A_009

An Improved Upper Bound for the Three Domatic Number Problem

M. M. Halldórsson *, O. Watanabe †, and M. Yamamoto ‡

1 Introduction

In this paper we consider the following problem and show an improved time bound for the existing algorithm for this problem.

Three Domatic Number Problem
Instance: An undirected graph $G = (V, E)$.
Question: Is there any partition (V_1, V_2, V_3) of V such that V_1, V_2, and V_3 are all dominating sets?

Tiege, etal [2] recently proposed a deterministic algorithm solving this problem, and they proved that it runs in $\widetilde{O}(2.6949^n)$-time for any graph with n vertices. Here we give a better bound for this algorithm. In the following, we call this algorithm *TRSY-algorithm*.

We briefly review TRSY-algorithm. Given a graph $G = (V, E)$ with n vertices, the algorithm first enumerates all minimal dominating sets for G with size at most $n/3$, which can be done in time $\widetilde{O}(1.7697^n)$ as shown in [1]. Secondly, for each obtained dominating set, it checks whether there are two other dominating sets for G. This can be regarded as the (general) satisfiability problem: Fix a dominating set D obtained by the enumeration. Let $\mathrm{N}[v]$ be a set of neighbors of v. For each $v \in V \backslash D$, generate a clause $C_v = \{v\} \cup (\mathrm{N}[v] \backslash D)$ as well as a clause $C_{\bar{v}} = \{\bar{v}\} \cup \{\bar{u} : u \in \mathrm{N}[v] \backslash D\}$; on the other hand, for each $v \in D$, generate a clause $C_v = \mathrm{N}[v] \backslash D$ as well as a clause $C_{\bar{v}} = \{\bar{u} : u \in \mathrm{N}[v] \backslash D\}$. Let F be a conjunction of all these $2n$ clauses. Then it is easy to see that a satisfying assignment of F determines two other dominating sets of G. TRSY-algorithm uses Yamamoto's algorithm [3] for searching this satisfying assignment, which is the currently fastest algorithm for the general CNF satisfiability problem. Since Yamamoto's algorithm runs in $\widetilde{O}(1.2335^m)$-time for a given CNF formula with m clauses, this part of TRSY-algorithm runs in $\widetilde{O}(1.2335^{2n})$ time; thus, the total running time is $\widetilde{O}(1.7697^n * 1.2335^{2n}) = \widetilde{O}(2.6949^n)$.

We further review Yamamoto's algorithm, which we call *Y-algorithm* in the following. A literal is called a *(i, j)-literal* if the literal positively occurs i times and negatively occurs j times. We similarly define a *(i^α, j^β)-literal* for $\alpha, \beta \in \{+, -\}$. We call a formula a *$(3, 3)$-formula* if it has no literal other than $(3, 3)$-literals. A pair of literals l and l' is *coincident* if there are two different clauses such that each clause contains l and l', and a formula is called *coincident-free* if it contains no coincident pair. It is shown [3] that a coincident-free $(3, 3)$-formula is the worst-case instance for Y-algorithm.

Now the crucial point for the argument in this paper is that so long as a formula has either some non $(3, 3)$-literal, or some coincident literal pairs, then Y-algorithm performs better. More specifically, for a given formula with m clauses, if Y-algorithm, on any computation path, can always eliminate m' clauses before reaching to some coincident-free $(3, 3)$-formula, then the algorithm runs in $\widetilde{O}(1.221^{m'} \times 1.234^{m-m'})$-time, which is better than its worst-case time bound $\widetilde{O}(1.234^m)$. In this paper, we make use of this point to show a better bound for TRSY-algorithm.

In what follows, we call a pair of clauses C_v and $C_{\bar{v}}$ for $v \in V \backslash D$ a *base clause pair*, and call the variable v its *base variable*. Note that the number n' of base clause pairs is at least $2n/3$ since the algorithm enumerates dominating sets of size at most $n/3$.

