Linear programming and the worst-case analysis of greedy algorithms on cubic graphs*

W. Duckworth †
Mathematical Sciences Institute
The Australian National University
Canberra, ACT 0200, Australia
billy.duckworth@gmail.com

N. Wormald ‡
Department of Combinatorics & Optimization
University of Waterloo
Waterloo ON, Canada N2L 3G1
nwormald@uwaterloo.ca

Submitted: Oct 20, 2009; Accepted: Jun 5, 2010; Published: Dec 10, 2010
Mathematics Subject Classification: 05C85

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. We show that for n-vertex connected cubic graphs, the size of an MMM is at most $9n/20 + O(1)$, which is a new result. We also show that the size of an MCDS is at most $3n/4 + O(1)$ and the size of a MIDS is at most $29n/70 + O(1)$. These results are not new, but earlier proofs involved rather long ad-hoc arguments. By contrast, our method is to a large extent automatic and can apply to other problems as well. We also consider n-vertex connected cubic graphs of girth at least 5 and for such graphs we show that the size of an MMM is at most $3n/7 + O(1)$, the size of an MCDS is at most $2n/3 + O(1)$ and the size of a MIDS is at most $3n/8 + O(1)$.

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

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

†Research supported by Macquarie University while the author was supported by the Macquarie University Research Fellowships Grants Scheme.

‡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 which are of bounded degree or regular of fixed degree. In some cases this applies even to 3-regular graphs, for example, Maximum Independent Set [12, problem GT20] and Minimum Dominating Set [12, 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 [12, problem GT1], Maximum Induced Matching [6] and Maximum 2-Independent Set [24]. 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 given 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 a cubic graph $G = (V, E)$. A (vertex) dominating set of G is a set $D \subseteq V$ such that for every vertex $v \in V$, either $v \in D$ or v has a neighbour in D. An edge dominating set is a set $F \subseteq E$ such that for every edge $e \in E$, either $e \in F$ or e shares a common end-point with an edge of F. An independent set of G is a set $I \subseteq V$ such that no two vertices of I are joined by an edge of E. A matching of G is a set $M \subseteq E$ such that no two edges of M share a common end-vertex.

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

dominating set (IDS) of G is an independent set that is also dominating. A *maximal matching* (MM) is a matching $\mathcal{E}$ for which every edge in $E(G) \setminus \mathcal{E}$ shares at least one end-vertex with an edge of $\mathcal{E}$. Equivalently, it is an IDS of the line graph of G. A *connected dominating set* (CDS) of G is a (vertex) dominating set that induces a connected subgraph.

For each of these types of sets, we consider such a set of minimum cardinality in G, which we denote by prefixing the acronym with M. Thus an MMM is a minimum maximal matching, and so on. Let $\mathcal{MIDS}$, $\mathcal{MMM}$ and $\mathcal{MCDS}$ denote the problems of finding a MIDS, an MMM and an MCDS 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 [13] 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, Shiu and Sun [20] 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 amount of ad hoc arguments by using computer calculations.

Note that, for n-vertex cubic graphs, it is simple to verify that the size of an 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 an MMM is at most $9n/20 + O(1)$, the size of an 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.

We also consider n-vertex connected cubic graphs of girth at least 5. For such graphs, we show that the size of an MMM is at most $3n/7 + O(1)$, the size of an MCDS is at most $2n/3 + O(1)$ and the size of a MIDS is at most $3n/8 + O(1)$. It turned out 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. This line of investigation was suggested by, for example, Denley [3] and Shearer [28, 29], who consider the similar problem of maximum independent set size in graphs with restricted girth. Ever-increasing bounds were obtained as the girth increases; see also [21]. For not-necessarily-independent dominating sets there has been a recent flurry of activity including the upper bound of $0.3572n$ by Fisher, Fraughnaugh and Seager [9] for girth 5, upper bounds of $(1/3 + 3/g^2)n$ by Kostochka and Stodolsky [18] and $(44/135 + 82/132g)n$ by Löwenstein and Rautenbach [22] when the girth g of the graph is at least 5. These are above $0.4n$ for $g = 5$. The most recent result for large girth is about $3n/10 + O(n/g)$ by Král, Škoda and J. Volec [19]. Hoppen and Wormald have announced an unpublished upper bound of $0.2794n$ for a MIDS in a cubic graph of sufficiently large girth.

Our basic idea involves considering the set of operations involved in a greedy algorithm for constructing the desired set of vertices or edges. The operations are classified in such a way that an operation of a given type has a known effect on the number of vertices whose neighbourhood intersects the set in a given way. There are restrictions on the numbers

of times that the various operations can be performed, which leads to a linear program. Due to the unique nature of the first step of the algorithm, our formulation of the linear program requires a small adjustment to the constraints, which is analysed post-optimally. We introduce prioritisation to the constraints in such a way that the solution of the linear program can be improved; this and the proof of validity of the linear program, including the post-optimal analysis, is the heart of our method.

The following section describes the type of greedy algorithms we will be using, and sets up our analysis of their worst-case performance 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.

