RAIRO-Theor. Inf. Appl. 44 (2010) 363–377
DOI: 10.1051/ita/2010018

Available online at:
www.rairo-ita.org

THE UNIFORM MINIMUM-ONES 2SAT PROBLEM AND ITS APPLICATION TO HAPLOTYPE CLASSIFICATION*

Hans-Joachim Böckenhauer[1], Michal Forišek[2], Ján Oravec[1], Björn Steffen[1], Kathleen Steinhöfel[3] and Monika Steinová[1]

Abstract. Analyzing genomic data for finding those gene variations which are responsible for hereditary diseases is one of the great challenges in modern bioinformatics. In many living beings (including the human), every gene is present in two copies, inherited from the two parents, the so-called *haplotypes*. In this paper, we propose a simple combinatorial model for classifying the set of haplotypes in a population according to their responsibility for a certain genetic disease. This model is based on the minimum-ones 2SAT problem with uniform clauses. The minimum-ones 2SAT problem asks for a satisfying assignment to a satisfiable formula in 2CNF which sets a minimum number of variables to true. This problem is well-known to be $\mathcal{NP}$-hard, even in the case where all clauses are uniform, *i.e.*, do not contain a positive and a negative literal. We analyze the approximability and present the first non-trivial exact algorithm for the uniform minimum-ones 2SAT problem with a running time of $\mathcal{O}(1.21061^n)$ on a 2SAT formula with n variables. We also show that the problem is fixed-parameter tractable by showing that our algorithm can be adapted to verify in $\mathcal{O}^*(2^k)$ time whether an assignment with at most k true variables exists.

Mathematics Subject Classification. 68Q25, 68R10, 92D20.

Keywords and phrases. Exact algorithms, fixed-parameter algorithms, minimum-ones 2SAT, haplotypes.

* *This work was partially supported by SNF grant 200021-109252/1.*

[1] Department of Computer Science, ETH Zurich, Switzerland; {hjb, oravecj, bjoern.steffen, monika.steinova}@inf.ethz.ch

[2] Department of Computer Science, Comenius University Bratislava, Slovakia; forisek@dcs.fmph.uniba.sk

[3] Department of Computer Science, King's College London, UK; kathleen.steinhofel@kcl.ac.uk

Article published by EDP Sciences

1. Introduction

With the enhancement of sequencing methods, more and more genetic data becomes available for research. One of the grand challenges in molecular biology is to interpret this genomic data and to make use of it, for example to detect genetic diseases. In this paper, we investigate a challenging combinatorial problem which arises in this context.

Humans and many other living organisms (including all vertebrates) have a diploid genome, *i.e.*, they have two copies for each chromosome. For an individual organism, these two copies usually differ from each other as a result of their inheritance from two separate parent entities. The most common differences are the so-called single nucleotide polymorphisms (SNPs), where a single nucleotide is replaced by another one. For a given chromosome, the complete sequence information of one copy is called a *haplotype*.

For the interpretation of such haplotype data, we consider the following classification problem: assume that the haplotype data of a population is given, together with some phenotypical data describing whether the individuals have symptoms of a specific genetically influenced disease or not. Our goal is to find a plausible subset of bad haplotypes occurring in the population which are responsible for this disease. We employ a very simple model by assuming that every ill individual carries at least one bad haplotype and that every healthy individual carries at least one good haplotype.

This problem can be formally described as the following variant of a satisfiability problem: we are given a satisfiable formula in 2CNF and the goal is to compute a satisfying assignment which minimizes the number of variables set to true. This problem is known as the *minimum-ones 2SAT problem*. It was first considered by Gusfield and Pitt [6] who presented a 2-approximation algorithm for it. Although it is possible to decide the satisfiability of a 2CNF formula in linear time [1], the minimum-ones 2SAT problem is $\mathcal{NP}$-hard and even $\mathcal{APX}$-hard since it is a generalization of the vertex cover problem.

In this paper, we concentrate on the *uniform* variant of the minimum-ones 2SAT problem, where the input formula does not contain any clause consisting of a negated and an unnegated variable. This problem variant exactly models our haplotype classification problem. This variant is hard as well, so we analyze its approximability and design an exact algorithm with an exponential running time of $\mathcal{O}(1.21061^n)$.

Additionally, we show that this uniform minimum-ones 2SAT problem is fixed-parameter tractable by showing that our algorithm can be adapted to verify in $\mathcal{O}^*(2^k)$ time whether an assignment with at most k true variables exists.

The paper is organized as follows. In Section 2, we fix our notation and give a formal definition of the minimum-ones 2SAT problem. In Section 3, we describe how to apply our result to the haplotype classification problem. Section 4 is devoted to the approximability of the minimum-ones 2SAT problem. In Section 5, we present our main result, namely, a fast exact algorithm for the uniform case of the minimum-ones 2SAT problem. We conclude the paper in Section 6.

