Optimizations in Decision Procedures for Propositional Linear Inequalities

Ofer Strichman

May 23, 2002

CMU-CS-02-133

School of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213

Abstract

Several decision procedures that were published in the last few years for sub-theories of propositional linear inequalities, i.e. a Boolean combination of predicates that belong to the theory, are based on a graph-based analysis of the formula's predicates. The analysis is always based on the predicates while ignoring the Boolean connectives between them. In this note we show how taking this information into account can significantly reduce the (practical) complexity of the decision procedure.

This research was supported in part by the Office of Naval Research (ONR) and the Naval Research Laboratory (NRL) under contract no. N00014-01-1-0796

Keywords: Verification, Decision-procedure, Equality-Logic, Conjunctions-matrices

1 Introduction

Several decision procedures that were published in the last few years [GSZ$^+$98], [PRSS99], [BGV99], [BV00], [BGV01], [SSB02] for equality logic (Boolean combinations of equalities) and for a logic of separation predicates (Boolean combinations of predicates of the form $x > y + c$ where x, y are variables and c is a constant) follow similar guidelines[1]. First, they represent the predicates of the examined formula φ in a graph. Second, they analyze this graph in order to apply the transitivity of the predicates. This can result in a list of explicit constraints [BV00,SSB02] or a finite range of values to each variable [PRSS99,BGV99]. In [GSZ$^+$98] the structure of the formula is not analyzed explicitly, but, as we will show in the appendix, our method is applicable to it as well.

We illustrate this approach with the work of Bryant et al. [BV00] on equality logic. Each predicate $i = j$ is encoded with a new Boolean variable $e_{i,j}$. Let φ' denote the encoded formula. In order to retain the transitivity of equality, they construct an undirected graph $G(V, E)$, where the nodes in V represent φ's variables, and there is an edge in E between two nodes i and j if and only if there is a predicate $i = j$ (or $i \neq j$) in φ. Transitivity of equality forbids an assignment in which all edges of a cycle except one are assigned TRUE. Thus, it is sufficient to add such a constraint to φ' for each (simple) cycle in the graph. This construction guarantees that φ' is satisfiable if and only if φ is satisfiable.

The added constraints reflect the transitivity constraints of equality, which implicitly exist in the original formula. But since each constraint adds to the complexity of the decision procedure, it is interesting to check whether all of these constraints are needed in order to preserve the soundness of the procedure. More specifically, it is possible that with a more careful analysis of the formula structure, the number of constraints can be reduced. This improvement is the subject of this note.

In all of these procedures, the graph analysis stage is based only on the predicates in φ, and ignores the Boolean connectives (i.e. conjunctions and disjunctions) between them. Thus the formulas $\varphi_1 : x = y \wedge y = z \wedge z = x$ and $\varphi_2 : x = y \vee y = z \vee z = x$ result in the same set of constraints (in this case preventing an assignment in which exactly two of these predicates are assigned TRUE). But as we will show in the next section, such a constraint is not needed for φ_2. Furthermore, it is possible to identify this fact in polynomial time.

While our method reduces the number of constraints, it has a drawback when it comes to reconstructing a satisfying assignment (rather than only deciding the formula). We will elaborate on this point in section 4.

2 A decision procedure

We demonstrate our suggested procedure for the case of equality predicates [BV00]. It can be applied in a very similar manner to [SSB02]. In the appendix we describe how it can also be applied to some of the other procedures that we referred to earlier.

Let φ be an equality formula in Disjunctive Normal Form (DNF), i.e., $\varphi : \bigvee \bigwedge p_k$ where p_k is an equality predicate of the form $i = j$ or $i \neq j$ for some variables i and j (we later abandon the requirement for having the formula in DNF. We only need it here for clarity of the explanation).

