Tightened Transitive Closure of Integer Addition Constraints

Peter Revesz
Department of Computer Science and Engineering
University of Nebraska-Lincoln
revesz@cse.unl.edu

Abstract

We present algorithms for testing the satisfiability and finding the tightened transitive closure of conjunctions of addition constraints of the form $\pm x \pm y \leq d$ and bound constraints of the form $\pm x \leq d$ where x and y are integer variables and d is an integer constrant. The running time of these algorithms is a cubic polynomial in the number of input constraints. We also describe an efficient matrix representation of addition and bound constraints. The matrix representation provides a easy, algebraic implementation of the satisfiability and tightened transitive closure algorithms. We also outline the use of these algorithms for the improved implementation of abstract interpretation methods based on the octagonal abstract domain.

1. Introduction

Many problems can be described by constraints of the form:

$$\textbf{Addition}: \quad \pm x \pm y \leq d$$

$$\textbf{Bound}: \quad \pm x \leq d$$

where x, y are rational or integer variables, and d is a constant (Revesz 2002). Addition constraints are also called *unit two variables per inequality (UTVPI)* constraints, and bound constraints are also called *single variable per inequality (SVPI)* constraints (Jaffar et al. 1994).

One recent application of these constraints is in the abstract interpretation of programs. Abstract interpretation derives either an over-approximation or an under-approximation of the collection semantics of a program (Cousot and Cousot 1976). (Miné 2001) pointed out that the above constraints can be used as an abstract domain, called the octagon abstract domain, and the program semantics can be abstracted in terms of a disjunctive normal form formula of the above constraints. This new abstract domain is more precise than intervals (Cousot and Cousot 1976) but simpler than polyhedra (Cousot and Halbwachs 1978).

Addition constraints include *difference constraints* of the form $x - y \leq d$. Since difference constraints play an important role in software verification (Alur et al. 1995; Clarke 1999; McMillan 1993), it is an interesting question whether addition constraints will also prove beneficial for software verification. It is well-known that difference constraints can be represented by *Difference-Bound Matrices* or DBMs, which allow the definition of many efficient operations on them when the variables range over either the rational numbers or the integers.

An abstract interpretation of Datalog queries with bound and difference constraints is implemented in the MLPQ constraint database system (Revesz et al. 2000), based on earlier constraint database concepts (Kanellakis et al. 1995, Revesz 1993). Using this abstract interpretation, the MLPQ system can be transformed into a software verification system of both logic programs and procedural programs (Delzanno and Podelski 1999; Fribourg and Olsén 1997; Fribourg and Richardson 1996; Revesz 2007). The MLPQ system also allows checking whether a given error condition and the under-approximation of the program semantics intersect, thereby proving that certain errors can occur during program executions (Anderson and Revesz 2005).

The applicability of the octagon abstract domain will depend largely on whether similarly efficient operators can be defined on it. There are currently some promising results for rational numbers. (Miné 2001) shows that testing the satisfiability of addition and bound constraints over n rational variables can be done in $O(n^3)$ time. However, (Miné 2001) conjectures the satisfiability problem to be $O(n^4)$ with integer variables.

Currently the best tightened transitive closure algorithm with n integer variables and m input constraints requires $O(mn^2)$ time (Harvey and Stuckey 1997). This algorithm also returns "unsatisfiable" if the input constraints have no integer solution. Since with n integer variables m is $O(n^2)$, this algorithm runs in $O(n^4)$ time in the worst case.

(Péron and Halbwachs 2007) recently discovered another interesting difference between the case of rational and integer variables. They show that conjunctions of rational difference constraints and disequality constraints of the form $x \neq y$ still allow satisfiability testing in $O(n^3)$ time. This is surprising because with integer variables the same problem is known to be NP-complete by a reduction of the graph 3-colorability problem (Rosenkrantz and Hunt 1980).

The result of (Péron and Halbwachs 2007) suggested the possibility that for conjunctions of bound and addition constraints too the basic operators will be harder with integer

Copyright © 2009, Association for the Advancement of Artificial Intelligence (www.aaai.org). All rights reserved.

variables than with rational variables. In spite of the earlier conjectures and suggestions, in this paper we present an $O(n^3)$ satisfiability testing algorithm with integer variables. We also present an $O(n^3 \log n)$ algorithm that finds the tightened transitive closure for conjunctions of integer addition and bound constraints.