2. Preliminaries

In this section, we define the basic notions we will be using throughout the paper. We consider Boolean formulas over a set of variables $X = \{x_1, \ldots, x_n\}$. A *literal* is either some variable $x \in X$ or its negation $\overline{x}$. A *clause* is a disjunction of literals, *i.e.*, a formula of the form $(y_1 \vee y_2 \vee \cdots \vee y_k)$ where the y_i are literals over X. A Boolean formula is said to be in *conjunctive normal form (CNF)* if it is a conjunction of disjunctions of literals, *i.e.*, a formula $C_1 \wedge \ldots \wedge C_m$ for some clauses $C_1, \ldots, C_m$. We say that a CNF formula is in *2CNF* if every clause contains exactly 2 literals. A 2CNF clause is called *positive* if both literals in it are unnegated variables; it is called *negative* if both literals are negated variables; *mixed*, if it contains one negated and one unnegated variable.

An *assignment* for a formula φ over X is a function $\beta\colon X \to \{0,1\}$ assigning a truth value to every variable. An assignment β *satisfies* a formula φ, if the standard Boolean evaluation of φ using the assignment β yields the value 1. A formula is *satisfiable*, if there exists a satisfying assignment for it.

Definition 2.1. Min-Ones-2SAT is the following optimization problem:

Input: A satisfiable 2CNF formula $\varphi = C_1 \wedge C_2 \wedge \cdots \wedge C_m$ over the set of variables $X = \{x_1, \ldots, x_n\}$.
Feasible solutions: all satisfying assignments for φ.
Costs: The cost of a satisfying assignment β is the number of variables set to 1 by β.
Goal: minimization.

Uniform Min-Ones-2SAT is the restriction of Min-Ones-2SAT to input instances without mixed clauses, and Positive Min-Ones-2SAT is the restriction of Uniform Min-Ones-2SAT to inputs containing only positive clauses.

For our algorithms, we are going to use a graph representation for the Uniform Min-Ones-2SAT where the vertices of the graph correspond to the variables and the edges correspond to the clauses.

Definition 2.2. For a formula φ in 2CNF without mixed clauses, we define the *satisfiability graph* $G_\varphi = (X, E_+ \cup E_-)$, where $E_+ = \{\{x, y\} \mid (x \vee y)$ is a positive clause in $\varphi\}$ and $E_- = \{\{x, y\} \mid (\overline{x} \vee \overline{y})$ is a negative clause in $\varphi\}$. We call the edges from E_+ *positive edges* and those from E_- *negative edges*. By G_φ^+ and G_φ^-, we denote the graph induced by the edges E_+ and E_-, respectively.

Note that the satisfiability graph is actually a multigraph, since two vertices may be connected by both a positive and a negative edge. Recall that a *vertex cover* of a graph $G = (V, E)$ is a set $C \subseteq V$ of vertices such that every edge is incident to at least one vertex from C. An *independent set* of a graph $G = (V, E)$ is a set $I \subseteq V$ of vertices that are pairwise non-adjacent. The *minimum vertex cover problem* (Min-VC) is the problem of finding a vertex cover of minimum cardinality in a given graph, whereas the *maximum independent set problem* is the problem of finding an independent set of maximum cardinality in a given graph.

For the ease of presentation, we use the short notation xy for an edge $\{x, y\}$ in any graph.

Observation 2.3. Positive Min-Ones-2SAT is equivalent to Min-VC.

Note that in every positive clause at least one variable has to be set to true and in every negative clause at most one variable may be set to true in order to satisfy the formula. Considering our graph representation, this leads to the following observation.

Observation 2.4. Uniform Min-Ones-2SAT is equivalent to the problem of finding a minimum vertex cover in G_φ^+ that is an independent set in G_φ^-.

3. Application to a haplotype classification problem

In this chapter, we will present the application of Uniform Min-Ones-2SAT for classifying haplotypes in more detail. The goal is to find genes which are responsible for the predisposition for some diseases.

Recall that the genome of every human is present in two copies in every cell. These two copies, called *haplotypes*, slightly differ from each other; one is inherited from the mother, the other from the father. In the following, we will not look at the whole genome of the individuals, but on smaller units, for example at the haplotypes of a single gene. This means that in a population of related individuals, some of the haplotypes will occur frequently, *e.g.*, because they have been conserved over some generations.

Some of these haplotypes might show genetic variations that are responsible for a predisposition for some kind of disease. It is of great importance to find these variations for a better understanding of such diseases and for the development of more successful treatments. Roughly speaking, our goal is to find these defective haplotypes in our data. The first problem is that the genomic data generated by many sequencing experiments does not produce the haplotypes of an individual, but rather a mixture of both haplotypes, the so-called *genotype*. It is a challenging problem by itself to compute the haplotypes from the genotype, but some successful algorithms have been presented in the literature; for an overview see, for example, [2,3,12].

We do not want to consider this problem here, rather we assume that we have access to some reliable haplotype data. Together with this haplotype data for some population of individuals, we are given some phenotypical data describing for each individual whether it shows some symptoms of the disease or not.

It is assumed that the predisposition for the disease is connected to some genetic defect on one or both haplotypes of the individual. The goal is to learn how to distinguish haplotypes with this defect from normal ones.

We now want to classify the individuals from the population into six different types according to the relationship between their haplotypes and the observed symptoms of the disease as shown in Table 1. Note that the information from the

TABLE 1. Types of individuals according to haplotype and phenotype data.

