Solving Connected Dominating Set Faster than 2^n

Fedor V. Fomin[1], Fabrizio Grandoni[2], and Dieter Kratsch[3]

[1] Department of Informatics, University of Bergen, N-5020 Bergen, Norway, fomin@ii.uib.no
[2] Dipartimento di Informatica, Università di Roma "La Sapienza", Via Salaria 113, 00198 Roma, Italy, grandoni@di.uniroma1.it
[3] LITA, Université Paul Verlaine - Metz, 57045 Metz Cedex 01, France, kratsch@univ-metz.fr

Abstract. In the connected dominating set problem we are given an n-node undirected graph, and we are asked to find a minimum cardinality connected subset S of nodes such that each node not in S is adjacent to some node in S. This problem is also equivalent to finding a spanning tree with maximum number of leaves.
Despite its relevance in applications, the best known exact algorithm for the problem is the trivial $\Omega(2^n)$ algorithm which enumerates all the subsets of nodes. This is not the case for the general (unconnected) version of the problem, for which much faster algorithms are available. Such difference is not surprising, since connectivity is a global property, and non-local problems are typically much harder to solve exactly.
In this paper we break the 2^n barrier, by presenting a simple $O(1.9407^n)$ algorithm for the connected dominating set problem. The algorithm makes use of new domination rules, and its analysis is based on the Measure and Conquer technique.

1 Introduction

Nowadays, it is common belief that NP-hard problems cannot be solved in polynomial time. For a number of NP-hard problems, we even have strong evidence that there are no sub-exponential algorithms [19]. Moreover, many relevant problems do not admit satisfactory approximation algorithms. For example, Maximum Independent Set is hard to approximate within $n^{1-\varepsilon}$ [17]. For these problems a promising approach is to design exact algorithms with smallest possible exponential running times.

The recent interest in exact exponential algorithms has several motivations. Indeed, there are applications that require exact solutions of NP-hard problems, although this might only be possible for moderate input sizes. Decreasing the exponential running time, say, from $\mathrm{O}(2^n)$ to $\mathrm{O}(2^{0.9\,n})$, increases the size of the instances solvable within a given amount of time by a constant *multiplicative* factor. This kind of improvements can be crucial in several applications. On the other hand, the study of exact algorithms leads to a better understanding of NP-hard problems, and initiates new combinatorial and algorithmic challenges.

In this paper we consider one of the classical NP-hard problems, the Connected Dominating Set problem (CDS). A *connected dominating set* of a graph $G = (V, E)$ is a subset of nodes $S \subseteq V$ such that S is a dominating set of G and the subgraph of G induced by S is connected. The Connected Dominating Set problem asks to find a connected dominating set of smallest possible cardinality. This problem is also equivalent to finding a spanning tree with maximum number of leaves. Connected Dominating Set is a fundamental problem in connected facility location and studied intensively in computer science and operations research [16, 26]. Another recent application of this problem is in wireless ad-hoc networks: a small connected dominating set is often a convenient backbone to route the flow throughout the network (see e.g. [2]). The problem is NP-hard [13] and there is a $(\ln \Delta + O(1))$-approximation algorithm, where Δ is the maximum degree [15]. Such approximation guarantee cannot be improved unless $NP \subseteq DTIME(n^{O(\log \log n)})$ [15]. Despite its relevance in applications, the current best exact algorithm for Connected Dominating Set is the trivial $\Omega(2^n)$ enumerative algorithm, which tries all possible subsets of nodes. Better results are known for the general (unconnected) version of the problem [8, 12, 14, 22]: the current best algorithm for Dominating Set has running time $O(2^{0.598n})$ [8].

Though apparently closely related, Connected Dominating Set and Dominating Set are rather different from the point of view of exact algorithms. In particular, the techniques used to solve Dominating Set do not seem to work for Connected Dominating Set. One of the main reasons of this discrepancy is that *connectivity* is a global property: very often exact algorithms are based on the local structure of the problem; these algorithms seem not able to capture global properties such as connectivity.

