Small edge dominating sets of regular graphs, Electronic Notes in Theoretical Computer Science, 91(C) (2004) 43–55.

[5] W. Duckworth, Minimum connected dominating sets of random cubic graphs, The Electronic Journal of Combinatorics, 9(1) (2002) #$R7$ 13 pages.

[6] W. Duckworth, D. Manlove and M. Zito, On the approximability of the maximum induced matching problem, The Journal of Discrete Algorithms, 3(1) (2005) 79–91.

[7] W. Duckworth and N.C. Wormald, Minimum independent dominating sets of random cubic graphs, Random Structures and Algorithms, 21(2) (2002) 147–161.

[8] W. Duckworth, N.C. Wormald and M. Zito, A PTAS for the sparsest 2-spanner problem in 4-connected planar triangulations, The Journal of Discrete Algorithms, 1(1) (2003) 67–76.

[9] T. Gál, Postoptimal Analyses, Parametric Programming and Related Topics, McGraw-Hill International Book Company, New York, 1979.

[10] G. Galbiati, F. Maffioli and A. Morzenti, A short note on the approximability of the maximum leaves spanning tree problem, Information Processing Letters, 52(1) (1994) 45–49.

[11] M.R. Garey and D.S. Johnson, Computers and Intractability: A Guide to the Theory of NP-Completeness, Freeman and Company, 1979.

[12] J.R. Griggs, D.J. Kleitman and A. Shastri, Spanning trees with many leaves in cubic graphs, The Journal of Graph Theory, 13(6) (1989) 669–695.

[13] S. Guha and S. Khuller, Approximation algorithms for connected dominating sets, Algorithmica, 20 (1988) 374–387.

[14] M.M. Halldórsson, Approximating the minimum maximal independence number, Information Processing Letters, 46(4) (1993) 169–172.

[15] J.D. Horton and K. Kilakos, Minimum edge dominating sets, SIAM Journal on Discrete Mathematics, 6(3) (1993) 375–387.

[16] V. Kann, On the Approximability of NP-Complete Optimisation Problems, Doctoral Thesis, Department of Numerical Analysis, Royal Institute of Technology, Stockholm, 1992.

[17] P.C.B. Lam, W.C. Shiu and L. Sun, On the independent domination number of regular graphs, Discrete Mathematics, 202 (1999) 135–144.

[18] H-I. Lu and R. Ravi, Approximating maximum leaf spanning trees in almost linear time, Journal of Algorithms, 29(1) (1998) 132–141.

[19] A. Meir and J.W. Moon, Relations between packing and covering numbers of a tree, Pacific Journal of Mathematics, 61(1) (1975) 225–233.

[20] M. Molloy and B. Reed, The dominating number of a random cubic graph, Random Structures and Algorithms, 7(3) (1995) 209–221.

[21] B. Reed, Paths, stars and the number three, Combinatorics, Probability and Computing, 5 (1996) 277–295.

[22] A. Schrijver, Theory of Linear and Integer Programming, Wiley, 1986.

[23] J.B. Shearer, A note on the independence number of triangle-free graphs, Discrete Mathematics, 46 (1983), 83–87.

[24] J.B. Shearer, A note on the independence number of triangle-free graphs II, Journal of Combinatorial Theory Series B, 53 (1991), 300-307

[25] R. Solis-Oba, 2-approximation algorithm for finding a spanning tree with maximum number of leaves, Lecture Notes in Computer Science, 1461 (1998) 441–452.

[26] M. Yannakakis and F. Gavril, Edge dominating sets in graphs, SIAM Journal on Discrete Applied Mathematics, 38(3) (1980) 364–372.

[27] M. Zito, Randomised Techniques in Combinatorial Algorithmics, Doctoral Thesis, Department of Computer Science, The University of Warwick, UK, 1999.

[28] M.Zito, Small maximal matchings in random graphs, Lecture Notes in Computer Science, 1776 (2000) 18–27.