Average-case complexity of a branch-and-bound algorithm for Min Dominating Set

Tom Denat[a], Ararat Harutyunyan[a], Nikolaos Melissinos[b], Vangelis Th. Paschos[a]

[a] *Université Paris-Dauphine, PSL University, CNRS, LAMSADE, 75016 Paris, France,* {*tom.denat,ararat.harutyunyan,vangelis.paschos*}*@lamsade.dauphine.fr*
[b] *Czech Technical University in Prague, Department of Theoretical Computer Science, Faculty of Information Technology, Prague, Czech Republic, melisnik@fit.cvut.cz*

Abstract

The average-case complexity of a branch-and-bound algorithm for Min Dominating Set problem in random graphs in the $\mathcal{G}(n,p)$ model is studied. We identify phase transitions between subexponential and exponential average-case complexities, depending on the growth of the probability p with respect to the number n of nodes.

Keywords: Approximation scheme, Subset-Sums ratio, Knapsack problems, Combinatorial optimization

1. Introduction

Given a graph $G = (V, E)$ of order n, a dominating set $S \subseteq V$ is a subset of V such that any $v_i \in V$ is either included in S or connected to a vertex of S by an edge of E. The Min Dominating Set problem consists of finding a minimum-size dominating set in G. Min Dominating Set is a very well-known **NP**-hard problem completely equivalent (from both complexity and polynomial approximation points of view) to min set cover problem.

Dealing with the exact solution of Min Dominating Set, besides the obvious $O(2^n)$ algorithm which considers the power set of V and chooses the smallest one that also forms a dominating set, several moderately exponential algorithms have been proposed mainly during the last fifteen years. To the best of our knowledge, the fastest one is the $O(1.4969^n)$ algorithm due to [14].

The main purpose of this paper is the study of the average case complexity of branch-and-bound algorithms for the Min Dominating Set problem in random graphs in the $\mathcal{G}(n,p)$ model. This model represents graphs on n vertices where each of the possible $\binom{n}{2}$ edges appears independently with probability p. For an extensive treatment of random graphs, we refer the reader to the monograph [4].

The branch-and-bound technique is one of the best known and most widely used for exactly solving **NP**-hard problems. A branch-and-bound algorithm

searches the solution space by recursively partitioning it. The progress of the algorithm is usually presented by a decision tree (also called branch-and-bound-tree) while the solution space (represented by number of nodes we need to explore) can be reduced by the introduction of pruning rules that allow us to completely ignore part of the solution space. An area where such algorithms have significant success is the mixed integer linear programming solvers, where all state-of-the-art algorithms use variable branching (see [3, 11]). One of the reasons of this success is that, in practice, the branch-and-bound-tree can be really small ([8]). More recently, in [5, 6, 10] the authors proved that there exist branch-and-bound algorithms that, with good probability, require polynomial time in order to return a solution of random instances of integer programming with fixed number of constrains.

Even though mathematical tools for average case analysis of algorithms have existed for decades [13] and have much advanced in sophistication [12], we do not know of many results on the average case complexity of branch-and-bound algorithms for graph-theoretical **NP**-hard problems in random graphs. The only works known to us are the ones of [2] where the authors study the complexity of a "pruning the search-tree algorithm" for MAX INDEPENDENT SET (the worst-case complexity of this algorithm is $O(1.1996^n)$, [15]) under the $\mathcal{G}(n,p)$ model, the one of [1], where the same algorithm is studied under the $\mathcal{G}(n,m)$ model and, finally, the one in [7] where the average-case complexity of a branch-and-bound algorithm for MAX INDEPENDENT SET is studied under the $\mathcal{G}(n,p)$ model. Here our goal is to study the complexity of a (rather intuitive and simple, with respect to the bounds in the nodes of the branch-and-bound-tree), branch-and-bound-algorithm for MIN DOMINATING SET. Even if, the obtained complexity-bounds are worse than the ones obtained for the problem by other more evolved methods (inclusion-exclusion, dynamic programming, etc.), we think that it is interesting to obtained some formal complexity results for the branch-and-bound method (the most used one for the exact solution of **NP**-hard problems).

In what follows, in Section 2 we present the branch-and-bound algorithm we considered while in Section 3 we analysed the complexity of this algorithm.