The rest of this paper is organized as follows. Section 2 gives a brief review of the basic concepts and previous work. Section 3 presents a variable elimination algorithm and a satisfiability testing algorithm, which is based on successive variable elimination. Section 4 presents a tightened transitive closure algorithm for conjunctions of addition and bound constraints. Finally, Section 5 discusses some related and future work.

2. Basic Concepts

2.1 Addition-Bound Matrices

Several researchers developed *Addition-Bound Matrices* or ABMs to represent conjunctions of addition and bound constraints in analogy to the well-known *Difference-Bound Matrices* or DBMs that represent conjunctions of bound and difference constraints.

(Miné 2001) represents a conjunction of addition and bound constraints over variables $V = \{x_1, \ldots, x_n\}$ by a conjunction of difference constraints over variables $V^+ = \{x_1^+, x_1^-, \ldots, x_n^+, x_n^-\}$, that is, every variable has a positive form x_i^+ equivalent to x_i and a negative form x_i^- equivalent to $-x_i$. While this representation takes advantage of already well-developed DBM algorithms, in this paper we prefer a simple three-matrices representation.

We say that an addition constraint $ax + by \leq d$ (or bound constraint $ax \leq d$) *trivially implies* another addition constraint $ax+by \leq d'$ (respectively, bound constraint $ax \leq d'$) if $d < d'$. We always simplify the given addition and bound constraints by deleting those constraints that are trivially implied by other given constraints.

We allow 0 to be a special variable, denoted as x_0, and we number the other variables as $x_1, \ldots, x_n$. Using x_0 as a dummy variable, we write every bound constraint as an addition constraint. Due to the deletion of trivially implied constraints, for each $a, b \in \{-1, 1\}$ combination and distinct variables x_i and x_j, we can have only one constraint of the form $ax_i + bx_j \leq d$.

We also rewrite every $-x_i + x_j \leq d$ into $x_j - x_i \leq d$, every $x_i + x_j \leq d$ where $i > j$ into $x_j + x_i \leq d$, and every $-x_i - x_j \leq d$ where $i > j$ into $-x_j - x_i \leq d$. We use matrix M_{+-} to represent constraints when $a = 1,\ b = -1$, matrix M_{++} to represent constraints when $a = b = 1$, and matrix M_{--} to represent constraints when $a = b = -1$, as follows:

$$M_{+-}[i,j] = \left\{ \begin{array}{ll} d & \text{if } (x_i - x_j \leq d) \in C \\ +\infty & \text{otherwise} \end{array} \right\}$$

$$M_{++}[i,j] = \left\{ \begin{array}{ll} d & \text{if } (x_i + x_j \leq d) \in C \\ +\infty & \text{otherwise} \end{array} \right\}$$

$$M_{--}[i,j] = \left\{ \begin{array}{ll} d & \text{if } (-x_i - x_j \leq d) \in C \\ +\infty & \text{otherwise} \end{array} \right\}$$

The above is a particularly efficient and convenient representation of addition and bound constraints because M_{++} and M_{--} are upper triangular matrices and do not need the variable 0.

Example 1 Consider the following conjunction of addition and bound constraints over the variables x and y:

$$-x \leq -25,\ y \leq 3,\ x-y \leq 4,\ x+y \leq 10,\ -x-y \leq -40$$

This set of addition and bound constraints can be represented by the following three matrices.

M_{+-}

	0	x	y
0	$+\infty$	-25	$+\infty$
x	$+\infty$	4	$+\infty$
y	3	$+\infty$	$+\infty$

M_{++}

	x	y
x		10
y		

M_{--}

	x	y
x		-40
y		

This representation is useful to check whether we have a constraint of a certain form and to avoid trivially implied constraints in our representation. In the following we assume that when we add a new constraint $ax_i + bx_j \leq d$ to matrix M_{ab}, then we overwrite $M_{ab}[i,j]$ with d if its current value is greater than d. We also assume that initially all entries are $+\infty$.

For simplicity we will not always use the above representation in describing our algorithms when the representation is not needed. However, it is easy to translate our algorithms into one that uses the above matrix representation even though we prefer to describe some of the algorithmic ideas using graphs. The translation from graphs to matrices is the basis of efficient, algebraic computer implementations.