Type	Phenotype	Haplotypes
1	Healthy	Both normal
2	Healthy	1 normal, 1 defective
3	Symptoms of disease	1 normal, 1 defective
4	Symptoms of disease	Both defective
5	Healthy	Both defective
6	Symptoms of disease	Both normal

second column of the table is known to us whereas the information from the third column is what we would like to deduce.

For a first attempt of classifying the haplotypes and finding the defective ones, we want to simplify the model by restricting ourselves to the first four rows of Table 1. This means, we assume that the population does neither include individuals which show symptoms of the disease but have two normal haplotypes nor healthy individuals with two defective haplotypes. Intuitively, this means, we assume that an individual with two defective haplotypes will almost surely show symptoms of the disease, whereas it is rare for an individual to show symptoms of the disease without genetic predisposition.

This problem can now be modelled by UNIFORM MIN-ONES-2SAT : Consider the haplotypes as the variables of the formula, where an assignment of 1 to a variable means to classify the respective haplotype as defective. Then every individual of the population describes one clause of the formula. For a healthy individual, at least one of its two haplotypes has to be normal; this corresponds to a negative clause. On the other hand, if an individual shows symptoms of the disease, this means that at least one of its haplotypes has to be defective, corresponding to a positive clause. Therefore, finding an assignment with as few as possible ones, represents an explanation for the disease with the smallest possible number of defective haplotypes. This mirrors our expectation that the population should contain more normal haplotypes than defective ones.

4. APPROXIMABILITY OF UNIFORM MIN-ONES-2SAT

In this section, we present some results on the approximability of UNIFORM MIN-ONES-2SAT. We start with an upper bound.

Observation 4.1. There exists a polynomial-time approximation algorithm for UNIFORM MIN-ONES-2SAT with ratio 2.

Proof. Gusfield and Pitt have shown that MIN-ONES-2SAT is 2-approximable [6]. Since UNIFORM MIN-ONES-2SAT is a special case of MIN-ONES-2SAT, the claim immediately follows. □

Since Uniform Min-Ones-2SAT is a generalization of Min-VC according to Observation 2.4, any approximation ratio better than $2 - \frac{1}{\Delta_G}$ (where Δ_G denotes the maximum degree of the satisfiability graph) or better than $2 - \frac{1}{\Theta(\sqrt{\log n})}$ would improve on the best known Min-VC approximation [8,10].

Regarding the lower bounds on the approximation ratio, Uniform Min-Ones-2SAT inherits the results from Min-VC. This leads to the following observation.

Observation 4.2. Uniform Min-Ones-2SAT is $\mathcal{APX}$-hard and not approximable within a ratio of 1.3606, unless $\mathcal{P} = \mathcal{NP}$ and not approximable within a factor better than 2 unless the unique games conjecture holds.

Proof. This follows immediately from the respective result for Min-VC [5,9]. □

We now prove that Uniform Min-Ones-2SAT is as hard to approximate as Min-Ones-2SAT.

Theorem 4.3. *For every polynomial-time α-approximation algorithm A_U for* Uniform Min-Ones-2SAT*, there exists a polynomial-time algorithm A_N for* Min-Ones-2SAT *with asymptotic approximation ratio α.*

Proof. Let A_U be a polynomial-time α-approximation algorithm for Uniform Min-Ones-2SAT. We show in the following how A_U can be used to find an asymptotic α-approximate solution to any Min-Ones-2SAT instance.

Consider an input instance φ for Min-Ones-2SAT over the set of variables $X = \{x_1, \ldots, x_n\}$. Let m denote the number of clauses of φ and let m_N denote the number of non-uniform (*i.e.*, mixed) clauses in φ. Without loss of generality, we assume that the clauses are numbered such that $C_1, \ldots, C_{m_N}$ are the mixed clauses. Moreover, we assume that φ is not satisfied by the all-zero assignment.

Define $k := \lceil (\alpha - 1) \cdot (m_N + n) \cdot f(n) \rceil$, where f denotes some unbounded, but arbitrarily slowly increasing function, *e.g.*, the inverse of the Ackermann function.

From φ, we construct a Uniform Min-Ones-2SAT formula ψ in two steps. In the first step, we add new variables $y_1, \ldots, y_n$ and $z_{1,1}, z_{1,2}, \ldots, z_{1,k}, \ldots, z_{n,1}, \ldots, z_{n,k}$ together with new clauses $(x_i \vee y_i)$, $(\overline{x_i} \vee \overline{y_i})$, $(y_i \vee z_{i,j})$, and $(\overline{y_i} \vee \overline{z_{i,j}})$, for all $i \in \{1, \ldots, n\}$ and all $j \in \{1, \ldots, k\}$.

In the second step, we add new variables $u_1, \ldots, u_{m_N}$ and $v_1, \ldots, v_{m_N}$ and replace the mixed clause $C_i = (x_{i_1} \vee \overline{x_{i_2}})$ by the four uniform clauses $(x_{i_1} \vee v_i)$, $(\overline{x_{i_2}} \vee \overline{v_i})$, $(v_i \vee u_i)$, and $(\overline{v_i} \vee \overline{u_i})$, for all $i \in \{1, \ldots, m_N\}$.

The satisfiability graph of the resulting formula ψ is shown for a sample formula in Figure 1.