2. The branch-and-bound algorithm

Let $G = (V, E)$ be a graph; set $n = |V|$ and fix an order $v_1, v_2, \ldots, v_n$ on V. The type of branch-and-bound algorithms for MIN DOMINATING SET studied here works by building a branch-and-bound binary tree, nodes of which are associated with a vector $\vec{x} \in \{0,1\}^n$ and a depth δ in the binary tree. The idea is that the vector $\vec{x}$ gives us the set that we consider as dominating set of the graph. In particular, $x_i = 1$ means that vertex v_i has been taken in the solution under construction and $x_i = 0$ means that v_i has not been taken. For a tree-node at level δ only vertices $v_1, v_2, \ldots, v_\delta$ have been fixed, i.e., only $x_1, x_2, \ldots, x_\delta$ of $\vec{x}$ have been assigned definite values. Also, the values for $x_{\delta+1}, \ldots, x_n$ are, for the moment, equal to 1. We remark that the superset of a dominating set is also a dominating set.

For a node x at level δ_x with vector $\vec{x}$, we define its cost $u(x)$ as the number of vertices that currently must be included in the solution. Formally:

$$u(x) = \sum_{i=1}^{\delta_x} x_i$$

Cost $u(x)$ can be seen as an optimistic prediction of the size of the optimal solution in the sense that, in the best case, we do not need to include any vertices form the set $\{v_i \mid i > \delta_x\}$ in order to have a feasible solution that respects the first δ_x choices (i.e., the set defined by the values x_i, $i \leqslant \delta_x$, is a dominating set).

The idea of the algorithm is to start from a root that represents the trivial dominating set including all the vertices ($\vec{x} = (1, 1, ..., 1)$) at the depth 0 and is initially un-visited. Then the algorithm repeats the following process. From all un-visited vertices, we select a node that has the minimum u-value. Let x be the selected node associated with a vector $\vec{x} \in \{0, 1\}^n$ and a depth δ in the binary tree. If $\delta < n$ then create both left and right children of x; ℓ and r. We associate ℓ and r with the vectors $\vec{\ell}$ and $\vec{r}$ respectively, where $\vec{\ell}_i = \vec{r}_i = x_i$ for all $i \neq \delta + 1$, $\ell_{\delta+1} = 1$ and $r_{\delta+1} = 0$. We set ℓ as un-visited. Also, if the set defined by $\vec{r}$ is a dominating set, we set r as as un-visited, otherwise we set r as non-interesting. If $\delta = n$ the algorithm terminates and return as minimum dominating set the set defined by the vector $\vec{x}$.

At each new step of the algorithm a new node will be visited. Also notice that a created node $(\vec{x}, \delta)$ that corresponds to a dominating set in G is either the left or the right child of an already visited node. Also, any vertex that is marked as non-interesting will be never visited by the algorithm.

Therefore, at each step of the algorithm, the nodes of the complete binary tree that represents the branch-and-boundtree can be divided in five categories:

1. un-visited nodes which correspond to dominating sets that have not been considered by the algorithm;
2. visited nodes which correspond to dominating sets that have been considered by the algorithm;
3. non-interesting nodes which correspond to sets that are not dominating sets in G, i.e., some infeasible solutions;
4. nodes that have not been created.

Nodes that have not been created either correspond to vertex sets that are not dominating sets or to vertex sets that are dominating sets but there parent nodes have not yet been visited (and may never be visited if the algorithm finds a minimum solution before it is needed).

As an example we consider the graph G in figure 1 which contains three vertices A, B and C. Therefore, the branch-and-bound tree contains 3 levels. The branch and bound algorithm starts by visiting the root $\{1, 1, 1\}, 0$. Now, the un-visited nodes are $\{1, 1, 1\}, 1$ and $\{0, 1, 1\}, 1$ with respective cost 1 and 0. Therefore, the node $\{0, 1, 1\}, 1$ is visited next. Then, since $\{0, 0, 1\}, 2$ does not