2.2 Harvey-Stuckey Tightened Transitive Closure

If $\pm x \pm y \leq d$ is an addition constraint, or $\pm x \leq d$ is a bound constraint, where d is a rational number, then $\pm x \pm y \leq \lfloor d \rfloor$ and $\pm x \leq \lfloor d \rfloor$ are *tightened* addition and bound constraints.

For example, $x \leq 3/2$ can be tightened to the constraint $x \leq 1$. Tightening is a valid operation in the case of integer variables because the tightened constraint always has the same solutions as the untightened constraint. In the following we always try to keep the constraints in a tightened form.

The *tightened transitive closure* of a set C of addition and bound constraints is the set of addition and bound constraints C^* that C implies, such that, we cannot derive any more addition and bound constraints which are not already in C^* or trivially implied by one constraint in C^*.

The tightened transitive closure allows easy variable elimination and testing simple implications of the original set of constraints. The first tightened transitive closure algorithm was presented by (Jaffar et al. 1994).

Theorem 1 [Jaffar et al. 1994] Let C be a set of addition and bound constraints that is closed under transitivity and tightening. Then C is integer satisfiable if and only if it does not contain a constraint of the form $0 \leq d$ where $d < 0$.

Although the algorithm of (Jaffar et al. 1994) is inefficient, it was already improved by (Harvey and Stuckey 1997).

Lemma 1 [Harvey and Stuckey 1997] Let A be a set of addition constraints, B be a set of bound constraints, such that $A \cup B$ is integer satisfiable, tight, and transitively closed. Let $a, b, e, f \in \{-1, 1\}$ and d, d', d'' be integers. Also let C be the addition constraint $ax + by \leq d$ where $x \neq y$.

$A \cup B \cup C$ is integer unsatisfiable exactly when there is at least one false inequality generated by the following rules:

$$0 \leq d + d' \quad \text{if} \quad -ax - by \leq d'$$

$$0 \leq d + d' + d'' \quad \text{if} \quad -ax \leq d',\ -by \leq d''$$

If $A \cup B \cup C$ is integer satisfiable, then its tightened transitive closure is $A \cup B \cup C \cup A' \cup B'$ where A' is the union of the constraints generated by the rules:

$$by + ex \leq d + d' \quad \text{if} \quad -ax + ez \leq d',\ z \neq y$$

$$ax + ft \leq d + d'' \quad \text{if} \quad -by + ft \leq d'',\ t \neq x$$

$$ex + ft \leq d + d' + d'' \quad \text{if} \quad \begin{array}{l} -ax + ez \leq d', \\ -by + ft \leq d'', \\ z \neq y,\ t \neq x,\ t \neq z \end{array}$$

and B' is the union of constraints generated by the rules:

$$by \leq d + d' \quad \text{if} \quad -ax \leq d'$$

$$ax \leq d + d' \quad \text{if} \quad -by \leq d'$$

$$ez \leq d + d' + d'' \quad \text{if} \quad \begin{array}{l} -ax \leq d', \\ -by + ez \leq d'', z \neq x \end{array}$$

$$ez \leq d + d' + d'' \quad \text{if} \quad \begin{array}{l} -by \leq d', \\ -ax + ez \leq d'', z \neq y \end{array}$$

$$by \leq \left\lfloor \frac{d + d'}{2} \right\rfloor \quad \text{if} \quad -ax + by \leq d'$$

$$ax \leq \left\lfloor \frac{d + d'}{2} \right\rfloor \quad \text{if} \quad ax - by \leq d'$$

$$ez \leq \left\lfloor \frac{d + d' + d''}{2} \right\rfloor \quad \text{if} \quad \begin{array}{l} -ax + ez \leq d', \\ -by + ez \leq d'' \end{array}$$

Based on the above, (Harvey and Stuckey 1997) presents a simple incremental tightened transitive closure algorithm that adds one constraint at a time to the tightened transitive closure. Initially, A' and B' are both empty. After adding each new inequality, as well as testing for satisfiability, the incremental algorithm updates A' and B' to ensure that $A \cup B \cup C \cup A' \cup B'$ remains transitively closed.

Theorem 2 [Harvey and Stuckey 1997] The tightened transitive closure of m number of bound and addition constraints over n variables can be computed in $O(mn^2)$ time.