Clearly, the formula ψ can be constructed from φ in polynomial time. We prove in the following that restricting an α-approximate assignment for ψ to the variables from X yields the desired asymptotic α-approximate solution for φ. For this, we prove the following claim:

> Let β be a satisfying assignment for φ setting c variables from X to one. Then every assignment γ for ψ which coincides with β on X satisfies ψ and has exactly $n + c \cdot k + m_N$ variables set to one. (1)

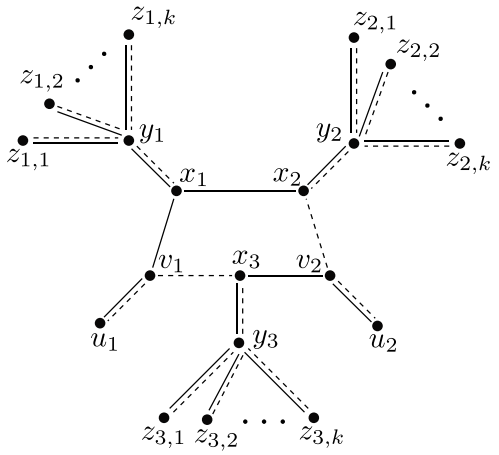

FIGURE 1. The satisfiability graph of the uniform 2CNF formula ψ as constructed in the proof of Theorem 4.3 for the 2CNF formula $\varphi = (x_1 \vee \overline{x_3}) \wedge (\overline{x_2} \vee x_3) \wedge (x_1 \vee x_2)$. Solid lines denote positive edges, dashed lines denote negative edges.

To prove this claim, we first consider the y- and z-variables. The respective clauses in ψ are constructed such that they formulate an exclusive-or constraint for x_i and y_i as well as an exclusive-or constraint for y_i and any of the $z_{i,j}$. Thus, if $\beta(x_i) = \gamma(x_i) = 1$, then $\gamma(y_i) = 0$ and thus $\gamma(z_{i_1}) = \ldots = \gamma(z_{i,k}) = 1$. If, on the other hand, $\beta(x_i) = \gamma(x_i) = 0$, then $\gamma(y_i) = 1$ and $\gamma(z_{i_1}) = \ldots = \gamma(z_{i,k}) = 0$. Overall, $c \cdot k$ of the z-variables and $n - c$ of the y-variables are set to one by γ.

We now consider the u- and v-variables. Let $C_i = (x_{i_1} \vee \overline{x_{i_2}})$ be any mixed clause in φ. The two clauses $(v_i \vee u_i)$ and $(\overline{v_i} \vee \overline{u_i})$ formulate an exclusive-or constraint for v_i and u_i. Thus, every satisfying assignment γ for ψ has to satisfy exactly m_N of these variables. It remains to show that it is possible to extend any satisfying assignment β for φ such that it also satisfies the clauses added in the second step of the construction. For this, we distinguish three cases according to how a mixed clause C_i is satisfied by β. At least one of the two literals has to be set to one by β. If $\beta(x_{i_1}) = 0$, then $\gamma(v_i) = 1$ is forced; if $\beta(x_{i_2}) = 1$, then this forces $\gamma(v_i) = 0$. If $\beta(x_{i_1}) = 1$ and $\beta(x_{i_2}) = 0$, then v_i can be set arbitrarily.

Overall, γ sets $c + c \cdot k + (n - c) + m_N = n + c \cdot k + m_N$ variables to 1 which proves the Claim (1).

We are now ready to analyze the approximation ratio achieved by this transformation. Let c_{opt} denote the minimum number of ones which have to be set to 1 in order to satisfy φ. Due to Claim (1), this implies that the minimum number of variables which have to be set to one for satisfying ψ is $n + c_{\text{opt}} \cdot k + m_N$. This means, an α-approximation algorithm for UNIFORM MIN-ONES-2SAT computes an assignment γ_{approx} for ψ which sets at most $\alpha \cdot (n + c_{\text{opt}} \cdot k + m_N)$ variables to one. Let x denote the number of variables from X which are set to one by γ_{approx}.

Following the argumentation above, the number x can also be counted as

$$x \leq \alpha \cdot (n + c_{\text{opt}} \cdot k + m_N) - m_N - x \cdot k - (n - x).$$

This leads to

$$x \leq \frac{1}{k} \cdot (\alpha \cdot (n + c_{\text{opt}} \cdot k + m_N) - m_N - n).$$

Thus, the approximation ratio of this approach can be estimated as

$$\begin{aligned} \frac{x}{c_{\text{opt}}} &\leq \frac{1}{c_{\text{opt}}} \cdot \frac{1}{k} \cdot (\alpha \cdot (n + c_{\text{opt}} \cdot k + m_N) - m_N - n) \\ &\leq \alpha + \frac{1}{k} \frac{1}{c_{\text{opt}}} \cdot (\alpha - 1)(n + m_N) \leq \alpha + \frac{1}{c_{\text{opt}} \cdot f(n)} \end{aligned}$$

which tends to α for n approaching infinity. □

5. An Exact algorithm for Uniform Min-Ones-2SAT

In this section, we design a fast exact algorithm for Uniform Min-Ones-2SAT. In the whole section, we use the graph representation of this problem. Given the satisfiability graph $G_\varphi = (X, E_+ \cup E_-)$ of a uniform 2CNF formula φ, we are looking for a subset $S \subseteq X$ of vertices, such that $S \cap V(G_\varphi^+)$ is a minimum vertex cover on G_φ^+ and $S \cap V(G_\varphi^-)$ is an independent set on G_φ^-. Our algorithm uses a standard branch-and-bound approach (see [7,11]).