Indeed, Connected Dominating Set belongs to a family of *non-local* problems which turns out to be particularly hard to solve exactly. Probably the best known example of this kind of problems is the Travelling Salesman Problem: find a minimum cost tour which visits all the nodes of a weighted graph. The fastest known algorithm for this problem, which dates back to the sixties [18], is based on dynamic programming and has running time $\Omega(2^n)$. Better results are known only for special graph classes, such as cubic graphs [6]. For many other non-local problems the current best known algorithms are still trivial. There are only a few exceptions to this. A relevant example is Steiner Tree: find a minimum size subtree of a given graph spanning a given subset of k nodes. For this problem an $O(1.4143^n)$ time algorithm can be obtained by combining the $O((2+\epsilon)^k n^{O(1)})$ dynamic-programming (exponential space) algorithm in [21] (for small k), with trivial $O(2^{n-k} n^{O(1)})$ enumeration of Steiner nodes (for large k). Finding a polynomial space algorithm faster than 2^n is still open. Another very recent example is a $O(1.9053^n)$ algorithm for Feedback Vertex Set: find a minimum cardinality subset of nodes of a graph whose removal makes the graph acyclic [23].

Our results. In this paper we make a further significant step in the design of faster exact algorithms for non-local problems, by presenting the first algorithm

for Connected Dominating Set which breaks the 2^n barrier: our recursive algorithm takes polynomial space and runs in time $O(1.9407^n)$. The algorithm is based on the simple strategy "stay connected", which means that all partial solutions generated recursively must be connected. Local choices are performed under this constraint. Our algorithm makes use of new domination rules, which were designed with the stay-connected framework in mind.

If analyzed in the standard way, our algorithm performs very poorly. The refined time bound is obtained with the Measure and Conquer approach described in [8]. The idea is to lower bound the progress made by the algorithm at each branching step according to non-standard measures of the size of the subproblems. However, the measure used in [8] for Dominating Set does not seem to work properly here. For this reason we designed a new, non-trivial measure: for every vertex v our measure reflects both the "need for domination" of v, and the ability of v "to dominate" the vertices that are not dominated yet. We remark that here Measure and Conquer is crucial to break the 2^n barrier. Moreover, we believe this approach is flexible enough to be applied to other non-local problems.

Measure and Conquer does not necessarily provide tight upper bounds for the worst case running time of recursive exponential algorithms, thus lower bounds are of great interest. As a second contribution of this paper, we establish a lower bound of $\Omega(4^{n/5})$ for the worst case running time of our algorithm.

Related Work. The design of exponential time algorithms has a long history dating back to Held and Karp's paper [18] on the travelling salesman problem in the early sixties. The last years have seen an emerging interest in constructing exponential time algorithms for combinatorial problems like Coloring [1, 4], Max-Cut [27], 3-SAT [3, 5], Dominating Set [8], Treewidth [11], and Independent Set [10]. There are two nice surveys of Woeginger [28, 29] describing the main techniques that have been established in the field. We also recommend the survey of Iwama [20] devoted to exponential algorithms for 3-SAT and the paper of Schöning [25] for its introduction to exponential time algorithms. In [9] we review some new techniques for the design and analysis of exponential-time algorithms, among which "Measure and Conquer" and "Lower Bounds".

One of the major techniques for constructing fast exponential time algorithms, which is also used in our CDS algorithm, is the *Branch and Reduce* paradigm. Roughly speaking, Branch and Reduce algorithms (also called search tree algorithms, Davis-Putnam-style exponential-time backtracking algorithms etc.) first apply some reduction rules, and then branch on two or more subproblems, which are solved recursively. Their running time analysis is based on a measure for the problem instance; reduction and branching rules lead to linear recurrences in the measure and their solution by standard methods provides upper bounds for the worst case running time. Recently, non-standard measures for problem instances have been used to improve the analysis of Branch and Reduce algorithms. This approach is called Measure and Conquer in [8]. The analysis of our algorithm for CDS is heavily based on this technique.

2 The Algorithm

Let $G = (V, E)$ be an n-node undirected and simple graph. The open *neighborhood* of a node v is denoted by $N(v) = \{u \in V : uv \in E\}$, and the closed neighborhood of v is denoted by $N[v] = N(v) \cup \{v\}$. The subgraph of G induced by a set $S \subseteq V$ is denoted by $G[S]$. A set $S \subseteq V$ of nodes of G is *connected*, if $G[S]$ is connected.

Without loss of generality, we can assume (i) that the graph is connected (otherwise there is no solution) and (ii) the minimum connected dominating set has cardinality at least two (otherwise the problem is trivially solvable in polynomial time). By the last assumption, we can consider the *total* variant of CDS, where each node v *dominates* its neighbors $N(v)$, but not the node v itself. This will turn out to be useful in the analysis.