3. Variable Elimination and Satisfiability

We present an incremental algorithm that eliminates each variable one-by-one. When C is any set of constraints and x is any fixed variable symbol, we denote by $C^{\setminus x}$ the subset of C that contains those constraints that do not contain x. We also denote by x/y the substitution of variable x by variable y.

Below we give an improved set of variable elimination rules.

Definition 1 Let $a, b, e \in \{-1, 1\}$ and d, d' be integers.

$$0 \le d + d' \quad \text{if} \quad -ax \le d, \;\; ax \le d'$$

$$0 \le d + d' \quad \text{if} \quad -ax + by \le d, \;\; ax - by \le d'$$

$$by \le d + d' \quad \text{if} \quad -ax + by \le d, \;\; ax \le d'$$

$$by \le \left\lfloor \frac{d + d'}{2} \right\rfloor \quad \text{if} \quad -ax + by \le d, \;\; ax + by \le d'$$

$$by + ez \le d + d' \quad \text{if} \quad -ax + by \le d, ax + ez \le d', y \ne z$$

It is easy to show that the above set of rules are valid.

Lemma 2 The rules in Definition 1 are valid.

Proof: One way to see the validity of rules in Definition 1 is to rewrite them into the following logically equivalent forms.

$$0 \le d + d' \quad \text{if} \quad -d \le ax, \;\; ax \le d'$$

$$0 \le d + d' \quad \text{if} \quad by - d \le ax, \;\; ax \le by + d'$$

$$by \le d + d' \quad \text{if} \quad by - d \le ax, \;\; ax \le d'$$

$$by \le \left\lfloor \frac{d + d'}{2} \right\rfloor \quad \text{if} \quad by - d \le ax, \;\; ax \le -by + d'$$

$$by + ez \le d + d' \quad \text{if} \quad by - d \le ax, ax \le d' - ez, y \ne z$$

It can be seen that in each case there are two constraints on the right hand side of "if." One of the constraints is of the form $l \le ax$ where l is some lower bound of ax and the other is a constraint of the form $ax \le u$ where u is an upper bound of ax. From these two constraints $l \le u$ follows by transitivity of the $\le$ relation. Finally the constraint on the left hand side of "if" is always some simplification of the $l \le u$ constraint. The simplification includes tightening in the case of fourth rule. □

Next we show that the rules of Definition 1 can be used to eliminate a variable from any set of addition and bound constraints.

Lemma 3 Let A be a set of addition constraints, and let B be a set of bound constraints with variables $x_1, \ldots, x_n$. Let I be the set of constraints derivable using Definition 1 rules (1-2) with substitutions x/x_1 and y/x_i for some $2 \le i \le n$. Let B' be the set of constraints derivable using Definition 1 rules (3-4) with substitutions x/x_1 and y/x_i for some $2 \le i \le n$. Let A' be the set of constraints derivable using Definition 1 rule (5) with substitutions x/x_1 and y/x_i and z/x_j where $2 \le i, j \le n$. Then $A \cup B$ is integer satisfiable if and only if $I \cup A^{\setminus x} \cup A' \cup B^{\setminus x} \cup B'$ is satisfiable.

Proof: Consider all the constraints in $A \cup B$ that have x_1 in them. They can be bound constraints of the form $x_1 \le d$ or $-x_1 \le d'$. The latter can be written as $d' \le x_1$. They can be also addition constraints of the form $x_1 + bx_i \le d_i$ or $-x_1 + bx_j \le d_j$. These latter ones can be written in the form $x_1 \le -bx_i + d_i$ and $bx_j - d_j \le x_1$. Therefore, every constraint that contains x_1 can be written as either some lower or some upper bound on x_1.

It can be also seen that for every pair of lower bound and upper bound constraints of the general form $l \le x_1 \le u$ one of the rules of Definition 1 derives a constraint that is equivalent to $l \le u$. In fact the rules only derive constraints that are equivalent to these types of constraints.

The derived constraints are contained in I if they contain no variables, in B' if they contain only one variable, or in A' if they contain two different variables. Hence it is easy to see that if $A \cup B$ is integer satisfiable, then $I \cup A^{\setminus x} \cup A' \cup B^{\setminus x} \cup B'$ is also integer satisfiable.

