Incremental Satisfiability and Implication for UTVPI Constraints

Andreas Schutt, Peter J. Stuckey
National ICT Australia Victoria Laboratory, Department of Computer Science & Software Engineering,
The University of Melbourne, Victoria, 3010, Australia, {aschutt,pjs}@csse.unimelb.edu.au

Unit two-variable-per-inequality (UTVPI) constraints form one of the largest class of integer constraints which are polynomial time solvable (unless P=NP). There is considerable interest in their use for constraint solving, abstract interpretation, spatial databases, and theorem proving. In this paper we develop new incremental algorithms for UTVPI constraint satisfaction and implication checking that require $\mathcal{O}(m + n \log n + p)$ time and $\mathcal{O}(n + m + p)$ space to incrementally check satisfiability of m UTVPI constraints on n variables and check implication of p UTVPI constraints. The algorithms can be straightforwardly extended to create non-incremental implication checking and generation of all (non-redundant) implied constraints, as well as generate minimal unsatisfiable subsets and minimal implicants.

Key words: unit two variable per inequality constraints; satisfaction; implication

1. Introduction

The unit two-variable-per-inequality (UTVPI) constraints form one of the largest class of integer constraints which are polynomial time solvable (unless P=NP). There is considerable interest in their use for constraint solving (Jaffar et al., 1994; Harvey and Stuckey, 1997), abstract interpretation (Miné, 2006), spatial databases (Sitzmann and Stuckey, 2000) and theorem proving (Lahiri and Musuvathi, 2005).

Most uses of UTVPI constraint are inherently incremental. UTVPI constraint solving repeatedly asks satisfiability questions in an incremental manner in order to drive a search in a large search space. Abstract interpretation uses of UTVPI (Miné, 2006) build descriptions of program points in an incremental manner by taking the description of the previous program point and adding new constraints to generate a description for the next program point. Theorem proving may be non-incremental as in Lahiri and Musuvathi (2005) where UTVPI problems arise as subproblems required by a verification system, but modern techniques such as SAT Modulo Theories (Niewenhuis et al., 2005), require incremental satisfaction and implication algorithms as well as algorithms for explanation. In this paper we develop new incremental algorithms for UTVPI constraint satisfaction and implication.

A UTVPI constraint has the form $ax + by \leq d$ where x, y are integer variables, $d \in \mathbb{Z}$ and $a, b \in \{-1, 0, 1\}$. For example $x + y \leq 2$, $x - y \leq -1$, $0 \leq -1$ and $x \leq 2$ are UTVPI constraints. UTVPI constraint solving is based on transitive closure: The constraints $ax - y \leq d_1$ and $y + bz \leq d_2$ imply the constraint $ax + bz \leq d_1 + d_2$. We can determine all the UTVPI consequences of a set of UTVPI constraints by transitive closure, but we need to *tighten* some constraints. The transitive closure procedure can generate constraints of the form $x + x \leq d$ and $-x - x \leq d$, which need to be tightened to $x \leq \left\lfloor \frac{d}{2} \right\rfloor$ and $-x \leq \left\lfloor \frac{d}{2} \right\rfloor$ respectively.

Jaffar et al. (1994) and Harvey and Stuckey (1997) present incremental consistency checking algorithms for adding a UTVPI constraint c to a set ϕ of UTVPI constraints. They are based on maintaining the transitive and tight closure of the set of UTVPI constraints ϕ involving n variables. Both algorithms require $\mathcal{O}(n^2)$ time and $\mathcal{O}(n^2)$ space for an incremental satisfaction check. Both algorithms can also be used to incrementally check implication of UTVPI constraints by $\phi \cup \{c\}$. These algorithms require $\mathcal{O}(n^2 + p)$ time and $\mathcal{O}(n^2 + p)$ space for incremental implication checking, where p is the number of constraints we need to check for implication. In order to (non-incrementally) check satisfiability of m UTVPI constraints on n variables these approaches require $\mathcal{O}(n^2 m)$ time, and to check implication they require $\mathcal{O}(n^2 m + p)$ time.