The proofs in this paper involve the creation of linear programs which are defined by a set of feasible operations in each case. The operations are determined by our proofs but are not listed in detail. In the Appendix[1] to this article, the operations actually used are all listed for each problem, along with the the associated linear program and its solution.

2 Worst-Case Analysis and Linear Programs

In this section we discuss the type of greedy algorithms we will consider, and our method of analysis. For this general description, let us call this algorithm ALG. One property of ALG we will require, to be made precise shortly, is that it can be broken down into repeated applications of a fixed set of operations. From these, we will derive an associated linear program (LP) giving a bound on the result obtained by the algorithm. Then we will describe how to improve the bound obtained by prioritising the operations.

In each of the problems that we consider in this paper, a graph is given and the task is to find a subset of the vertices (or edges) of small cardinality that satisfies given local conditions. ALG is a greedy algorithm based on selecting vertices (that have particular properties) from an ever-shrinking subgraph of the input graph. It takes a series of steps. In each step, a vertex called the *target* is chosen, then a vertex (or edge) near the target is selected to be added to a growing set $\mathcal{S}$, called the *chosen* set. Once this selection has been made, a set of edges and vertices are deleted from the remaining graph. Then the next step is performed, and so on until no vertices remain. The final output of each algorithm is $\mathcal{S}$. It is the appropriate choice of vertices and edges to delete in each step that will guarantee that the final set $\mathcal{S}$ satisfies the required property (domination, independence, *etc.*).

2.1 Operations

For our general method to be applicable to ALG, it must use a fixed set OPS of "operations" such that each step of ALG can be expressed as the application of one of the operations in OPS. Associated with each operation Op, there is a graph H. When Op

[1]Published on the same page as this article

is applied, an induced subgraph H' of the main graph isomorphic to H is selected, one or more elements (vertices or edges) of H' are added to the chosen set, and certain vertices and edges of H' are deleted. Associated with Op we give a diagram showing which elements are deleted and which are added to $\mathcal{S}$.

For instance, consider $\mathcal{MIDS}$ in which $\mathcal{S}$ is an IDS. One step of the algorithm may call for the target vertex v to be any vertex of degree 2 adjacent to precisely one vertex of degree 1. The target vertex chosen might be the vertex 2 in Figure 1. The step of the algorithm in this instance will be required to add the target vertex v to $\mathcal{S}$, delete v and its neighbouring vertices, and then to add to $\mathcal{S}$ any vertices that consequently become isolated, and also to delete the latter from the graph. With the neighbourhood of the target vertex as shown in this figure, vertex 5 is added to $\mathcal{S}$ and is also deleted. In figures such as this, vertices added to the chosen set $\mathcal{S}$ are shown as black, and the dotted lines indicate edges that are deleted. It is understood that all vertices that become isolated are automatically deleted. In this way, the operation Op is defined by the figure.

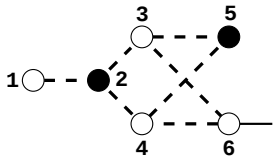

Figure 1: An example operation

Each step of the algorithm can thus be re-expressed as choosing both an operation and an induced subgraph of the graph to apply it to. (Strictly, in the above figure, the induced subgraph has six vertices and the vertex 6 must have degree exactly three, as shown by the incomplete edge leading to the right. All our figures should be read this way: any incomplete edge represents an edge joining to any other vertex of the graph or to another incomplete edge, thereby making a full edge. If there is more than one incomplete edge, the figure can therefore represent any of several possible induced subgraphs.) Naturally, this operation can only be applied if the target vertex lies in the appropriate induced subgraph.

2.2 A linear program

The idea behind our approach derives from the following observation. For many greedy algorithms, there are certain operations which will appear to be 'wasteful' in the sense that they add a relatively large number of vertices to the chosen set $\mathcal{S}$ (which is supposed to be kept small) and, at the same time, delete a relatively small number of vertices of the graph (though, presumably more than were added to the chosen set). However, such operations tend to create many vertices of some given degree, so there is a limit to how many times the algorithm can use such an operation. To take advantage of this, we classify the vertices of the graph according to their degree. In the case of $\mathcal{MCDS}$, we additionally classifying them according to their "colour," which will be defined in the

relevant section. Let $V_1, \ldots, V_t$ denote the classes obtained by any such classification, and let Y_i denote $|V_i|$ for $1 \leq i \leq t$.

For each operation $Op \in OPS$, and for each i, we require that the *net* change in Y_i must be the same in each application of the operation. Let $\Delta Y_i(Op)$ denote this constant. In addition, the increase in the size of the chosen set must be a constant, denoted by $m(Op)$. For instance, with the operation given in Figure 1, the number of vertices of degree 3 decreases by 4, one vertex of degree 1 is deleted but one is created, and so on. Thus $\Delta Y_3 = -4$, $\Delta Y_2 = -1$, $\Delta Y_1 = 0$ and $m = 2$.