Let us consider now the other direction. Consider any given integer solution of $I \cup A^{\setminus x} \cup A' \cup B^{\setminus x} \cup B'$. Obviously, it must satisfy all the derived constraints as it contains $I \cup A' \cup B'$. Among the constraints that contain x_1, when rewritten as explained in the first paragraph, there is some $l_{\max}$ which under the given integer solution gives the largest lower bound on x_1 and there is some u_{min} which under the same integer solution gives the smallest upper bound on x_1. As we saw above, one of the rules must have derived a constraint that is equivalent to $l_{\max} \le u_{\min}$. Since $I \cup A^{\setminus x} \cup A' \cup B^{\setminus x} \cup B'$ is satisfiable, $l_{\max} \le u_{\min}$ must be satisfiable.

Further, $l_{\max}$ and $u_{\min}$ are sums of integers. Hence they must be integers themselves. Since $l_{\max} \le u_{\min}$ holds and $l_{\max}$ and $u_{\min}$ are integers, there must be some integer x_1 such that $l_{\max} \le x_1 \le u_{\min}$. Clearly this integer solution for x_1 satisfies all other lower bound and upper bound constraints on x_1. Therefore $A \cup B$ must be satisfiable. □

Lemma 3 gives the idea that we successively eliminate the variables from $A \cup B$. When the last variable is eliminated, and we have not found yet any unsatisfiable constraint between constants, then $A \cup B$ is satisfiable. The following satisfiability testing algorithm is based on this idea.

Satisfiability(A,B)

input: A is addition constraints, B is bound constraints with variables $x_1, \ldots, x_n$.

output: "satisfiable" or "unsatisfiable" depending on whether $A \cup B$ is satisfiable.

for $k = 1$ to n **do**
  $I = A' = B' = \emptyset$

  for $i = k$ to n **do**
    $I = I \cup$ constraints by Definition 1 rules (1-2) with x/x_k and y/x_i
    $B' = B' \cup$ constraints by Definition 1 rules (3-4) with x/x_k and y/x_i
  end-for

  if any constraint in I is false **then**
    return "unsatisfiable"
  end-if

  for $i = k$ to n **do**
    for $j = k$ to n **do**
      $A' = A' \cup$ constraints by Definition 1 rule (5) with x/x_k, y/x_i and z/x_j
    end-for
  end-for

  $A = A^{\backslash x} \cup A'$
  $B = B^{\backslash x} \cup B'$
end-for

return "satisfiable"

Based on the above algorithm we can show the following theorem.

Theorem 3 Satisfiability of addition and bound constraints over n variables can be computed in $O(n^3)$ time.

Proof: We show that algorithm SATISFIABILITY is correct and runs in $O(n^3)$ time. Each execution of the outermost for-loop of the algorithm will eliminate the kth variable. Since all the variables $x_1, \ldots, x_{k-1}$ are already eliminated by that time, the algorithm correctly considers only substitutions from the still active variables $x_k, \ldots, x_n$. Otherwise, the proof of correctness follows from repeated applications of Lemma 3.

The computational complexity is easily seen to be in $O(n^3)$ time because we have triple nested for loops. Otherwise, each step can be done in constant time. When we try out for a particular rule particular variable substitutions with either -1 or 1 value of the coefficients, we have only a constant number of substitutions to try. The derived constraints are added to the known constraints when they are tighter than the ones already known (this can be checked easily using the ABM three-matrices representation of Section 2). Hence each rule application with one particular substitution of the variables can be done in constant time. □

The following practical theorem also follows as a corollary of the above.

Theorem 4 We can eliminate k variables from a set of addition and bound constraints over n variables in $O(kn^2)$ time. □

4. Tightened Transitive Closure

We now develop an efficient tightened transitive closure algorithm. For the sake of a clearer view of the algorithm, let us represent the addition and bound constraints as a graph. In the graph each variable is represented as a vertex. We also introduce a special vertex that represents the constant 0.

Each addition constraint of the form $ax + by \leq d$ where $a, b \in \{-1, 1\}$ is represented as an undirected edge between vertices x and y with three labels. Label a adjacent to vertex x, label b adjacent to vertex y, and label d, which we will call the *weight*, in the middle of the edge.

Further, each bound constraint of the form $ax \leq d$ where $a \in \{-1, 1\}$ is represented as an undirected edge between vertices x and 0. The label a appears adjacent to vertex x, in the middle the weight d is written, but we do not put any label adjacent to vertex 0 because the label is irrelevant in that case.

