Journal on Satisfiability, Boolean Modeling and Computation 6 (2008) 13-32

Linear Satisfiability Algorithm for 3CNF Formulas of Certain Signaling Networks

Utz-Uwe Haus[*] haus@imo.math.uni-magdeburg.de
Institute for Mathematical Optimization
Otto-von-Guericke University Magdeburg
39106 Magdeburg, Germany

Klaus Truemper truemper@utdallas.edu
Department of Computer Science
University of Texas at Dallas
Richardson, Texas 75080, USA

Robert Weismantel[*] weismantel@imo.math.uni-magdeburg.de
Institute for Mathematical Optimization
Otto-von-Guericke University Magdeburg
39106 Magdeburg, Germany

Abstract

A simple model of signal transduction networks in molecular biology consists of CNF formulas with two and three literals per clause. A necessary condition for correctness of the network model is satisfiability of the formulas. Deciding satisfiability turns out to be $\mathcal{NP}$-complete. However, for a subclass that still is of practical interest, a linear satisfiability algorithm and related characterization of unsatisfiability can be established. The subclass is a special case of so-called closed sums of CNF formulas.

Keywords: *SAT algorithm, 2CNF, 3CNF, closed sum*

Submitted December 2007; revised September 2008; published September 2008

1. Introduction

A signal transduction network is a simplified model of a biological unit that reacts to external signals or environmental challenges like stimulation or infection. The relevant molecules of the biological unit are classified as input substances, for example, receptors, intermediate substances, and output substances, for example, transcription factors. The network depicts how substances produce or trigger other substances.

In the setting considered here, the actions are represented by logic formulas where the propositional variables represent the substances under consideration, and where logic implications represent experimentally proven knowledge. For example, "MEK *activates* ERK" is encoded by MEK $\to$ ERK, and *"In the absence of (activated)* pten *and* ship1 *we find that* pi3k *produces* pip3" is handled by $(\neg\text{pten} \wedge \neg\text{ship1} \wedge \text{pi3k}) \to \text{pip3}$ [9].

Using the notation of propositional logic, we can assume that all implications are of the form $(\bigvee_{j\in J} A_j) \to C$, where the A_j, $j \in J$, and C are literals of propositional variables. In

∗ Supported by FORSYS grant of the German Ministry of Education and Research.

the networks of interest here, it is assumed that the reverse implications hold as well. This is equivalent to claiming that we know all input patterns that may produce a given output. The formulation and interpretation is related to the query evaluation process of [5]. That process uses the *negation as failure* inference rule, where the negation of a conclusion is declared to have been proved if every possible proof of the conclusion fails.

Let R be the conjunction of the implications of a network. If R represents the behavior of the biological unit correctly, then necessarily R is satisfiable. On the other hand, if R is unsatisfiable, then localizing the cause helps understand which part of R is defective. Accordingly, establishing satisfiability as well as localizing causes of unsatisfiability are important. Identifying causes of unsatisfiability is also important when some variables of R represent external inputs. In the typical case, each possible assignment of values to the input variables corresponds to a satisfying solution of a second formula T, and conversely. We want to verify that for each satisfying solution of T, the corresponding input values leave R satisfiable. The problem of deciding this question is called Q-ALL SAT in [8], where an effective solution algorithm is described. If for some satisfying solution of T the formula R does become unsatisfiable, then localizing the reason is important.

If an implication has on the left-hand side a disjunction with at least three literals, then this case can be transformed with additional variables to implications with two literals on the left-hand side. For example, the implication $(A_1 \vee A_2 \vee A_3 \vee A_4) \leftrightarrow C$ has the equivalent representation $(A_1 \vee A_2) \leftrightarrow Z_1$, $(Z_1 \vee A_3) \leftrightarrow Z_2$, and $(Z_2 \vee A_4) \leftrightarrow C$. Thus, we assume from now on that all implications have two literals on the left-hand side. We call the variables of those two literals the *input variables* of the implication and the variable of the single right-hand side literal the *conclusion variable*.

A network may also impose side conditions in the form of disjunctions. Until Section 5, we consider only short disjunctions having at most two literals. Due to the possibility of such disjunctions, we may assume that any variable occurs at most once as conclusion variable of the implications. Indeed, if two implications have literals of the same conclusion variable c, then we replace the literals by literals of two new conclusion variables c_1 and c_2 and add the condition $c_1 \leftrightarrow c_2$, which is equivalent to the two disjunctions $\neg c_1 \vee c_2$ and $c_1 \vee \neg c_2$.

To summarize, the formula R is a conjunction of implications of the form $(A \vee B) \leftrightarrow C$ and of disjunctions with at most two literals. Furthermore, any variable occurs at most once as conclusion variable of the implications.

Since $(A \vee B) \leftrightarrow C$ can be represented by the conjunction of $\neg A \vee C$, $\neg B \vee C$, and $A \vee B \vee \neg C$, the formula R is equivalent to the formula S that is a conjunction of the following disjunctions. First, any disjunction with at most two literals is allowed. Second, disjunctions of the form $A \vee B \vee \neg C$ may occur, in which case there must also be the disjunctions $\neg A \vee C$ and $\neg B \vee C$. For the second case, let c be the variable of the literal C. Then there cannot be another disjunction of the second type with the same conclusion variable c.

It turns out that the satisfiability problem of formulas S defined above is $\mathcal{NP}$-complete, but becomes solvable in linear time under certain assumptions. The presentation proceeds as follows. Section 2 introduces a number of definitions and basic results, including $\mathcal{NP}$-completeness of the satisfiability problem of formulas S. Section 3 characterizes a subclass where the formulas have totally unimodular (t.u.) representation matrices, which are integer

$\{0, \pm 1\}$ matrices where each square submatrix has determinant equal to 0, +1, or −1. It is well known that total unimodularity assures linear-time checking of satisfiability and compact identification of causes of unsatisfiability. The characterization of the t.u. case involves two conditions, one of them rather severe, the second one much less so. Section 4 describes a linear satisfiability algorithm assuming just the second, less severe, condition. The linear complexity is due to a certain decomposition of logic formulas. The section also provides a simple characterization of unsatisfiability under the same assumption. Section 5 shows that the decomposition used in Section 4 is a special case of the so-called *closed sum* decomposition of [13]. That viewpoint also produces polynomial satisfiability algorithms for some settings where implications have any number of input variables and additional disjunctions have more than three literals.

2. Definitions and Basic Results

This section summarizes and extends the above definitions for logic formulas. It also covers definitions and basic results for the satisfiability problem and for certain graphs and matrices. Most of the material is well-known to experts of logic or combinatorics. We include it to make the paper self-contained and more accessible to readers outside logic or combinatorics.

Logic Formulas

Definition 2.1. Let S be a propositional logic formula. The formula is in *conjunctive normal form* (CNF) if it is the conjunction of CNF *clauses*, where each clause is a disjunction of possibly negated propositional variables. Each occurrence of a possible negated variable is a *literal* of S. A clause with exactly one literal is a *unit clause*. The *empty clause* has no literals.

Definition 2.2. A 2CNF (resp. 3CNF) *formula* is a CNF formula that has at most two (resp. exactly three) literals in each 2CNF (resp. 3CNF) *clause*.