We assume that all vertices initially belong to the same class, which we may select as V_t by definition. So, initially, $Y_t = n$ and $Y_i = 0$ for all $1 \leq i < t$. Another assumption we make, which is easy to verify for each instance of ALG we will use, is that at the end, all vertices have been deleted, so $Y_i = 0$ for $1 \leq i \leq t$. This implies that the *net* change in Y_t over the execution of ALG is $-n$, and for $1 \leq i < t$, the net change in Y_i is 0. For an operation $Op \in OPS$, we use $r(Op)$ to denote the number of times operation Op is performed during the algorithm's execution. Then the solution to the linear program LP_0 given in Figure 2 gives an upper bound on the size of the chosen set returned by the algorithm. Here, $\mathbf{C}_i$ denotes the constraint imposed by the net change in Y_i.

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

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

Figure 2: The linear program LP_0

2.3 Prioritisation, and two more linear programs

In the examples we have examined, the upper bound obtained via LP_0 is quite weak, and can be improved by prioritising the operations, which will result in an LP with more constraints. Before each operation, we may (implicitly or explicitly) define a list of subsets $S_1, S_2, \ldots$ of the vertex set, called a *priority list*, where S_1 has the priority index 1 (highest), S_2 has priority index 2 (second-highest), and so on. 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_3: vertices of degree 2 (and their neighbours) that have precisely two vertices at distance 2,

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

S_5: vertices of degree 2 (and their neighbours) that have precisely four vertices at distance 2.

The priority index of a vertex v is defined to be $\min\{i : v \in S_i\}$ (taking the minimum of the empty set as ∞). We then impose the condition that vertices can be chosen as the target only when no vertices of higher priority (i.e. smaller priority index) exist in the graph at the time.

To analyse the effect of prioritisation, we consider the effect of an operation Op on a set V_i as the simultaneous destruction of some vertices of V_i (by deleting them or changing their degrees) and creation of new vertices of V_i. Denote by $Y_i^+(Op)$ the number of vertices of V_i created, and by $Y_i^-(Op)$ the *negative* of the number of vertices of V_i destroyed. It follows that $Y_i^+(Op) + Y_i^-(Op) = \Delta Y_i(Op)$.

Prioritisation will lead to extra constraints, but first we examine the effect it has on eliminating operations. Since the input graph is assumed to be connected, the first step of ALG is unique in the sense that it is the only application of an operation where the minimum degree of the vertices is 3. Thus, an operation is feasible to be applied as the first step of ALG only if it belongs to the set OPS_0 of operations Op satisfying $Y_i^-(Op) = 0$ for all $1 \leq i < t$. The algorithms we consider will achieve good results by giving a higher priority to all operations that destroy a vertex of degree less than 3. This will ensure that no operation in OPS_0 may be applied after the first step. By changing our focus to what the algorithm does after the first step, we will be able to exclude the operations in OPS_0 (and hence obtain an LP with a better objective function). This will be formalised below.

Prioritisation will also prevent certain other operations from ever occurring. Continuing the $\mathcal{MIDS}$ example, consider the operations given in Figure 3 and assume that vertex v has been selected to be added to $\mathcal{S}$. The operation in Figure 3(a) 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 excluded: an operation adding the neighbour of the vertex of degree 1 to $\mathcal{S}$ will be used instead. 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).

In each of the algorithms and problems considered, we will define a set OPS_1 of operations such as these, that are excluded due to the prioritisation. We define $OPS_2 = OPS \setminus (OPS_0 \cup OPS_1)$, which contains all operations that can feasibly occur after the first step.

We are about to define two new LP's. For these, the variable r is redefined so as to refer only to operations after the first one. Thus, for each excluded operation Op, we may add the constraint $r(Op) = 0$ to the LP. In addition, further significant constraints result from prioritisation. We assume (as will be true for each algorithm we consider) that there

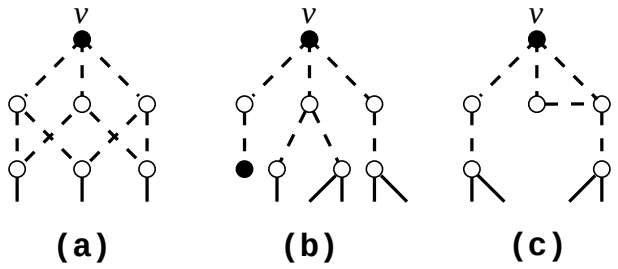

Figure 3: Excluded operations

will be a set V_γ, such that

(A) *all vertices in V_γ have degree less than 3,*

all operations Op with $Y_i^-(Op) < 0$ have priority over all other operations, and, moreover, when $Y_\gamma > 0$, at least one such operation with $Y_i^-(Op) < 0$ can be applied. It follows that

(B) *when $Y_\gamma > 0$, the next operation Op applied must have $Y_\gamma^-(Op) < 0$.*

(Conversely, of course if $Y_\gamma = 0$, the next operation Op must have $Y_\gamma^-(Op) = 0$ since there are no vertices in the class V_γ available in the graph.) Let K denote the range of nonzero (hence negative) values taken by $Y_\gamma^-(Op)$ over all $Op \in OPS_2$. 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).

