DIMACS Technical Report 97-34
July 1997

Horn Minimization by Iterative Decomposition[1]

by

Endre Boros[2,3] Ondřej Čepek[4] Alexander Kogan[5,6]

[1]The authors gratefully acknowledge the partial support by the Office of Naval Research (grants N00014-92-J1375 and N00014-92-J4083). The first author also grateful for the partial support by NSF (grant INT 9321811) and NATO (grant CRG 931531). The second author thanks the travel support provided by DIMACS, the National Science Foundation Center for Discrete Mathematics and Theoretical Computer Science.

[2]Permanent Member

[3]RUTCOR, Rutgers University, P.O. Box 5062, New Brunswick, NJ 08903, USA (boros@rutcor.rutgers.edu)

[4]Dept. of Theoretical Informatics, Charles University, Malostranské nám. 25, 118 00 Praha 1, Czech Republic (cepek@ksi.ms.mff.cuni.cz)

[5]Permanent Member

[6]Department of Accounting and Information Systems, Faculty of Management, Rutgers University, Newark, NJ 07102, and RUTCOR, Rutgers University, P.O. Box 5062, New Brunswick, NJ 08903, USA (kogan@rutcor.rutgers.edu)

DIMACS is a partnership of Rutgers University, Princeton University, AT&T Labs, Bellcore, and Bell Labs.

DIMACS is an NSF Science and Technology Center, funded under contract STC–91–19999; and also receives support from the New Jersey Commission on Science and Technology.

ABSTRACT

The problem of Horn minimization can be stated as follows: given a Horn CNF representing a Boolean function f, find a CNF representation of f which consists of a minimum possible number of clauses. This problem is the formalization of the problem of knowledge compression for speeding up queries to propositional Horn expert systems, and it is known to be NP-hard. In this paper we present a linear time algorithm which takes a Horn CNF as an input, and through a series of decompositions reduces the minimization of the input CNF to the minimization problem on a "shorter" CNF. The correctness of this decomposition algorithm rests on several interesting properties of Horn functions which, as we prove here, turn out to be independent of the particular CNF representations.

1 Introduction

Horn functions constitute a very important subclass of Boolean functions. Their importance stems from the fact that the satisfiability problem (SAT), which is NP-complete for general Boolean formulae (see e.g. [7]) is solvable in linear time for Horn formulae [6, 14, 17]. This implies that certain real-life problems which require solving SAT become tractable if the underlying Boolean function in the problem is Horn. Such problems arise in several application areas, among others in artificial intelligence [4, 11, 12] and database design [5, 16]. Horn clauses are by far the most popular type of knowledge representation which is due to both the computational efficiency of reasoning and their sufficient richness for capturing essential features of real-life problems.

In some applications an important problem is to find a shortest possible representation of a given Boolean function. For instance, in artificial intelligence this problem is equivalent to finding a most compact representation of a given knowledge base [11, 12]. Such transformation of a knowledge base accomplishes knowledge compression, since the actual knowledge does not change, while the size of the representation can be significantly reduced. The computational complexity of reasoning in the case of Horn knowledge bases is reduced accordingly, since the compressed knowledge base is guaranteed to remain Horn. The procedure of knowledge compression preprocesses the knowledge base, and can be done off-line. This results in speeding up on-line operation while answering queries. Therefore, the computational expense of a single run of knowledge compression will be quickly amortized over a large number of queries to the knowledge base.

Unfortunately, unlike satisfiability, the representation minimization problem is NP-hard not only in the general case, but also for Horn CNFs [1, 2, 3, 11, 16]. The Horn Minimization (HM) problem can be stated as follows: given a Horn CNF $\mathcal{F}$ find a CNF $\mathcal{F}'$ representing the same function and such that $\mathcal{F}'$ consists of a minimum possible number of clauses. Paper [13] introduced two subclasses of Horn functions, acyclic and quasi-acyclic functions, for which HM is solvable in polynomial time. However, these subclasses are rather special, and therefore it is still desirable to find techniques for tackling HM for Horn functions in general.

In this paper we shall show how HM on a given CNF can be transformed into HM on a shorter CNF which can be then more easily solved by heuristic or approximate algorithms. This is achieved through a series of decompositions of the input CNF. Each decomposition step splits the "current" CNF into two parts (both nonempty) with the property that the minimization of the original CNF is equivalent to the minimization of one of the parts. The iterative decomposition process is based on "switching" of propositional variables. We shall also show that the "short" CNF produced by this iterative decomposition is always the same independently of the order in which the propositional variables are switched. In other words, we shall show that although in each iteration our algorithm makes a random choice of how to split the "current" CNF into two shorter CNFs, these random choices do not influence the final output. It will follow from this "invariance" property that the obtained CNF is in some sense the best (shortest) possible.

The paper is structured as follows. In Section 2 we introduce the necessary notation and state several elementary results important for the subsequent presentation. We commence Section 3 by stating the decomposition theorems for Horn functions from [9]. We proceed by showing that if we employ a simple trick involving "switching" of propositional variables, these theorems can be used in an iterative manner to reduce the minimization of a given CNF to the minimization of a CNF consisting only of a subset of clauses from the original CNF. We also design a naive algorithm performing this iterative decomposition.

In Section 4 we prove an interesting property of Horn functions which rests on the correspondence between Horn CNFs and directed graphs first introduced in [13]. This property shows that every Horn function possesses a certain structure which is independent of the particular CNF representation of the function. In Section 5 we use this structural property to prove that the output CNF of the naive algorithm is independent of the number of iterations (successive decompositions) and the order in which the clauses are eliminated from the input CNF. In other words, the random choices that the naive algorithm makes during its run have, quite surprisingly, no influence on the final output. We conclude Section 5 by presenting a more sophisticated iterative decomposition algorithm which achieves the same decomposition of the input CNF as the naive one, and has, however, a much better (linear time) complexity.

Section 6 investigates how the iterative decomposition algorithm performs when run on two distinct CNFs representing the same Horn function. We show that there exists a certain relationship between those clauses in both CNFs that are not eliminated by the algorithm. Moreover, if we restrict our attention to irredundant and prime CNFs of f, then the number of eliminated negative clauses and the "unswitchable" pure Horn function do not depend on the initial CNF and can be thought of as properties of the function f. Finally, in Section 7 we present the classes of acyclic and quasi-acyclic Horn CNFs for which Horn minimization is known to be solvable in polynomial time, and show that the iterative decomposition algorithm in fact minimizes these CNFs.

2 Basic Notation, Definitions, and Results

In this section we shall introduce the necessary notation and summarize the basic known results that will be needed later in the text. The first subsection will present some basic facts about (general) Boolean functions and about the subclass of Horn functions. The second subsection will introduce a so called "forward chaining procedure" which constitutes a very useful tool for the study of Horn functions. Finally, the third subsection will present the standard graph terminology that will be used throughout this paper.

2.1 Boolean functions

A *Boolean function* f on n propositional variables $x_1, \ldots, x_n$ is a mapping $\{0,1\}^n \rightarrow \{0,1\}$. The propositional variables $x_1, \ldots, x_n$ and their negations $\overline{x}_1, \ldots, \overline{x}_n$ are called *literals* (*positive* and *negative literals*, respectively). An elementary disjunction of literals

$$C = \bigvee_{i \in I} \overline{x}_i \vee \bigvee_{j \in J} x_j \tag{1}$$

is called a *clause*, if every propositional variable appears in it at most once, i.e. if $I \cap J = \emptyset$. For two Boolean functions f and g we write $f \leq g$ if

$$\forall (x_1, \ldots, x_n) \in \{0,1\}^n \ : \ f(x_1, \ldots, x_n) = 1 \quad \Longrightarrow \quad g(x_1, \ldots, x_n) = 1 \tag{2}$$

Since each clause is in itself a Boolean function, formula (2) also defines the meaning of inequalities $C_1 \leq C_2$, $C_1 \leq f$, and $f \leq C_1$, where C_1, C_2 are clauses and f is a Boolean function.

We say that a clause C_1 *subsumes* another clause C_2 if $C_1 \leq C_2$ (e.g. the clause $\overline{x} \vee z$ subsumes the clause $\overline{x} \vee \overline{y} \vee z$). A clause C is called an *implicate* of a function f if $f \leq C$. An implicate C is called *prime* if there is no distinct implicate C' subsuming C, or in other words, an implicate of a function is prime if dropping any literal from it produces a clause which is not an implicate of that function.

It is a well-known fact that every Boolean function f can be represented by a conjunction of clauses. Such an expression is called a *conjunctive normal form* (or CNF) of the Boolean function f. It should be noted that a given Boolean function may have many CNF representations (typically exponentially many in the number of propositional variables). Any two CNFs representing the same function are called *equivalent*. A CNF $\mathcal{F}$ representing a function f is called *prime* if each clause of $\mathcal{F}$ is a prime implicate of the function f. A CNF $\mathcal{F}$ representing a function f is called *irredundant* if dropping any clause from $\mathcal{F}$ produces a CNF that does not represent f.