Definition 2.3. A 3IFF *formula* R is a conjunction of 2CNF clauses and of 3IFF *clauses* of the form $(A \vee B) \leftrightarrow C$ where A, B, and C are literals of three distinct variables a, b, and c, respectively; the variables a and b are the *input variables* and variable c is the *conclusion variable*. In R, any variable may occur at most once as conclusion variable. A 3IFF formula is *linearly ordered* if the variables of the formula are linearly ordered such that, for each 3IFF clause, the conclusion variable is larger than both input variables.

Definition 2.4. A 3IFFCNF *formula* S is the CNF formula obtained from a 3IFF formula R by representing each 3IFF clause $(A \vee B) \leftrightarrow C$ by the three equivalent CNF clauses $\neg A \vee C$, $\neg B \vee C$, and $A \vee B \vee \neg C$.

Satisfiability Problems

For details, see [7].

Definition 2.5. The satisfiability problem SAT has a CNF formula as instance, and the question "Can S be satisfied?" is to be answered.

Definition 2.6. The problems 2SAT (resp. 3SAT, 3IFFSAT) are special SAT cases where S is 2CNF (resp. 3CNF, 3IFFCNF). The problem ONE-IN-THREE 3SAT has as input a 3CNF formula S, and the question "Can S be satisfied such that in each clause exactly one literal evaluates to *True*?" is to be answered.

Lemma 2.1. (see [7]) *Problem* 2SAT *can be solved in linear time, while the problems* SAT*,* 3SAT*, and* ONE-IN-THREE 3SAT *are $\mathcal{NP}$-complete.*

We use the $\mathcal{NP}$-completeness of ONE-IN-THREE 3SAT to prove the same conclusion for 3IFFSAT and two special cases. A third special case that properly contains the intersection of the first two special cases, is solvable in linear time.

Theorem 2.1. *Problem* 3IFFSAT *is $\mathcal{NP}$-complete. Remains $\mathcal{NP}$-complete if the instances do not have* 2CNF *clauses or if they can be linearly ordered. Is solvable in linear time if the* 2CNF *clauses of the instances do not contain any conclusion variables and if the instances can be linearly ordered.*

Proof. All cases are clearly in $\mathcal{NP}$. To prove completeness of 3IFFSAT, we reduce ONE-IN-THREE 3SAT to 3IFFSAT. Take an instance S of ONE-IN-THREE 3SAT. We derive the following formula R. For each 3CNF clause of S, say $A \vee B \vee C$, we assign the 3IFF clause $(A \vee B) \leftrightarrow \neg C$ and the 2CNF clause $\neg A \vee \neg B$ to R. Simple checking shows that the satisfying solutions of $A \vee B \vee C$ where exactly one literal evaluates to *True* are precisely the satisfying solutions of the two clauses added to R. When all clauses of ONE-IN-THREE 3SAT have been processed as described, we satisfy the condition that each variable of R can occur at most once as conclusion variable, by the transformation described in Section 1. The resulting 3IFF formula has a satisfying solution if and only if the ONE-IN-THREE 3SAT instance has a satisfying solution where in each clause exactly one literal evaluates to *True*.

To prove $\mathcal{NP}$-completeness of the first special case, where the instances do not have any 2CNF clauses, we derive from an arbitrary 3IFF formula R an equivalent 3IFF formula observing the special condition, as follows. We introduce a new variable w and replace each 2CNF clause $A \vee B$ by the 3IFF clause $(A \vee B) \leftrightarrow w$. Then we introduce three new variables x, y, and z, and add the three 3IFF clauses $(w \vee \neg z) \leftrightarrow x$, $(z \vee \neg x) \leftrightarrow y$, and $(x \vee \neg y) \leftrightarrow z$. The three 3IFF clauses force $w = \textit{True}$ in any satisfying solution, which makes each 3IFF clause $(A \vee B) \leftrightarrow w$ equivalent to the corresponding 2CNF clause $A \vee B$.

For an $\mathcal{NP}$-completeness proof of the second special case, where the 3IFF clauses can be linearly ordered, we convert an arbitrary 3IFF formula R to an equivalent, linearly ordered 3IFF formula, as follows. Let R have the variables $x_1, x_2, \dots, x_m$. Suppose R has n 3IFF clauses. For $i = 1, 2, \dots, n$, we replace the conclusion literal of the ith 3IFF clause by a new variable z_i. Let R' be the resulting 3IFF formula. We order the variables of R' as x_1, $x_2, \dots, x_m, z_1, z_2, \dots, z_n$, with z_n largest. Finally, we add to R' 2CNF clauses enforcing the equivalence of each z_i with the conclusion literal it has replaced. The resulting 3IFF formula R'' is equivalent to R and linearly ordered.

Finally, we describe a linear-time algorithm for the third special case, where the 2CNF clauses of the instances do not contain any conclusion variables and the instances can be linearly ordered. Let R be one such 3IFF formula. Define $\triangleright$ to be the partial binary relation on the variables of R where $c \triangleright a$ holds if some 3IFF clause has a as input variable and c

as conclusion variable. Since any linear ordering of R must be consistent with $\triangleright$, and since at least one such ordering of R exists, we know that $\triangleright$ is a partial order and thus can be extended to a linear order of R. That extension can be done in linear time via a graph representation of $\triangleright$. We are ready to discuss the desired satisfiability algorithm. We first check satisfiability of the 2CNF clauses of R using the linear-time algorithm of [6]. If the 2CNF clauses are unsatisfiable, then R is unsatisfiable, and we stop. Otherwise, we proceed as follows. By assumption, the 2CNF clauses of R do not involve any conclusion variables of the 3IFF clauses of R. Thus, we can use the ordering of the conclusion variables implied by the linear ordering of R to assign *True/False* values so that all 3IFF clauses become satisfied. □

Graphs

For details, see, for example, [3].

Definition 2.7. An *arc* is a directed edge connecting two nodes, while the term *edge* refers to the undirected case. Both the arc from node A to B and the edge connecting A and B are denoted by (A, B). A graph is *directed* (resp. *undirected*) if it has only arcs (resp. edges). In a directed graph, a *directed* cycle or path allows a traversal in the direction of the arcs. A *cutnode* of a connected graph is a node whose removal increases the number of connected components. A 2-*connected component* of an undirected graph is a maximal connected subgraph that has no cutnode. A *strong component* of a directed graph is a maximal subgraph where any two nodes i and j are connected by two directed paths from i to j and from j to i.

Note that we utilize only directed or undirected graphs and thus do not make use of graphs having both arcs and edges.

For any CNF formula S, we define two graphs $G(S)$ and $H(S)$. The graph $G(S)$ is based solely on the clauses of S with two literals, while $H(S)$ is based on those clauses as well as the unit clauses.

Definition 2.8. Let S be a CNF formula.

(a) The undirected *graph* $G(S)$ has for each variable of S a node and, for each clause of S with two literals, one edge that connects the two nodes corresponding to the two variables of the clause. The edge is *regular* if exactly one of the two variables occurs negated in the clause, and it is *special* otherwise. All unit clauses and clauses with at least three literals are ignored in the construction.

(b) The directed *graph* $H(S)$ has a node for each literal of S. Each clause $A \vee B$ produces two arcs $(\neg A, B)$ and $(\neg B, A)$. Each unit clause A generates one arc $(\neg A, A)$. All clauses with at least three literals are ignored.

The two arcs $(\neg A, B)$ and $(\neg B, A)$ specified in Definition 2.8(b) represent the implication $\neg A \rightarrow B$ and its contrapositive $\neg B \rightarrow A$. Each of these implications is equivalent to $A \vee B$.

Matrices

For details, see, for example, [4].