We now bound from above the increase in S_k from an operation Op. From property (B), if $Y_\gamma^-(Op) = 0$, then Op cannot be performed due to the priority constraints unless $Y_\gamma = 0$. Thus, S_k increases by $\max(0, Y_\gamma^+(Op) - k + 1)$. If $Y_\gamma^-(Op) < 0$ and $\Delta Y_\gamma(Op) \geq 0$, then S_k increases by at most $\Delta Y_\gamma(Op)$, a bound which is valid for all operations. 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 either decrease S_k by $-\Delta Y_\gamma(Op)$ (if that is smaller than S_k) or send S_k to 0. In the latter case, note that by definition, Op requires at least k vertices in V_γ before it can be applied, and so we may assume that

$$Y_\gamma \geq -Y_\gamma^-(Op).$$

By assumption, $-Y_\gamma^-(Op) \geq k$, so S_k must equal $Y_\gamma - k + 1$ before Op is performed. Hence, the change in S_k due to Op in this case is exactly $-(Y_\gamma - k + 1) \leq Y_\gamma^-(Op) + k - 1$ by the inequality above. Combining these cases, such an Op must *subtract* at least

$$m_{\gamma,k} := \min(-\Delta Y_\gamma(Op), -Y_\gamma^-(Op) - k + 1)$$

from S_k.

The net increase in S_k throughout the algorithm, including the initial step, must be 0 since (A) implies that initially $Y_\gamma = 0$, and of course at the end no vertices remain. Let $s = s(k, Op_{\text{init}})$ denote the value of S_k after the first operation Op_{init}, and note that all subsequent operations are in OPS_2. The considerations above produce the following constraint, which we call $\mathbf{C}_{\mathbf{P}_k}(s)$:

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

We refer to these constraints, for each $-k \in K$, as *priority constraints*. Note that they do not need to hold for every k and s, but for each k, in any application of the algorithm, $\mathbf{C}_{\mathbf{P}_k}(s)$ must hold for some s which is a feasible value of S_k after the first operation.

With the same definition of s, we will also establish the following additional priority constraint $\mathbf{C}'_{\mathbf{P}_k}(s)$ for each positive k:

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

The justification for this constraint is as follows. Let $Y_{\gamma,k} = \lfloor Y_\gamma/k \rfloor$. As before, the net change in $Y_{\gamma,k}$ over the course of the whole algorithm is 0. The first two summations provide an upper bound on the net increase in $Y_{\gamma,k}$ due to all operations with $\Delta Y_\gamma(Op) > 0$, apart from the increase s due to the first operation. The operations in the first summation can only be performed, in view of condition (B), 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 with $\Delta Y_\gamma(Op) > 0$ must have $Y_\gamma^-(Op) < 0$, and $\lceil \Delta Y_\gamma(Op)/k \rceil$ is the maximum possible increase in $Y_{\gamma,k}$ in such a step, which yields the terms in the second summation. For any operation Op with $\Delta Y_\gamma(Op) < 0$, the magnitude of the decrease in $Y_{\gamma,k}$ is at least $\lfloor -\Delta Y_\gamma(Op)/k \rfloor$. The third summation, which is subtracted, is hence a lower bound on the net decrease in $Y_{\gamma,k}$ due to such operations.

For any possible initial operation Op_{init}, consider the linear program obtained from LP_0 by altering the right hand side constants in the constraints $\mathbf{C}_i$ to represent the part of the algorithm remaining after the first step, adding any prescribed set of the priority constraints described above, with appropriate value of s as determined by Op_{init}, and excluding the operations in $\{OPS_0 \cup OPS_1\}$ by adding the appropriate equations. Solving this LP will again give an upper bound on the size of the set $\mathcal{S}$. However, it gives a different LP for each value of n, and we need to remove this dependence on n.

First, scale all variables in the problem by $1/n$ (effectively, the only change is to multiply the right hand side of the constraints by $1/n$ and, after solving, scale the solution back up by a factor of n) and denote this linear program by $\mathrm{LP}_1(Op_{\mathrm{init}})$. There are still $O(1/n)$ variations in the right hand side constants in the constraints, depending on n and on the initial operation Op_{init}. To remove these, define the linear program LP_2 from $\mathrm{LP}_1(Op_{\mathrm{init}})$ by setting the right hand side of all constraints (except $\mathbf{C}_t$) to 0. LP_2 is then independent of Op_{init} and, apart from scaling by $1/n$, 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 now have the task of estimating the error in approximating $\mathrm{LP}_1(Op_{\mathrm{init}})$ by LP_2. This can be done using the theory of post-optimal analysis of solutions of LP's.