5.1. Deduction Rules

Below, we present a list of deduction rules that will be applied during the branch-and-bound algorithms whenever their premises apply.

Dnegative: If v is incident only to negative edges, we can deduce that $v \notin S$.

Dspread: If we already decided that $v \in S$, and we have $uv \in E_-$, we can deduce that $u \notin S$. Similarly, from $v \notin S$ and $uv \in E_+$ we have $u \in S$.

Dtriangle: If we have positive edges xy and xz and a negative edge yz, we can deduce that $x \in S$. If we have negative edges xy and xz and a positive edge yz, we can deduce that $x \notin S$.

Dgreedy: If $v_1, \ldots, v_k$ (for some $k \geq 1$) and u are vertices such that:

- for each i there is a positive edge $v_i u$;
- no v_i has any other positive edges;
- there are no negative edges between u and vertices other than v_i

then we can just consider the case where $u \in S$ and $\forall i : v_i \notin S$.

Lemma 5.1. *All presented deduction rules are correct. That is, applying any of the rules will never cause our search to skip all optimal valid assignments.*

Proof. We will consider the deduction rules one at a time.

Dnegative: If a variable v only appears in negative clauses, we take any valid assignment where v is true and change it to false, thereby obtaining a better valid assignment. Hence we can just consider the case where v is false.

Dspread: Let $v \in S$ and $uv \in E_-$. Then the only remaining way to satisfy the clause $(\overline{u} \vee \overline{v})$ is to set u to false. If $v \notin S$ and $uv \in E_+$, then the only way to satisfy the clause $(u \vee v)$ is to set u to true.

Dtriangle: At least one of y and z must be false in any valid assignment. As there is a positive edge connecting that variable to x, it follows that x must be true in any valid assignment.

The other case is proved similarly. At least one of y and z must be true in any valid assignment. As there is a negative edge connecting that variable to x, it follows that x must be false in any valid assignment.

Dgreedy: Consider any valid assignment. Assume that, for some i, we have $v_i \in S$, that is, the variable v_i is true. This assignment can be changed as follows. Set all v_i to false. If u is false, set it to true. This change did not increase the number of true variables and it preserved the validity of the assignment. To see why the latter holds, note that all clauses that only contain u and one of the v_i are satisfied by our new assignment, all other clauses containing u are positive and therefore satisfied, and all other clauses containing the v_i are negative and therefore satisfied.

Hence, when looking for an assignment that minimizes the number of true variables, it is sufficient to consider the one chosen by this rule. □

5.2. The Algorithm

Our algorithm consists of several phases, as described below.

Phase 1: We use **Dnegative** to ensure that each vertex is incident to a positive edge.

Phase 2: While there is some vertex incident to at least 5 negative edges, we pick any such vertex v and branch on it (with applying **Dspread** afterwards).

Phase 3: For any vertex v incident to some negative edges, we analyze all possible cases how its neighborhood looks like. In each case, we either show that we can deduce the state of some vertex, or we find a vertex such that branching on it produces two sufficiently smaller instances.

Phase 4: Once only positive edges are left, we apply a general algorithm to find a minimum vertex cover.

In some situations, our algorithm will branch on a given vertex v, meaning that we sequentially try first the case $v \in S$ and then the case $v \notin S$. In each case, we will be applying the deduction rules (usually the rule **Dspread**) to deduce as much information as possible. We call a branching an (a, b)-*branching* if in one case we

can deduce the state of a vertices and in the other case the state of b vertices (including the one we branch on).

Now we give a detailed description of Phases 2 and 3 of our branch-and-bound algorithm. To achieve the desired running time, any branching must be at least a $(6,2)$-, a $(5,3)$-, or a $(4,4)$-branching. We will call all such branchings *acceptable*.

We introduce some additional notation for the sake of the proof. The phrase *positive neighbor* is shorthand for "neighbor in the graph (X, E_+)", similarly for *negative neighbor*. The degree of v in G_φ^+ is denoted $\Delta_+(v)$, $\Delta_-(v)$ is defined similarly. We also define $\Delta_\pm(v) = (\Delta_+(v), \Delta_-(v))$ and $\Delta(v) = \Delta_+(v) + \Delta_-(v)$.

In each of the cases presented below, v will be some vertex incident to negative edges, n_i will be its negative neighbors and p_j will be its positive neighbors. If some of the n_i coincide with some of the p_j, we will label them so that the last few n_i are equal to the first few p_j.

In all figures, the vertices shown as squares are already final, i.e., there are no other edges leaving these vertices. Positive edges are shown as full lines, negative edges are shown as dashed lines, unknown edges are dotted.

5.2.1. *Phase 2: Handling vertices of large negative degree*

As the outcome of Phase 1, we know that $\Delta_+(v) > 0$, for all v.