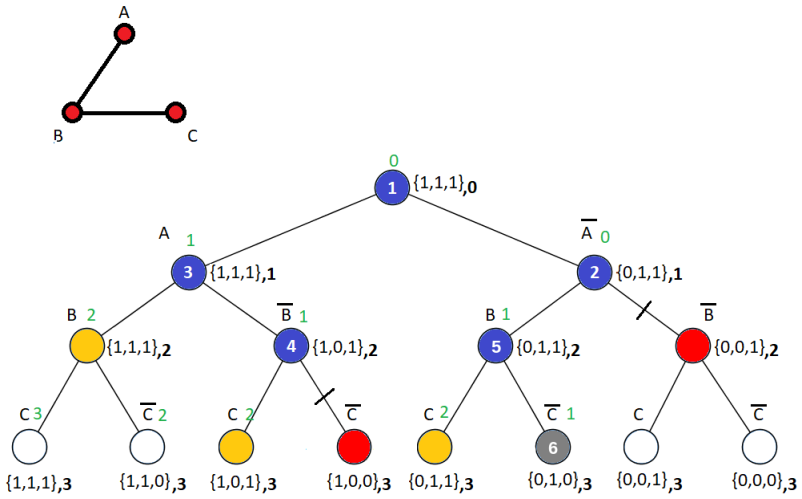

Figure 1: Illustration of a possible execution of the algorithm on graph G. The graph G appears on the upper left corner of the figure. In this case, the algorithm has visited 6 nodes in the order presented. In blue we present the visited nodes that do not belong in the last level. In yellow we have the un-visited nodes (i.e. feasible solutions). With red we have the non-interested nodes and with white we have nodes that have not been created. Finally, with grey, we have the first visited vertex of the last level which corresponds to the returned solution.

correspond to dominating set it is marked as visited. Therefore, the two un-visited nodes are $\{1,1,1\},1$ and $\{0,1,1\},2$, both with potential 1. The next node to be visited is chosen randomly. Assume that the next visited node is $\{1,1,1\},1$ and so on. Finally, the complete solution $\{0,1,0\},3$ is found. As its depth is 3, this solution $\{B\}$ is the minimum dominating set.

3. Analysis of the branch-and-bound algorithm

In what follows, we denote by $\mathbb{T}(n,p)$ the average complexity of branch-and-bound in a binomial random graph $G=(V,E)$ with parameters (n,p), that is, the total number of created nodes. For a node x in depth δ_x related to the vector $\vec{x}$, we will denote with S_x the set defined by the values x_i, $i \leqslant \delta_x$. Also, for any set $S \subseteq \{1,\ldots,i\}$ and $j \in \{i+1,\ldots,n\}$ we will denote S^j the set $S \cup \{j,\ldots,n\}$. Notice that if $i+1 > n$ then $\{j,\ldots,n\}$ is empty and $S^j = S$.

Let x be the first node at the nth level, where n is the order of G, visited by the algorithm. The fact that we visited x means that the set S_x is a dominating set of G. Also, any node x' that has not been visited yet must have $u(x') \geqslant u(x) = |S_x|$ as otherwise it would have been visited before. Finally, any other un-visited node x' has $u(x') \geqslant |S_x|$ therefore S_x must be a minimum dominating set, and the algorithm terminates. It follows that, during the running of the

algorithm, a node x is visited only if $u(x) \leqslant |S^*|$, where S^* is a minimum dominating set.

Thus, for a visited node x in depth i with $S_x \subseteq \{1, \ldots, i\}$ we have dominating set S_x^{i+1}, satisfying $|S_x| \leqslant |S^*|$. We can therefore say that the number of nodes in the branch-and-boundtree is at most $\sum_{k=1}^{|S^*|} \binom{n}{k}$ and the average running time $\mathbb{T} \leqslant \mathbb{E}(\sum_{k=1}^{|S^*|} \binom{n}{k})$. Finally, notice that in order to bound $\mathbb{T}$ it suffices to bound the value $\binom{n}{k} \Pr[\gamma \geqslant k]$ for $1 \leqslant k \leqslant n$. To do so, we define the function $f(n)$ as follows: for each $n > 0$, let $f(n) = k^* \leqslant n$ where $\binom{n}{k^*} \Pr[\gamma \geqslant k^*] = \max_{1 \leqslant k \leqslant n} \binom{n}{k} \Pr[\gamma \geqslant k]$. Thus, we need to upper-bound only $M := \binom{n}{f(n)} \Pr[\gamma \geqslant f(n)]$ for all $n \geqslant 1$.

3.1. Upper bounds

The following theorem provides upper bounds for the complexity of the branch-and-bound algorithm presented in Section 2 for random graphs in the $\mathcal{G}(n, p)$ model.