For simplicity we can assume that the constraints were already tested for satisfiability and are satisfiable. Therefore, we do not need to use the first two rules of Definition 1. We can translate the third and the fourth rules of Definition 1 into a graph representation as follows where $x \neq y$ and $x \neq z$.

$$y \overset{b}{\underset{d+d'}{\text{———}}} 0 \quad \text{if} \quad y \overset{b}{\underset{d}{\overset{-a}{\text{———}}}} x \overset{a}{\underset{d'}{\text{———}}} 0$$

$$y \overset{b}{\underset{\lfloor (d+d')/2 \rfloor}{\text{———}}} 0 \quad \text{if} \quad y \overset{b}{\underset{d}{\overset{-a}{\text{———}}}} x \overset{a}{\underset{d'}{\overset{b}{\text{———}}}} y$$

By the symmetry of every undirected edge, the second line above, that is, the fourth rule of Definition 1, can be rewritten as the first graphical rule of Figure 1. Finally, the fifth rule of Definition 1 is shown as the second graphical rule in Figure 1. Note that the third rule of Definition 1 is now a special case of the fifth rule of Definition 1, assuming that for 0 we can introduce either a -1 or $+1$ label any time. Hence we need to use only the graphical rules in Figure 1 to derive new constraints.

This alternative graphical view of the rules of Definition 1 allows us to make some important observations. Imagine that in the initial graph there is between vertices v_1 and v_l a path of the following form:

$$v_1 \overset{a_1}{\underset{d_1}{\overset{-a_2}{\text{———}}}} v_2 \ \ldots \ v_{l-1} \overset{a_{l-1}}{\underset{d_{l-1}}{\overset{-a_l}{\text{———}}}} v_l$$

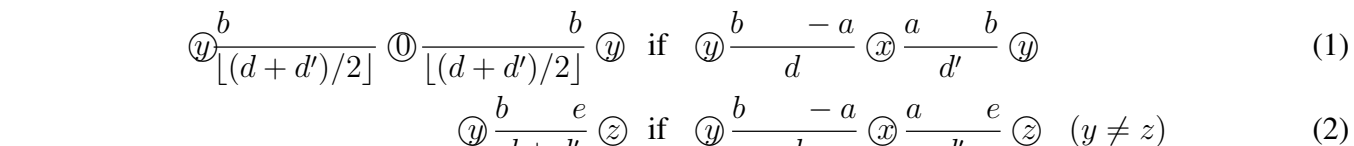
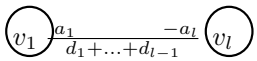


Figure 1: Graphical visualizations of the rules of Definition 1.

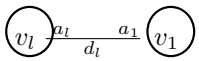
where the v_is for $1 \leq i \leq l$ are distinct.

We can assume that the initial graph does not contain any edge from 0 to 0. (If it did with a positive value, it would mean that the entire graph is unsatisfiable. If it did with a non-positive value, then it would be superfluous.) Hence we can assume that if v_i is the 0 vertex, then the adjacent vertices on the path, that is v_{i-1} and v_{i+1}, are not 0.

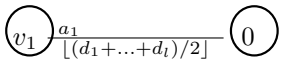
Let us apply rule (2) at each place where it is applicable. Further let us repeat this $\log_2 l$ times. Then we will get a single edge between v_1 to v_l of the form:



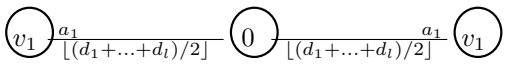
In general, in the entire graph $\log_2 n$ repeated parallel applications of rule (2) will generate a similar single edge in place of every path where the vertices are all distinct. Now suppose that there is another edge from v_l to v_1 of the form:



Note that there is now a cycle from v_1 to itself. With the previous single edge and the new single edge, we can apply rule (1), which will yield:



By rule (1) of Figure 1, this is equivalent to:



Hence we have replaced each cycle of the form where rule (1) applies into a pair of tightened edges. Further, if there was a path between any pair of vertices in the initial graph, there is now a path that leads through such tightened pair of edges instead of untightened cycles of the form we had in v_1 to itself.

Now in the new graph consider any shortest path. A shortest path between a pair of vertices u_1 and u_m is a path between them that has a minimal number of edges among those paths that have a minimal sum of weights between them.

Lemma 4 In the new graph between any two vertices there is always a shortest path that uses at most two occurrences of each distinct variable.