Take any vertex v and let $\Delta_\pm(v) = (a, b)$. Regardless of the rest of the graph, if we branch on v, this will be at least an $(a+1, b+1)$-branching. If $b > 4$, this branching is guaranteed to be acceptable and we can do it immediately.

5.2.2. *Phase 3: Other vertices of large degree*

Let v be an arbitrary vertex with $\Delta(v) > 5$ and $\Delta_-(v) > 0$. Then the branching at v is clearly acceptable and we can do it immediately. After this process terminates, we get a graph in which each vertex is incident to at least one positive edge and at most four negative edges. Additionally, a vertex adjacent to some negative edges has degree at most 5. We will now show how to process the remaining vertices incident to negative edges.

5.2.3. *Phase 3: Vertices with four negative edges*

Suppose that $\Delta_-(v) = 4$. Then we must have $\Delta_\pm(v) = (1, 4)$. There has to be a positive edge incident to n_1. If it leads into a new vertex, by **Dspread** we get at least a $(6,2)$-branching at v, otherwise we can apply the rule **Dtriangle**.

5.2.4. *Phase 3: Vertices with three negative edges*

The case $\Delta_\pm(v) = (2, 3)$ is handled in the same way as $\Delta_\pm(v) = (1, 4)$, except this time the branching is at least $(5, 3)$ instead of at least $(6, 2)$.

Now let $\Delta_\pm(v) = (1, 3)$. There is at least one positive edge incident to each of n_1 and n_2. If there is a positive edge xy with $x \in \{n_1, n_2\}$ and $y \in \{n_1, n_2, n_3, p_1\}$, we can apply **Dtriangle**. If two positive edges lead from n_1 and n_2 into two distinct new vertices, we get at least a $(6,2)$-branching at v.

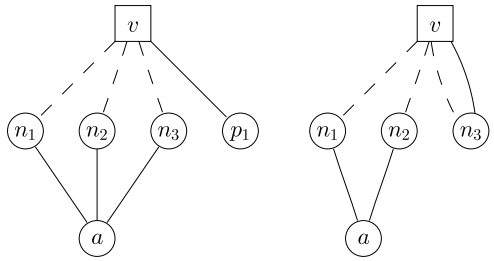

FIGURE 2. Two cases while handling a vertex v with three negative edges.

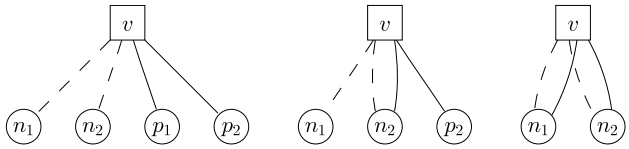

FIGURE 3. Three cases for v with two positive and two negative edges.

Finally, we are left with one of the cases shown in Figure 2. If there is a negative edge from a into a new vertex, we get a $(6,2)$-branching at v. If there is a negative edge ap_1, we get a $(5,3)$-branching at v. In all other cases we can apply **Dgreedy** at a.

From this moment we may assume that each vertex in our graph is incident to at most two negative edges.

5.2.5. *Phase 3: Vertices with Two Negative Edges*

Three positive edges: We have $\Delta_{\pm}(v) = (3,2)$.

If there is no positive edge vn_1, there must be a positive edge leaving n_1. If its second endpoint is a neighbor of v, we apply **Dtriangle**, otherwise we have at least a $(4,4)$-branching at v.

In the remaining case, we have $n_1 = p_1$ and $n_2 = p_2$. If $\Delta(n_1) = \Delta(n_2) = 2$, we apply **Dgreedy** at v. If there is any (positive or negative) edge xy with $x \in \{n_1, n_2\}$ and $y \in \{n_1, n_2, p_3\}$, we can always apply **Dtriangle**. Finally, if there is an edge from one of n_1 and n_2 to a new vertex, we get an acceptable branching at v. (Either $(5,3)$ or $(4,4)$, depending on the type of the new edge.)

Two positive edges: We have $\Delta_{\pm}(v) = (2,2)$.

If there is a positive edge n_1n_2, a negative edge p_1p_2, or any edge xy with $x \in \{n_1, n_2\}$ and $y \in \{p_1, p_2\}$, we apply **Dtriangle**.

We will now consider three sub-cases depending on the number of n_i and p_j that coincide. These cases are shown in Figure 3.

In the left case, we must have positive edges leaving n_1 and n_2 into new vertices. If these are two distinct vertices, we get a $(5,3)$-branching at v. Otherwise label

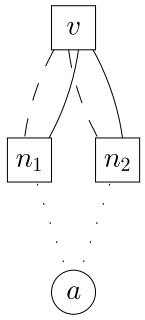

FIGURE 4. The last remaining case for two positive and two negative edges.

their common vertex a. If there is a negative edge from a to a new vertex or to one of the p_i, we get an acceptable branching at v. In the other case, we can apply **Dgreedy** at a.

In the middle case, we must have a positive edge n_1a where a is a new vertex. We apply the same reasoning at a as in the previous case.

We are left with the case on the right. If one of n_1 and n_2 has degree 2, we can remove this vertex and v from the graph, find the optimal solution for the smaller graph, and then reattach the two vertices and determine their values. Otherwise, we have at least one more edge leaving each of the vertices. If there is an edge n_1n_2, we apply **Dtriangle**. If there are edges into two new vertices, we get an acceptable branching at v.