Theorem 1. *The following two facts hold:*

(a) If p is constant, then the branch-and-bound algorithm takes subexponential time as $n \to \infty$.

(b) If $pn = c$, where $c \geqslant 20$ is a constant, then the branch-and-bound algorithm takes time at most $(2 - \varepsilon)^n$, where we can take $\varepsilon = 0.01$.

Proof. We first focus on the case where p is fixed. We show that in this case, M is subexponential. We use the fact that $\gamma(G) \leqslant \alpha(G)$ for every graph G, where $\gamma(G)$ denotes the size of the minimum dominating set of G and $\alpha(G)$ denotes the stability number of G (indeed, a maximal independent set is a dominating set) and the union bound.

We recall that $1 - x \leqslant e^{-x}$ for all values of x (this follows from the fact that $1 - x$ is the tangent line of e^{-x} at $x = 0$). We remark that $\alpha(G) > x$ for some x, implies that one of the $\binom{n}{x}$ subsets of vertices of size x induces an independent set in G. Thus:

$$\begin{aligned} M \leqslant \binom{n}{f(n)} \Pr[\alpha \geqslant f(n)] &\leqslant \binom{n}{f(n)} \binom{n}{f(n)} (1-p)^{\binom{f(n)}{2}} \\ &\leqslant \left(\binom{n}{f(n)} \right)^2 e^{-pf(n)(f(n)-1)/2}. \end{aligned}$$

If $f(n) = o(n)$, then $\binom{n}{f(n)}^2 < (en/f(n))^{2f(n)}$ is clearly subexponential. Thus, we may assume $f(n) = \Theta(n)$. Quantity M clearly satisfies:

$$M \leqslant \left(\left(\frac{en}{f(n)} \right)^2 e^{-p(f(n)-1)/2} \right)^{f(n)}$$

Since $f(n) = \Theta(n)$ and p is fixed, clearly, $(en/f(n))^2 e^{-p(f(n)-1)/2} < 1$ for n sufficiently large. In fact, with a slightly more careful analysis, we can obtain that M is subexponential in the regime $pn \to \infty$. Indeed, we may suppose as before that $f(n) = \Theta(n)$ and now it follows that $(en/f(n))^2$ is bounded by a constant and since $p = \omega(1)$ we have that $e^{-p(f(n)-1)/2} \to 0$, as $n \to \infty$. Therefore, $(en/f(n))^2 e^{-p(f(n)-1)/2} < 1$ for n sufficiently large and the claim follows.

It remains to consider the case (b). In this case we have $p = c/n$, where $c \geqslant 20$ is some constant; we will assume in the computations that follow that $p = 20/n$ since this would give an upper bound on the number of steps of the algorithm. As before, we consider the quantity $\binom{n}{f(n)}^2 e^{-pf(n)(f(n)-1)/2}$. We set $\varepsilon := \frac{f(n)}{n}$, where $0 < \varepsilon \leqslant 1$. We will show that for any value of ϵ, the running time of the branch-and-bound algorithm takes time at most 1.99^n.

Thus, from above, it suffices to show that that $\left(\binom{n}{\varepsilon n}\right)^2 e^{-p\varepsilon n(\varepsilon n-1)/2} \leqslant 1.99^n$. We use the fact that $\binom{n}{\varepsilon n} \leqslant 2^{H(\varepsilon)n}$, where $H(\varepsilon)$ is the binary entropy function. Now,

$$\binom{n}{\varepsilon n}^2 e^{-p\varepsilon n(\varepsilon n-1)/2} \leqslant (2^{nH(\varepsilon)})^2 e^{-10\varepsilon^2 n+10\epsilon} \leqslant e^{10}((\varepsilon^{-\varepsilon}(1-\varepsilon)^{-(1-\varepsilon)})^2)^n e^{-10\varepsilon^2 n}$$