For example, in the (Horn) CNF

$$(\overline{x}_1 \vee x_2) \wedge (\overline{x}_1 \vee \overline{x}_3) \wedge (\overline{x}_1 \vee x_4) \wedge (\overline{x}_2 \vee \overline{x}_3 \vee x_4) \wedge (\overline{x}_3 \vee \overline{x}_4)$$

the 2nd clause can be dropped (although it is prime), and the 4th clause can be shortened (i.e. it is not prime). In fact, the same (Horn) function can be represented by the CNF

$$(\overline{x}_1 \vee x_2) \wedge (\overline{x}_1 \vee x_4) \wedge (\overline{x}_2 \vee \overline{x}_3) \wedge (\overline{x}_3 \vee \overline{x}_4)$$

which is both prime and irredundant.

Throughout this paper we shall also use the following notation. For an arbitrary CNF $\mathcal{F}$ the symbol $\widehat{\mathcal{F}}$ denotes the set of all clauses in $\mathcal{F}$. For an arbitrary CNF $\mathcal{F}$ and a set of clauses S the CNF

$$\mathcal{F}^S = \bigwedge_{C \in \widehat{\mathcal{F}} \cap S} C \tag{3}$$

is called the *projection of* $\mathcal{F}$ *on* S. The CNF $\mathcal{F}^S$ results from $\mathcal{F}$ by deleting all the clauses that are not in S.

For the purpose of measuring the complexity of algorithms we need ways of measuring the "size" of a given CNF. The following two definitions are the ones used most commonly in the literature. For a CNF $\mathcal{F}$ let

- $|\mathcal{F}|_c = \sum_{C \in \widehat{\mathcal{F}}} 1$ (the number of clauses in $\mathcal{F}$),
- $|\mathcal{F}|_\ell = \sum_{C \in \widehat{\mathcal{F}}} |C|$ (the number of literals in $\mathcal{F}$).

Let us now turn our attention to the subclass of Boolean functions which is the focus of this paper, i.e. the class of Horn functions.

A clause C defined by (1) is called *negative* if it contains no positive literals (i.e. if $J = \emptyset$). It is called *pure Horn* (or in some literature *definite Horn*) if it contains exactly one positive literal (i.e. if $|J| = 1$). To simplify notation, we shall sometimes write a pure Horn clause $C = \bigvee_{x \in S} \overline{x} \vee y$ simply as $C = S \vee y$. Each propositional variable $x \in S$ is called a *subgoal of* C and the propositional variable y is called the *head of* C.[1]

A CNF is called *Horn* if it contains only negative and pure Horn clauses. A CNF is called *pure Horn* if it contains only pure Horn clauses. Finally, a Boolean function is called *Horn* if it has at least one representation by a Horn CNF, and similarly a Boolean function is called *pure Horn* if it has at least one representation by a pure Horn CNF.

It is known (see [9]) that each prime implicate of a Horn function is either negative or pure Horn, and each prime implicate of a pure Horn function is pure Horn. Thus, in particular, any prime CNF representing a Horn function is Horn, and any prime CNF representing a pure Horn function is pure Horn. The next statement was proved in [9].

Theorem 2.1 *Given a Horn CNF $\mathcal{F}$ one can find in $O(|\mathcal{F}|_\ell^2)$ time an irredundant and prime CNF $\mathcal{F}'$ equivalent with $\mathcal{F}$.*

Theorem 2.1 allows us to make an assumption that the CNF we work with (input CNF) is prime (and sometimes also irredundant).

[1] This terminology comes from the area of artificial intelligence, where the clause C is thought of as a "rule" $S \longrightarrow y$.

2.2 Forward chaining procedure

In verifying that a given clause is an implicate of a given pure Horn function, a very useful and simple procedure is the following. Let $\mathcal{F}$ be a pure Horn CNF of a pure Horn function f. We shall define a *forward chaining* procedure which associates to any subset Q of the propositional variables of f a set R in the following way. The procedure takes as input the subset Q of propositional variables, initializes the set $R = Q$, and at each step it looks for a pure Horn clause $S \vee y$ in $\mathcal{F}$ such that $S \subseteq R$, and $y \notin R$. If such a clause is found, the propositional variable y is included into R, and the search is repeated as many times as possible.

In the relational database terminology the propositional variables in $R_{\mathcal{F}}(Q)$ are said to be "chained" to the subset Q (see e.g. [5]). In the expert system terminology the usage of the clause $S \vee y$ is called "firing the rule" $\bigwedge_{x \in S} x \to y$ (see e.g. [10]).

Forward Chaining Procedure	
Input:	A pure Horn CNF $\mathcal{F} = \bigwedge_{i=1}^{m}(S_i \vee x_{j_i})$ and a subset Q of the propositional variables.
Initialization:	Set $R = Q$.
Main Step:	while $\exists\ i$ such that $S_i \subset R$ and $x_{j_i} \notin R$ do $R = R \cup \{x_{j_i}\}$.
Stop:	Output $R_{\mathcal{F}}(Q) = R$.

The following lemma was proved in [11].

Lemma 2.2 *A pure Horn clause $Q \vee y$ is an implicate of a pure Horn function f given by a pure Horn CNF $\mathcal{F}$ if and only if $y \in R_{\mathcal{F}}(Q)$.*

Lemma 2.2 shows how to verify whether a given clause is an implicate of a given CNF. The following statement proved in [13] shows that the forward chaining procedure can be efficiently implemented, i.e. that the complexity of performing the above mentioned verification is low.

Lemma 2.3 *Given a pure Horn CNF $\mathcal{F}$ and a subset Q of the propositional variables used in $\mathcal{F}$, the set $R_{\mathcal{F}}(Q)$ can be determined in $O(|\mathcal{F}|_\ell)$ time.*

Obviously, the complexity guaranteed by Lemma 2.3 is the best possible one, since any implementation of the forward chaining procedure must at least read the entire input CNF. Let us conclude this section with a notational remark.

If $\mathcal{F}_1$ and $\mathcal{F}_2$ are two distinct CNF representations of a given pure Horn function f and if Q is an arbitrary subset of the propositional variables, then by Lemma 2.2 $R_{\mathcal{F}_1}(Q) = R_{\mathcal{F}_2}(Q)$

because $\mathcal{F}_1$ and $\mathcal{F}_2$ have the same set of implicates. Therefore, the set of propositional variables reachable from Q by forward chaining depends only on the underlying function rather than on a particular CNF representation. For this reason, we shall use the expression $R_f(Q)$ instead of $R_{\mathcal{F}}(Q)$ whenever we do not want to refer to a specific CNF.

2.3 Graphs and digraphs

We shall recall here a few standard notions from graph theory.

A *graph* is an ordered pair $G = (V, E)$ where V is a set of *vertices* and E is a set of *edges*. An edge is an unordered pair of vertices. A *path* is a sequence of edges, say $e_1, e_2, \ldots, e_p$ such that $e_i = (x_i, x_{i+1})$ for some vertices $x_1, x_2, \ldots, x_{p+1}$. A path is called *simple* if $x_i \neq x_j$ for $i \neq j$. A *cycle* is a path such that $x_1 = x_{p+1}$. A cycle is called *simple* if $x_i \neq x_j$ for $i \neq j$ with the only exception of $x_1 = x_{p+1}$. A graph is called *connected* if for any two vertices x and y there exists a path between them. If a graph is not connected then its maximal connected parts are called *connected components*. A graph is called acyclic if it contains no cycle. A *degree* of a vertex is the number of (distinct) edges adjacent to it.

A *directed graph* (sometimes abbreviated to *digraph*) is an ordered pair $D = (N, A)$ where N is the set of *nodes* and A is the set of *arcs*. An arc is an ordered pair of nodes. The concepts of a *directed path* and *directed cycle* are defined similarly as in the undirected case. However, unlike the undirected case, the concept of a path is not symmetric. Instead of talking about a path between two nodes x and y, we talk about a directed path from x to y and a directed path from y to x, noting that the existence of one does not imply the existence of the other.

A directed graph is called *strongly connected* if for any two nodes x and y there exist both a directed path from x to y and a directed path from y to x. If a graph is not strongly connected then its maximal strongly connected parts are called *strong components*. The *outdegree* of a node x is the number of arcs leaving the node, i.e. $|\{y \mid (x, y) \in A\}|$, and the *indegree* of a node x is the number of arcs entering the node, i.e. $|\{y \mid (y, x) \in A\}|$. A directed graph is called *acyclic* if it contains no directed cycle. Note that in such a case every strong component consists of a single node. If D is an acyclic directed graph with a node set $N = \{x_1, \ldots, x_n\}$, then an ordering of the nodes $(x_{i_1}, \ldots, x_{i_n})$ is called a *topological order* on N if for every pair of distinct indices i_j, i_k the inequality $i_j > i_k$ implies that there is no directed path from x_{i_j} to x_{i_k}.

If $D = (N, A)$ is a directed graph, then the *transitive closure* of D is a directed graph $\overline{D} = (N, \overline{A})$ where $(x, y) \in \overline{A}$, whenever there is a directed path from x to y in the digraph D. Obviously, for a given D the transitive closure $\overline{D}$ is uniquely defined.