The last remaining case is shown in Figure 4. If the edges n_1a and n_2a are of different types, we have an acceptable branching at v. If they are both negative, we apply **Dgreedy** at v. If they are both positive, we can remove the vertices v and n_1, add a negative edge n_2a, solve the smaller problem and use the solution to reconstruct the optimal solution. To see the correctness of this construction, note that there are two possible assignments for v, n_1, n_2, namely $v \notin S$, $n_1, n_2 \in S$ or $v \in S$, $n_1, n_2 \notin S$. The latter, more desirable, assignment is only feasible if $a \in S$. Thus, n_2 has a different value than a in any optimal solution which can be guaranteed by the additional negative edge.

One positive edge: We have $\Delta_{\pm}(v) = (1, 2)$.

If there is a positive edge n_1n_2 or any edge n_ip_1, we apply **Dtriangle**.

Again, we now have two separate cases: either $n_2 = p_1$ or not.

Case $n_2 \neq p_1$. If there is no negative edge at p_1, we apply **Dgreedy** at p_1. If there are positive edges from n_1 and n_2 into two distinct new vertices, we get at least a $(5, 3)$-branching at v. Otherwise, there are exactly two positive edges n_1a and n_2a. If there are no negative edges incident to a or the only negative edges incident to a are n_1a or n_2a, we apply **Dgreedy** at a, otherwise we get an acceptable branching at a.

Case $n_2 = p_1$. Note that there is no edge between n_1 and n_2. If there is no other negative edge at n_2, we apply **Dgreedy** at n_2. Otherwise we have a negative edge

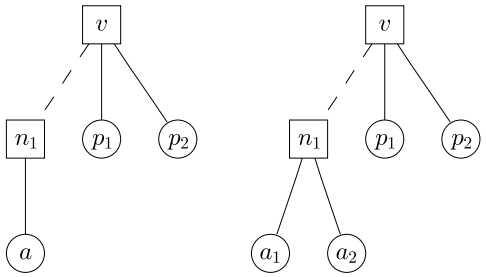

FIGURE 5. Difficult cases with a single negative edge.

n_2a. Both n_1 and a must be incident to positive edges. If there is a positive edge n_1a, we get at least a $(4,4)$-branching at v. If there are positive edges from n_1 and a to new vertices (not necessarily distinct ones), we get at least a $(4,4)$-branching at v. The last remaining possibility are positive edges n_1b and n_2a, which gives us a $(5,3)$-branching at v.

5.2.6. *Phase 3: Vertices with a single negative edge*

We are left with a graph in which each vertex is only incident to at most one negative edge, and each vertex incident to a negative edge has degree at most 5.

We skip the analysis for cases $\Delta_\pm(v) = (4,1)$ and $\Delta_\pm(v) = (3,1)$. The analysis for these cases is just a simpler case of the analysis for $\Delta_\pm(v) = (2,1)$.

Two positive edges: we have $\Delta_\pm(v) = (2,1)$.

If there is any edge n_1p_i, or a negative edge p_1p_2, we apply **Dtriangle**.

We will first consider the case $n_1 \neq p_1$. There are either one or two positive edges incident to n_1. Both cases are shown in Figure 5.

In the case on the left side of Figure 5 consider the vertex a. If there is no negative edge leaving a, we can apply **Dgreedy**. If there is one into some p_i, we get an acceptable branching at v. Finally, let there be a negative edge ab. The new vertex b must have a positive edge. If it leads into a, we get at least a $(6,2)$-branching at a, otherwise we get at least a $(5,3)$-branching at v.

The case in Figure 5 on the right can be solved as follows. If v and n_1 are both false, then p_1, p_2, a_1, a_2 must all be true and we find an optimal solution for the rest of the graph. Otherwise exactly one of v and n_1 is true. We remove vertices v and n_1, add all four positive edges p_ia_j, find the optimal solution for the new graph and then deduce the values for v and n_1. Note that this is a special $(6,2)$-branching.

Now for the case $n_1 = p_1$. If $\Delta(n_1) = 2$, we apply **Dgreedy** at v. Otherwise we get a positive edge n_1a. Now note that exactly one of v and n_1 is true. Also, as a consequence, p_2 and a may not be false at the same time. We now can add a positive edge p_2a, remove the vertices v and n_1, find the optimal solution for the smaller graph and afterwards deduce the truth values for v and n_1.

A single positive edge: Each vertex is either incident to positive edges only, or it is incident to a positive and a negative edge. Let v be such that $\Delta_\pm(v) = (1,1)$.

If $p_1 = n_1$, we apply **Dgreedy** at n_1. If the positive edge from n_1 goes to p_1, we apply **Dtriangle**, otherwise let its endpoint be a. If there are no negative edges from a or p_1, we can apply **Dgreedy** at that vertex. If there is a negative edge ap_1, we have a $(4,4)$-branching at v. Finally, we might have negative edges ab and p_1c. Together with a positive edge that must be incident to c we again have a $(4,4)$-branching at v.

This concludes Phase 3 of our algorithm.