Definition 2.9. Let S be a CNF formula. The integer *representation matrix* $M(S)$ of S has for each variable of S a column, and for each clause of S a row. The nonzero entries of $M(S)$ are $\{0, \pm 1\}$ and are defined as follows. If variable y occurs nonnegated (resp. negated) in clause x, then the entry of $M(S)$ in column y and row x is equal to $+1$ (resp. -1).

Balancedness and Total Unimodularity

We introduce some graph and matrix properties and results. For details, see, for example, [4] and [10].

Definition 2.10. A cycle of a graph $G(S)$ is *even* (resp. *odd*) if it has an even (resp. odd) number of special edges. The graph $G(S)$ is *balanced* if it has no odd cycles.

A standard graph result is the following.

Lemma 2.2. *A graph $G(S)$ is balanced if and only if the graph obtained from $G(S)$ by contraction of all regular edges is bipartite.*

Definition 2.11. A *hole* of an integer matrix M with $\{0, \pm 1\}$ entries is a minimal submatrix of M having exactly two $\{\pm 1\}$ entries in each row and column. A hole is *even* (resp. *odd*) if the entries of the hole sum to $0 (\text{mod } 4)$ (resp. $2 (\text{mod } 4)$).

Definition 2.12. An integer $\{0, \pm 1\}$ matrix M is *balanced* if it has no odd holes.

Definition 2.13. An integer matrix M is *totally unimodular* (t.u.) if each square submatrix has determinant equal to 0, +1, or −1.

The following two lemmas have simple proofs. For details, see Chapter 19 of [10].

Lemma 2.3. *The determinant of an even (resp. odd) hole is equal to* 0 *(resp.* ± 2*).*

Lemma 2.4.

(a) *If S is a* 2CNF *formula, then $G(S)$ is balanced if and only if $M(S)$ is balanced.*

(b) *If S is a* 3CNF *formula, then $M(S)$ is balanced if and only if it is totally unimodular. Furthermore, if $M(S)$ is balanced, then $G(S)$ is balanced.*

In subsequent sections, we sometimes replace logic variables by their complements to simplify the discussion. The next lemma says that such a change does not affect any of the properties introduced above. The proof involves trivial checking of cases.

Lemma 2.5. *In the cases below, S' is derived from a CNF formula S by replacing an arbitrary variable by its complement.*

(a) *Each cycle of $G(S)$ has the same parity as the corresponding cycle of $G(S')$. Furthermore, $G(S)$ and $G(S')$ have the same cutnodes and are either both balanced or both not balanced.*

(b) *For any variable t of S, the nodes t and $\neg t$ of $H(S)$ are in the same strong component if and only if this holds for $H(S')$.*

(c) *Each hole of $M(S)$ has the same parity as the corresponding hole of $M(S')$. Furthermore, $M(S)$ is t.u. if and only if $M(S')$ is t.u.*

Polyhedra

For a given CNF formula S, define the polyhedron $P(S)$ as

$$\begin{aligned} P(S) &= \{w \mid M(S)\cdot w \geq \underline{1} - n(M(S))\} \\ 0 &\leq w \leq \underline{1} \end{aligned} \tag{1}$$

where the $\underline{1}$s are vectors of 1s of suitable length, and where the ith entry of the vector $n(M(S))$ is the number of -1s in row i of $M(S)$.

Lemma 2.6. (see Chapter 2 of [4]) *If S is a* CNF *formula without unit clauses, then $P(S)$ contains the vector $x = \frac{1}{2}\cdot\underline{1}$ and thus is nonempty.*

The above definitions and Chapter 19 of [10] imply the following.

Lemma 2.7. *If $M(S)$ is t.u., then either $P(S)$ is empty, or all vertices of $P(S)$ are $\{0, \pm 1\}$. In the former case, S is unsatisfiable. In the latter case, the vertices are in one-to-one correspondence with the satisfying solutions of S where the value $+1$ (resp. -1) for a variable of $P(S)$ corresponds to True (resp. False) for the related logic variable of S.*

Assume $M(S)$ to be t.u. Then Lemmas 2.6 and 2.7 justify the following linear-time satisfiability algorithm for S.

1. Reduce S by recursively fixing the variables of unit clauses. If the empty clause is produced, declare S to be unsatisfiable, and stop. If there are no clauses left, declare the values on hand, plus arbitrary *True/False* values for variables not yet decided, to be a satisfying solution of S, and stop.

2. Arbitrarily assign a *True/False* value to a variable of S, and go to Step 1.

Still assume $M(S)$ to be t.u. If S is unsatisfiable, then the Farkas Lemma of Linear Programming applied to the inequalities of (1) provides a compact identification of the clauses causing that conclusion. Finally, if minimum cost versions of SAT are to be solved, then general linear programming techniques or, alternately, special methods for t.u. matrices provide optimal answers in polynomial time.

These attractive results motivate us to look for a compact characterization of total unimodularity of $M(S)$. We already have one such a characterization, since Lemma 2.4 implies that $M(S)$ is t.u. if and only if $M(S)$ has no odd holes. Also, there is a cubic algorithm for testing total unimodularity for general matrices [12]. The next section supplies a characterization that provides better insight and supports more efficient testing.

3. Total Unimodularity

This section characterizes total unimodularity of $M(S)$ and provides a linear algorithm for deciding whether that property is present. We begin with a small case where S is derived from one 3IFF clause where no variable occurs negated. That is, the 3IFF clause is $(a \vee b) \leftrightarrow c$ where a, b, and c are the variables. The equivalent 3IFFCNF clauses are

$$\begin{aligned} &\neg a \vee c \\ &\neg b \vee c \\ &a \vee b \vee \neg c \end{aligned} \tag{2}$$

The matrix $M(S)$ for this case is

$$M(S) = \begin{array}{c} \\ \left[\begin{array}{rrr} -1 & 0 & 1 \\ 0 & -1 & 1 \\ 1 & 1 & -1 \end{array}\right] \end{array} \begin{array}{c} \\ a \\ b \\ c \end{array} \tag{3}$$

with columns labelled a, b, c.

Note that we use the variables to index both the columns and rows of the matrix. Thus, we may say "column a" or "row a" of $M(S)$ without risk of confusion. In the general case of S representing a single 3IFF clause, some variables are replaced by their complements. The replacement by complements corresponds to scaling of some columns of $M(S)$ of (3) by -1. Thus, $M(S)$ becomes the matrix

$$M(S) = \begin{array}{c|ccc|c} & a & b & c & \\ \hline & x' & 0 & z' & a \\ & 0 & y' & z'' & b \\ & x & y & z & c \end{array} \tag{4}$$

where the various symbols denote $\{\pm 1\}$ entries obeying the following relationships.

$$\begin{aligned} x &= -x' \\ y &= -y' \\ z &= -z' = -z'' \end{aligned} \tag{5}$$

We use z' and z'' even though they have identical value, since later we need to refer to specific entries. Direct checking confirms the following equations implied by (5).

$$\begin{aligned} (x+z)(\text{mod } 4) &= (\mathrm{x}' + \mathrm{z}')(\text{mod } 4) \\ (y+z)(\text{mod } 4) &= (\mathrm{y}' + \mathrm{z}'')(\text{mod } 4) \\ (x+y+2)(\text{mod } 4) &= (\mathrm{x}' + \mathrm{y}' + \mathrm{z}' + \mathrm{z}'')(\text{mod } 4) \end{aligned} \tag{6}$$

It is easy to check that $M(S)$ of (3) is t.u. By Lemma 2.5, this also holds for $M(S)$ of (4).