Finally, if $D = (N, A)$ is a directed graph with strong components $C_1, \ldots, C_s$, then the directed graph $D' = (N', A')$ on the set of nodes $N' = \{C_1, \ldots, C_s\}$ with arcs

$$(C_i, C_j) \in A' \text{ iff } \exists x \in C_i \ \exists y \in C_j \text{ such that } (x, y) \in A$$

is called the *acyclic condensation* of the graph D.

3 Blind Switching

We shall start this section by recalling two theorems proved in [9].[2] Let f be a Horn function on a set S of propositional variables and let $\mathcal{H}$ denote the set of all pure Horn clauses on S and $\mathcal{N}$ the set of all negative clauses on S.

Theorem 3.1 *Let $\mathcal{F}_1$ and $\mathcal{F}_2$ be two prime CNFs of a Horn function f. Then $\mathcal{F}_1^{\mathcal{H}}$ and $\mathcal{F}_2^{\mathcal{H}}$ are equivalent (and the function they represent is called the pure Horn component of f).*

Theorem 3.2 *Let $\mathcal{F}_1$ and $\mathcal{F}_2$ be two irredundant and prime CNFs of a Horn function f. Then $|\mathcal{F}_1^{\mathcal{N}}|_c = |\mathcal{F}_2^{\mathcal{N}}|_c$ (i.e. $\mathcal{F}_1$ and $\mathcal{F}_2$ contain the same number of negative clauses).*

These two theorems imply the following. Suppose that we are given an irredundant and prime Horn CNF $\mathcal{F}$ and we want to minimize it. Clearly, a minimum CNF must also be irredundant (and can be assumed prime), and therefore, by Theorems 3.1 and 3.2, we can simply put aside all the negative clauses and minimize the pure Horn part independently.

This leads to an intriguing question. Suppose we can replace some of the propositional variables in the pure Horn part by their complements (i.e. replace x by $\overline{x}$ and vice versa) in such a way that the resulting CNF is again Horn and contains some negative clauses.[3] Then it is again possible to put the negative clauses aside and iterate the process. In order to carry out such a procedure, one has to be able to answer the following question. Given a pure Horn clause $T = A \vee x$ in a pure Horn CNF $\mathcal{F}$, is it possible to make T negative by "switching" the propositional variable x while at the same time keeping $\mathcal{F}$ Horn? The difficulty is that in every clause where x appears negative, switching of x forces the head of the clause to be switched as well (so that the CNF stays Horn), and it is not immediately obvious whether this "domino effect" will cycle, thus making the switch impossible. However, this problem can be handled by a simple procedure SWITCH (see below) which performs the forced switching described above until it either succeeds to make T negative (while keeping $\mathcal{F}$ Horn) or gets stuck, the latter meaning that T cannot be switched.

[2] These two theorems appear in [9] as Corollary 4.4 and Corollary 5.4. Both are restated here using the notation of this text.

[3] Although strictly speaking the CNF resulting from such a switch represents a different function than the original CNF, it is clear that if we perform a minimization of the "new" function, we can always obtain the corresponding CNF of the original function by switching back the appropriate propositional variables.

PROCEDURE <u>SWITCH</u>($\mathcal{F}, C$)

Input: A pure Horn CNF $\mathcal{F}$ and a clause $C = A \vee x \in \widehat{\mathcal{F}}$.

Initialization: Set $fix = A$, set up a *queue*, and put x in it.

Algorithm:
```
repeat
   let y be the first propositional variable from queue;
   set fix = fix ∪ {y};
   loop over all clauses T ∈ F̂
      if y is in T then replace y by ȳ in T else
      if ȳ is in T then
         if T contains two or more positive literals then failure;
         if T contains exactly one positive literal (say z̄) then
            if z ∈ fix then failure;
            if z ∉ fix then replace ȳ by y in T and put z in queue;
         if T is negative then replace ȳ by y in T;
   remove y from queue
until (queue is empty or failure)
```

Output:
```
if failure
   then output SWITCH failed and return the original CNF
   else output SWITCH succeeded and return the modified CNF.
```

Once we are able to decide the "switchability" of an arbitrary clause in a given pure Horn CNF, we can proceed to implement the "iterative decomposition" algorithm. In each iteration, the procedure <u>DECOMPOSITION</u> (see below) finds a "switchable" clause, performs the switch, and separates the pure Horn and the negative clauses.

PROCEDURE DECOMPOSITION($\mathcal{F}$)

Input: A pure Horn CNF $\mathcal{F}$.
Initialization: Set up an empty *clause-queue*.
Algorithm:
```
repeat
   put all (pure Horn) clauses of F into a clause-queue;
   while (clause-queue is nonempty) do
      let T be the first clause in clause-queue;
      run SWITCH(F, T);
      if SWITCH succeeded
         then empty clause-queue
            and remove all negative clauses from F
         else remove T from clause-queue
until (no clause was switched to negative in the last step)
```
Output: All the removed negative clauses and the remaining "unswitchable" pure Horn CNF.

Let us denote $c = |\mathcal{F}|_c$ and $\ell = |\mathcal{F}|_\ell$. The complexity of DECOMPOSITION is bounded by $O(c^2)$ calls to the subroutine SWITCH. If implemented carefully, SWITCH can be carried out in $O(\ell)$ time. Therefore[4], the overall complexity of DECOMPOSITION is $O(c^2\ell)$. What is not clear, however, is how the order in which the clauses are picked and switched influences the result of the procedure. In particular, it is not immediately obvious whether one sequence of switches may lead to a larger number of pure Horn clauses being switched to negative clauses (and subsequently removed from the CNF) than some other sequence. Surprisingly, the set of the removed clauses is independent of the order in which the clauses are switched. This result will be obtained as a byproduct of developing a more sophisticated procedure for the "iterative decomposition" of a given pure Horn CNF. Rather than blindly trying to switch clauses in a random order, the forthcoming procedure will first determine the set of all clauses that are "candidates" for a switch, and then sequentially perform the switching of all these "candidate" clauses in such an order which minimizes the number of necessary iterations.

For this purpose, we shall first prove some interesting properties of Horn functions. These properties will be stated in terms of graphs which can be associated with Horn functions. The next section is devoted to this topic.

[4]We skip a detailed complexity analysis of the procedures SWITCH and DECOMPOSITION. Later in this paper a "better" procedure performing the same task will be developed and analyzed in detail.

4 Graph Properties of Horn Functions

Following [13], we associate with each Horn CNF a directed graph as follows :

Definition 4.1 *For a Horn CNF $\mathcal{F}$ let $G_{\mathcal{F}} = (N, A_{\mathcal{F}})$ be the digraph defined by*

$$N = \{x \mid x \text{ is a propositional variable in } \mathcal{F}\}$$

$$A_{\mathcal{F}} = \{(x, y) \mid \exists \text{ a clause } T \in \widehat{\mathcal{F}} \text{ such that } T \geq \overline{x} \vee y\}.$$

Described in words, for each pure Horn clause T in $\mathcal{F}$, the graph $G_{\mathcal{F}}$ contains as many arcs as the number of negative literals (subgoals) in T, with each arc going from the corresponding negative literal to the only positive literal (head) in T.

The following lemma proved in [13] describes a useful property of the graph $G_{\mathcal{F}}$.

Lemma 4.2 *If a pure Horn clause $T = \bigvee_{x \in S} \overline{x} \vee y$ is a prime implicate of a pure Horn function f given by a pure Horn CNF $\mathcal{F}$, then $G_{\mathcal{F}}$ contains a directed path from x to y for all $x \in S$.*

In the rest of this section we shall show that although the graph $G_{\mathcal{F}}$ depends on the particular CNF $\mathcal{F}$ (i.e. for two distinct CNFs representing the same function we may get two distinct graphs), there are certain features of the graph which are independent of the particular CNF representation and depend only on the underlying function.

Theorem 4.3 *Let $\mathcal{F}_1$ and $\mathcal{F}_2$ be two distinct prime CNFs representing the same Horn function f and let x, y be arbitrary propositional variables from f. Then there is a directed path from x to y in $G_{\mathcal{F}_1}$ if and only if there is a directed path from x to y in $G_{\mathcal{F}_2}$.*

Proof. Due to the symmetry it is enough to prove one of the implications. Let us for instance assume that there exists a directed path from x to y in $G_{\mathcal{F}_1}$. Let this path be $x = z_0, z_1, \ldots, z_k = y$. By the definition of $G_{\mathcal{F}_1}$, for each arc (z_{i-1}, z_i) there exists a clause T_i in $\mathcal{F}_1$ that contains literals $\overline{z}_{i-1}$ and z_i. Every T_i is a prime implicate of $\mathcal{F}_2$ because $\mathcal{F}_1$ is prime and represents the same function as $\mathcal{F}_2$. Moreover, every T_i is pure Horn and hence it is also a prime implicate of the pure Horn component $\mathcal{F}_2^{\mathcal{H}}$ of $\mathcal{F}_2$ ($\mathcal{F}_2^{\mathcal{H}}$ consists of all pure Horn clauses in $\mathcal{F}_2$). Therefore, by Lemma 4.2 there exists a directed path from z_{i-1} to z_i in $G_{\mathcal{F}_2^{\mathcal{H}}}$ for every $i = 1, \ldots, k$. However, obviously $G_{\mathcal{F}_2^{\mathcal{H}}} = G_{\mathcal{F}_2}$, and so the concatenation of these directed paths gives us a path from x to y in $G_{\mathcal{F}_2}$.