Lemma 1 *If the solution of LP_2 is finite, then for any fixed initial operation, the solutions of $LP_1(Op_{\mathrm{init}})$ and LP_2 differ by at most c/n for some constant c independent of n.*

Proof: We may assume that the first constraint listed in LP_2 is $\mathbf{C}_t$, so the column vector of the right hand sides of the constraints of LP_2 is

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

Let t' denote the total number of constraints, including priority constraints, in LP_1 and hence also in LP_2. For $1 \leq i \leq t'$, let $\Delta\mathbf{b}_i$ be the change in the right hand side of the i-th constraint in passing back from LP_2 to $\mathrm{LP}_1(Op_{\mathrm{init}})$. For example, if the j-th constraint is one of the original constraints $\mathbf{C}_i$ of LP_0, then $-n\Delta\mathbf{b}_j$ is the change in Y_i due to the initial operation. The difference in the constant column vectors, between LP_2 and $\mathrm{LP}_1 = \mathrm{LP}_1(Op_{\mathrm{init}})$, is now

$$\Delta\mathbf{b} = [\Delta\mathbf{b}_1, \ldots, \Delta\mathbf{b}_{t'}]^{\mathbf{T}}.$$

Each operation Op_{init} can alter the right hand sides of the constraints by at most a constant before scaling. Hence, $\Delta\mathbf{b}_i = c_i/n$ for some constant c_i depending on i.

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. By [27, equation (20), p. 126]), $\kappa_1 \leq \kappa_2 - y^*\Delta\mathbf{b}$ provided that both LP's have finite optima. LP_2 has a finite optimum by the assumption of this theorem. That $\mathrm{LP}_1(Op_{\mathrm{init}})$ has a finite optimum is shown below. Since $\Delta\mathbf{b}_i = c_i/n$ 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.

It only remains to show that $\mathrm{LP}_1(Op_{\mathrm{init}})$ has a finite optimum. We only need to show that it is feasible (so the optimum is not $-\infty$, under the interpretation in [27]) and that the objective function is bounded above by the constraints. Feasibility follows by fact that the constraints were built based on an argument that $r(OP)$ represents the number of operations in the algorithm ALG. We assumed explicitly near the start of Section 2.2 that ALG can always process the graph and terminate with all vertices deleted. Hence, all constraints must be satisfied in any one run of ALG, which proves that there is a feasible solution. □

Note that, when we use an LP solver to find a solution, we use one that also gives a solution to the dual. The duality theorem of linear programming then gives us a simple way to check the claimed upper bound on the solution given by the primal LP.

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 conceivably 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. In several cases, by chaining multiple copies of this subgraph together, we are able to construct an infinite family of cubic graphs for which the worst case performance of the given algorithm is equal to that indicated by the solution of the LP, to within a constant number of vertices. In this way, we show that the upper bound given by the LP is essentially sharp.

For a given algorithm, we sometimes impose additional priorities on operations in order to cut down on the number of operations that can possibly occur in the solution. No extra constraints need to be added but this permits us to exclude certain operations. This can result in reducing the number of operation variables $r(OP)$ that have non-zero value in the solution of the LP. We found no case where this reduced the value of the solution, but it did lead to simpler example graphs.

Figure 4 represents one of the many possible ways to form these example graphs. Each shaded diamond (and an incident edge) represents a copy of the repeating subgraph. If this is done in a suitable way, it can be argued that the black vertex can be selected to be added to the set by the initial operation of the algorithm, and that each repeated subgraph in the chains is then processed in turn until the last operation is performed.

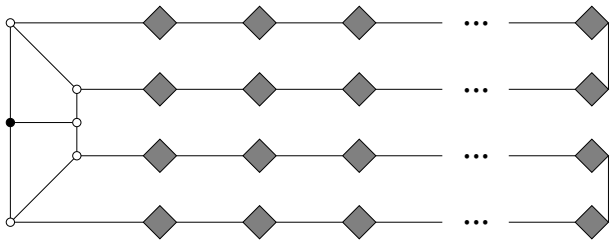

Figure 4: Forming a cubic graph

3 Small Maximal Matchings

Yannakakis and Gavril [31] showed that the size of a smallest edge dominating set of a graph is the same as the size of a smallest maximal matching (MM) and that the problem of finding a *minimum* maximal matching (MMM) 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 an MMM of trees. Horton and Kilakos [16] showed that the problem of finding an MMM remains NP-hard for planar bipartite graphs and planar cubic graphs. They also gave a polynomial time algorithm that finds an MMM for various classes of chordal graphs. More recently, Zito [32], 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 an MMM is approximable within the ratio 5/3. Zito [33] showed that for a *random* n-vertex cubic graph G, the size of an 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 an MM of size at most $9n/20 + O(1)$. We also show that there exist 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 an 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.

The operation in Figure 5 has $\Delta Y_3 = -4$, $\Delta Y_2 = 0$, $\Delta Y_1 = 0$ and $m(Op) = 2$ (as the edges incident with the vertices 1, 2, 4 and 5 are deleted, vertices 3 and 6 are changed

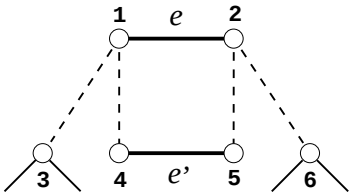

Figure 5: An example operation for Edge_Greedy

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. For this we must define the function $\mathbf{MinN}(T)$ which, given a set of vertices T, 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

Figure 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. (See the Appendix for the list of operations not excluded.) 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 (see the Appendix). The solution is shown in Figure 7. Maple also returns a dual solution, which can be substituted directly into the problem to give a simple proof that the upper bound on the solution is correct. By Lemma 1 this shows that for n-vertex cubic graphs, algorithm Edge_Greedy returns an MM of size at most $9n/20 + O(1)$.