In the general case, any number of 3IFF clauses define S. For 3IFF clause i, we assume the variables to be a_i, b_i and c_i. The three corresponding rows of $M(S)$ are indexed by these variables, as are the related columns. Note that this may produce several column labels, since a variable may occur in several 3IFF clauses. But that fact shall not trouble us. The 3×3 submatrix M_i of $M(S)$ representing the ith 3IFF clause has the form of (4), except that the column and row labels have a subscripted i. We display M_i since we use that matrix repeatedly.

$$M_i = \begin{array}{c} \\ \\ \\ \\ \end{array}\begin{array}{ccc|c} a_i & b_i & c_i & \\ \hline x' & 0 & z' & a_i \\ 0 & y' & z'' & b_i \\ x & y & z & c_i \end{array} \tag{7}$$

We call M_i *block* i of $M(S)$.

We define cutnodes of $G(S)$ that are of particular interest.

Definition 3.1. Let a, b, and c be three nodes of a graph. The node c is an a/b *cutnode* if removal of c from the graph disconnects the nodes a and b. Two nodes (resp. edges) of a graph are in *distinct* 2-connected components of the graph if there is no 2-connected component that contains both nodes (resp. edges).

Lemma 3.1. *Assume that one of the* 3IFF *clauses involved in the definition of* S *has input variables* a *and* b *and conclusion variable* c. *Then the node* c *is an* a/b *cutnode of* $G(S)$ *if and only if the nodes* a *and* b *are in distinct* 2*-connected components of* $G(S)$.

Proof. Suppose c is an a/b cutnode of $G(S)$. If the nodes a and b are in the same 2-connected component, then removal of any node from that component cannot disconnect any two remaining nodes, in particular a and b.

Suppose that the nodes a and b are in distinct 2-connected components of $G(S)$. Since c is the conclusion variable, the two 2CNF clauses produced by the 3IFF clause create in $G(S)$ an edge connecting nodes a and c and a second edge connecting nodes b and c. Since the nodes a and b are in distinct 2-connected components, so must be the two edges. This is possible only if c is a cutpoint. □

Definition 3.2. The *cutnode condition* holds for $G(S)$ if for each 3IFF clause involved in the definition of S, say with input variables a and b and conclusion variable c, the node c is an a/b cutnode of $G(S)$.

Next, we investigate holes of $M(S)$.

Lemma 3.2. *If a hole of* $M(S)$ *uses the entry in column* c_i *and row* c_i *of block* i, *then the hole can be changed to another hole of same parity using the first row* a_i *or the second row* b_i *of the block* i *instead of row* c_i.

Proof. We assume that the nonzero entries of block i are x, y, z, x', y', z', z'' as depicted in (7). Thus, the lemma states that the given hole of $M(S)$ contains the entry z of block i.

Due to symmetry, we may assume that the hole uses in row c_i the entry x besides the entry z. Instead of row c_i, we now use row a_i to define the hole. This change replaces the entries x and z by x' and z' of the row a_i of block i. Since row a_i of block i has just two nonzero entries, the resulting submatrix of $M(S)$ must also be a hole. By (6), $(x + z)(\text{mod } 4) = (\mathrm{x}' + \mathrm{z}')(\text{mod } 4)$, so the two holes must have the same parity. □

Definition 3.3. The row c_i of the assumed hole of Lemma 3.2 is a *type* I row of the hole, and the exchange of rows is a *type* I *exchange.*

Lemma 3.3. *If a hole has no type* I *rows, but contains, for some block i, the two entries x and y of row c_i, then the hole can be changed to a hole of opposite parity by deleting row c_i and instead using rows a_i and b_i and column c_i of the block.*

Proof. We first argue that replacement of the row c_i by the rows a_i and b_i and the column c_i results in a hole. Indeed, the column c_i cannot be part of the original hole since otherwise the row c_i would induce three nonzero entries in the hole, a contradiction. But conceivably another row of the hole, taken from another block, may have a nonzero entry in column c_i. Thus, that row of $M(S)$ has three nonzeros. By assumption, the corresponding row of the hole is not of type I. Thus, the row of $M(S)$ must have variable c_i as conclusion variable. But Definition 2.3 rules out such duplicate use of variable c_i as conclusion variable.

To establish the change of parity, we use the notation of M_i of (7). Thus, the two entries x and y of row c_i are replaced by the four entries x', y', z', z'' of rows a_i and b_i of block i. By (6), $x + y + 2 (\text{mod } 4) = \mathrm{x}' + \mathrm{y}' + \mathrm{z}' + \mathrm{z}'' (\text{mod } 4)$. Thus, the parity of the two holes differs as claimed. □

Definition 3.4. The row c_i of the given hole of Lemma 3.3 is a *type* II row of the hole, and the exchange of rows and addition of column c_i is a *type* II *exchange.*

We are ready for the characterization of t.u. matrices $M(S)$.

Theorem 3.1. *Let S be a* 3IFFCNF *formula. Then M(S) is t.u. if and only if $G(S)$ is balanced and satisfies the cutnode condition.*

Proof. Assume that $M(S)$ is t.u. By Lemma 2.4, $G(S)$ is balanced. Suppose for some block i, the nodes a_i and b_i are in one 2-connected component of $G(S)$. Hence, some path connects a_i and b_i while avoiding node c_i. That path plus the two edges of $G(S)$ connecting node c_i with the nodes a_i and b_i form a cycle of $G(S)$. Since $M(T)$ is t.u., the corresponding hole of $M(T)$ is balanced. We carry out the inverse of a type II exchange, where the rows a_i and b_i are replaced by row c_i, and where column c_i is deleted. According to Lemma 3.3, the resulting hole is odd, a contradiction of the total unimodularity of $M(S)$.

For proof of the converse, assume balancedness of $G(S)$ and the cutnode condition. If $M(S)$ is not t.u., then by Lemma 2.4 it has an odd hole. Due to the parity of the hole and the general structure of holes, the hole contains, for any block i, nonzero entries of at most two rows of the block; in the case of two rows, the entries must be from rows a_i and b_i. Since $G(S)$ is balanced, the odd hole cannot correspond to a cycle of $G(S)$ and thus must have a nonzero number of type I and/or type II rows. If there are type I rows, we eliminate them by repeated application of Lemma 3.2 and get an odd hole without type I rows. If that hole corresponds to a cycle of $G(S)$, then we have a contradiction of the balancedness of $G(S)$. Thus, the odd hole has no type I rows and at least one type II row. We now exchange each type II row as described in Lemma 3.3. Eventually we get a hole that corresponds to a cycle of $G(S)$. The parity of the resulting hole depends on the number of type II exchanges. But the parity does not matter here. Indeed, upon termination of the exchange process, we have a hole corresponding to a cycle of $G(S)$ where for at least one block i, the nodes

a_i, c_i, b_i occur consecutively on the cycle. Thus, c_i is not an a_i/b_i cutnode. By Lemma 3.1, the cutnode condition does not hold for $G(S)$, a contradiction. □

Theorem 3.1 supports the following linear-time algorithm for testing total unimodularity of $M(S)$.

1. Use Lemma 2.2 and breadth-first search to decide bipartiteness of the graph derived from $G(S)$ by contraction of all regular edges. If $G(S)$ is not bipartite, $M(S)$ is not t.u., and we stop.
2. Employ the linear-time algorithm of [11] to determine the 2-connected components of $G(S)$ to verify the cutnode condition. If the condition is not satisfied, $M(S)$ is not t.u., and we stop.
3. Declare $M(S)$ to be t.u.