An improvement on the complexity of non-incremental satisfiability for UTVPI constraints was devised by Lahiri and Musuvathi (2005). They define a non-incremental satisfiability algorithm requiring $\mathcal{O}(nm)$ time and $\mathcal{O}(n+m)$ space. The key behind their approach is to map UTVPI constraints to difference constraints (also called separation theory constraints) of the form $x - y \leq d$, where x and y are integer variables and $d \in \mathbb{Z}$.

The difference constraints are a well studied class of constraints because of their connection to shortest path problems. We can consider the constraint $x - y \leq d$ as a directed edge $x \rightarrow y$ with weight d. Satisfiability of difference constraints corresponds to the problem of negative weight cycle detection, and implication of difference constraints corresponds to finding shortest paths (see e.g. Cotton and Maler (2006) for details).

The mapping of UTVPI to difference constraints by Lahiri and Musuvathi (2005) is a relaxation of the problem. The relaxed problem is solved by a negative (weight) cycle detection algorithm but it guarantees only the satisfiability in $\mathbb{Q}$ for the UTVPI problem.

In order to check satisfiability in $\mathbb{Z}$ they need to construct an auxiliary graph and check for certain paths in this graph.

In this paper we first extend Lahiri and Musuvathi's algorithm (see Lahiri and Musuvathi (2005)) to check satisfaction incrementally in $\mathcal{O}(n \log n + m)$ and $\mathcal{O}(n + m)$ space. Then we show how to build an incremental satisfiability and implication algorithm using the relaxation of Lahiri and Musuvathi and incremental implication approaches for difference constraints of Cotton and Maler (2006), which can incrementally check implication in $\mathcal{O}(n \log n + m + p)$ time and $\mathcal{O}(n + m + p)$ space.

We give experimental results showing that our algorithms improve upon the previous incremental algorithms for UTVPI satisfaction and implication checking unless the constraint graph is dense.

We then consider using our incremental approach as a basis for a non-incremental algorithm for implication checking in $\mathcal{O}(n^2 \log n + nm + p)$ time and $\mathcal{O}(n + m + p)$ space. Similarly we can generate all implied constraints in $\mathcal{O}(n^2 \log n + nm)$ time and $\mathcal{O}(n + m + p)$ space, where here p is the number of generated implied constraints.

One of the interests of solving UTVPI constraints is in solving Boolean combinations of UTVPI constraints, e.g. $(x - y \leq 3 \vee y - x \leq 4) \rightarrow (x - z \leq 2 \wedge z - y \leq 1)$. Seshia et al. (2007) discuss how to encode Boolean combinations of UTVPI constraints in CNF by giving a tight bound on the region of satisfiability. An alternate approach is to use incremental satisfaction and implication algorithms in a Satisfiability Modulo Theories (SMT) solver (Niewenhuis et al., 2005). This requires that one can efficiently discover unsatisfiable subsets and implicants of implied constraints. We discuss how to extend the incremental algorithm to discover minimal unsatisfiable subsets and minimal implicants in $\mathcal{O}(n)$ time. Surprisingly this is not as simple as the case for difference constraints.

In summary the contributions of this paper are:

- A new incremental satisfiability algorithm for UTVPI based on the approach of Lahiri and Musuvathi, which is asymptotically better than previous algorithms for this problem.

- A new incremental implication algorithm for UTVPI, based on a fundamental new understanding of how we can compute the tightened transitive closure, which is asymptotically better than previous algorithms for this problem.

- Experiments illustrating that for sparse problems our algorithms are significantly better than existing algorithms for these problems

- New non-incremental algorithms for implication checking and implication generation which are asymptotically better than existing algorithms for this.

- The first algorithms we are aware of specifically for generation of minimal unsatisfiable subsets and minimal implicants for UTVPI problems.

The remainder of the paper is organized as follows. In the next section we give preliminary definitions. In Section 3 we explain the approach of Lahiri and Musuvathi (2005) to UTVPI satisfaction. In Section 4 we show how can modify their approach to perform incremental satisfaction. In Section 5 we show how to do incremental implication checking, which also introduces a new way to do incremental satisfiability checking. In Section 6 we give experimental comparisons of the algorithms for satisfiability and implication. In Section 7 we show how we can create non-incremental implication checking from the incremental algorithms. In Section 8 we explain how to generate minimal explanations of the unsatisfiability and implication for use in a SMT solver. Finally in Section 9 we conclude.