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

Figure 7: A solution to the LP for Edge_Greedy

The operations Op_i (for $i \in \{1, 2, 3\}$) are shown in Figure 8. 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.

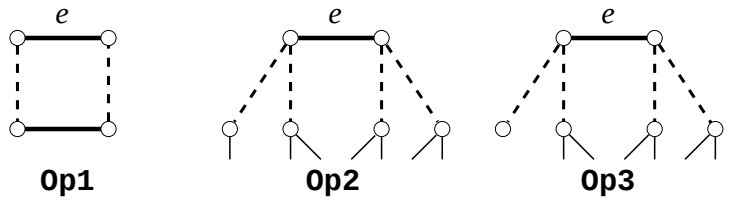

Figure 8: Operations in the LP solution for Edge_Greedy

□

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

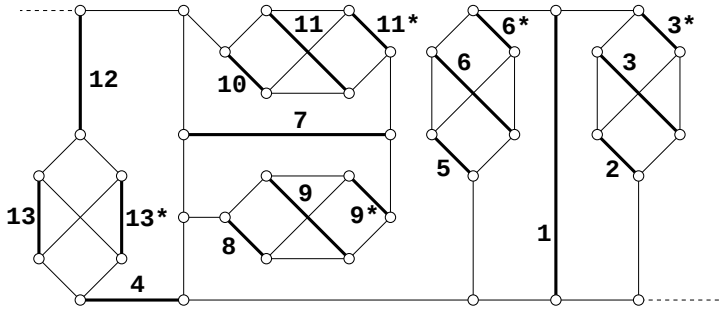

Figure 9: Repeating component

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 an MM of size at most $9n/20 + O(1)$.

Having considered an upper bound on the size of an MMM of a cubic graph, we now consider the maximum, over all n-vertex cubic graphs, of the the size of an 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 $3n/8$.

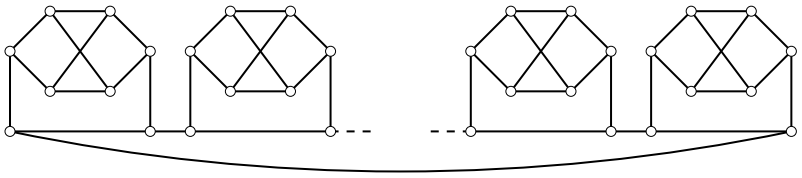

Figure 10: No MM of size less than $3n/8$

3.3 Cubic Graphs with Girth at least 5

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 an MM, $\mathcal{E}$, of G.

Theorem 2 *Given a connected, n-vertex, cubic graph of girth at least 5, the 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.

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

Figure 11: Edge_Greedy5 solution

The operations Op_i (for $i \in \{1, 2, 3, 4, 5\}$) are shown in Figure 12.

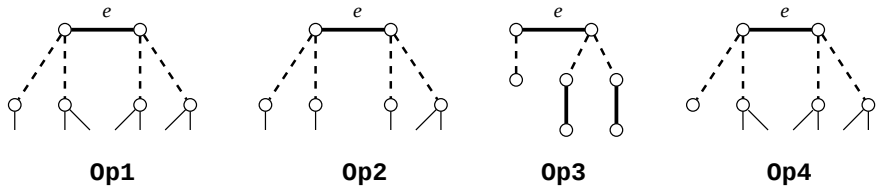

Figure 12: Operations in the LP solution for Edge_Greedy5

□

4 Small Connected Dominating Sets

The problem of finding an MCDS is a well known NP-hard optimisation problem [12] 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 [30] showed that $\mathcal{MLST}$ is approximable with approximation ratio 2, improving the previous best known approximation ratio of 3 by Lu and Ravi [23]. Galbiati, Maffioli and Morzenti [11] 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 [13] 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 [14] that "*grows a tree*". The idea behind the algorithm of [14] 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"

Figure 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 an 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 [13]. 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 [14], 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° white neighbours	Set	Colour	N° 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

Figure 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 $m(Op) = 1$.

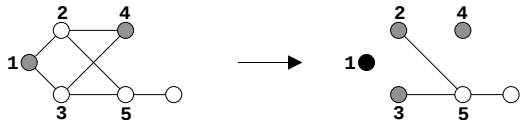

Figure 15: An example operation for Build_Tree

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

Figure 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.

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

Figure 17: Build_Tree LP solution

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.

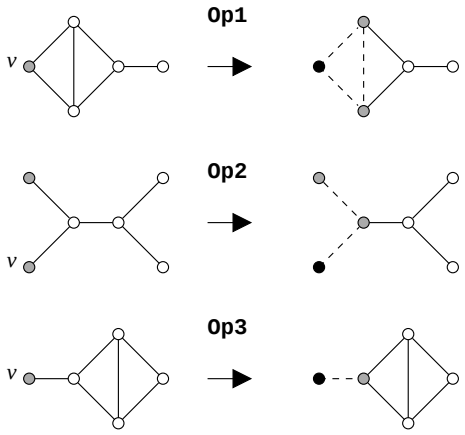

Figure 18: Operations in the LP solution for Build_Tree

□

The subgraph that forms part of a cubic graph that realises the solution of the linear program is shown in Figure 19.