The following structural result can be established for t.u. matrices $M(S)$ and may be used to characterize unsatisfiability and to solve minimum cost versions of SAT by network flow techniques. For details of the latter techniques, see, for example, [1].

Define I to be an identity matrix of appropriate size, and let the superscript t denote the transpose operation. A *pivot* in a given matrix consists of elementary row operations that convert one column to a unit vector.

A *network matrix* is an integer $\{0, \pm 1\}$ matrix with at most two nonzeros in each column. In the case of two nonzeros, they have opposite sign.

Theorem 3.2. *If $M(S)$ is t.u., then the matrix $\left[I \mid M(S)^t \right]$ can by row scaling and pivots be transformed to a network matrix.*

Proof. The arguments use the closed k-sum of Section 5 and matrix/matroid techniques of [14]. Due to space constraints, we omit details. □

The requirement of balancedness of $G(S)$ is rather severe. For example, if S contains the two, rather innocuous, clauses $a \vee b$ and $\neg a \vee b$, then these two clauses already produce an odd cycle in $G(S)$. Not quite so stringent is the cutnode condition. In the next section, it is shown that the cutnode condition, by itself, implies a linear-time satisfiability algorithm for S.

As an aside, the algorithm to come is of no use for the ONE-IN-THREE 3SAT problem. Even if an instance of that problem has just one clause, say $a \vee b \vee c$, where a, b, and c are the variables, then the equivalent 3IFFSAT instance has the clauses $a \vee b \vee c$, $\neg a \vee \neg b$, $\neg a \vee \neg c$, and $\neg b \vee \neg c$. Accordingly, $G(S)$ is a cycle with three edges, the nodes a and b are in the same 2-connected component of $G(S)$, and the cutnode condition does not hold.

4. Satisfiability Test and Characterization of Unsatisfiability

This section describes, for the instances S of 3IFFSAT observing the cutnode condition, a linear-time algorithm that for satisfiable S produces a satisfying solution and for unsatisfiable S computes a compact characterization of unsatisfiability. We recall the cutnode condition for S. It demands that, for each block i of $M(S)$, the node c_i of $G(S)$ is an a_i/b_i cutnode of $G(S)$.

The characterization of unsatisfiability produced by the algorithm builds upon the characterization of unsatisfiability for 2CNF formulas S of [2] and uses the directed graph $H(S)$ of Definition 2.8(b). Recall that in $H(S)$, each literal of the given 2CNF formula corresponds to a node. Each clause $A \vee B$ of the formula produces the arcs $(\neg A, B)$ and $(\neg B, A)$, while each unit clause generates the arc $(\neg A, A)$. The clauses with at least three literals are ignored in the construction.

Theorem 4.1. [2] *A* 2CNF *formula S is unsatisfiable if and only if, for some variable r of S, the graph $H(S)$ has a directed path from node r to node $\neg r$ and a second directed path from node $\neg r$ to node r. The variable r and the two paths can be found in linear time.*

Theorem 4.1 has the following corollary.

Corollary 4.1. *Suppose a* 2CNF *formula S is satisfiable. Let S' be obtained from S by adding, for some variable t, the unit clause $\neg t$. Then S' is unsatisfiable if and only if $H(S)$ has a directed path from node $\neg t$ to node t. The path can be found in linear time.*

Proof. By Theorem 4.1, $H(S')$ has, for some node r, directed paths from r to $\neg r$ and from $\neg r$ to r. These paths can be found in linear time. The arc $(t, \neg t)$ corresponding to the unit clause $\neg t$ must be part of the paths, since otherwise the paths prove S to be unsatisfiable, a contradiction. Then it is trivial to extract a directed path from $\neg t$ to t from the two paths. □

Unsatisfiability of S' of Corollary 4.1 is equivalent to the fact that in all satisfying solutions of S, we necessarily have $t = \mathit{True}$.

The characterization of unsatisfiable 3IFFCNF formulas S observing the cutnode condition involves an extension of the two directed paths of Theorem 4.1 that we refer to as *collection Q*. The collection consists of directed paths and logic implications. In general, the collection Q is recursively constructed as follows. Initially, there are two directed paths P_1 and P_2 just as in the case of unsatisfiable 2CNF formulas. Thus, P_1 and P_2 have common endpoints, say r and $\neg r$, and one of the paths goes from r to $\neg r$, while the other one goes from $\neg r$ to r. In the recursive construction step, a directed arc $(\neg C, C)$ of the collection Q on hand is replaced by a directed path whose direction is in conformance with the arc direction; or the path $P(A, \neg A)$ and the implication $\neg A \to (C \to B)$ are assigned to whichever arc (C, B) or $(\neg B, \neg C)$ is present in Q. We call the pair $(P(A, \neg A), \neg A \to (C \to B))$ an *assigned path/implication* pair.

The interpretation of the path $(A, \neg A)$ and the implication $\neg A \to (C \to B)$ is as follows. If $A = \mathit{True}$ is assumed, then the clauses represented by the path $P(A, \neg A)$ force $A = \mathit{False}$. Thus, $A = \mathit{False}$ must hold. Based on that conclusion, the implication $\neg A \to (C \to B)$ justifies the implication $C \to B$ and the equivalent $\neg B \to \neg C$. The latter fact then validates presence of whichever arcs (C, B) and $(\neg B, \neg C)$ occur in Q.

The collection Q is such that each arc without assigned path/implication pair corresponds to a clause of S. Furthermore, each assigned implication is equivalent to a clause of S with three literals.

Theorem 4.2. *A collection Q as defined above exists if and only if S is unsatisfiable.*

Proof. Assume that Q exists. If P_1 or P_2 has at least one arc with assigned path/implication pair, then there must be an assigned pair $(P(A, \neg A), \neg A \to (C \to B))$ where the arcs of $P(A, \neg A)$ themselves have no assigned pair. As argued above, the pair $(P(A, \neg A), \neg A \to (C \to B))$ justifies existence of the arc(s) to which it is assigned. We now erase the pair $(P(A, \neg A), \neg A \to (C \to B))$ associated with those arcs. Carrying out this process recursively, eventually we arrive at the situation where each arc of the two paths P_1 and P_2 either has been justified or represents an original 2CNF clause of S. Thus, the contradictory conclusions by the two paths prove S to be unsatisfiable.

We prove the converse below by constructing a collection Q of the prescribed form when a 3IFFCNF formula S is found to be unsatisfiable. □

In the recursive part of the algorithm, we repeatedly fix some variables to *True/False* values. Formally, we accomplish this by adding unit clauses. Below, we refer to a *node of a block*, meaning a node corresponding to a column of a block.

We are ready for the description of the algorithm. Comments are added in parentheses. We suppose that the given formula S cannot be partitioned into subformulas with disjoint variable sets, since otherwise the algorithm can be applied to the subformulas one-by-one until either a subformula turns out to be unsatisfiable or all subformulas have been proved to be satisfiable.

1. If S is not a 2SAT instance, then determine the 2-connected components of $G(S)$ with the linear-time algorithm of [11].

(Steps 2–5 constitute the recursive part of the algorithm.)

2. If S is a 2SAT instance: Solve the instance with the linear-time 2SAT algorithm of [6].

   If S satisfiable: Begin backtracking with the satisfying solution.