∎

Corollary 4.4 *Let $\mathcal{F}_1$ and $\mathcal{F}_2$ be two distinct prime CNFs representing the same Horn function f. Then $G_{\mathcal{F}_1}$ and $G_{\mathcal{F}_2}$ have identical transitive closures, and therefore identical strong components and identical acyclic condensations.*

As an example let us consider the following pure Horn CNF

$$\mathcal{F} = (\overline{x}_1 \vee \overline{x}_5 \vee x_6) \wedge (\overline{x}_1 \vee \overline{x}_6 \vee x_7) \wedge (\overline{x}_2 \vee \overline{x}_3 \vee x_5) \wedge (\overline{x}_2 \vee \overline{x}_4 \vee x_3) \wedge (\overline{x}_3 \vee \overline{x}_5 \vee x_4) \wedge (\overline{x}_4 \vee \overline{x}_7 \vee x_8) \wedge (\overline{x}_7 \vee x_6) \tag{4}$$

consisting of 7 *clauses on* 8 *propositional variables. The corresponding graph* $G_{\mathcal{F}}$*, shown on Figure 1, has five strong components, which in their topological order are* $C_1 = \{x_1\}$*,* $C_2 = \{x_2\}$*,* $C_3 = \{x_3, x_4, x_5\}$*,* $C_4 = \{x_6, x_7\}$ *and* $C_5 = \{x_8\}$*.*

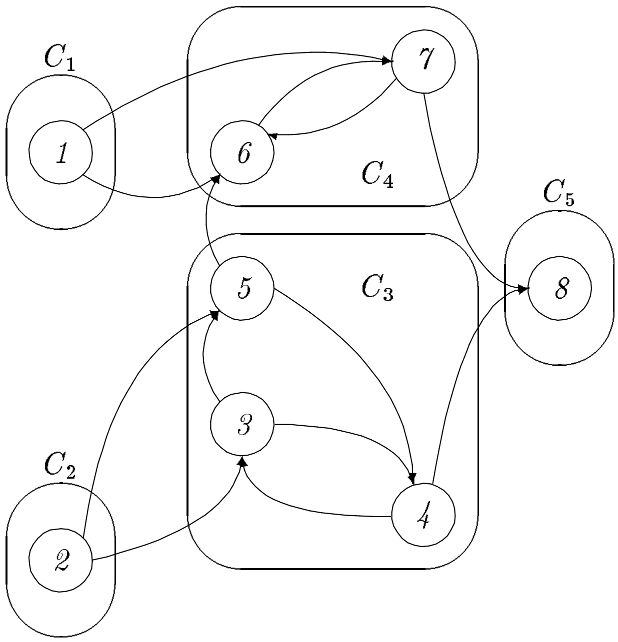

Figure 1: The graph $G_{\mathcal{F}}$ corresponding to the CNF in (4). The five strong components C_1, C_2, C_3, C_4 and C_5 are labeled according to their topological order.

Now we may associate a graph G_f directly to a given Horn function f.

Definition 4.5 *Let us consider an arbitrary prime CNF* $\mathcal{F}$ *representing the given Horn function* f*, and let us define* G_f *as the transitive closure of* $G_{\mathcal{F}}$*.*

Corollary 4.4 implies that G_f is well defined. This, and the fact that G_f can be efficiently constructed from any prime CNF representing f, will enable us to work with a graph that corresponds directly to a Horn function rather than to a particular CNF representing it. We will utilize this fact in the upcoming sections.

5 Graph-Based Switching

In this section we shall show how, given a pure Horn CNF $\mathcal{F}$, we can use the corresponding graph $G_\mathcal{F}$ to decide which clauses in $\mathcal{F}$ can be switched to negative clauses (while the resulting CNF remains Horn) and which clauses cannot. We shall also use $G_\mathcal{F}$ to determine the "best order" in which the "switchable" clauses should be picked in order to keep the complexity of the "iterative decomposition" algorithm low. We shall start by introducing some necessary notation.

Let $\mathcal{F}$ be a prime CNF of a Horn function f and let $G_\mathcal{F}$ be its graph. For arbitrary two strong components C_i, C_j of $G_\mathcal{F}$ we define a partial order

$$C_i \preceq C_j \text{ if there is a path from } C_i \text{ to } C_j \text{ in } G_\mathcal{F}. \tag{5}$$

Consistently with the definition in Section 2.3, any linear extension of the partial order "$\preceq$" defined by (5) is called a *topological order* of the strong components on the graph $G_\mathcal{F}$. Finally, for each C_i let us denote the set of its predecessors and successors by

- $P(C_i) = \{x \mid x \in C_j \text{ for some } C_j \preceq C_i\} = \bigcup_{j:C_j \preceq C_i} C_j$
- $S(C_i) = \{x \mid x \in C_j \text{ for some } C_i \preceq C_j\} = \bigcup_{j:C_i \preceq C_j} C_j$

Note that $C_i = P(C_i) \cap S(C_i)$. We are now ready to state several simple lemmas.

Lemma 5.1 *Let $\mathcal{F}$ be a pure Horn CNF, let x be an arbitrary propositional variable used in $\mathcal{F}$, and let C_i be the strong component of $G_\mathcal{F}$ for which $x \in C_i$. Let a Horn CNF $\mathcal{F}'$ result from $\mathcal{F}$ by switching a subset Z of propositional variables. Then $x \in Z$ implies $S(C_i) \subseteq Z$.*

We omit a formal proof since Lemma 5.1 is a direct corollary of Definition 4.1. It just describes in more precise terms the "domino effect" mentioned in Section 3. Indeed, once the propositional variable x is switched, it forces the switching of all propositional variables that can be reached from x via a path in $G_\mathcal{F}$, i.e. of all propositional variables in $S(C_i)$.

The next lemma proves that the strong component structure of $G_\mathcal{F}$ is invariant under such switching.

Lemma 5.2 *Let $\mathcal{F}$ be a pure Horn CNF and let a Horn CNF $\mathcal{F}'$ result from $\mathcal{F}$ by switching a subset Z of propositional variables. Then the graphs $G_\mathcal{F}$ and $G_{\mathcal{F}'}$ have the same strong components.*

Proof. By Lemma 5.1, for every strong component C_i either $C_i \subseteq Z$ or $C_i \cap Z = \emptyset$. So, if we denote $W = \{x \mid x \notin Z\}$, then for every C_i either $C_i \subseteq W$ or $C_i \subseteq Z$. The following three simple observations provide the proof:

- The graphs $G_{\mathcal{F}}$ and $G_{\mathcal{F}'}$ are identical on the set W.
- On the set Z the graph $G_{\mathcal{F}'}$ results from $G_{\mathcal{F}}$ by reversing the direction of all arcs, and therefore $G_{\mathcal{F}}$ and $G_{\mathcal{F}'}$ have the same strong component structure on the set Z.
- By Lemma 5.1, all arcs in $G_{\mathcal{F}}$ between W and Z are directed from W to Z. If an arc (x, y), where $x \in W$ and $y \in Z$, comes from a clause $T = A \vee \overline{x} \vee y \in \widehat{\mathcal{F}}$, then $|A \cap Z| \leq 1$ must hold (otherwise T would give rise to a non-Horn clause in $\mathcal{F}'$) and therefore either
  - $|A \cap Z| = 0$ in which case T gives rise to a negative clause and contributes no arcs in $G_{\mathcal{F}'}$, or
  - $|A \cap Z| = 1$, say $A \cap Z = \{z\}$, in which case T gives rise to a pure Horn clause and contributes arcs from $(A \setminus \{z\}) \cup \{x\}$ to z in $G_{\mathcal{F}'}$ (all of which are directed from W to Z) and an arc from y to z in $G_{\mathcal{F}'}$ which is within Z.

  Therefore, in $G_{\mathcal{F}'}$ all arcs between W and Z are directed from W to Z, and hence they cannot have any influence on the strong component structure.

■

We shall continue by stating two sufficient conditions which guarantee that a given strong component C "cannot be switched", i.e. which guarantee that if any the propositional variables in C were switched the resulting CNF would not be Horn.

Lemma 5.3 *Let $\mathcal{F}$ be a pure Horn CNF and let x, y, z be three propositional variables such that there exists a clause $T \in \widehat{\mathcal{F}}$ for which $T \geq \overline{x} \vee \overline{y} \vee z$ (i.e. $\overline{x}$, $\overline{y}$, and z are literals in T). Let C_i be a strong component of $G_{\mathcal{F}}$ such that $\{x, y, z\} \subseteq C_i$ and let a Horn CNF $\mathcal{F}'$ result from $\mathcal{F}$ by switching a subset Z of the propositional variables. Then $P(C_i) \cap Z = \emptyset$.*

Proof. Let us assume that there exists a propositional variable $v \in P(C_i) \cap Z$, say $v \in C_k \subseteq P(C_i)$. By Lemma 5.1, all propositional variables in $S(C_k)$ must be switched (i.e. $S(C_k) \subseteq Z$), so in particular $\{x, y, z\} \subseteq Z$ since $C_i \subseteq S(C_k)$. But that means that after the switch the clause corresponding to T contains two positive literals x and y, contradicting the assumption that $\mathcal{F}'$ is Horn.