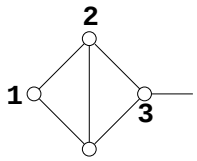

Figure 19: Repeating component

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 [13] using the example given in Figure 20. 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. Since each component of four vertices must contribute at

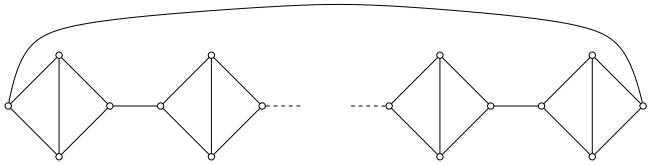

Figure 20: No CDS of size less than $3n/4$

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 21.

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, the 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 equations associated with the operations Op_i (for $i \in \{1, 2, 3\}$) may be derived from Figure 22. For each operation, black vertices are added to the CDS and deleted edges are indicated by dotted lines.

The solution to LP_2 and the non-zero variables in the solution are shown in Figure 23.

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

Figure 21: Algorithm Build_Tree5

□

Now consider the maximum, over all n-vertex cubic graphs of girth 5, of the the size of an MCDS. The graph of Figure 24 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 $4n/7$.

5 Small Independent Dominating Sets

The problem of finding a MIDS is one of the core NP-hard optimisation problems in graph theory [12]. Halldórsson [15] showed that for general graphs, this problem is not approximable within $n^{1-\epsilon}$ for any $\epsilon > 0$. Kann [17] showed that when restricted to graphs

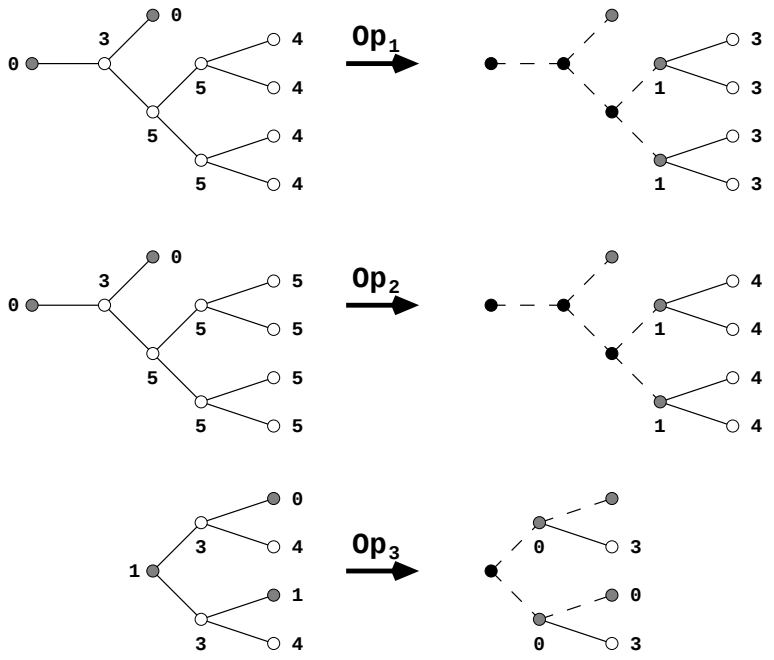

Figure 22: Operations in the LP solution for Build_Tree5

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

Figure 23: Build_Tree5 LP solution

of maximum degree at least 3, the problem is APX-Complete. Lam, Shiu and Sun [20] 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.

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 connected 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 $3n/8 + O(1)$.

Suppose we relax the independence constraint on the dominating set. Reed [26] 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 [25] 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]

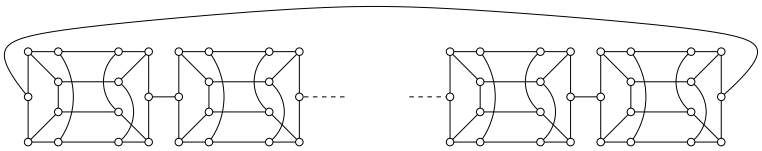

Figure 24: No CDS of size less than $4n/7$

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 25.

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 26.

$v \leftarrow$ **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

Figure 25: Algorithm Min_Ratio

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

Figure 26: Min_Ratio solution

The operations Op_i (for $i \in \{1, 2, 3\}$) are as shown in Figure 27.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.