Our recursive CDS algorithm is based on the following approach. Suppose we are given two subsets of nodes S (*selected* nodes), and D (*discarded* nodes), where $|S| \geq 2$ and $G[S]$ is connected. We will describe a recursive algorithm which finds an optimum solution OPT, if any, under the constraint that all the nodes in S and no node in D belong to OPT:

$$S \subseteq OPT \quad \text{and} \quad D \cap OPT = \emptyset.$$

In order to solve CDS it is sufficient to guess two adjacent nodes v' and v'' of some optimum solution, and run the algorithm above on the instance $(S, D) = (\{v', v''\}, \emptyset)$. So we run the algorithm $O(n^2)$ times.

Clearly, the instance is infeasible when $V \setminus D$ is not a connected dominating set. For notational convenience, we will sometimes allow S and D to overlap, and in that case we say that the instance is infeasible as well.

Before describing the algorithm, let us introduce some notation. The *available* nodes $A = V \setminus (S \cup D)$ are the nodes which are neither selected nor discarded. An available node v is a *candidate* if it is adjacent to S, and a *promise* if its removal makes the instance infeasible, i.e. $V \setminus (D \cup \{v\})$ is not a connected dominating set of G. Intuitively, a candidate is a node that might be added to S in the current step, while a promise is a node that must be added to S at some point (if the instance is feasible). We say that a node is *dominated* if it is adjacent to some node in S, and *free* otherwise. By F we denote the set of the free nodes

$$F = V \setminus \cup_{v \in S} N(v).$$

The algorithm halts if either the instance is infeasible or S is a (connected) dominating set. In the first case the algorithm returns *no*, while in the second one it returns $OPT = S$. Otherwise the algorithm performs some reductions on the problem instance, and then it branches on one or more subproblems, which are solved recursively. In each subproblem the algorithm adds available nodes to either S or D but always keeping S connected. The best solution of the subproblems, that is the one which minimizes the size $|OPT|$ of the solution returned, is the solution to the original problem.

The reduction rules are:

(a) If there is a candidate v which is a promise, select it (add it to S);
(b) If there are two candidates v and w (which by (a) are not promises) such that $N(v) \cap F \subseteq N(u) \cap F$, discard v (add it to D);
(c) If there is an available node v which does not dominate any free node, discard v.

The algorithm branches according to the following rules:

(A) If there is a candidate v which dominates at least three free nodes w_1,w_2 and w_3, or which dominates an available node w such that, after selecting v, w does not dominate any free node, branch on the two subproblems

- $(S_1, D_1) = (S \cup \{v\}, D)$;
- $(S_2, D_2) = (S, D \cup \{v\})$.

(B) If there is a candidate v which dominates a unique free node w, let

$$U = \{u_1, u_2, \ldots, u_k\} = N(w) \cap A \setminus N[v]$$

be the set of the available neighbors of w which are not in the closed neighborhood of v. Branch on the three subproblems:

- $(S_1, D_1) = (S, D \cup \{v\})$;
- $(S_2, D_2) = (S \cup \{v, w\}, D)$;
- $(S_3, D_3) = (S \cup \{v\}, D \cup \{w\} \cup U)$.

Observe that w might be discarded or a promise. Moreover one of the u_i's could be a promise. In those cases one or more subproblems are infeasible, and the algorithm simply halts on such infeasible subproblems. The same kind of situation may happen also in the following cases.

(C) If there is a candidate v which dominates two free nodes w_1 and w_2, name w_1 and w_2 such that if w_2 is available (a promise), so is w_1. Let

$$U_i = \{u_{i,1}, u_{i,2}, \ldots, u_{i,k_i}\} = N(w_i) \cap A \setminus N[v]$$

be the available neighbors of w_i which are not in the closed neighborhood of v. There are three different subcases:

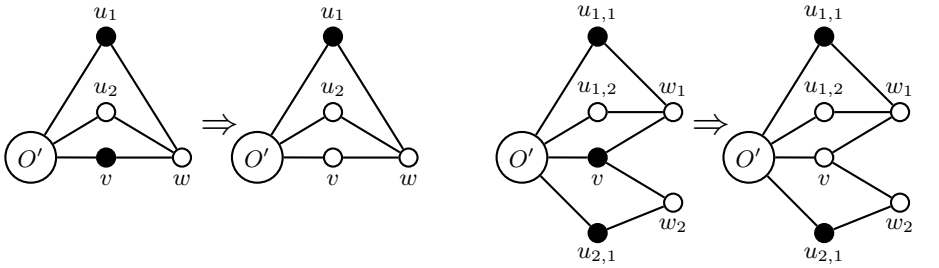
(C.1) If w_1 and w_2 are adjacent, w_1 is available and w_2 is discarded, branch on the three subproblems:

- $(S_1, D_1) = (S, D \cup \{v\})$;
- $(S_2, D_2) = (S \cup \{v, w_1\}, D)$;
- $(S_3, D_3) = (S \cup \{v\}, D \cup \{w_1\} \cup U_1)$.