Proposition 1. *φ is unsatisfiable if and only if each of its clauses contain a sub-formula of the form $\psi : i_1 \neq i_2 \wedge i_2 = i_3 \ldots \wedge i_n = i_1$ for $n \geq 2$. We call ψ an unsatisfiable cycle.*

[1] We assume in this note that the reader is familiar at least with [BV00] and [SSB02]

The following procedure relies on proposition 1:

1. Encode each predicate of the form $i = j$ in φ with a new Boolean variable $e_{i,j}$. Let φ'' denote the encoded formula.
2. Construct a graph $G(V, E)$ s.t. the nodes are the variables in φ and there is an edge (i, j) in E if and only if there is a predicate $i = j$ in φ.
3. For each cycle C of predicates that belong to the same clause, add a constraint to φ'' that forbids an assignment in which exactly $|C| - 1$ edges are assigned TRUE.

The formula φ'' is a conjunction of two subformulas, which we denote by φ''_f and φ''_c. φ''_f is the Boolean encoding of φ resulting from step 1. φ''_c is a conjunction of the constraints that we add in step 3.

The difference between φ'', as derived in this procedure, and φ', the formula derived in [BV00], is that in the latter transitivity constraints are added regardless of the Boolean connectives of φ. In other words, a constraint is added to φ' for each cycle of predicates, even if these predicates do not share a clause. The following proposition justifies this reduction:

Proposition 2. *φ'' is satisfiable if and only if φ' is satisfiable.*

Proof. ($\Leftarrow$) φ'' has, by construction, only a subset of the constraints of φ'. Therefore it is trivial that if φ' is satisfiable then so is φ''. ($\Rightarrow$) Assume φ, and consequently φ', is unsatisfiable. In this case, according to proposition 1, each of the clauses of φ has an unsatisfiable cycle C. Since the predicates of C are conjuncted, in order to satisfy the corresponding encoded cycle in φ'' ($\neg e_{i_1,i_2} \wedge e_{i_2,i_3} \wedge \ldots \wedge e_{i_n,i_1}$) the first variable ($e_{i_1,i_2}$) has to be FALSE and the rest have to be TRUE, i.e. exactly $|C| - 1$ edges are assigned TRUE. But this contradicts the constraints that are added to φ'' in step 3. Hence, the encoded cycle can not be satisfied, and therefore φ'' is unsatisfiable. Thus, if φ'' is satisfiable, then so is φ'. $\square$

The above decision procedure is computationally expensive, because it requires φ to be in DNF, which may impose an exponential growth in the size of the formula. Fortunately this is not really required. We can predict, in polynomial time, which predicates would share a clause *if* the formula was transformed to DNF. This can be done with a syntactic analysis of φ''_f, as explained in the next subsection.

2.1 Conjunctions matrices

Assume φ is given in Negation Normal Form (all negations are pushed to the atomic predicates). Consequently all the internal nodes of the parse tree of φ''_f, except those that are immediately adjacent to the leafs, correspond to either disjunctions or conjunctions (recall that φ''_f represents a Boolean encoding of φ's predicates) . For each pair of leafs, there is a single internal node from which these two leafs can be reached via non overlapping directed paths. We call the Boolean operand represented by this node the *joining operand* of these two leafs.

Example 1. Consider the formula:

$$\varphi : (x_1 = x_2) \wedge (x_3 \neq x_4 \wedge (x_4 = x_5 \vee x_5 = x_6))$$

The Boolean encoding of φ is given by

$$\varphi''_f : e_{x_1,x_2} \wedge (\neg e_{x_3,x_4} \wedge (e_{x_4,x_5} \vee e_{x_5,x_6})$$

The joining operand of e_{x_3,x_4} and e_{x_5,x_6} is '$\wedge$'. The joining operand of e_{x_4,x_5} and e_{x_5,x_6} is '$\vee$'. $\square$

For simplicity, we first assume that no predicates appear in φ more than once. Denote by φ^D the formula φ after it is transformed to DNF. The following proposition is the basis for the prediction technique:

Proposition 3. *Two predicates share a clause in φ^D if and only if their joining operand is a conjunction.*

We construct a matrix that holds this information for all pairs of predicates:

Definition 1. *Let m denote the number of predicates in φ. The* conjunctions matrix M_φ *of φ is an $m \times m$ binary, symmetric matrix, where $M_\varphi[e_{i,j}][e_{k,l}] = 1$ if and only if the joining operand of the two predicates $i = j$ and $k = l$ is a conjunction.*

In the graph G, the entries of M_φ can be thought of as 'edges that connect edges'.