■

We shall call a triplet $\{x, y, z\}$ with the properties described in Lemma 5.3 an *inside fork blocking the component C_i*. The next lemma will introduce an *outside fork blocking the component C_i*.

Lemma 5.4 *Let $\mathcal{F}$ be a pure Horn CNF and let x, y, z be three propositional variables such that there exists a clause $T \in \widehat{\mathcal{F}}$ for which $T \geq \overline{x} \vee \overline{y} \vee z$. Let C_i, C_j be two distinct strong components of $G_{\mathcal{F}}$ such that $x \in C_i$ and $\{y, z\} \subseteq C_j$ (note that by definition $C_i \preceq C_j$), and let a Horn CNF $\mathcal{F}'$ result from $\mathcal{F}$ by switching a subset Z of the propositional variables. Then $P(C_i) \cap Z = \emptyset$.*

Proof. It is sufficient to notice that for every $C_k \subseteq P(C_i)$ we have $\{x, y, z\} \in S(C_k)$. The rest of the proof is similar to the proof of Lemma 5.3.

■

Let us call both the inside and outside forks *blocking patterns.* The following definition makes use of them.

Definition 5.5 *Let $\mathcal{F}$ be a pure Horn CNF and let C_i be a strong component of $G_{\mathcal{F}}$. We say that C_i is* blocked *if (at least) one of the following three conditions holds:*

- *there is an inside fork blocking C_i, or*
- *there is an outside fork blocking C_i, or*
- *there is a blocked component C_j such that $C_i \preceq C_j$.*

If C_i is blocked due to the third condition, then we say that C_i is blocked by inheritance. *If C_i is not blocked, then it is called* switchable.

Let us return to the example of the CNF given in (4) and shown in Figure 1. We can see that the components C_5 and C_4 are not blocked, component C_3 is blocked by the inside fork $\overline{x}_3 \vee \overline{x}_5 \vee x_4$, component C_1 is blocked by the outside fork $\overline{x}_1 \vee \overline{x}_6 \vee x_7$, while component C_2 is blocked by C_3.

We shall show later in this paper that the term "switchable" is well justified, i.e. that every strong component that is not blocked can be eventually switched. The word "eventually" is important here. It may happen that a given CNF $\mathcal{F}$ contains a clause $T = \overline{x} \vee \overline{y} \vee z$ such that $x \in C_i$, $y \in C_j$, and $z \in C_k$ where C_i, C_j, C_k are three distinct switchable strong components of $G_{\mathcal{F}}$. By definition, $C_i \preceq C_k$ and $C_j \preceq C_k$. However, if also $C_i \preceq C_j$ holds then switching C_i would cause T to yield a non-Horn clause. Thus C_i cannot be switched. The point is that switching C_k while leaving C_i and C_j unswitched turns T into a negative clause which can be eliminated from $\mathcal{F}$ thus "unblocking" C_i. It will be shown that after several rounds of such eliminations every switchable component can indeed be switched.

Lemma 5.2 states that the strong components of $G_{\mathcal{F}}$ are invariant under switching. We shall now use Lemmas 5.3 and 5.4 to show that another invariant of switching is which components are blocked and which are not.

Lemma 5.6 *Let $\mathcal{F}$ be a pure Horn CNF and let a Horn CNF $\mathcal{F}'$ result from $\mathcal{F}$ by switching a subset Z of the propositional variables. Let C_i be an arbitrary strong component of $G_{\mathcal{F}}$. Then C_i is a blocked component in $G_{\mathcal{F}}$ if and only if C_i is a blocked component in $G_{\mathcal{F}'}$.*

Proof. First of all, since C_i is a strong component in $G_{\mathcal{F}}$, it is also a strong component in $G_{\mathcal{F}'}$ by Lemma 5.2. Secondly, due to the symmetry[5] it is sufficient to prove only one of the implications, say that if C_i is blocked in $G_{\mathcal{F}}$ then it is also blocked in $G_{\mathcal{F}'}$. We shall proceed by handling the three possible blocking patterns separately.

- Let C_i be blocked in $G_{\mathcal{F}}$ by an inside fork $\{x, y, z\}$ contained in some clause $T \in \widehat{\mathcal{F}}$. All the literals of T are in $P(C_i)$ and so by Lemma 5.3 no literal of T is in Z. Thus, T itself is a clause in $\mathcal{F}'$, and hence the inside fork $\{x, y, z\}$ blocks C_i in $G_{\mathcal{F}'}$.

- Let C_i be blocked in $G_{\mathcal{F}}$ by an outside fork $\{x, y, z\}$ contained in some clause $T = A \vee \overline{x} \vee \overline{y} \vee z$, where $x \in C_i$, $\{y, z\} \subseteq C_j$, and $A \cap C_j = \emptyset$ for some component C_j such that $C_i \preceq C_j$. By Lemma 5.4 we get $(A \cup \{x\}) \cap Z = \emptyset$, and by Lemma 5.1 either $C_j \subseteq Z$ or $C_j \cap Z = \emptyset$. Therefore, there are two possibilities. In the case $C_j \subseteq Z$ the clause T gives rise to a clause $T' = A \vee \overline{x} \vee \overline{z} \vee y$ in $\mathcal{F}'$, and in the case $C_j \cap Z = \emptyset$ we have $T' = T$, i.e. T appears in $\mathcal{F}'$ unchanged. In both cases T' contains an outside fork blocking the component C_i in $G_{\mathcal{F}'}$.

- Let C_i be blocked in $G_{\mathcal{F}}$ by inheritance, i.e. there is a blocked component C_j in $G_{\mathcal{F}}$ such that $C_i \preceq C_j$. From Definition 5.5 it is clear that we can assume that C_j is blocked by a blocking pattern. The previous two cases imply that C_j is also blocked in $G_{\mathcal{F}'}$. Finally, the fact that C_j is blocked guarantees that $P(C_j) \cap Z = \emptyset$, and so all the clauses from $\mathcal{F}$ which are entirely in $P(C_j)$ appear also in $\mathcal{F}'$. Therefore, the relation $C_i \preceq C_j$ is preserved in $G_{\mathcal{F}'}$, and hence C_i is blocked by inheritance in $G_{\mathcal{F}'}$.

■

The proof of Lemma 5.6 actually shows a little more than the lemma states. It shows that certain "groups" of clauses are so to speak "closed under switching", i.e. a clause from one group can give rise only to another clause from the same group. In order to state this observation in more precise terms, let us first split the clauses in $\mathcal{F}$ into three categories as follows:

- $B(\mathcal{F}) = \{T = A \vee x \in \widehat{\mathcal{F}} \mid x \in C_i \text{ for some blocked component } C_i\}$

- $N_0(\mathcal{F}) = \{T = A \vee x \in \widehat{\mathcal{F}} \mid x \in C_i \text{ for some switchable component } C_i \text{ and } A \cap C_i = \emptyset\}$

- $N_1(\mathcal{F}) = \{T = A \vee x \in \widehat{\mathcal{F}} \mid x \in C_i \text{ for some switchable component } C_i \text{ and } |A \cap C_i| = 1\}$

[5]Note that by switching propositional variables in Z we can go back and forth between $\mathcal{F}$ and $\mathcal{F}'$

Note that since no switchable component can contain an inside fork, the sets $B(\mathcal{F})$, $N_0(\mathcal{F})$, and $N_1(\mathcal{F})$ constitute a partition of $\widehat{\mathcal{F}}$. Using these sets, the discussion in the proof of Lemma 5.6 yields the following corollary.

Corollary 5.7 *Let $\mathcal{F}$ be a pure Horn CNF and let $T \in \widehat{\mathcal{F}}$ be an arbitrary clause. Let a Horn CNF $\mathcal{F}'$ result from $\mathcal{F}$ and a clause T' result from T by switching a subset Z of propositional variables. Then*

- *if $T \in B(\mathcal{F})$ then $T' \in B(\mathcal{F}')$, and moreover $T = T'$*
- *if $T \in N_1(\mathcal{F})$ then $T' \in N_1(\mathcal{F}')$*
- *if $T \in N_0(\mathcal{F})$ then either $T' \in N_0(\mathcal{F}')$ or T' is negative*

Now we are finally ready to analyze the behavior of the procedure DECOMPOSITION introduced in Section 3. We shall show that the procedure eliminates the same set of clauses regardless of the order in which the clauses are picked, and that it eliminates every clause that can be eliminated in this way.

Theorem 5.8 *Let $\mathcal{F}$ be a pure Horn CNF and let $T \in \widehat{\mathcal{F}}$ be an arbitrary clause. Then T is switched to a negative clause and eliminated from the CNF by the procedure* DECOMPOSITION$(\mathcal{F})$ *if and only if $T \in N_0(\mathcal{F})$.*

Proof. Let $Q \subseteq \widehat{\mathcal{F}}$ be the set of all the clauses in $\mathcal{F}$ that are switched to negative clauses and eliminated from the CNF by the procedure DECOMPOSITION$(\mathcal{F})$. We want to prove that $Q = N_0(\mathcal{F})$.