(C.2) If w_1 and w_2 are adjacent and both available, branch on the four subproblems:

- $(S_1, D_1) = (S, D \cup \{v\})$;
- $(S_2, D_2) = (S \cup \{v, w_1\}, D)$;
- $(S_3, D_3) = (S \cup \{v, w_2\}, D \cup \{w_1\})$;
- $(S_4, D_4) = (S \cup \{v\}, D \cup \{w_1, w_2\} \cup U_1 \cup U_2)$.

Figure 1 Examples of cases (B) and (C.3). Black nodes are selected.



(C.3) Otherwise (either w_1 and w_2 are not adjacent, or they are adjacent and both discarded), branch on the five subproblems

- $(S_1, D_1) = (S, D \cup \{v\})$;
- $(S_2, D_2) = (S \cup \{v, w_1\}, D)$;
- $(S_3, D_3) = (S \cup \{v, w_2\}, D \cup \{w_1\})$;
- $(S_4, D_4) = (S \cup \{v\}, D \cup \{w_1, w_2\} \cup U_1)$;
- $(S_5, D_5) = (S \cup \{v\}, D \cup \{w_1, w_2\} \cup U_2)$.

Theorem 1. (correctness) *The algorithm above computes a minimum cardinality connected dominating set.*

Proof. The correctness of the halting rules is trivial.

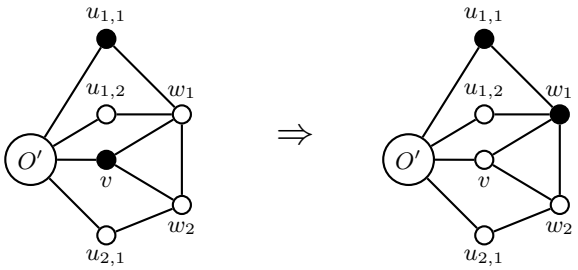
A reduction rule is *feasible* if it does not modify the value of the optimum. Reduction rule (a) is feasible since removing a candidate v which is a promise would lead to an infeasible instance. Reduction rule (b) is feasible since if $v \in OPT$, then $OPT' = OPT \cup \{w\} \setminus \{v\}$ is a feasible solution of cardinality at most $|OPT|$. Reduction (c) is feasible since all the available neighbors of v are already connected to S, and thus removing v from any feasible solution keeps the solution feasible.

Let us consider the branching rules. First observe that, as required, every set S_i induces a connected subgraph of the original graph. A branching rule is *feasible* if at least one subproblem preserves the value of the optimum solution. Branching rule (A) is trivially feasible: every connected dominating set either contains candidate v or does not. This simple fact is also used in the remaining branching rules.

Consider now branching rule (B). It is sufficient to show that if we select v and discard w, then we must also discard U. Assume by contradiction that $OPT = O' \cup \{v, u_i\}$ is an optimum solution of $(S', D') = (S \cup \{v\}, D \cup \{w\})$, where $u_i \in U$. Since w is discarded, $OPT' = O' \cup \{u_i\}$ is also connected. Moreover, since v dominates only w, and w is dominated by u_i as well, we have that OPT' is a dominating set (see Figure 1). Thus OPT' is a connected dominating set of size $|OPT| - 1$, which is a contradiction.

The feasibility of (C.3) follows by observing that if we select v and discard both w_1 and w_2, then we must also discard either U_1 or U_2 (or both). This can be proved by essentially the same argument as in case (B).

Figure 2 Example of cases (C.1) and (C.2). Here we are assuming that w_1 is available.



The remaining two cases are slightly more complicated. Consider first case (C.2). It is sufficient to show that, if we select v and discard both the w_i's, then we can also discard U_1 and U_2. By the same argument used in case (C.3), we already know that in the optimum solution OPT to $(S \cup \{v\}, D \cup \{w_1, w_2\})$ we must discard either U_1 or U_2. For sake of contradiction, suppose that $OPT = O' \cup \{v, u_{1,i}\}$ contains one $u_{1,i} \in U_1$ and no node in U_2 (a symmetric analysis holds if OPT contains one $u_{2,j} \in U_2$ and no node in U_1). Since w_1 and w_2 are adjacent, and w_1 is available, we have that by replacing v with w_1 in OPT, we obtain another feasible solution of the same cardinality (see Figure 2). Thus we do not need to consider this case because if OPT is the optimum solution to the original problem, the algorithm will find a solution of the same cardinality while solving subproblem $(S_1, D_1) = (S, D \cup \{v\})$.