For a given pair of predicates, it is a linear operation (in the height h of the parse tree) to check whether their joining operand is a conjunction or disjunction. Therefore constructing M_φ has the complexity of $O(m^2h)$.

The information in M_φ is sufficient for concluding whether a set of predicates share a clause in φ^D:

Proposition 4. *A set of predicates share a DNF clause in φ^D if and only if their associated entries in the conjunctions matrix form a clique.*

Note that the 'clique' refers to the edges between the edges of G, and not to the edges of G itself.

Given a set of predicates, finding whether they form a clique in M_φ is quadratic in the size of the set. Thus, we do not require anymore that φ is given in DNF, and pay a quadratic price for this in step 3 of the procedure.

Example 2. Consider the formula $\varphi : x = y \wedge (y = z \vee x = z)$. The corresponding conjunctions matrix is

$$M_\varphi = \begin{array}{c|ccc} & e_{x,y} & e_{y,z} & e_{x,z} \\ \hline e_{x,y} & 0 & 1 & 1 \\ e_{y,z} & 1 & 0 & 0 \\ e_{x,z} & 1 & 0 & 0 \end{array}$$

(Note that M_φ is symmetric by definition).

The graph G includes the cycle $x - y - z$. But since $M_\varphi[e_{y,z}][e_{x,z}] = 0$, there is no clique between its edges, and we can therefore conclude that these three predicates do not share a clause in φ^D. Consequently no constraint is added to φ''. $\square$

2.2 Handling repeating predicates

Practically most formulas contain predicates that appear more than once, in different parts of the formula. We will denote by $e^k_{i,j}$, $k \geq 1$ the k instance of the predicate $e_{i,j}$ in φ''_f. It is possible that the same pair of predicates will have different joining operands. There are two possible solutions to this problem:

1. Represent each predicate instance as a separate edge. This will make G a multigraph, where the number of edges between two nodes i, j is equal to the number of instances of the predicate $i = j$ in φ. The size of the matrix M_φ will grow accordingly.
2. Assign $M_\varphi[e_{x,y}][e_{z,l}] = 1$ if there exists an instance of $x = y$ and of $z = l$ with a joining operand '$\wedge$'.

For comparison between the two options, denote the matrix constructed in option i by M^i_φ. Then $M^2_\varphi[e_{x,y}][e_{z,l}] = 1$ if and only if there exists k_1, k_2 s.t. $M^1_\varphi[e^{k_1}_{x,y}][e^{k_2}_{z,l}] = 1$.

The second option has a more concise representation, but may result in redundant constraints, as the example below demonstrates.

Example 3. Consider the formula $\varphi : (x = y \wedge (y = z \vee x = z)) \vee (y = z \wedge x = z)$. The two options are depicted in Fig. 1.

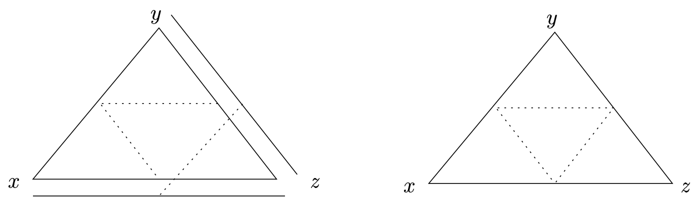

Fig. 1. Handling multiple instances of predicates through (a) separate edges and (b) joint edges.