   If S unsatisfiable: Construct $H(S)$. Use the linear-time algorithm of Theorem 4.1 to initialize the collection Q as two directed paths $P_1 = P(r, \neg r)$ and $P_2 = P(\neg r, r)$ of $H(S)$. Begin backtracking with this Q.

3. (S is not a 2SAT instance. By the cutnode condition, $G(S)$ is not 2-connected. By induction, the 2-connected components of $G(S)$ are on hand.)

   The 2-connected components are joined in a tree structure. Arbitrarily select a leaf of the tree, define G_1 be the 2-connected subgraph of $G(S)$ corresponding to that leaf, and let G_2 be the subgraph of $G(S)$ represented by the remaining edges of the tree. Thus, identification of one node of G_1, say c, with node c of G_2 creates $G(S)$.

   If G_1 minus node c, denoted by $G_1 - \{c\}$, contains a node of a block, say indexed by i, then go to Step 4. Otherwise, go to Step 5.

4. By the cutnode condition, the node of block i occurring in $G_1 - \{c\}$ cannot be c_i and thus must be a_i or b_i. By symmetry we may suppose that it is a_i. Furthermore, c_i must be the node c, and b_i must be a node of $G_2 - \{c\}$. Due to scaling, we also may suppose that the 3CNF clause associated with block i is $a_i \vee b_i \vee \neg c_i$.

Define S_1 to contain the clauses of S all of whose variables are represented by nodes of G_1, and declare S_2 to have the remaining clauses of S save the clause $a_i \vee b_i \vee \neg c_i$. Due to this definition, S_1 is a 2SAT instance, and all variables of the clauses of S_2 occur as nodes in G_2.

We now drop the index i and denote the nodes and variables a_i, b_i, and c_i just by a, b, and c. The logic clauses connected with the block are shown in (2).

With the linear-time 2SAT algorithm of [6], we solve three 2SAT cases derived from S_1 by fixing the variables a and c as follows.

Case 1: $a = \mathit{True}$, $c = \mathit{True}$

Case 2: $a = \mathit{False}$, $c = \mathit{False}$

Case 3: $a = \mathit{False}$, $c = \mathit{True}$

We have several subcases, depending on the outcomes for the three instances defined by Cases 1–3. In each subcase, we define a formula S'_2 from S_2.

(We proceed roughly as follows. Recursively, we compute a satisfying solution for S'_2 or determine S'_2 to be unsatisfiable. In the former case, we combine the satisfying solution for S'_2 with one of the three solutions computed for Cases 1–3 above, and get a solution for S that satisfies the clause $a \vee b \vee \neg c$ of (2). In the latter case, the construction of S'_2 implies that S is unsatisfiable. We assume that the recursive processing of S'_2 produces a collection Q'_2 that characterizes the unsatisfiability of S'_2.)

(In the detailed discussion of the subcases below, we explicitly define S'_2 and show that the solution for S'_2 plus a solution selected for S_1 satisfy the clause $a \vee b \vee \neg c$. But we omit repetition of the statement that we solve S'_2 recursively, that the combined solutions constitute a satisfying solution of S, and that unsatisfiability of S'_2 implies unsatisfiability of S and produces a collection Q'_2 characterizing that situation.)

4.1 Cases 1 and 2 satisfiable: Define $S'_2 = S_2$.

If S'_2 is satisfiable: If the satisfying solution for S'_2 has $c = \mathit{True}$ (resp. $c = \mathit{False}$), then the solution for Case 1 (resp. Case 2) is combined with that for S'_2. Due to the values defined in Cases 1 and 2 for a and c, the clause $a \vee b \vee \neg c$ is satisfied.

If S'_2 is unsatisfiable: Since $S'_2 = S_2$, the collection Q'_2 characterizes the unsatisfiability not just of S'_2 but also of S_2 and S. Hence, the collection $Q = Q'_2$ characterizes the unsatisfiability of S.

4.2 Case 1 satisfiable, Case 2 unsatisfiable: In S_1, the clause $\neg a \vee c$ of (2) rules out the case $a = \mathit{True}$, $c = \mathit{False}$. Thus, the outcomes for Cases 1 and 2 imply that in any satisfying solution of S_1, necessarily $c = \mathit{True}$. Accordingly, we define S'_2 to be S_2 with an added unit clause forcing $c = \mathit{True}$.

If S'_2 is satisfiable: We combine the solution for S'_2 with the solution for Case 1 of S_1. Since Case 1 has $a = \mathit{True}$, the clause $a \vee b \vee \neg c$ is satisfied.

If S'_2 is unsatisfiable: If the arc $(\neg c, c)$ occurs in Q'_2, then we construct the graph $H(S_1)$ and use the linear-time algorithm of Corollary 4.1 to find a directed path $P(\neg c, c)$ in the graph $H(S_1)$, and replace in Q'_2 the arc $(\neg c, c)$ by the path $P(\neg c, c)$ to get a collection Q for S. If the arc $(\neg c, c)$ does not occur in Q'_2, we declare $Q = Q'_2$.

4.3 Case 1 unsatisfiable, Case 2 satisfiable, Case 3 satisfiable: Arguing as in Step 4.2, we must have $a = \mathit{False}$ in any satisfying solution of S_1 or S. We use that value to reduce the clause $a \vee b \vee \neg c$ of (2), getting $b \vee \neg c$. We define S_2' to be S_2 with the added clause $b \vee \neg c$.

If S_2' is satisfiable: If the solution specifies $c = \mathit{True}$ (resp. $c = \mathit{False}$), then we combine that solution with the one of Case 3 (resp. Case 2) of S_1. Due to the specification of $b \vee \neg c$ in S_2', the solution satisfies the clause $a \vee b \vee \neg c$.

If S_2' is unsatisfiable: If the arc $(\neg b, \neg c)$ or (c, b) occurs in Q_2', then we use the linear-time algorithm of Corollary 4.1 to find a directed path $P(a, \neg a)$ in the graph $H(S_1)$. To the arcs $(\neg b, \neg c)$ and (c, b), whichever occur in Q_2', we assign the path $P(a, \neg a)$ and the implication $\neg a \rightarrow (c \rightarrow b)$. This produces Q for S.

4.4 Case 1 unsatisfiable, Case 2 satisfiable, Case 3 unsatisfiable: We must have the values $a = c = \mathit{False}$ of Case 2 in any satisfying solution of S_1 or S. Accordingly, we add a unit clause enforcing $c = \mathit{False}$ to S_2 to obtain S_2'.

If S_2' is satisfiable: We combine its solution with that of Case 2 for S_1. Since $c = \mathit{False}$, the clause $a \vee b \vee \neg c$ is satisfied.

If S_2' is unsatisfiable: This is handled like the case of unsatisfiable S_2' in Step 4.2, except that c plays the role of $\neg c$ and *vice versa*.

4.5 Cases 1 and 2 unsatisfiable, Case 3 satisfiable: The values $a = \mathit{False}$, $c = \mathit{True}$ of Case 3 would allow us to reduce $a \vee b \vee \neg c$ to the unit clause b. Instead, we use the two equivalent, seemingly more cumbersome, clauses c and $b \vee \neg c$. We make this choice since it simplifies the treatment of unsatisfiability, as will be seen below. Add the two clauses c and $b \vee \neg c$ to S_2 to obtain S_2'.

If S_2' is satisfiable: We combine the solution with that of Case 3 of S_1. Since b must have the value *True*, the clause $a \vee b \vee \neg c$ is satisfied.