Basically the same argument shows that in case (C.1), if we select v and we discard both w_1 and w_2, then we can also discard U_1. Hence the feasibility of (C.1). Note that, differently from case (*C*.2), we cannot use a symmetric argument to show that also U_2 can be discarded. This is because $w_2 \in D$, and thus the optimum solution to $(S_1, D_1) = (S, D \cup \{v\})$ cannot contain w_2. □

3 Analysis

Consider a given instance (S, D) of the problem (where the graph G is fixed). We will measure the size of the problem as described below. This measure will be used to bound the progress made by the algorithm at each branching step.

We associate two different weights to each node v of the graph. The first weight $\alpha(v) \geq 0$ is used to take into account the need for domination of v. In particular, if v is already dominated by S, $\alpha(v) = 0$. The second weight $\beta(v) \geq 0$ instead reflects the capability of v to dominate free nodes. For this reason, we assume $\beta(v) = 0$ if either $v \in S$ or $v \in D$.

Altogether the weight of the problem is

$$k = k(G, S, D) = k(S, D) = \sum_{v \in F} \alpha(v) + \sum_{v \in A} \beta(v). \tag{1}$$

In order to simplify the analysis, we make the following assumptions:

- for a given available node v, $\beta(v) = \beta \in (0, 1]$ if v is a promise and $\beta(v) = 1$ otherwise.
- $\alpha(v) = \alpha_{|v|}$ is a non-decreasing function of the *frequency* of v, denoted by $|v|$, that is the number of available nodes which dominate v (recall that v does not dominate itself). More precisely we assume

$$0 = \alpha_0 = \alpha_1 < \alpha_2 < \alpha_3 = \alpha_i, \quad \forall i \geq 4.$$

The reasons for the simplifying assumptions above will be clearer from the analysis. Note that the size of the original problem is upper bounded by $(1 + \alpha_3)n$.

For notational convenience, we define

$$\Delta\alpha_i = \alpha_i - \alpha_{i-1}, \quad i \geq 1.$$

Intuitively, $\Delta\alpha_i$ is the reduction of the size of the problem due to the reduction from i to $i - 1$ of the frequency of a free node.

Fact 1 *Observe that when we discard a candidate node v, we decrease the size of the problem (i) by $\beta(v)$, because v is not available any more, and (ii) by $\Delta\alpha_{|w|}$ for each free neighbor w of v, because of the decrease of the frequency of w. Moreover, the size could further decrease (iii) by $(1 - \beta)$, if some available node becomes a promise.*

On the other hand, when we select a node v, we decrease the size of the problem (i) by $\beta(v)$, because v is not available any more, and (ii) by $\alpha_{|w|}$ for each free neighbor w of v, because w is not free any more.

Fact 1 will be repeatedly applied in the proof of the following theorem.

Theorem 2. (running time) *The running time of the CDS algorithm of Section 2 is $O(1.9407^n)$.*

Proof. Let $P(k)$ be the number of subproblems generated to solve an instance of size $k = k(S, D)$, where k is defined as in (1). For notational convenience we assume $P(k) = 1$ for $k \leq 0$.

Of course, if the algorithm halts, $P(k) = 1$. Now observe that when we apply the reduction rules (a)-(c), the size of the problem decreases by at least β:

$$P(k) \leq 1 + P(k - \beta). \tag{2}$$

Consider now the case when the algorithm branches. Note that, by (a), the candidate v selected is not a promise (and thus $\beta(v) = 1$). Following the algorithm, we distinguish different subcases:

(A) Suppose v dominates three free nodes w_1, w_2, and w_3 (and possibly more). By Fact 1,

$$\begin{aligned} P(k) \leq 1 &+ P(k - 1 - \Delta\alpha_{|w_1|} - \Delta\alpha_{|w_2|} - \Delta\alpha_{|w_3|}) \\ &+ P(k - 1 - \alpha_{|w_1|} - \alpha_{|w_2|} - \alpha_{|w_3|}). \end{aligned} \tag{3}$$

Now suppose v dominates an available node w, such that $N(w) \cap F \setminus N(v) = \emptyset$. Then when we select v, w is discarded by (c). Note that w cannot be a promise ($\beta(w) = 1$). By Fact 1 and the observation above,

$$P(k) \leq 1 + P(k - 1 - \Delta\alpha_{|w|}) + P(k - 1 - \alpha_{|w|} - 1). \tag{4}$$