5.3. Proof of correctness and time complexity

Lemma 5.2. *Whenever our algorithm branches, the branching vertex will be chosen to achieve at least a* $(6,2)$*-, a* $(5,3)$*-, or a* $(4,4)$*-branching.*

Proof. This follows from the detailed presentation of the third phase of our algorithm. □

Theorem 5.3. *Our algorithm correctly solves the* Uniform Min-Ones-2SAT *and its time complexity is* $\mathcal{O}(1.21061^n)$.

Proof. The correctness immediately follows from Lemma 5.1 and the detailed analysis of the algorithm above.

Using Robson's algorithm [13], we can find the minimum vertex cover in any graph with n vertices in $\mathcal{O}(1.18882^n)$.

By Lemma 5.2, each branching of our algorithm is at least a $(6,2)$-, a $(5,3)$-, or a $(4,4)$-branching. Consider the recurrences that describe the time complexity of a branch and bound algorithm in each of these cases separately:
The recurrence $T(n) = 2T(n-4) + \mathcal{O}(1)$ solves to $T(n) = \mathcal{O}(1.18921^n)$,
the recurrence $T(n) = T(n-3) + T(n-5) + \mathcal{O}(1)$ solves to $T(n) = \mathcal{O}(1.19386^n)$,
and the recurrence $T(n) = T(n-2) + T(n-6) + \mathcal{O}(1)$ solves to $\mathcal{O}(1.21061^n)$.

Hence, even if the worst case occurs and our algorithm is forced to perform (6,2)-branchings until it runs out of vertices, its time complexity will be $\mathcal{O}(1.21061^n)$. □

Additionally, our algorithm can be easily modified to obtain a parameterized complexity algorithm.

Theorem 5.4. *Given a positive integer* k *and a* Uniform Min-Ones-2SAT *instance, we can check whether there is a satisfying assignment with at most* k *true variables in* $\mathcal{O}^*(2^k)$.

Proof. Phase 1 runs in polynomial time. In Phase 2 and Phase 3 we can easily verify that each branching is done on a vertex incident to a positive and a negative edge. Hence, in each branch we set at least one new variable to true. This implies that we can limit the search tree depth to k. If a solution with at most k true variables exists, we will still find it.

For Phase 4, we can use the algorithm by Chen *et al.* [4], which is an $\mathcal{O}(kn + 1.2852^k)$ algorithm for the parameterized version of the minimum vertex cover problem. □

6. Conclusion

In this paper, we introduced the Uniform Min-Ones-2SAT problem as a means for modelling a problem of haplotype classification. We analyzed its approximability and presented a moderately exponential-time exact algorithm for it. While we have seen that Uniform Min-Ones-2SAT and general Min-Ones-2SAT are equally hard with respect to approximability, it remains as an interesting open problem to extend the exact algorithm to the general case.

Moreover, from the viewpoint of the biological application, it would be very interesting to generalize the classification model to include all types of individuals as listed in Table 1 or to allow for some errors in the data.

References

[1] B. Aspvall, M.F. Plass and R.E. Tarjan, A linear-time algorithm for testing the truth of certain quantified boolean formulas. *Inf. Proc. Lett.* **8** (1979) 121–123.

[2] H.-J. Böckenhauer and D. Bongartz, *Algorithmic Aspects of Bioinformatics*. Natural Computing Series, Springer-Verlag (2007).

[3] P. Bonizzoni, G.D. Vedova, R. Dondi and J. Li, The haplotyping problem: An overview of computational models and solutions. *J. Comput. Sci. Technol.* **18** (2003) 675–688.

[4] J. Chen, I. Kanj and W. Jia, Vertex cover: further observations and further improvements. *J. Algorithms* **41** (2001) 280–301.

[5] I. Dinur and S. Safra, On the hardness of approximating minimum vertex cover. *Ann. Math.* **162** (2005) 439–485.

[6] D. Gusfield and L. Pitt, A bounded approximation for the minimum cost 2-sat problem. *Algorithmica* **8** (1992) 103–117.

[7] J. Hromkovič, *Algorithmics for Hard Problems. Introduction to Combinatorial Optimization, Randomization, Approximation, and Heuristics*. Texts in Theoretical Computer Science, An EATCS Series, Springer-Verlag, Berlin (2003).

[8] G. Karakostas, *A better approximation ratio for the vertex cover problem*. Technical Report TR04-084, ECCC (2004).

[9] S. Khot and O. Regev, Vertex cover might be hard to approximate to within 2-epsilon. *J. Comput. Syst. Sci.* **74** (2008) 335–349.

[10] J. Kiniwa, Approximation of self-stabilizing vertex cover less than 2, in *Self Stabilizing Systems* (2005) 171–182.

[11] E.L. Lawler and D.E. Wood, Branch-and-bound methods: A survey. *Operat. Res.* **14** (1966) 699–719.

[12] J. Li and T. Jiang, A survey on haplotyping algorithms for tightly linked markers. *J. Bioinf. Comput. Biol.* **6** (2008) 241–259.

[13] J.M. Robson, *Finding a maximum independent set in time* $O(2^{n/4})$. Technical Report 1251-01, LaBRI, Université Bordeaux I (2001).

Communicated by J. Hromkovic.
Received July 1st, 2010. Accepted July 1st, 2010.