2. Preliminaries

In this section we give notations and preliminary concepts.

A *weighted directed graph* $G = (V, E)$ is made up of vertices V and a set E of weighted directed edges (u, v, d) from vertex $u \in V$ to vertex $v \in V$ with weight d. We also use the notation $u \xrightarrow{d} v$ to denote the edge (u, v, d).

A *path* P from v_0 to v_k in graph G, denoted $v_0 \rightsquigarrow v_k$, is a sequence of edges $e_1, \ldots, e_k$ where $e_i = (v_{i-1}, v_i, d_i) \in E$ and $k \in \mathbb{N}$. Let $|P|$ be the number of edges appearing in P. A *simple path* P is a path where $v_i \neq v_j, 0 \leq i < j \leq k$.

A (simple) *cycle* P is a path P where $v_0 = v_k$ and $v_i \neq v_j, 0 \leq i < j \wedge k \wedge (i \neq 0 \vee j \neq k)$.

The *path weight* of a path P, denoted $w(P)$ is $\Sigma_{i=1}^{k} d_i$.

Let G be a graph without negative weight cycles, that is without a cycle P where $w(P) < 0$. Then we can define the *shortest path* from v_0 to v_k, which we denote by $SP(v_0, v_k)$, as the (simple) path P from v_0 to v_k such that $w(P)$ is minimized.

Let $wSP(x, y) = w(SP(x, y))$ or $+\infty$ if no path exists from x to y.

Given a graph G and vertex x define the functions $\delta_x^{\leftarrow}, \delta_x^{\rightarrow} : V \to \mathbb{R}$ as

$$\delta_x^{\leftarrow}(y) = wSP(y, x) \qquad \text{and} \qquad \delta_x^{\rightarrow}(y) = wSP(x, y) \ .$$

Let G be a graph without negative weight cycles. Then π is a *valid potential function* for G if $\pi(u) + d - \pi(v) \geq 0$ for every edge (u, v, d) in G. A edge (u, v, d) is called *tight* if $\pi(u) + d - \pi(v) = 0$.

There are many algorithms (see e.g. Cherkassky and Goldberg (2006)) for detecting negative weight cycles in a weighted directed graph, which either detect a cycle or determine a valid potential function for the graph.

Given a valid potential function π for graph $G = (V, E)$ we can define the *reduced-cost graph* $rc(G)$ as $(V, \{(x, y, \pi(x) + d - \pi(y)) \mid (x, y, d) \in E\})$. All weights in the reduced cost graph are non-negative and we can recover the original path length $w(P)$ for path P from x to y from paths in the reduced cost graph since $w(P) = w + \pi(y) - \pi(x)$ where w is the weight of the corresponding path in the reduced-cost graph.

Since edges in the reduced-cost graph are non-negative we can use Dijkstra's algorithm to calculate the shortest paths in the reduced-cost graph in time $\mathcal{O}(n \log n + m)$ instead of $\mathcal{O}(nm)$.

2.1 Difference constraints

Difference constraints have the form $x - y \leq d$ where x and y are integer variables and $d \in \mathbb{Z}$. We can map difference constraints to a weighted directed graph.

Definition 1. Let C be a set of difference constraints and let $G = (V, E)$ be the graph comprised of one weighted edge $x \xrightarrow{d} y$ for every constraint $x - y \leq d$ in C. We call G the *constraint graph* of C.

The following well-known result characterizes how the constraint graph can be used for satisfiability and implication checking of difference constraints.

Theorem 1. *Let C be a set of difference constraints and G its corresponding graph. C is satisfiable iff G has no negative weight cycles, and if C is satisfiable then $C \models x - y \leq d$ iff $wSP(x, y) \leq d$.* $\square$

2.2 UTVPI constraints

A UTVPI constraint is of the form $ax + by \leq d$, where x and y are integer variables, $a, b \in \{-1, 0, 1\}$ and $d \in \mathbb{Z}$.