From this point on we can assume that every available node w adjacent to the candidate v dominates at least one free node z not in $N(v)$. In the following we denote such free nodes by

$$Z(w) = N(w) \cap F \setminus N(v).$$

(B) Recall that v dominates a unique free node w, and U is the set of available neighbors of w, excluding $N[v]$. By Fact 1,

$$\begin{aligned} P(k) \leq 1 &+ P(k - 1 - \Delta\alpha_{|w|}) \\ &+ P(k - 1 - \alpha_{|w|} - \beta(w) - \sum_{z \in Z(w)} \delta_{w \in A} \cdot \alpha(z)) \\ &+ P(k - 1 - \alpha_{|w|} - \beta(w) - \sum_{z \in Z(w)} \delta_{w \in A} \cdot \Delta\alpha(z) - \sum_{u \in U} \beta(u)), \end{aligned} \tag{5}$$

where $\delta_{\mathcal{P}} = 1$ if predicate $\mathcal{P}$ is true, and $\delta_{\mathcal{P}} = 0$ otherwise.

Since v is not a promise, we have that $|U| = |w| - 1 \geq 1$. Moreover, by case (A), if w is available, it must dominate at least one free node z ($|Z(w)| \geq 1$). If w is not a promise, such a neighbor z must have frequency at least two.

It is worth to mention that there might be subproblems which are infeasible because we either select nodes which are discarded, or we discard nodes which are promises. In those cases we can replace the corresponding $P(k')$ with 1 in the recurrences above, since the algorithms halts on such subproblems. The same holds also in next cases.

(C) Recall that v dominates two free nodes w_1 and w_2, where U_i are the available neighbors of w_i, excluding $N[v]$. Moreover the w_i's are named such that, if w_2 is available (a promise), so is w_1. In particular this implies that, if w_1 is discarded, the same holds for w_2.

(C.1) In this case w_1 and w_2 are adjacent, w_1 is available and w_2 is discarded. Observe that, if $|w_1| = 2$, which implies $U_1 = \{u_{1,1}\}$, and $u_{1,1}$ is not a promise, then $u_{1,1}$ becomes a promise when we remove v. By this observation and Fact 1,

$$\begin{aligned} P(k) \leq 1 &+ P(k - 1 - \Delta\alpha_{|w_1|} - \Delta\alpha_{|w_2|} - \delta_{|w_1|=2}(\beta(u_{1,1}) - \beta)) \\ &+ P(k - 1 - \alpha_{|w_1|} - \alpha_{|w_2|} - \beta(w_1) - \sum_{z \in Z(w_1)} \delta_{w_1 \in A} \cdot \alpha_{|z|}) \\ &+ P(k - 1 - \alpha_{|w_1|} - \alpha_{|w_2|} - \beta(w_1) - \sum_{z \in Z(w_1)} \delta_{w_1 \in A} \cdot \Delta\alpha_{|z|} - \sum_{u \in U_1} \beta(u)). \end{aligned} \tag{6}$$

Note that $|U_1| = |w_1| - 1 \geq 1$.

(C.2) In this case w_1 and w_2 are adjacent and both available. Observe that, if $|w_1| = 2$ ($|w_1| = 2$) and w_2 (w_1) is not a promise, then w_2 (w_1) becomes a promise when we remove v. By this observation and Fact 1,

$$\begin{aligned}
P(k) \leq 1 &+ P(k-1-\sum_{i=1}^{2} \Delta\alpha_{|w_i|} - \sum_{i=1}^{2} \delta_{|w_i|=2} \cdot (\beta(w_i) - \beta)) \\
&+ P(k-1-\sum_{i=1}^{2} \alpha_{|w_i|} - \beta(w_1) - \sum_{z \in Z(w_1)} \delta_{w_1 \in A} \cdot \alpha(z)) \\
&+ P(k-1-\sum_{i=1}^{2} \alpha_{|w_i|} - \sum_{i=1}^{2} \beta(w_i) - \sum_{z \in Z(w_1)} \delta_{w_1 \in A} \cdot \Delta\alpha(z)) \\
&+ P(k-1-\sum_{i=1}^{2} \alpha_{|w_i|} - \sum_{i=1}^{2} \beta(w_i) - \sum_{z \in Z(w_1)} \delta_{w_1 \in A} \cdot \Delta\alpha(z) - \sum_{u \in U_1 \cup U_2} \beta(u)).
\end{aligned} \tag{7}$$

Note that it cannot be $|w_1| = |w_2| = 2$ since otherwise v would be a promise. Moreover $|U_1 \cup U_2| \geq \max\{|U_1|, |U_2|\} \geq \max\{|w_1| - 2, |w_2| - 2\}$.