We remark that $\varepsilon^{-\varepsilon}(1-\varepsilon)^{-(1-\varepsilon)}$ is increasing on the interval $(0, 1/2)$ and decreasing on $(1/2, 1)$, with its maximum at $\varepsilon = 1/2$, since $H(\varepsilon)$ is the logarithm of this function. Moreover, this function is symmetric with respect to $\varepsilon = 1/2$, and as $e^{-10\varepsilon^2 n+10}$ decreases with ε, we can assume that the maximum for $((\varepsilon^{-\varepsilon}(1-\varepsilon)^{-(1-\varepsilon)})^2)^n e^{-10\varepsilon^2 n+10}$ is attained for some $\varepsilon \leqslant 1/2$ (for otherwise, we could obtain a larger value by taking $1-\varepsilon$ instead of ε). Since $\varepsilon^{-\varepsilon}(1-\varepsilon)^{-(1-\varepsilon)}$ is increasing on the interval $(0, 1/2)$, there is some constant $\varepsilon_0 < 1/2$ such that, for all $\varepsilon \leqslant \varepsilon_0$ we have $\varepsilon^{-\varepsilon}(1-\varepsilon)^{-(1-\varepsilon)} < \sqrt{1.99}$. It is easy to check that we can take $\varepsilon_0 = 1/10$. Thus, we may assume that $\varepsilon > 1/10$. Indeed, for otherwise $((\varepsilon^{-\varepsilon}(1-\varepsilon)^{-(1-\varepsilon)})^2)^n < 1.99^n$ and we are done. Therefore, to complete the proof, it suffices to show that:

$$\left[\left(\varepsilon^{-\varepsilon}(1-\varepsilon)^{-(1-\varepsilon)}\right)\right]^2 e^{-10\varepsilon^2} < 1.99$$

for all $\varepsilon \in [1/10, 1/2]$, since this implies that $e^{10}((\varepsilon^{-\varepsilon}(1-\varepsilon)^{-(1-\varepsilon)})^2)^n e^{-10\varepsilon^2 n} \leqslant 1.99^n$. Since the first of the two products is increasing and the second is decreasing with ε, we will dominate each separately. To this end, we refine the intervals of ε. First consider the interval $\varepsilon \in [1/10, 1/8]$. To bound our product, it is sufficient to substitute $1/8$ in the first term and $1/10$ in the second. By doing this, we obtain that the product is less than 1.99. It is easily verified that we can repeat this argument on the following intervals, thus finishing the theorem. In each case, we obtain a bound of less than 1.99. The precise bounds are given below, where $g(\varepsilon) := [(\varepsilon^{-\varepsilon}(1-\varepsilon)^{-(1-\varepsilon)})]^2 e^{-10\varepsilon^2}$:

- $\varepsilon \in [1/8, 1/7]; g(\varepsilon) < 1.943$;
- $\varepsilon \in [1/7, 0.15]; g(\varepsilon) < 1.9$;

- $\varepsilon \in [0.15, 0.17]; g(\varepsilon) < 1.988$;
- $\varepsilon \in [0.17, 0.19]; g(\varepsilon) < 1.981$;
- $\varepsilon \in [0.19, 0.21]; g(\varepsilon) < 1.95$;
- $\varepsilon \in [0.21, 0.25]; g(\varepsilon) < 1.982$;
- $\varepsilon \in [0.25, 0.35]; g(\varepsilon) < 1.955$;
- $\varepsilon \in [0.35, 0.5]; g(\varepsilon) < 1.2$.

The proof of the theorem is now completed. □

3.2. Lower bounds

The following result shows that the upper bound on complexity of the algorithm given by Item (b) of Theorem 1 cannot be drastically improved in order that a subexponential bound is taken.

Theorem 2. *Let $p = c/n$, where c is a positive fixed constant. Then, the branch-and-bound algorithm takes at least $(1/\varepsilon)^{\varepsilon n}$ time for $G(n,p)$, where $\varepsilon := \min\{0.01, 1/10c\}$*

Proof. Note that for any k, $\binom{n}{k} \Pr[\gamma \geqslant k]$ is a lower bound on the complexity of the algorithm. Thus, it is sufficient to show that:

$$\binom{n}{\varepsilon n} \Pr[\gamma \geqslant \varepsilon n] = \Omega\left(\left(\frac{1}{\varepsilon}\right)^{\varepsilon n}\right)$$

for n sufficiently large.