If S_2' is unsatisfiable: Update Q_2' as described in Steps 4.2 and 4.3. This produces Q for S.

4.5 Cases 1–3 unsatisfiable: Then S_1 and hence S are unsatisfiable. Initialize the collection Q as done for the unsatisfiable case of Step 2, except that S_1 here plays the role of S of Step 2. Begin backtracking with this Q.

5. Since the node c is not the conclusion of a 3IFF clause, the procedure is a simplified version of the above steps. That is, S_1 is solved twice, using first $c = \mathit{True}$ and then $c = \mathit{False}$, and the four possible outcomes are processed analogously to the above procedures.

The proof of the linear time bound of the algorithm follows directly from Steps 2–5, except for the claim made in Step 3 that the 2-connected components are on hand for $G(S)$. We examine that claim. Step 1 computes the 2-connected components for the original S. By induction, in each of the Steps 4 and 5, the 2-connected components of $G(S_2)$ are available, and it suffices that we show that the 2-connected components of $G(S_2')$ can be obtained from those of $G(S_2)$ in constant time. Indeed, except for Step 4.3 and 4.5, all steps add at most unit clauses to S_2 to get S'_2, and $G(S'_2) = G(S_2)$. In Steps 4.3 and 4.5, the clause $b \vee \neg c$ is added to S_2 to create S'_2. Since S_2 contains the clause $\neg b \vee c$ of (2), the additional edge representing $b \vee \neg c$ is parallel to the edge for $\neg b \vee c$. Thus, the 2-connected components of $G(S'_2)$ are the 2-connected components of $G(S_2)$ save for the addition of a parallel edge in one 2-connected component.

We should point out that the unsatisfiable Q obtained by the algorithm may not be minimal. The root cause is the fact that the configuration of the two paths produced by the linear algorithm of Theorem 4.1 need not be minimal. Of course, minimality of Q can be achieved by iteratively removing one clause of Q at a time, checking satisfiability, and reinstating the clause if the reduced case is satisfiable. As an alternative, in the recursive construction of Q, we can use instead of the linear algorithm of Theorem 4.1 the polynomial algorithm (6.3.6) of [13], which finds minimal unsatisfiable 2CNF formulas and describes them in terms of cycles and so-called ladders of a certain graph. Details of the cases are provided in Corollary (6.3.5) of the cited reference. The description of the minimal unsatisfiable subformula in terms of cycles and ladders may be useful for interpreting the unsatisfiability in the biological unit. We should mention that Corollary (6.3.5) of [13] assumes that the given unsatisfiable 2CNF formula has no unit clauses. But if there are unit clauses, the formula is easily transformed to a slightly larger, equivalent one having no unit clauses.

The decomposition used in the processing of 3IFFCNF formulas satisfying the cutnode condition is a special case of a logic decomposition called *closed sum*. The next section provides details.

5. Closed Sum

We show that the class of 3IFFCNF formulas observing the cutnode condition is a subset case of a class of satisfiability instances that is recursively constructed from 2CNF formulas and single CNF clauses by the *closed sum* operation of [13]. The instances of the class can be solved in polynomial time. Unfortunately, characterization of unsatisfiability of the instances is rather cumbersome, and we do not cover that aspect here.

The closed sum is based on a concept of *Boolean closed*, for short *closed*, integer $\{0, \pm 1\}$ matrices. A review of the definition, properties, and various characterizations of closed matrices is beyond the scope of this paper. Instead, we turn a part of Theorem (7.3.5) of [13], which characterizes closed matrices in various ways, into a definition. We say that a given matrix D *contains a column-scaled version of a matrix* N if column scaling with $\{\pm 1\}$ factors and reordering of columns and rows can turn N into a submatrix of D.

Definition 5.1. (based on Theorem (7.3.5) of [13]) A matrix D is *closed* if it does not contain any column-scaled version of any one of the matrices N_1–N_4 below.

$$N_1 = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \tag{8}$$

$$N_2 = \begin{bmatrix} 1 & 0 \\ 1 & 1 \\ -1 & 1 \end{bmatrix} \tag{9}$$

$$N_3 = \begin{bmatrix} 1 & 1 \\ 1 & -1 \\ -1 & 1 \end{bmatrix} \tag{10}$$

$$N_4 = \begin{bmatrix} 1 & 1 \\ -1 & 1 \\ -1 & -1 \end{bmatrix} \tag{11}$$

Of particular interest are the $\{0, \pm 1\}$ closed matrices that, when viewed over the ternary field GF(3), have GF(3)-rank of at most 2. These matrices can be characterized as follows.

Theorem 5.1. (Theorem (7.4.8) of [13]) *Let D be a $\{0, \pm 1\}$ matrix, considered to be integer or over* GF(3) *as appropriate.*

(a) *If* GF(3)-rank$(D) \leq 1$, *then D is closed.*

(b) *If* GF(3)-rank$(A) = 2$, *then D is closed if and only if column scaling followed by deletion of duplicate columns and rows can reduce D to a matrix that has* GF(3)-rank *equal to* 2 *and that is a submatrix of one of the matrices F^1–F^3 below.*

$$F_1 = \begin{bmatrix} 1 & -1 & 0 \\ 1 & 1 & 1 \end{bmatrix} \tag{12}$$

$$F_2 = \begin{bmatrix} -1 & 0 \\ 1 & -1 \\ 1 & 1 \\ 1 & 0 \end{bmatrix} \tag{13}$$

$$F_3 = \begin{bmatrix} -1 & 0 \\ -1 & -1 \\ 1 & 1 \\ 1 & 0 \end{bmatrix} \tag{14}$$

For any integer $k \geq 1$, define a $\{0, \pm 1\}$ matrix M to be a *closed k-sum* if

$$M = \begin{bmatrix} M_1 & 0 \\ D & M_2 \end{bmatrix} \tag{15}$$

where the submatrix D is closed and has GF(3)-rank equal to $k-1$. The submatrices $\left[\frac{M_1}{D}\right]$ and $[D \mid M_2]$ are the *components* of the closed k-sum. Note that any one of the submatrices of M may have no columns or no rows.

Let $\mathcal{M}$ be a class of $\{0, \pm 1\}$ matrices that is closed under submatrix taking. That is, if a matrix is in $\mathcal{M}$, then every submatrix of the matrix is in $\mathcal{M}$ as well. Further, assume that two polynomial algorithms exist, one for determining membership in $\mathcal{M}$, and the second one for the satisfiability problem of the formulas S whose representation matrix is in $\mathcal{M}$. When these conditions hold for $\mathcal{M}$, then the class $\mathcal{M}$ is SAT *central* [13].

Define $\mathcal{C}$ to be the class of matrices created from the matrices of a given class $\mathcal{C}_0$ by recursive closed k-sum steps, $k \geq 1$, where in each sum operation one component is taken from $\mathcal{C}_0$ while the second component is also taken from $\mathcal{C}_0$ or has been created so far by the recursive process. We say that the class $\mathcal{C}$ is *created from* $\mathcal{C}_0$ *by repeated closed* k*-sum steps.*

In [13], the following result is established for $\mathcal{C}$.

Theorem 5.2. (Theorem (10.4.21) of [13]) *Let* $\mathcal{C}_0$ *be a* SAT *central class of matrices. Define* $\mathcal{C}$ *to be the class of matrices created from* $\mathcal{C}_0$ *by repeated closed* k*-sum steps with* $k \leq 3$*. Then* $\mathcal{C}$ *is* SAT *central.*

The corollary of Theorem 5.2 included next establishes a SAT central class $\mathcal{C}$ from a $\mathcal{C}_0$ that is based on 2CNF formulas and single CNF clauses.