(C.3) Recall that if w_1 and w_2 are adjacent, they are both discarded. In any case, $|U_1| = |w_1| - 1 \geq 1$ and $|U_2| = |w_2| - 1 \geq 1$. If $|w_1| = 2$, which implies $U_1 = \{u_{1,1}\}$, and $u_{1,1}$ is not a promise, $u_{1,1}$ becomes a promise when we remove v. A symmetric argument holds for w_2. By this observation and Fact 1,

$$\begin{aligned}
P(k) \leq 1 &+ P(k-1-\sum_{i=1}^{2} \Delta\alpha_{|w_i|} - \delta_{|w_1|=2 \text{ or } |w_2|=2} \max_h\{\beta(u_{h,1}) - \beta\}) \\
&+ P(k-1-\sum_{i=1}^{2} \alpha_{|w_i|} - \beta(w_1) - \sum_{z \in Z(w_1)} \delta_{w_1 \in A} \cdot \alpha_{|z|}) \\
&+ P(k-1-\sum_{i=1}^{2} \alpha_{|w_i|} - \sum_{i=1}^{2} \beta(w_i) - \sum_{z \in Z(w_1)} \delta_{w_1 \in A} \cdot \Delta\alpha_{|z|}) \\
&+ P(k-1-\sum_{i=1}^{2} \alpha_{|w_i|} - \sum_{i=1}^{2} \beta(w_i) - \sum_{z \in Z(w_1)} \delta_{w_1 \in A} \cdot \Delta\alpha_{|z|} - \sum_{u \in U_1} \beta(u)) \\
&+ P(k-1-\sum_{i=1}^{2} \alpha_{|w_i|} - \sum_{i=1}^{2} \beta(w_i) - \sum_{z \in Z(w_1)} \delta_{w_1 \in A} \cdot \Delta\alpha_{|z|} - \sum_{u \in U_2} \beta(u)).
\end{aligned} \tag{8}$$

Observe that, if w_1 is available (and thus w_1 and w_2 are not adjacent), by (A) w_1 must dominate at least one free node z: $|Z(w_1)| \geq 1$.

From recurrences (2)-(8), $P(k) \leq c^k \leq c^{(1+\alpha_3)n}$, where $c = c(\beta, \alpha_2, \alpha_3)$ is a quasi-convex function of the weights [7]. Thus the estimation of the running time reduces to choosing the weights minimizing $c^{1+\alpha_3}$. Note that it is sufficient to

consider only a finite number of recurrences. This is because each recurrence R where the frequency of some node considered is larger than 5 is *dominated* by a recurrence R' where the same element has frequency 4, that is the upper bound on c given by R is not larger than the one given by R'. We numerically obtained $\beta = 0.5004$, $\alpha_2 = 0.0600$, and $\alpha_3 = 0.1215$, and thus the claimed running time $O(1.9407^n)$. □

4 An Exponential Lower Bound

Since the known tools to analyze the worst case running time of Branch and Reduce algorithms (including Measure and Conquer) do not provide tight upper bounds, it is natural to ask for lower bounds: A lower bound may give an idea of how far is the established upper bound from the real worst case running time.

Theorem 3. (lower bound) *The worst case running time of the CDS algorithm of Section 2 is* $\Omega(4^{n/5}) = \Omega(1.3195^n)$.

The proof of Theorem 3 is omitted here for lack of space.

References

1. R. Beigel and D. Eppstein. 3-coloring in time $O(1.3289^n)$. *Journal of Algorithms* 54:168–204, 2005.
2. J. Blum, M. Ding, A. Thaeler, and X. Cheng. Connected dominating set in sensor networks and MANETs. In *Handbook of combinatorial optimization. Supplement Vol. B*, pages 329–369. Springer, New York, 2005.
3. T. Brueggemann and W. Kern. An improved deterministic local search algorithm for 3-SAT. *Theoretical Computer Science* 329:303–313, 2004.
4. J. M. Byskov. Enumerating maximal independent sets with applications to graph colouring. *Operations Research Letters* 32:547–556, 2004.
5. E. Dantsin, A. Goerdt, E. A. Hirsch, R. Kannan, J. Kleinberg, C. Papadimitriou, P. Raghavan, and U. Schöning. A deterministic $(2-2/(k+1))^n$ algorithm for k-SAT based on local search. *Theoretical Computer Science* 289:69–83, 2002.
6. D. Eppstein. The traveling salesman problem for cubic graphs. In *Workshop on Algorithms and Data Structures (WADS)*, pages 307–318, 2003.
7. D. Eppstein. Quasiconvex analysis of backtracking algorithms. Procedings of the *ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pp. 781–790, 2004.
8. F. V. Fomin, F. Grandoni, and D. Kratsch. Measure and Conquer: Domination - A Case Study, Proceedings of the *32nd International Colloquium on Automata, Languages and Programming (ICALP 2005)*, Springer LNCS vol. 3580, 2005, pp. 191–203.
9. F. V. Fomin, F. Grandoni, D. Kratsch. Some new techniques in design and analysis of exact (exponential) algorithms. *Bulletin of the EATCS* 87:47–77, 2005.
10. F. V. Fomin, F. Grandoni, D. Kratsch. Measure and Conquer: A simple $O(2^{0.288\,n})$ independent set algorithm. Procedings of the *17th ACM-SIAM Symposium on Discrete Algorithms (SODA)*, 2006, pp. 18–25.