We will prove that $\Pr[\gamma < \varepsilon n]$ is arbitrarily small. This is clearly sufficient. Let A be the event that a fixed set S of size εn is a dominating set. Then, $\Pr[\gamma < \varepsilon n] \leqslant \binom{n}{\varepsilon n} \Pr[A] < 2^n \Pr[A]$. We use the fact that $1 - x \geqslant e^{-2x}$ for all $x \in (0, 1/2)$; this can be seen, for example, by noticing that e^{-2x} is a convex function and that $1 - x = e^{-2x}$ has two solutions at $x = 0$ and at some $x \in (0.5, 1)$. Now, $\Pr[A] = (1 - (1-p)^{\varepsilon n})^{n - \varepsilon n} \leqslant (1 - e^{-2c\varepsilon})^{n - \varepsilon n}$. Thus:

$$\Pr[\gamma \leqslant n - \varepsilon n] \leqslant \left(2\left(1 - e^{-2c\varepsilon}\right)^{(1-\varepsilon)}\right)^n$$

Note that by definition of ε, $2c\varepsilon < 1/5$. Thus, $(1 - e^{-2c\varepsilon})^{1-\varepsilon} \leqslant (1 - e^{-1/5})^{0.99} \leqslant 1/2$. It follows that:

$$\binom{n}{\varepsilon n} \Pr[\gamma > n - \varepsilon n] = \Omega\left(\left(\frac{1}{\varepsilon}\right)^{\varepsilon n}\right)$$

as required. □

4. Conclusion

We have studied in this paper the average-case complexity of a branch-and-bound algorithm for Min Dominating Set in random graphs under the $\mathcal{G}(n,p)$ model. It has been proved that this complexity is: (a) *subexpontial* when p is constant; (b) *exponential* when $p = {}^{c}/_{n}$. For the latter case, it was proved that the smaller the constant c, the closer to 2^n is the average case complexity of the algorithm.

References

[1] C. Banderier, H. Hwang, V. Ravelomanana, and V. Zacharovas. Average case analysis of NP-complete problems: Maximum independent set and exhaustive search algorithms. *International Meeting on Probabilistic, Combinatorial, and Asymptotic Methods in the Analysis of Algorithms, AofA09*, 2009.

[2] C. Banderier, H. Hwang, V. Ravelomanana, and V. Zacharovas. Analysis of an exhaustive search algorithm in random graphs and the $n^{c \log n}$-asymptotics. *SIAM J. Disc. Math.*, 28(1):342–371, 2014.

[3] R. E. Bixby and E. Rothberg Progress in computational mixed integer programming - A look back from the other side of the tipping point. *Annals of Operations Research*, 149:37 – 41, 2007.

[4] B. Bollobás. *Random graphs.* Academic Press, London, 1985.

[5] S. Borst, D. Dadush, S. Huiberts and S. Tiwari. On the integrality gap of binary integer programs with Gaussian data. *Mathematical Programming*, 197:1221–1263, 2023.

[6] S. Borst, D. Dadush and D. Mikulincer. Integrality Gaps for Random Integer Programs via Discrepancy. *Proceedings of the 2023 ACM-SIAM Symposium on Discrete Algorithms, SODA 2023, SIAM*, 1692–1733, 2023.

[7] N. Bourgeois, R. Catellier, T. Denat, and V. Th. Paschos. Average-case complexity of a branch-and-bound algorithm for maximum independent set, under the G(n, p) random model. *CoRR*, abs/1505.04969, 2015.

[8] K. K. H. Cheung, A. Gleixner, and D. E. Steffy, Verifying Integer Programming Results. Integer Programming and Combinatorial Optimization. IPCO 2017. *Lecture Notes in Computer Science, Springer.*

[9] V. Chvátal Hard Knapsack Problems. *Operations Research*, 28:1402 – 1411, 1980.

[10] S. S. Dey, Y. Dubey and M. Molinaro Branch-and-bound solves random binary IPs in poly(n)-time. *Mathematical Programming*, 200:569–587, 2023.

[11] S. S. Dey, M. Molinaro and Q. Wang Analysis of Sparse Cutting Planes for Sparse MILPs with Applications to Stochastic MILPs. *Mathematics of Operations Research*, 43:304 – 332, 2018.

[12] Ph. Flajolet and R. Sedgewick. *Analytic combinatorics*. Cambridge University Press, 2008.

[13] D. E. Knuth. *The art of computer programming: fundamental algorithms*, volume 1. Addison-Wesley, Reading MA, 1969.

[14] J. M.M. van Rooij and H. L. Bodlaender. Exact algorithms for dominating set. *Discrete Appl. Math.*, 159(17):2147–2164, 2011.

[15] M. Xiao and H. Nagamochi. Exact algorithms for maximum independent set. *Information and Computation*, 255:126 – 146, 2017.