Corollary 5.1. *Let* $\mathcal{S}_0$ *be the class of* CNF *formulas each of which is a* 2CNF *formula or a single* CNF *clause. Define* $\mathcal{C}_0 = \{M(S) \mid S \in \mathcal{S}_0\}$*, and declare* $\mathcal{C}$ *to be the class of matrices created from* $\mathcal{C}_0$ *by repeated closed* k*-sum steps with* $k \leq 3$*. Then* $\mathcal{C}$ *is* SAT *central.*

Proof. Due to the linear-time algorithm of [6] for the 2CNF instances, the class $\mathcal{C}_0$ is SAT central. By Theorem 5.2, the class $\mathcal{C}$ is SAT central as well. □

In a moment, we also need the following lemma.

Lemma 5.1. *Let* $\mathcal{C}$ *be created from a given class* $\mathcal{C}_0$ *by repeated closed* k*-sums,* $k \leq 3$*. Suppose each* $\{0, \pm 1\}$ *column or row vector is in* $\mathcal{C}_0$*. If a zero or unit vector is added as column or row to any matrix of* $\mathcal{C}$*, then the enlarged matrix is also in* $\mathcal{C}$*.*

Proof. Any such an extension can be viewed as a closed 1- or 2-sum where one component is the given matrix of $\mathcal{C}$ and the second component is just one column or row vector. □

Here is the main result of this section.

Theorem 5.3. *Let* S *be a* 3IFFCNF *formula for which the cutnode condition holds. Then the matrix* $M(S)$ *is in the class* $\mathcal{C}$ *of Corollary 5.1.*

Proof. In the nontrivial case of the inductive proof, S is not a 2CNF formula. In the satisfiability algorithm of Section 4, such S is decomposed in Step 4 or 5. We examine the more complex case of Step 4 and leave that of Step 5 to the reader. Recall that each clause of S save the clause $a \vee b \vee \neg c$ is assigned to either S_1 or S_2 such that the variables a and c are in S_1, the variables b and c are in S_2, and the variable c is the single variable common to S_1 and S_2. Derive $\overline{S}_2$ from S_2 by deleting all literals of the variable c occurring in the clauses of S_2. Define M_1 to be $M(S_1)$, and let M_2 be $M(\overline{S}_2)$ plus one additional row unit vector representing the subclause b of $a \vee b \vee \neg c$. Declare D to be an appropriately sized matrix whose nonzeros represent the subclause $a \vee \neg c$ of $a \vee b \vee \neg c$ and the literals c and $\neg c$ dropped a moment ago from S_2. Suitable column scaling and deletion of duplicate rows and columns reduce D to a submatrix of the following matrix D'.

$$D' = \begin{bmatrix} 1 & 1 \\ 1 & 0 \\ -1 & 0 \end{bmatrix} \tag{16}$$

Now D' is a submatrix of F_3 of (14), and D is closed by Theorem 5.1. Accordingly, the matrix M of (15) created by the just-defined M_1, M_2, and D is a closed 2- or 3-sum.

Since S_1 is a 2CNF formula, and since each row of D has at most two nonzeros, the component $\left[\frac{M_1}{D}\right]$ of the closed sum represents a 2CNF formula and thus is in $\mathcal{C}_0$.

From the second component $[D \mid M_2]$, we delete all zero columns of D and the column unit vector defined by variable a, getting a matrix $\left[\overline{D} \mid M_2\right]$ that represents a 3IFFCNF formula $\overline{S}$ smaller than S. By induction, $\left[\overline{D} \mid M_2\right]$ is in $\mathcal{C}$, and by Lemma 5.1, $[D \mid M_2]$ is in $\mathcal{C}$ as well. Then the closed sum M is in $\mathcal{C}$, as desired.

□

The construction of SAT central matrix classes via closed k-sums with $k \leq 3$ supports several generalizations of the 3IFFSAT formulas observing the cutnode condition. For example, instead of two input variables for each 3IFF clause, we may consider any number of input variables and thus may have implications of the form $(\vee_{j \in J} A_j) \leftrightarrow C$ for any $|J| \geq 2$. In the introduction it was argued that this case can always be reduced to 3IFF clauses. That is correct, but it is easily seen that, if the ultimately resulting 3IFFCNF formula is to obey the cutnode condition, then the original formula must observe a very severe condition. Instead, we can treat that case directly by closed k-sums, using for the definition of the class $\mathcal{C}_0$ single CNF clauses and 2CNF formulas that possibly are augmented by exactly one clause with at least three literals. The class $\mathcal{C}_0$ is SAT central, so the closed k-sum construction with $k \leq 3$ can still be carried out.

For the definition of another SAT central class $\mathcal{C}_0$, define a CNF formula to be *Horn* if each clause has at most one nonnegated variable, and to be *hidden Horn* if suitable complementation of variables produces a Horn formula. Then $\mathcal{C}_0$ consists of the matrices of formulas that after deletion of at most one clause become hidden Horn.

The linearly solvable case defined in Theorem 2.1 leads to yet another SAT central class $\mathcal{C}_0$. For additional classes, see [13].

References

[1] Ravindra K. Ahuja, Thomas L. Magnanti, and James B. Orlin. *Network Flows: Theory, Algorithms, and Applications.* Prentice Hall, 1993.

[2] Bengt Aspvall, Michael F. Plass, and Robert E. Tarjan. A linear-time algorithm for testing the truth of certain quantified boolean formulas. *Information Processing Letters*, **8**:121–123, 1979.

[3] J.A. Bondy and U.S.R. Murty. *Graph Theory with Applications.* Elsevier North-Holland, 1976.

[4] Vijay Chandru and John Hooker. *Optimization Methods for Logical Inference.* John Wiley & Sons, 1999.

[5] Keith L. Clark. Negation as failure. In *Logic and Databases*, pages 293–322. Plenum Press, 1978.

[6] S. Even, A. Itai, and A. Shamir. On the complexity of timetable and multicommodity flow problems. *SIAM Journal on Computing*, **5**:691–703, 1976.

[7] Michael R. Garey and David S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman & Co, 1979.

[8] Anja Remshagen and Klaus Truemper. An effective algorithm for the futile questioning problem. *Journal of Automated Reasoning*, **34**:31–47, 2005.

[9] Julio Saez-Rodriguez, Luca Simeoni, Jonathan A. Lindquist, Rebecca Hemenway, Ursula Bommhardt, Boerge Arndt, Utz-Uwe Haus, Robert Weismantel, Ernst D. Gilles, Steffen Klamt, and Burkhart Schraven. A logical model provides insights into T cell receptor signaling. *PLoS Computational Biology*, **3**(8):1580–1590, August 2007.

[10] Alexander Schrijver. *Theory of Linear and Integer Programming*. John Wiley & Sons, 1986.

[11] Robert Tarjan. Depth-first search and linear graph algorithms. *SIAM Journal on Computing*, **1**:146–160, 1972.

[12] Klaus Truemper. A decomposition theory for matroids. V. Testing of matrix total unimodularity. *Journal of Combinatorial Theory Series (B)*, **49**:241–281, 1990.

[13] Klaus Truemper. *Effective Logic Computation*. John Wiley & Sons, 1998.

[14] Klaus Truemper. *Matroid Decomposition (revised edition)*. Leibniz, 1998.