Definition 2. The *transitive closure* $TC(\phi)$ of a set of UTVPI constraints ϕ is defined as the smallest set S containing ϕ such that

$$ax - y \leq d_1 \in S \wedge y + bz \leq d_2 \in S \quad \Rightarrow \quad ax + bz \leq d_1 + d_2 \in S$$

The *tightened closure* $TI(\phi)$ of a set of UTVPI constraints ϕ is defined as the smallest set S containing ϕ such that

$$ax + ax \leq d \in S \quad \Rightarrow \quad ax \leq \left\lfloor \frac{d}{2} \right\rfloor \in S, \qquad a \in \{-1, 1\}$$

The *tightened transitive closure* $TTC(\phi)$ of ϕ is the smallest set containing ϕ that satisfies both conditions.

The fundamental results for UTVPI constraints solving are (see Jaffar et al. (1994)):

Theorem 2 (Unsatisfiability, Jaffar et al. (1994))**.** *Let ϕ be a set of UTVPI constraints. Then ϕ is unsatisfiable iff exists $0 \leq d \in TTC(\phi)$ where $d < 0$.* □

Theorem 3 (Implication, Jaffar et al. (1994))**.** *Let $c \equiv ax + by \leq d$ be a UTVPI constraint and let ϕ be a satisfiable set of UTVPI constraints. Then $\phi \models c$ iff either $c \equiv 0 \leq d$ is a tautology, there exists $\{ax \leq d_1, by \leq d_2\} \subseteq TTC(\phi)$ with $d_1 + d_2 \leq d$, there exists $ax + by \leq d' \in TTC(\phi)$ with $d' \leq d$.* □

Example 1. Consider the UTVPI constraints $\phi \equiv \{x - y \leq 2,\ x + y \leq -1,\ -x - z \leq -4\}$, Then $TC(\phi)$ includes in addition $\{x + x \leq 1,\ -y - z \leq -2,\ y - z \leq -5, -z - z \leq -7, x - z \leq -3\}$. And $TI(TC(\phi))$ includes in addition $\{x \leq 0, -z \leq -4\}$ and $TTC(\phi) = TI(TC(\phi))$ in this case. The constraint $-z \leq -3$ is implied by ϕ as is $y - z \leq 0$.

3. Lahiri and Musuvathi's approach

Lahiri and Musuvathi map UTVPI constraints ϕ to difference constraints or equivalently a weighted directed graph G_ϕ, and they use graph algorithms to detect satisfiability.

Table 1: Transformation from UTVPI constraint c to associated difference constraints $D(c)$ to edges in the constraint graph $E(c)$.

UTVPI c	Diff. Constr. $D(c)$	Edges $E(c)$
$x - y \leq d$	$x^+ - y^+ \leq d$ $y^- - x^- \leq d$	$x^+ \xrightarrow{d} y^+$ $y^- \xrightarrow{d} x^-$
$x + y \leq d$	$x^+ - y^- \leq d$ $y^+ - x^- \leq d$	$x^+ \xrightarrow{d} y^-$ $y^+ \xrightarrow{d} x^-$
$-x - y \leq d$	$x^- - y^+ \leq d$ $y^- - x^+ \leq d$	$x^- \xrightarrow{d} y^+$ $y^- \xrightarrow{d} x^+$
$x \leq d$	$x^+ - x^- \leq 2d$	$x^+ \xrightarrow{2d} x^-$
$-x \leq d$	$x^- - x^+ \leq 2d$	$x^- \xrightarrow{2d} x^+$

We denote the constraint graph arising from ϕ as $G_\phi = (V, E)$. The graph G contains two vertices x^+ and x^- for every variable x. These variables are used to convert UTVPI constraints into difference constraints. The vertex x^+ represents $+x$ and x^- represents $-x$.

Let ϕ be a set of UTVPI constraints. Each UTVPI constraint $c \in \phi$ is mapped to a *set of difference constraints* $D(c)$, or equivalently a *set of weighted edges* $E(c)$. The mapping is shown in the Table 1. Each UTVPI constraint on two variables generates two difference constraints and accordingly two edges in the constraint graph. Each UTVPI constraint on a single variable generates a single constraint, and hence a single edge.

Let $-u$ denote the counterpart of a vertex $u \in V$, i.e. $-x^+ := x^-$ and $-x^- := x^+$. Clearly, for each edge $(u, v, d) \in E$ the graph G_ϕ also includes the edge $(-v, -u, d)$ (called its *counteredge*) which has equal weight. This correspondence extends to paths.