11. F. V. Fomin, D. Kratsch, and I. Todinca. Exact algorithms for treewidth and minimum fill-in. Proceedings of the *31st International Colloquium on Automata, Languages and Programming (ICALP 2004)*, Springer LNCS vol. 3142, 2004, pp. 568–580.
12. F. V. Fomin, D. Kratsch, and G. J. Woeginger. Exact (exponential) algorithms for the dominating set problem. Proceedings of the *30th Workshop on Graph Theoretic Concepts in Computer Science (WG 2004)*, Springer LNCS vol. 3353, 2004, pp. 245-256.
13. M. R. Garey and D. S. Johnson. *Computers and Intractability. A Guide to the Theory of NP-Completeness.* Freemann, 1979.
14. F. Grandoni. A note on the complexity of minimum dominating set. *Journal of Discrete Algorithms*, 4(2):209–214, 2006.
15. S. Guha and S. Khuller. Approximation algorithms for connected dominating sets. *Algorithmica*, 20(4):374–387, 1998.
16. A. Gupta, A. Kumar, and T. Roughgarden. Simpler and better approximation algorithms for network design. Proceedings of the *35th Annual ACM Symposium on Theory of Computing (STOC 2003)*, pages 365–372, New York, 2003. ACM.
17. J. Håstad. Clique is hard to approximate within $n^{1-\epsilon}$. *Acta Math.* 182 (1):105–142, 1999.
18. M. Held and R.M. Karp. A dynamic programming approach to sequencing problems. *Journal of SIAM*, pages 196–210, 1962.
19. R. Impagliazzo, R. Paturi, and F. Zane. Which problems have strongly exponential complexity. *Journal of Computer and System Sciences* 63:512–530, 2001.
20. K. Iwama. Worst-case upper bounds for k-SAT. *Bulletin of the EATCS* 82:61–71, 2004.
21. D. Mölle, S. Richter, and P. Rossmanith. A faster algorithm for the steiner tree problem. Proceedings of the *23d Symposium on Theoretical Aspects of Computer Science (STACS 2006)*, Springer LNCS vol. 3884, 2006, pp. 561-570.
22. B. Randerath and I. Schiermeyer. Exact algorithms for MINIMUM DOMINATING SET. Technical Report, zaik-469, Zentrum für Angewandte Informatik Köln, April 2004.
23. I. Razgon. Exact computation of maximum induced forest. *Proceedings of the 10th Scandinavian Workshop on Algorithm Theory (SWAT 2006).* To appear.
24. J. M. Robson. Algorithms for maximum independent sets. *Journal of Algorithms* 7(3):425–440, 1986.
25. U. Schöning. Algorithmics in exponential time. Proceedings of the *22nd International Symposium on Theoretical Aspects of Computer Science (STACS 2005)*, Springer LNCS vol. 3404, 2005, pp. 36–43.
26. C. Swamy and A. Kumar. Primal-dual algorithms for connected facility location problems. *Algorithmica* 40(4):245–269, 2004.
27. R. Williams. A new algorithm for optimal constraint satisfaction and its implications. Proceedings of the *31st International Colloquium on Automata, Languages and Programming (ICALP 2004)*, Springer LNCS vol. 3142, 2004, pp. 1227–1237.
28. G. J. Woeginger. Exact algorithms for NP-hard problems: A survey. *Combinatorial Optimization – Eureka, You Shrink*, Springer LNCS vol. 2570, 2003, pp. 185–207.
29. G. J. Woeginger. Space and time complexity of exact algorithms: Some open problems. Proceedings of the *1st International Workshop on Parameterized and Exact Computation (IWPEC 2004)*, Springer LNCS vol. 3162, 2004, pp. 281–290.