The inclusion $Q \subseteq N_0(\mathcal{F})$ is a direct consequence of Corollary 5.7. Clearly, no clause from $B(\mathcal{F}) \cup N_1(\mathcal{F})$ can be ever switched to a negative clause.

Proving the reverse inclusion $N_0(\mathcal{F}) \subseteq Q$ requires a little more effort. Suppose the procedure DECOMPOSITION discovers a clause in $N_0(\mathcal{F})$ that can be switched to a negative clause, and performs the switch resulting in a CNF $\mathcal{F}'$. By Corollary 5.7 every clause in $N_0(\mathcal{F})$ gives rise either to a clause in $N_0(\mathcal{F}')$ or to a negative clause, the latter happening for at least one clause. Therefore, the cardinality of the set $N_0(\mathcal{F})$ decreases by at least one after every iteration of DECOMPOSITION, and thus the procedure cannot cycle. On the other hand, if there is at least one clause in $N_0(\mathcal{F})$ that can be switched to a negative clause, DECOMPOSITION is going to discover such a clause, since it performs an exhaustive search until either it succeeds or it fails on every single clause. Hence, it is sufficient to prove that if $N_0(\mathcal{F})$ is nonempty then at least one clause in $N_0(\mathcal{F})$ can be switched to a negative clause.

Let us first see what can go wrong if we pick a clause from $N_0(\mathcal{F})$ at random.[6] If we try to switch the head of the randomly picked clause (let us say the head is in a strong

[6] This is what DECOMPOSITION does. Actually, the procedure is even more "stupid" – it picks a clause at random from the set of all clauses, not just $N_0(\mathcal{F})$.

component C_i), we are forced (by Lemma 5.1) to switch all its successors in the set $S(C_i)$. However, if there is a clause T that has two or more negative literals in $S(C_i)$ then the switch fails, because T would give rise to a non-Horn clause. Therefore, we should try to pick the candidate for switching in such a way that $S(C_i)$ is as small as possible. We shall show that this rather simple idea actually works.

Let us concentrate on the subgraph G' of $G_{\mathcal{F}}$ induced by the switchable components of $G_{\mathcal{F}}$. It is easy to see that all the arcs between the strong components of G' come from clauses in $N_0(\mathcal{F})$. Let us pick one of the "sink" strong components (i.e. components with no outgoing arcs) such that at least one clause (say T) from $N_0(\mathcal{F})$ has its head in it. Let this component be C_i. Clearly, such a component exists whenever $N_0(\mathcal{F}) \neq \emptyset$.[7] Note that since C_i is a sink component, we have $S(C_i) = C_i$, and so $S(C_i)$ is rather "small". We claim that switching all propositional variables in C_i produces a Horn CNF and T gives rise to a negative clause. However, this is obvious: no clause in $\mathcal{F}$ can have two or more negative literals in C_i since that would be an inside fork, contradicting the assumption that C_i is switchable.

■

At this point it should be quite clear that the complexity of <u>DECOMPOSITION</u>($\mathcal{F}$) can be vastly improved by utilizing the graph $G_{\mathcal{F}}$ in a way suggested by the proof of Theorem 5.8. The procedure <u>GRAPH-ALGO</u> (see below) formalizes this "graph approach" in an algorithmic manner.

The fact that <u>GRAPH-ALGO</u> outputs the same eliminated negative clauses (although, perhaps, in a different order) and then terminates with the same "unswitchable" pure Horn CNF as the procedure <u>DECOMPOSITION</u> follows from the proof of Theorem 5.8. Moreover, it is not hard to see that <u>GRAPH-ALGO</u> can be implemented to have an $O(\ell)$ complexity, which is a substantial speed-up compared to the $O(\ell c^2)$ complexity of the procedure <u>DECOMPOSITION</u> (recall that here $\ell = |\mathcal{F}|_\ell$ and $c = |\mathcal{F}|_c$).

[7]The only case when a sink component C of G' does not contain a head of a clause from $N_0(\mathcal{F})$ is if C is isolated in G' and every propositional variable in C is a head of a clause from $N_1(\mathcal{F})$.

PROCEDURE GRAPH-ALGO

Input: A pure Horn CNF $\mathcal{F}$.

Initialization: Construct the directed graph $G_{\mathcal{F}}$ and detect and order topologically the strong components $C_1, \ldots, C_s$ of $G_{\mathcal{F}}$.

Algorithm:

for every clause $T = A \vee x \in \widehat{\mathcal{F}}$ **do** (assume $x \in C_i$)
  if $|A \cap C_i| \geq 2$ **then** mark all components in $P(C_i)$ as blocked;
  (* inside fork blocking C_i, back-propagate blocking in $G_{\mathcal{F}}$ *)
  if $|A \cap C_i| = 1$ (say $A \cap C_i = \{y\}$) **then for** each $z \in A \setminus \{y\}$ **do**
    mark all components in $P(C_j)$ as blocked; (assuming $z \in C_j$)
  (* outside fork blocking C_j, back-propagate blocking in $G_{\mathcal{F}}$ *)
for every clause $T = A \vee x \in \widehat{\mathcal{F}}$ **do** (assume $x \in C_i$)
  if C_i is blocked **then** put T in a *B-queue*;
  (* one queue for all clauses in $B(\mathcal{F})$ *)
  if C_i is switchable **then**
    if $|A \cap C_i| = 1$ **then** put T in an $N_1(i)$-*queue*;
    if $|A \cap C_i| = 0$ **then** put T in an $N_0(i)$-*queue*;
  (* two queues per switchable component for clauses in $N_1(\mathcal{F})$ and $N_0(\mathcal{F})$ *)
for $i = s$ downto 1 **do if** C_i is switchable **then**
  (* note that C_i is a sink component of the current graph *)
  for every clause $T = A \vee x$ in the $N_1(i)$-*queue* **do**
    switch x and the (unique) propositional variable in $A \cap C_i$;
  for every clause $T = A \vee x$ in the $N_0(i)$-*queue* **do**
    switch x and eliminate T from $\mathcal{F}$;
  (* eliminated negative clause *)

Output: All the eliminated negative clauses from all N_0-*queues* and the remaining "unswitchable" pure Horn CNF consisting of the clauses in the *B-queue* and all N_1-*queues*.

The preprocessing phase where the graph $G_{\mathcal{F}}$ is constructed and its strong components are discovered and topologically ordered can be implemented to have an $O(\ell)$ complexity using the well-known algorithms of [18] and [15]. Each of the three main cycles in GRAPH-ALGO examines every clause at most once, and the amount of work per clause is linear in the size of the clause (i.e. number of literals in the clause), which gives an overall $O(\ell)$ complexity. The only exception to this rule (i.e. more than linear amount of work per clause) is the extra work needed for the back-propagation of marking blocked components in the first main cycle. However, the back-propagation can be implemented in such a way that every arc of $G_{\mathcal{F}}$ is examined at most once, and therefore the overall complexity of the back-propagation is also bounded by $O(\ell)$.

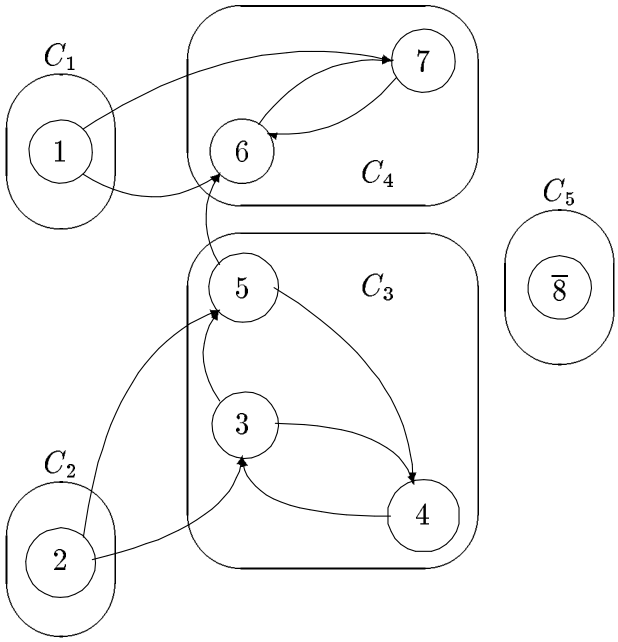

Figure 2: The graph $G_{\mathcal{F}'}$.

To illustrate the above procedure, let us consider the pure Horn CNF $\mathcal{F}$ of (4), the corresponding graph $G_{\mathcal{F}}$ of which is shown in Figure 1. In this example we have five strong components, the last two of which are switchable. Thus the variables in the last strong component C_5 can be switched first, i.e. x_8 will be replaced by $\overline{x}_8$, and vice versa, yielding the CNF

$$(\overline{x}_1 \vee \overline{x}_5 \vee x_6) \wedge (\overline{x}_1 \vee \overline{x}_6 \vee x_7) \wedge (\overline{x}_2 \vee \overline{x}_3 \vee x_5) \wedge (\overline{x}_2 \vee \overline{x}_4 \vee x_3) \wedge (\overline{x}_3 \vee \overline{x}_5 \vee x_4) \wedge (\overline{x}_4 \vee \overline{x}_7 \vee \overline{x}_8) \wedge (\overline{x}_7 \vee x_6)$$

which contains a negative clause. Thus the minimization of $\mathcal{F}$ is reduced to finding a minimum representation for

$$\mathcal{F}' = (\overline{x}_1 \vee \overline{x}_5 \vee x_6) \wedge (\overline{x}_1 \vee \overline{x}_6 \vee x_7) \wedge (\overline{x}_2 \vee \overline{x}_3 \vee x_5) \wedge (\overline{x}_2 \vee \overline{x}_4 \vee x_3) \wedge (\overline{x}_3 \vee \overline{x}_5 \vee x_4) \wedge (\overline{x}_7 \vee x_6). \quad (6)$$

The graph $G_{\mathcal{F}'}$ is shown in Figure 2. It has the same strong components as $G_{\mathcal{F}}$, but some edges between the strong components are missing, so that the switched strong component became isolated.