Following option 2 (Fig. 1(b)) will yield a redundant constraint, because the information that the three predicates never appear together in the same clause is lost. On the other hand this is a more concise representation than the first option. □

Redundant constraints can not make the method incomplete. Recall that all of these constraints are part of φ', which, according to [BV00], is satisfiable if and only if φ is satisfiable.

3 Chordal graphs

Since there can be an exponential number of cycles in G, both [BV00] and [SSB02] first make the graph *chordal* by adding edges to the graph. An undirected graph is chordal iff every cycle of size 4 or more has an internal chord. In the case of directed and weighted graphs the definition is more complicated: it refers to directed cycles of size 4 or more, and each chord from node i to j is a projection of the path between these two nodes on the cycle. In both of these references, it is proven that in a chordal graph, if a transitivity constraint is violated, then a transitivity constraint of a cycle of size 3 is violated as well. This allows them to add constraints only for cycles of size 3 or less. Consequently the number of constraints reduces to a polynomial in [BV00] and have a similar effect in some cases considered by [SSB02]. We now examine the question of how to integrate conjunctions matrices with chordal graphs. Again, we concentrate on [BV00]. A known procedure for making a graph chordal is the following:

While there are vertices in the graph {
  i) select a variable v.
  ii) add chords between all immediate neighbors of v.
  iii) remove v and its adjacent edges from the graph.
}

In the case of [SSB02] the procedure is slightly more complicated in line ii), because the graph is directed and weighted. One may think of this as an iterative procedure of eliminating variables and projecting their associated constraints to the remaining variables (similar to the Fourier-Motzkin technique).

Integrating conjunctions matrices into this process is simple: we only need to project information from pairs of constraints (edges) that have a conjunction as their joining operand. Thus, we change line ii) as follows:

ii) add a chord (x, y) if and only if $(x, v) \in E$ and $(v, y) \in E$ and $M_\varphi[e_{x,v}][e_{v,y}] = 1$.

We demonstrate the effectiveness of this method in the case of the directed weighted graphs of [SSB02]. The main problem in applying [SSB02] is that the process of making the graph chordal may add an exponential number of edges. Conjunctions matrices can significantly reduce this number, as demonstrated by the example below.

Example 4. Consider the diamond shaped topology of Fig. 2. Assume that all edges are conjoined with one another (they all appear in the same clause) except the edge between v_0 and v_1, which appears in a separate clause. Also assume some arbitrary distribution of weights over these edges. We now analyze what different procedures do with such a formula:

1. In methods based on case splitting, such as Pratt's method [Pra77] (which performs a graph analysis similar to [SSB02] but assumes that all edges are conjoined), the formula is first transformed to DNF, and then each clause is analyzed separately. In this case an exponential number of graphs will be constructed, only to discover that non of them contains a cycle.
2. In [SSB02], in the worst case an exponential number of chords is added to the graph, and an exponential number of constraints is added to the formula.
3. When using a combination of [SSB02] and conjunction matrices, no chords are added to the graph, no constraints are added to the formula, and the procedure terminates in linear time. This means that this combination of methods is able to capture, efficiently, the fact that there are no conjoined cycles in this graph. Note that constraints are always associated with cycles and therefore a sub-graph without cycles shouldn't impose any constraints. With the help of conjunctions matrices we can take this rule one step further, and say that only conjoined cycles, i.e., cycles of predicates that share the same clause, should impose a constraint. In the graph of Fig. 2 there are no such cycles and therefore an optimal procedure will not add constraints at all.

Yet, the process of making the graph chordal, even if it doesn't contain a cycle, may add chords (assuming we do not explicitly look for such sub-graphs and discard them). Chordal graphs are not unique, i.e., for a given graph G, there are many possible completions of G that make it chordal. The order in which nodes are chosen in step i) determines the exact chordal completion, and therefore various heuristics can be used in order to minimize the number of added chords. The implementation of [SSB02] uses a simple greedy criterion: at each iteration it picks the node that imposes the minimal number of added chords. For the graph of Fig. 2 this criterion is sufficient for identifying that no additional chords are needed: due to the conjunctions matrix analysis no chord is added if v_0 is removed, and therefore it is removed first. Now the removal of v_1 will not add any edges (because it doesn't have incoming edges), and so forth. Thus, nodes are removed from right to left without adding chords or constraints.