Proof: Consider a shortest path between u_1 and u_m. Let us traverse the path from u_1 to u_m. Suppose that some vertex w appears at least three times on this shortest path. Consider on the path the vertex w' before and the vertex w'' after the second occurrence of w.

If $w' \neq w''$, then by rule (2) there must be an edge between w' and w'' such that its weight is equal to the sum of the weights on the edge from w' to w and from w to w''. Then there is a path from u_1 to u_m which is shorter than the one given, which is a contradiction.

If $w' = w''$, then there are two cases.

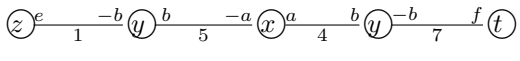
Case I: The outgoing label of w' is the opposite of the incoming label of w''. Then rule (2) still applies and we can find a shorter path when we replace w' w w'' by a single edge between w' and w''. This is a contradiction to the assumption that the path between u_1 and u_m is a shortest path.

Case II: The outgoing label of w' is the same as the incoming label of w''. Then by rule (1) we can replace the path w' w w'' by a path w' 0 w''. This path will have the same number of edges, the same or possibly smaller sum of weights (because of the tightening) and have one fewer occurrence of w.

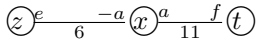
By a repetition of the above argument we can eliminate all the other intermediate occurrences of w or derive a contradiction. □

Lemma 4 implies that if we now apply again rule (2) in parallel $\log_2 n$ times, then we get a single edge between every pair of vertices such that the weight of the edge is the minimal weight that can be obtained by any application of rules (1-2).

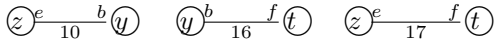
Example 2 Consider the following path between vertices z and t.



The first time we apply rule (2) in parallel we add the edges:



The second time we apply rule (2) in parallel we add the edges:



Then we apply rule (1) and get:

$$y \overset{b}{\underset{8}{\text{———}}} b\, y$$

Then again we apply rule (2) in parallel and get:

$$z \overset{e}{\underset{9}{\text{———}}} b\, y \qquad y \overset{b}{\underset{15}{\text{———}}} f\, t$$

Finally we apply rule (2) again in parallel and get:

$$z \overset{e}{\underset{16}{\text{———}}} f\, t$$

Hence we conclude that the shortest path between z and t has length 16.

The tightened transitive closure algorithm is shown below.

TransitiveClosure(G)
input: An integer satisfiable set of addition and bound constraints as a graph G with variables $x_0, x_1, \ldots, x_n$.
output: The tightened transitive closure of G.

```
G' = ∅
for m = 1 to log n do
    for i = 0 to n do
        for j = 0 to n do
            for k = 0 to n do
                G' = G' ∪ apply Figure 1 rule (2)
                          with y/x_i, x/x_k, z/x_j
            end-for
        end-for
    end-for
    G = G ∪ G'
    G' = ∅
end-for

for i = 0 to n do
    for k = 0 to n do
        G' = G' ∪ apply Figure 1 rule (1)
                  with y/x_i and x/x_k
    end-for
end-for

G = G ∪ G'
G' = ∅
for m = 1 to log n do
    for i = 0 to n do
        for j = 0 to n do
            for k = 0 to n do
                G' = G' ∪ apply Figure 1 rule (2)
                          with y/x_i, x/x_k, z/x_j
            end-for
        end-for
    end-for
    G = G ∪ G'
    G' = ∅
end-for
return G
```

Although for simplicity we presented the above algorithm using a graph representation, it can be rewritten into a logically equivalent algorithm that uses the matrix representation of Section 2. The addition of the constraints in G' to G can then be implemented efficiently, and we can keep the size of the representation of the graph $O(n^2)$.

Theorem 5 The tightened transitive closure of addition and bound constraints over n variables can be computed in $O(n^3 \log n)$ time.

Proof: The proof is based on showing the correctness of algorithm TRANSITIVECLOSURE. The key condition is expressed in Lemma 4. Based on Lemma 4, it is easy to show that the actual shortest path can be computed in $\log n + 1$ parallel applications of the rules in Figure 1. □

5. Related and Future Work

One advantage of our satisfiability testing and tightened transitive closure algorithms is that they can be relatively easily implemented based on matrices. We are currently working on implementing in the MLPQ system the evaluation of the under-approximation and the over-approximation of the least fixed point semantics of Datalog queries with addition and bound constraints. An interesting open problem is to find conditions when the under-approximation and the over-approximation of the program semantics are the same, resulting in a precise evaluation.