We can continue by switching the variables in the next switchable component C_4. After switching x_6 and x_7 we obtain

$$(\overline{x}_1 \vee \overline{x}_5 \vee \overline{x}_6) \wedge (\overline{x}_1 \vee x_6 \vee \overline{x}_7) \wedge (\overline{x}_2 \vee \overline{x}_3 \vee x_5) \wedge (\overline{x}_2 \vee \overline{x}_4 \vee x_3) \wedge (\overline{x}_3 \vee \overline{x}_5 \vee x_4) \wedge (x_7 \vee \overline{x}_6)$$

which again contains a negative clause. Thus the minimization of $\mathcal{F}$ is further reduced to finding a minimum representation of

$$\mathcal{F}'' = (\overline{x}_1 \vee x_6 \vee \overline{x}_7) \wedge (\overline{x}_2 \vee \overline{x}_3 \vee x_5) \wedge (\overline{x}_2 \vee \overline{x}_4 \vee x_3) \wedge (\overline{x}_3 \vee \overline{x}_5 \vee x_4) \wedge (x_7 \vee \overline{x}_6). \qquad (7)$$

The graph $G_{\mathcal{F}''}$ is shown in Figure 3. Since there are no more switchable components, the decomposition algorithm stops here.

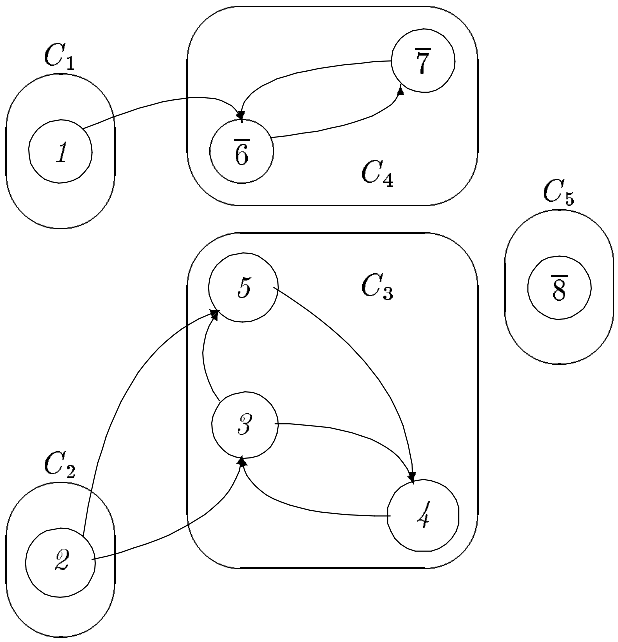

Figure 3: The graph $G_{\mathcal{F}''}$.

As a conclusion, we can claim that the CNF $\mathcal{F}$ of (4) has a minimum length CNF representation of the following form

$$\mathcal{F} = \mathcal{G} \wedge (\overline{x}_1 \vee \overline{x}_5 \vee x_6) \wedge (\overline{x}_4 \vee \overline{x}_7 \vee x_8), \qquad (8)$$

where $\mathcal{G}$ is an arbitrary minimum length CNF representation of the shorter sub-CNF

$$(\overline{x}_1 \vee \overline{x}_6 \vee x_7) \wedge (\overline{x}_2 \vee \overline{x}_3 \vee x_5) \wedge (\overline{x}_2 \vee \overline{x}_4 \vee x_3) \wedge (\overline{x}_3 \vee \overline{x}_5 \vee x_4) \wedge (\overline{x}_7 \vee x_6). \qquad (9)$$

Theorem 5.8 proves that the set of clauses that can be switched to negative ones depends only on the input CNF and not on the order in which the clauses are switched. This leads to a question of what happens if we start the decomposition process on two distinct CNFs of the same Horn function. The next section deals with such a situation. Given two distinct prime CNFs $\mathcal{F}_1$ and $\mathcal{F}_2$ representing the same Horn function, we shall investigate the relation between the graphs $G_{\mathcal{F}_1}$ and $G_{\mathcal{F}_2}$.

6 Blocked Components and Eliminated Clauses in Distinct CNFs

In this section we shall show another interesting property of the graph G_f associated with a Horn function f. In Section 5 we defined (recall Definition 5.5) for every CNF representation $\mathcal{F}$ of f which strong components of $G_{\mathcal{F}}$ are blocked. Here we shall prove that if we restrict our attention to prime CNFs of f then the property of being blocked does not depend on the particular CNF representation and hence can be thought of as a property of the function f and its associated graph G_f. Moreover, the function represented by the "unswitchable" pure Horn CNF will not depend on the initial CNF either. If in addition to being prime, the CNFs we consider are irredundant as well, then the number of clauses eliminated by switching will not depend on the initial CNF either. Thus, the number of eliminated negative clauses and the "unswitchable" pure Horn function can also be thought of as properties of the function f.

Theorem 6.1 *Let $\mathcal{F}_1$ and $\mathcal{F}_2$ be two distinct prime CNFs of a pure Horn function f and let C be an arbitrary strong component of G_f. Then C is blocked in $G_{\mathcal{F}_1}$ if and only if it is blocked in $G_{\mathcal{F}_2}$.*

Proof. Let $C_1, C_2, \ldots, C_s$ be the strong components of G_f given in a topological order and let C_i be an arbitrary component. By Corollary 4.4 the graphs $G_{\mathcal{F}_1}$ and $G_{\mathcal{F}_2}$ have the same strong components and have the same acyclic condensation as G_f (here we use the primality of $\mathcal{F}_1$ and $\mathcal{F}_2$). To prove that C_i is blocked in $G_{\mathcal{F}_1}$ if and only if it is blocked in $G_{\mathcal{F}_2}$ it is sufficient to show the following two facts:

- If there is a clause $T_1 = A \vee \overline{x} \vee \overline{y} \vee z \in \widehat{\mathcal{F}}_1$ such that $\{x, y, z\} \subseteq C_i$ (where A is a possibly empty disjunction of additional negative literals), then there is a clause $T_2 = B \vee \overline{u} \vee \overline{v} \vee w \in \widehat{\mathcal{F}}_2$ such that $\{u, v, w\} \subseteq C_i$. In other words, if C_i is blocked by an inside fork in $G_{\mathcal{F}_1}$ then it is also blocked by an inside fork in $G_{\mathcal{F}_2}$.
- If there is a clause $T_1 = A \vee \overline{x} \vee \overline{y} \vee z \in \widehat{\mathcal{F}}_1$ such that $\{x\} \subseteq C_i$ and $\{y, z\} \subseteq C_k$ for some $k > i$, then there is a clause $T_2 = B \vee \overline{u} \vee \overline{v} \vee w \in \widehat{\mathcal{F}}_2$ such that $\{u\} \subseteq C_j$ for some $C_i \preceq C_j$ and $\{v, w\} \subseteq C_k$. In other words, if C_i is blocked by an outside fork in $G_{\mathcal{F}_1}$, then there is a component C_j (possibly C_i itself) which is accessible by a directed path from C_i and which is blocked in $G_{\mathcal{F}_2}$, thus blocking C_i in $G_{\mathcal{F}_2}$ by inheritance.

Let us start the proof of the first statement by denoting (recall the definition of forward chaining in Section 2)

$$X = R_{\mathcal{F}_2}(A \cup \{x\}) \cap (\bigcup_{\ell=1}^{i} C_\ell)$$

$$Y = R_{\mathcal{F}_2}(A \cup \{y\}) \cap (\bigcup_{\ell=1}^{i} C_\ell)$$

$$Z = R_{\mathcal{F}_2}(A \cup \{x, y\}) \cap (\bigcup_{\ell=1}^{i} C_\ell)$$

Since T_1 is an implicate of f, $z \in Z$ must hold. Moreover, the fact that T_1 is prime implies $z \notin X$ and $z \notin Y$. Therefore, there must exist $u \in X \setminus Y$, $v \in Y \setminus X$, and a clause $T_2 = B \vee \overline{u} \vee \overline{v} \vee w \in \widehat{\mathcal{F}}_2$ (for some B and w) which is used in the forward chaining to eventually include z into Z. It is obvious that the sets X and Y coincide on each C_ℓ for $\ell < i$ and differ only on the component C_i, i.e. that $X \setminus Y \subseteq C_i$ and $Y \setminus X \subseteq C_i$. Hence $\{u, v\} \subseteq C_i$. Moreover, since T_2 is used to include $z \in C_i$ into Z, $w \in C_\ell$ for some $\ell \leq i$ must hold. On the other hand, the fact that $\{u, v\} \subseteq C_i$ and $T_2 \in \widehat{\mathcal{F}}_2$, forces $w \in C_\ell$ for some $\ell \geq i$. Thus $w \in C_i$ which finishes the proof of the first statement.