2 Bound

At each step of Y-algorithm, the algorithm selects one variable (by a certain rule), assigns true and false to the variable, creating two slightly simpler formulas, and searches for a satisfying assignment recursively on these two formulas. Clauses already satisfied by a current partial assignment are eliminated, and the goal of Y-algorithm is to obtain a null formula by eliminating all clauses. We estimate the number of clauses (in the worst-case) that are eliminated until the current formula becomes coincident-free $(3, 3)$-formula.

Y-algorithm uses "resolution" for obtaining a simpler formula, which produces a formula whose structure may be quite different from the original formula. Here we keep a table $\mathcal{T}$ of $2n$ clauses that are given originally, and analyze how these clauses are eliminated during the execution of the algorithm. When the algorithm assigns a value to some variable explicitly (which we call an *explicit assignment*), we can simply assign the value to the variable in the clauses of $\mathcal{T}$, and cross out (i.e., eliminate) those clauses that are satisfied by this assignment. On the other hand, when a resolution is conducted in Y-algorithm, even if it reduces the number of clauses from the formula that Y-algorithm maintains, we just keep its record and do not make any change on the table. More specifically, we keep a record $(x, \{C_1, ..., C_u\}, \{D_1, ..., D_v\})$, where $C_1, ..., C_u$ are clauses containing a variable x, and $D_1, ..., D_v$ are

*Dept. of Computer Science, Faculty of Engineering, University of Iceland, IS-107, Reykjavik, Iceland, mmh@hi.is
†Dept. of Mathematical and Computing Sciences, Tokyo Institute of Technology, Tokyo, Japan, watanabe@is.titech.ac.jp
‡Graduate School of Informatics, Kyoto University, Kyoto, Japan, masaki.yamamoto@kuis.kyoto-u.ac.jp

those containing its negation $\bar{x}$, and the resolution is conducted between $C_1, ..., C_u$ and $D_1, ..., D_v$ w.r.t. x. We call this x a *pivot variable*. Though the resolution itself does not invoke any clause elimination on $\mathcal{T}$, some clauses may be crossed out *later* due to the resolution. More specifically, the following two eliminations are possible: (1) At some point, if all $C_1, ..., C_u$ (resp., all $D_1, ..., D_v$) are crossed out in the table $\mathcal{T}$, then set $x = 0$ (resp., $x = 1$), thereby crossing out all $D_1, ..., D_v$ (resp., $C_1, ..., C_u$). (2) At some point, if all $C_1, ..., C_u$ but one $C_i = (x \vee \ell_1 \vee \cdots \ell_p)$ (resp., all $D_1, ..., D_v$ but one $D_j = \bar{x} \vee \ell'_1 \vee \cdots \ell'_q$) are crossed out in the table $\mathcal{T}$, then set $x = \neg(\ell_1 \vee \cdots \vee \ell_p)$ (resp., $\bar{x} = \neg(\ell'_1 \vee \cdots \vee \ell'_q)$), thereby crossing out C_i because it now becomes a tautology. Note that at the same time, all occurrences of $\bar{x}$ in $D_1, ..., D_v$ (resp., all occurrences of x in $C_1, ..., C_u$) are replaced with $\ell_1 \vee \cdots \vee \ell_p$ (resp., $\ell'_1 \vee \cdots \vee \ell'_q$). These assignments are called *symbolic assignments*. Clauses that become tautology by such symbolic assignments are also crossed out from $\mathcal{T}$.

For an initial formula F, let F' be a formula that Y-algorithm obtains after making several explicit assignments; that is, F' is a formula that the algorithm yields from F on a path of the search tree. Following this execution, the original table $\mathcal{T}$ is changed to a table $\mathcal{T}'$. We say that a clause (resp., a literal or a variable) is *alive in* $\mathcal{T}'$ if it is (resp., it is in a clause) not crossed out in $\mathcal{T}'$. Between this table $\mathcal{T}'$ and the formula F' that the algorithm maintains, we can show the following relationship by induction.