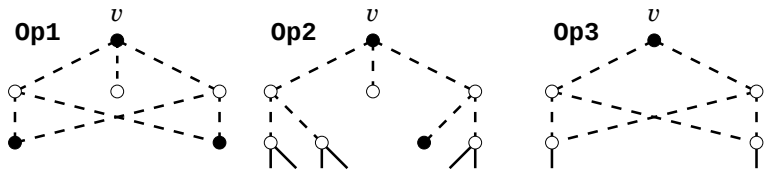

Figure 27: Operations performed in the worst case

□

The subgraph that forms part of a cubic graph that realises the solution of the linear program is shown in Figure 28. 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

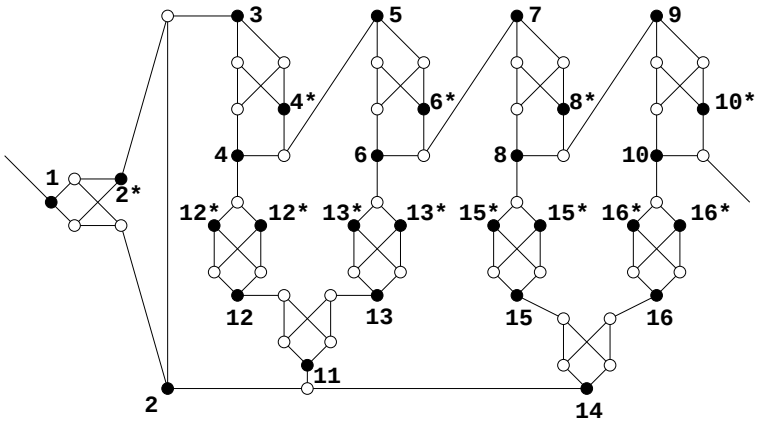

Figure 28: Repeating component

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 29 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 $3n/8$.

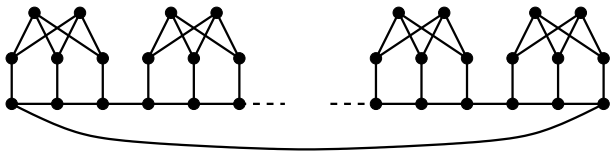

Figure 29: No independent dominating set of size less than $3n/8$

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, the 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 30.

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

Figure 30: Min_Deg_One solution

The operations Op_i (for $i \in \{1, 2, 3, 4, 5, 6\}$) represent the operations shown in Figure 31.

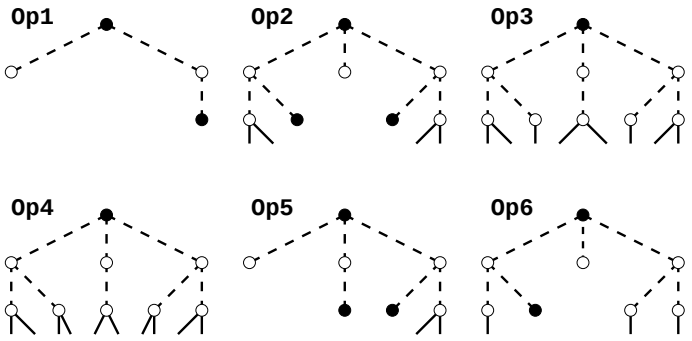

Figure 31: Operations performed in the worst case

□

Regarding sharpness, the graph of Figure 32 represents a family of cubic graphs which contain a chain of k repeating components. Each component has eighteen vertices indi-

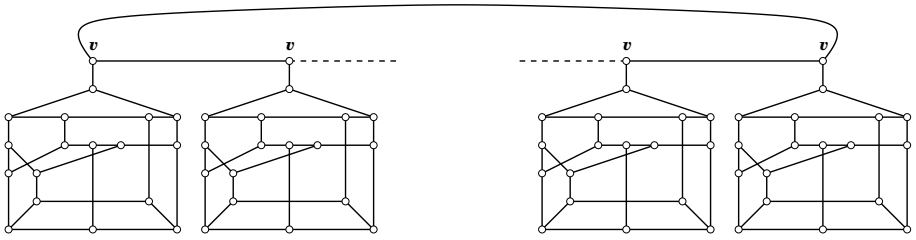

Figure 32: No IDS of size less than $n/3$

cating 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 was achieved, the same result is relatively easily obtained by means of a simpler analysis.

Acknowledgments The authors would like to thank anonymous referees whose thoughtful comments led us to correct some errors and improve the presentation.

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] D. Fisher, K. Fraughnaugh and S. Seager, The domination number of cubic graphs with girth at least five. The Ninth Quadrennial International Conference on Graph Theory, Combinatorics, Algorithms and Applications, 9 pp. (electronic), Electron. Notes Discrete Math. 11, Elsevier, Amsterdam, 2002.

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

[11] 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.

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

[13] 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.

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

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

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

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

[18] A.V. Kostochka and B.Y. Stodolsky. An upper bound on the domination number of n-vertex connected cubic graphs, Discrete Math., 309 (2009) 1142–1162.

[19] D. Král, P. Škoda and J. Volec, Domination number of cubic graphs with large girth (submitted).

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

[21] J. Lauer and N. Wormald, Large independent sets in regular graphs of large girth, J. Combinatorial Theory, Series B, 97 (2007), 999–1009.

[22] C. Löwenstein and D. Rautenbach, Domination in graphs with minimum degree at least two and large girth, Graphs Combin. 24 (2008) 37-46.

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

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

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

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

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

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

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

[30] 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.

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

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

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