□

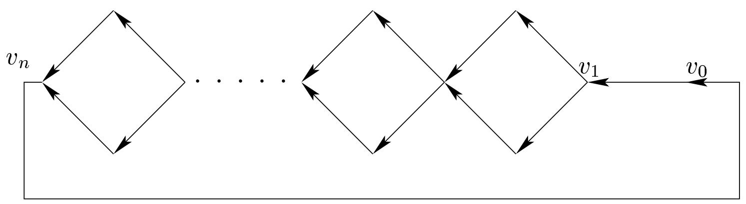

Fig. 2. An n diamonds topology, where the edge (v_0, v_1) is disjointed from the rest of the formula.

4 Reconstruction of a satisfying assignments

In many applications knowing whether a formula is satisfiable or not is not enough. The actual satisfying assignment to φ has to be reconstructed. The satisfying assignment to φ' (recall that φ' denotes the encoded formula of [BV00] without conjunctions matrices) indicates the Boolean value that each predicate should evaluate to in order to satisfy φ. It is not hard to derive the values for φ's variables from this information. Let α be a satisfying assignment to φ' and let $G_T(V, E_T)$ be a graph s.t. V correspond to φ's variables, and $E_T = \{e | e \in E \wedge \alpha(e) = T\}$, i.e., the edges in E_T correspond to the edges in $G(V, E)$ that were assigned TRUE by α. To construct a satisfying assignment to φ, decide on a unique value to each maximal connected component on G_T, and then assign this value to each of the nodes (variables) in the component. By construction this assignment will evaluate all predicates in φ the same way as their encoding was evaluated in φ'.

This becomes more complicated with conjunctions matrices, because the assignments to φ'''s predicates can not always be reconstructed in φ simultaneously. For example in the formula $\varphi'' : e_{x,y} \vee e_{y,z} \vee e_{x,z}$ a possible satisfying assignment is one in which the first two predicates are assigned TRUE and the third predicate is assigned FALSE. Since there is a disjunction between these predicates, no constraint is added to φ'' that will prevent such an assignment. Clearly no values can be assigned to x, y, z in the original formula φ s.t. $x = y$,$y = z$ will be true and $x = z$ is false. With conjunctions matrices, we are able to reconstruct a satisfying assignment of one or more clauses of φ^D, but not necessarily all of them at the same time. Clearly one clause is sufficient to satisfy φ, but the problem is that we do not know which one it is.

We could not find a polynomial solution to this problem in each of the procedures that are based on Boolean encoding [GSZ+98,BV00,SSB02], and we leave it as an open problem. If there is no such solution and the satisfying assignment is needed, then conjunctions matrices can not be used. Procedures that are not based on encoding, but rather on allocation of a finite domain to each variable [PRSS99,BGV99] do not have this problem, because the satisfying assignment assigns values directly to φ's variables.

The relevance of conjunctions matrices to [PRSS99,BGV99] as well as to [GSZ+98] is briefly described in the appendix.

References

[BGV99] R. Bryant, S. German, and M. Velev. Exploiting positive equality in a logic of equality with uninterpreted functions. In *Proc. 11^{th} Intl. Conference on Computer Aided Verification (CAV'99)*, 1999.

[BGV01] R. Bryant, S. German, and M. Velev. Processor verification using efficient reductions of the logic of uninterpreted functions to propositional logic. *ACM Transactions on Computational Logic*, 2(1):1–41, 2001.