References

R. Alur, C. Courcoubetis, N. Halbwachs, T. Henzinger, P.-H. Ho, X. Nicollin, A. Olivero, J. Sifakis, and S. Yovine. The algorithmic analysis of hybrid systems. *Theoretical Computer Science*, 138(1):3–34, 1995.

S. Anderson and P. Revesz. Verifying the incorrectness of programs and automata. In *Proc. 6th International Symposium on Abstraction, Reformulation, and Approximation*, volume 3607 of *Lecture Notes in Computer Science*, pages 1–13. Springer-Verlag, 2005.

E. M. Clarke, O. Grumberg, and D. A. Peled. *Model Checking*. MIT Press, 1999.

P. Cousot and R. Cousot. Abstract interpretation: A unified lattice model for static analysis of programs by construction or aproximation of fixpoints. In *Proc. ACM Principles on Programming Languages*, pages 238–252. ACM Press, 1977.

P. Cousot and N. Halbwachs. Automatic discovery of linear restraints among variables of a program. In *Proc. ACM Principles on Programming Languages*, pages 84–97. ACM Press, 1978.

G. Delzanno and A. Podelski. Model checking in CLP. In *2nd International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, volume 1579 of *Lecture Notes in Computer Science*, pages 74–88. Springer-Verlag, 1999.

L. Fribourg and H. Olsén. A decompositional approach for computing least fixed-points of Datalog programs with Z-counters. *Constraints*, 2(3–4):305–36, 1997.

L. Fribourg and J. D. C. Richardson. Symbolic verification with gap-order constraints. In *Proc. Logic Program Synthesis and Transformation*, volume 1207 of *Lecture Notes in Computer Science*, pages 20–37. Springer-Verlag, 1996.

W. Harvey and P. Stuckey. A unit two variable per inequality integer constraint solver for constraint logic programming. In *Proc. Australian Computer Science Conference (Australian Computer Science Communications)*, pages 102–11, 1997.

J. Jaffar, M. J. Maher, P. Stuckey, and R. H. Yap. Beyond finite domains. In A. Borning, editor, *Proc. 2nd International Workshop on Principles and Practice of Constraint Programming*, volume 874 of *Lecture Notes in Computer Science*, pages 86–94. Springer-Verlag, 1994.

P. C. Kanellakis, G. M. Kuper, and P. Revesz. Constraint query languages. *Journal of Computer and System Sciences*, 51(1):26–52, 1995.

K. McMillan. *Symbolic Model Checking*. Kluwer, 1993.

A. Miné. The octagon abstract domain. In *Proc. Analysis, Slicing and Transformation*, pages 310–319. IEEE Press, 2001.

M. Péron and N. Halbwachs. An abstract domain extending difference-bound matrices with disequality constraints. In *Proc. 8th International Conference on Verification, Model Checking, and Abstract Interpretation*, volume 4349 of *Lecture Notes in Computer Science*, pages 268–282. Springer-Verlag, 2007.

P. Revesz. A closed-form evaluation for Datalog queries with integer (gap)-order constraints. *Theoretical Computer Science*, 116(1):117–49, 1993.

P. Revesz. Reformulation and approximation in model checking. In *Proc. 4th International Symposium on Abstraction, Reformulation, and Approximation*, volume 1864 of *Lecture Notes in Computer Science*, pages 124–43. Springer-Verlag, 2000.

P. Revesz. *Introduction to Constraint Databases*. Springer-Verlag, 2002.

P. Revesz. The constraint database approach to software verification. In *Proc. 8th International Conference on Verification, Model Checking, and Abstract Interpretation*, volume 4349 of *Lecture Notes in Computer Science*, pages 329–345. Springer-Verlag, 2007.

P. Revesz, R. Chen, P. Kanjamala, Y. Li, Y. Liu, and Y. Wang. The MLPQ/GIS constraint database system. In *Proc. ACM SIGMOD International Conference on Management of Data*, 2000.

D. J. Rosenkrantz and H. B. Hunt. Processing conjunctive predicates and queries. In *Proc. IEEE International Conference on Very Large Databases*, pages 64–72, 1980.