Proposition 1 *For any alive clause C in $\mathcal{T}'$, there exists a clause C' in F' such that C' contains all literals of C except for pivot variables.*

Corollary 1 *(1) For any alive literal except for pivot literals, the number of alive occurrences of the literal in $\mathcal{T}'$ is less than or equal to the one in F'. (2) For any alive literals x and y except for pivot literals, if they occur in the same alive clause of $\mathcal{T}'$, then they occur simultaneously in some clause of F'.*

We estimate a lower bound of the number of clauses eliminated from F (on any computation path) until Y-algorithm encounters a coincident-free $(3,3)$-formula. By the corollary above, it suffices to show such a lower bound for the table $\mathcal{T}'$. That is, we first fix any computation path such that $\mathcal{T}'$ becomes coincident-free and contains only $(3,3)$-literals, and we estimate the number m'' of clauses crossed out in $\mathcal{T}'$. Here we only focus on base clauses, i.e., clauses C_v and $C_{\bar{v}}$ for v not in the first dominating set. Recall that the number n' of base clause pairs satisfies $n' \geq 2n/3$.

We classify n' pairs of base clauses into several types based on the status in $\mathcal{T}'$. First let $\mathcal{C}_{\mathrm{EA}}$ be the set of clause pairs whose base variable is explicitly assigned true or false values. On the other hand, let $\mathcal{C}_{\mathrm{RA}}$ be the set of clause pairs whose base variable is assigned some value (true, false, or symbolic one) due to resolutions. Clause pairs not in $\mathcal{C}_{\mathrm{EA}} \cup \mathcal{C}_{\mathrm{RA}}$ are those with an unassigned base variable, which are classified into the following four types: (1) a clause pair C_v and $C_{\bar{v}}$ such that both are already crossed out, (2) a clause pair such that C_v is alive while its partner $C_{\bar{v}}$ is already crossed out, (3) a clause pair such that $C_{\bar{v}}$ is alive while C_v is already crossed out, and (4) a clause pair C_v and $C_{\bar{v}}$ that are both alive. Let n_1, n_2, n_3, and n_4 respectively denotes the number of clause pairs of each type.

Then by using the assumption that $\mathcal{T}'$ is coincident-free and contains only $(3,3)$-literals, it is easy to derive the following relations.

Proposition 2 *Every clause of type (4) clause pair consists of its base variable and base literals from type (1) clauses. Furthermore, we have $n_4 \leq 3n_1$.*

Now we estimate a lower bound for m''. Noting $n' = n_1 + n_2 + n_3 + n_4 + t + s$ and $n_4 \geq 3n_1$, the following is immediate by definition.

$$\begin{aligned} m'' &\geq 2n_1 + n_2 + n_3 + t + s \geq 2n_1 \\ &= n' + n_1 - n_4 \geq n' - 2n_1. \end{aligned}$$

Then the worst-case is the case $n' - 2n_1 = 2n_1$, which, by using a lower bound $n' \geq 2n/3$, derives $n_1 = n/6$, implying $m'' \geq n/3$. Therefore, as explained in Introduction, the total running time is

$$\widetilde{O}(1.2208^{n/3} \times 1.2335^{2n-n/3}) = \widetilde{O}(1.5163^n)$$

Theorem 1 *The three domatic number problem can be solved in $\widetilde{O}(1.7697^n \times 1.5163^n) = \widetilde{O}(2.6834^n)$.*

This slightly beats the existing bound $\widetilde{O}(2.6949^n)$ [2].

References

[1] F. Fomin, F. Grandoni, A. Pyatkin, and A. Stepanov, Bounding the number of minimal dominating sets: A measure and conquer approach, In Proceedings of the 16th International Symposium on Algorithms and Computation, pp. 573-582, 2005.

[2] T. Tiege, J. Rothe, H. Spakowski, and M. Yamamoto, An improved exact algorithm for the domatic number problem, In Proceedings of the Second IEEE International Conference on Information & Communication Technologies: From Theory to Applications, pp. 1021-1022, 2006.

[3] M. Yamamoto, An improved $\widetilde{O}(1.234^m)$-time deterministic algorithm for SAT, In Proceedings of the 16th International Symposium on Algorithms and Computation, pp. 644-653, 2005.