[BV00] R. Bryant and M. Velev. Boolean satisfiability with transitivity constraints. In E.A. Emerson and A.P. Sistla, editors, *Proc. 12^{th} Intl. Conference on Computer Aided Verification (CAV'00)*, volume 1855 of *Lect. Notes in Comp. Sci.* Springer-Verlag, 2000.

[GSZ+98] A. Goel, K. Sajid, H. Zhou, A. Aziz, and V. Singhal. BDD based procedures for a theory of equality with uninterpreted functions. In A.J. Hu and M.Y. Vardi, editors, *CAV98*, volume 1427 of *LNCS*. Springer-Verlag, 1998.

[Pra77] V. Pratt. Two easy theories whose combination is hard. Technical report, Massachusetts Institute of Technology, 1977. Cambridge, Mass.

[PRSS99] A. Pnueli, Y. Rodeh, O. Shtrichman, and M. Siegel. Deciding equality formulas by small-domains instantiations. In *Proc. 11^{th} Intl. Conference on Computer Aided Verification (CAV'99)*, Lect. Notes in Comp. Sci. Springer-Verlag, 1999.

[SSB02] O. Strichman, S.A. Seshia, and R.E. Bryant. Deciding separation formulas with SAT. In *Proc. 14^{th} Intl. Conference on Computer Aided Verification (CAV'02)*, LNCS, Copenhagen, Denmark, July 2002. Springer-Verlag. (To appear).

A Conjunctions matrices applied to other procedures

We shortly describe how conjunctions matrices can be combined with the methods of Goel et al. [GSZ+98], Bryant et al [BGV99] and Pnueli et al. [PRSS99]. All three methods decide the same type of formulas as [BV00], i.e. a propositional combination of equalities. In the following description it is assumed that the reader is familiar with these methods.

Goel et al. [GSZ+98]: The method is based on e_{ij} encoding of the formula, similar to [BV00]. After encoding, the resulting formula is represented as a BDD. Then the procedure looks for consistent paths in the BDD that lead to '1'. A consistent path is an assignment to the Boolean variables that do not contradict the transitivity of equality. For example, a path in which e_{ij} and e_{jk} are TRUE but e_{ik} is FALSE is inconsistent, because it contradicts the implicit transitivity constraint $i = j \wedge j = k \rightarrow i = k$. With conjunctions matrices, an inconsistent path is one that includes the above assignment and all three predicates share the same clause in φ^D. For example, consider the formula $\varphi : x = y \lor y = z \lor z \neq x$. An assignment TRUE to all three predicates is a valid assignment although it can not be reconstructed in φ. The only information that this assignment gives us is that it is possible to satisfy separately each of these clauses.

Bryant et al. [BV00]: The method is based on assigning unique constant values to positive terms (p-terms), while assigning an increasing range of values to other terms (g-terms). For a connected component of size n of g-terms, the resulting range imposes a state-space of $n!$. The definition of a 'connected component' can be changed to incorporate the information given by the conjunctions matrix. A connected component is a set of g-terms that, in addition to the fact that they are connected on the graph G, they also share a clause in φ^D.

Pnueli et al. [PRSS99]: The method is based on finding a small domain (a range of values) for each variable which is sufficient for preserving satisfiability. This is also a graph-based algorithm, but there are two types of edges in this graph, $G_=$ and $G_{\neq}$, corresponding to equalities and disequalities in φ respectively (G is thus a union of these two graphs, and some of the nodes are shared by both sub-graphs). In each step, a shared node v is given a new unique value *char*(v) (called the *characteristic value* of v). Then, *char*(v) is added to the range of all other nodes that have a $G_=$ path to v. Finally, v is removed from the graph, and the process is repeated until there are no more shared nodes. With conjunctions matrices, this can be altered: the value needs to be given to every node that has a *conjoined* $G_=$ path to v, i.e. the edges in the path share a clause in φ^D. Other stages in this algorithm can be altered in a similar way, although the above step has the largest effect on the size of the computed domains.