To prove the second statement we can proceed in a similar manner. We define the sets X, Y, and Z as above. By the same arguments, there must exist $u \in X \setminus Y$, $v \in Y \setminus X$, and a clause $T_2 = B \vee \overline{u} \vee \overline{v} \vee w \in \widehat{\mathcal{F}}_2$ (for some B and w) which was used in the forward chaining to eventually include z into Z. Considerations similar to the above show that $Y \setminus X \subseteq C_k$, implying $v \in C_k$ and $w \in C_k$. Let us assume $u \in C_j$. What remains to be shown is that $C_i \preceq C_j$. However, $u \in X \setminus Y$ implies that $A \vee \overline{x} \vee u$ is an implicate of $\mathcal{F}_2$ while $A \vee u$ is not. Therefore, there is a subset $A' \subseteq A$ such that $A' \vee \overline{x} \vee u$ is a prime implicate of $\mathcal{F}_2$, and therefore by Lemma 4.2 there is a path from x to u in $G_{\mathcal{F}_2}$ implying $C_i \preceq C_j$. Now there are two cases to consider. If $j = k$ then C_j is blocked by an inside fork in $G_{\mathcal{F}_2}$, and if $j < k$ then C_j is blocked by an outside fork in $G_{\mathcal{F}_2}$.

■

Now we are ready to show that for all the irredundant and prime CNFs of the same pure Horn function the number of negative clauses eliminated by switching and the remaining "unswitchable" pure Horn *function* are the same (although the remaining "unswitchable" pure Horn *CNFs* may be different).

Theorem 6.2 *Let $\mathcal{F}_1$ and $\mathcal{F}_2$ be two distinct prime CNFs of a pure Horn function f. Then the procedure* GRAPH-ALGO *(and therefore, the procedure* DECOMPOSITION*) applied to each of these two CNFs terminates with "unswitchable" pure Horn CNFs representing the same pure Horn function. If, additionally, both CNFs $\mathcal{F}_1$ and $\mathcal{F}_2$ are also irredundant, then the number of eliminated negative clauses is also the same.*

Proof. It follows from Corollary 4.4 that without loss of generality we can assume that the procedures GRAPH-ALGO($\mathcal{F}_1$) and GRAPH-ALGO($\mathcal{F}_2$) use the same topological order of strong components. Theorem 6.1 implies that since the same strong components are switchable in both $G_{\mathcal{F}_1}$ and $G_{\mathcal{F}_2}$, both procedures will be renaming the same strong components in the same order.

Let $\mathcal{F}_1'$ and $\mathcal{F}_2'$ denote the Horn CNFs resulting from the first switch. Clearly, they are both prime, and represent the same Horn function (since the same variables are switched). Then by Theorem 3.1, $\mathcal{F}_1'^{\mathcal{H}}$ and $\mathcal{F}_2'^{\mathcal{H}}$ are prime CNFs representing the same pure Horn function. Therefore, by induction, at each step of both procedures logically equivalent prime pure Horn CNFs will be produced.

If $\mathcal{F}_1$ and $\mathcal{F}_2$ are also irredundant, then $\mathcal{F}_1'$ and $\mathcal{F}_2'$ are both irredundant and prime, and by Theorem 3.2, $\mathcal{F}_1'^{\mathcal{N}}$ and $\mathcal{F}_2'^{\mathcal{N}}$ contain the same number of negative clauses. Therefore, by induction, at each step of both procedures the same number of negative clauses will be eliminated.

■

7 Acyclic and Quasi-Acyclic CNFs

In this section we shall investigate how the algorithm GRAPH-ALGO performs on CNFs from the two classes, for which Horn Minimization is known to be polynomially solvable (see [13]).

A Horn CNF $\mathcal{F}$ is said to be *acyclic* if the graph $G_{\mathcal{F}}$ is acyclic. In such a case each node of $G_{\mathcal{F}}$ constitutes a one-element strong component. Therefore, no inside or outside forks may exist and thus no strong component is blocked. That means that all clauses in $\mathcal{F}$ are in $N_0(\mathcal{F})$ and as such, they are eliminated from $\mathcal{F}$ by the algorithm GRAPH-ALGO, i.e. there is nothing left to minimize. This gives an alternative proof of the fact that each acyclic irredundant and prime CNF is minimal.[8]

A Horn CNF $\mathcal{F}$ is said to be *quasi-acyclic* if the graph $G_{\mathcal{F}}$ has the following property: if nodes x and y belong to the same strong component of $G_{\mathcal{F}}$ and (x, y) is an arc, then $\overline{x} \vee y$ is a clause in $\mathcal{F}$. In other words, a CNF $\mathcal{F}$ is quasi-acyclic if all arcs that lie inside strong components of $G_{\mathcal{F}}$ come from quadratic clauses in $\mathcal{F}$. This again means that no inside or outside forks may exist, and hence no strong component is blocked. In this case the partition of clauses in $\mathcal{F}$ looks as follows

- $B(\mathcal{F}) = \emptyset$
- $N_1(\mathcal{F}) = \{C = \overline{x} \vee y \mid x, y \text{ are from the same strong component of } G_{\mathcal{F}}\}$

[8]In fact, more is true (as shown in [13]). An acyclic Horn function has a unique irredundant and prime CNF representation which is then, of course, also minimal.

- $N_0(\mathcal{F}) = \{$ all other clauses $\}$

It follows that the algorithm GRAPH-ALGO eliminates all clauses of $\mathcal{F}$ except for those quadratic clauses that create the strong components of $G_{\mathcal{F}}$. This gives an alternative proof of the fact (see [13]) that a minimal CNF of a quasi-acyclic function can be obtained from an arbitrary irredundant and prime CNF by replacing the quadratic clauses in $N_1(\mathcal{F})$ by a set of simple cycles of quadratic clauses, one cycle per strong component of $G_{\mathcal{F}}$ (each cycle spanning the entire component).

References

[1] G. Ausiello, A. D'Atri and D. Sacca. Minimal representation of directed hypergraphs. *SIAM Journal on Computing*, **15**(1986), 418–431.

[2] E. Boros and O. Čepek. On the complexity of Horn minimization. *RUTCOR Research Report RRR 1-94*, Rutgers University, New Brunswick, NJ, January 1994.

[3] O. Čepek. *Structural Properties and Minimization of Horn Boolean Functions. Doctoral dissertation*, Rutgers University, New Brunswick, NJ, October 1995.

[4] R. Dechter and J. Pearl. Structure identification in relational data. *Artificial Intelligence* **58**(1992), 237-270.

[5] C. Delobel and R.G. Casey. Decomposition of a Data Base and the Theory of Boolean Switching Functions. *IBM Journal of Research and Development*, **17**(1973), 374-386.

[6] W.F. Dowling and J.H. Gallier. Linear time algorithms for testing the satisfiability of propositional Horn formulae. *Journal of Logic Programming*, **3**(1984), 267–284.

[7] M.R. Garey and D.S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness.* W.H. Freeman and Company, San Francisco, 1979.

[8] M.R. Genesereth and N.J. Nilsson. *Logical Foundations of Artificial Intelligence.* Morgan Kaufmann, Los Altos, CA, 1987.

[9] P.L. Hammer and A. Kogan. Horn functions and their DNFs. *Information Processing Letters*, **44**(1992), 23–29.

[10] P.L. Hammer and A. Kogan. Horn function minimization and knowledge compression in production rule bases. *RUTCOR Research Report RRR 8-92*, Rutgers University, New Brunswick, NJ, March 1992.

[11] P.L. Hammer and A. Kogan. Optimal compression of propositional Horn knowledge bases: Complexity and approximation. *Artificial Intelligence*, **64**(1993), 131-145.

[12] P.L. Hammer and A. Kogan. Knowledge compression – logic minimization for expert systems. *Computers As Our Better Partners. Proceedings of the IISF/ACM Japan International Symposium.* World Scientific, Singapore, 1994, 306-312.

[13] P.L. Hammer and A. Kogan. Quasi-acyclic propositional Horn knowledge bases: Optimal compression. *IEEE Transactions on Knowledge and Data Engineering*, **7**(5)(1995), 751-762.

[14] A. Itai and J.A. Makowsky. Unification as a complexity measure for logic programming. *Journal of Logic Programming*, **4**(1987), 105-117.

[15] D.E. Knuth. *Fundamental Algorithms. Vol.1 of The Art of Computer Programming*, Addison-Wesley, 1968 (second edition 1973).

[16] D. Maier. Minimal covers in the relational database model. *Journal of the ACM*, **27**(1980), 664-674.

[17] M. Minoux. LTUR: A simplified linear time unit resolution algorithm for Horn formulae and computer implementation. *Information Processing Letters*, **29**(1988), 1-12.

[18] R.E. Tarjan. Depth first search and linear graph algorithms. *SIAM Journal on Computing*, **2**(1972), 146-160.