Lemma 1 (Lahiri and Musuvathi (2005))**.** *If there is a path P from u to v in the constraint graph G_ϕ, then there is a* counterpath *path P' from $-v$ to $-u$ such that $w(P) = w(P')$.* □

If we relax the restriction on variables to take values in $\mathbb{Z}$ and allow them to take values in $\mathbb{Q}$ we can check satisfiability in $\mathbb{Q}$ using G_ϕ.

Lemma 2 (Lahiri and Musuvathi (2005))**.** *A set of UTVPI constraints ϕ is unsatisfiable in $\mathbb{Q}$ iff the constraint graph $G_\phi = (V, E)$ contains a negative weight cycle.* □

The reason why the satisfiability in $\mathbb{Z}$ cannot be tested with G_ϕ arises from the possible implication of constraints of the form $x + x \leq d$ or $-x - x \leq d$ through the transitivity of constraints in ϕ. If d is odd (equivalently $d/2 \in \mathbb{Q} \setminus \mathbb{Z}$) then ϕ may be satisfiable with $x = d/2$ but not with $x = \lfloor d/2 \rfloor$.

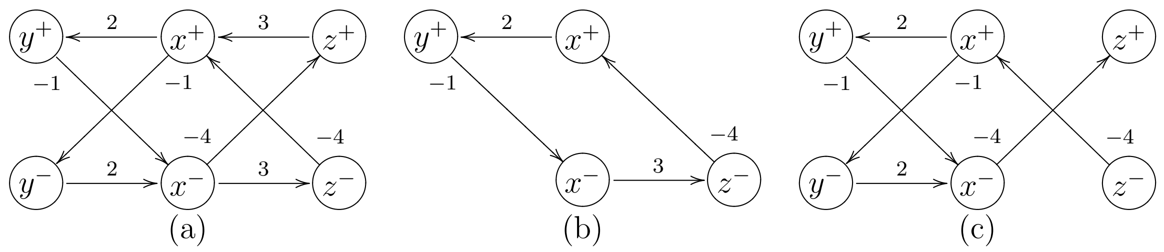


Figure 1: (a) $G_{\phi'}$ for ϕ' of Example 2 which is $\mathbb{Q}$-satisfiable but not $\mathbb{Z}$ satisfiable. (b) a zero weight cycle in $G_{\phi'}$. (c) G_ϕ for ϕ of Example 1.

Example 2. Consider the UTVPI problem $\phi' \equiv \{x - y \leq 2,\ x + y \leq -1,\ -x - z \leq -4,\ -x + z \leq 3\}$, then a transitive consequence of the first two is $x + x \leq 1$, while a consequence of the second two is $-x - x \leq -1$. Together these require $x = \frac{1}{2}$.

The graph $G_{\phi'}$ is shown in Figure 1(a). A zero weight cycle is extracted in Figure 1(b). This cycle has solutions in $\mathbb{Q}$ but not in $\mathbb{Z}$.

The satisfaction algorithm of Lahiri and Musuvathi (2005) is based on Lemma 2 and the following result.

Lemma 3 (Lahiri and Musuvathi (2005))**.** *Suppose G_ϕ has no negative cycles and ϕ is unsatisfiable in $\mathbb{Z}$. Then G_ϕ contains a zero weight cycle containing vertices u and $-u$ such that $wSP(u, -u)$ is odd.* □

The Lahiri and Musuvathi algorithm is shown in Algorithm 1: LaMu. The algorithm first checks $\mathbb{Q}$ satisfiability using a negative cycle detection algorithm (line 3), and then checks that no such zero weight cycles exists in G_ϕ (lines 5–13) while building up an auxiliary graph H_ϕ containing all tight edges E' and determining all its strongly connected components (SCC).

Example 3. A valid potential function for the graph shown in Figure 1(a) is $\pi(y^+) = 5$, $\pi(x^+) = 3$, $\pi(z^+) = 0$, $\pi(y^-) = 2$, $\pi(x^-) = 4$, $\pi(z^-) = 7$. Each of the edges is tight, so E' contains all edges, and all nodes are in the same SCC. Both x^+ and x^- occur in the same SCC and $SP(x^+, x^-) = \pi(x^-) - \pi(x^+) = 1$ is odd, hence the system is unsatisfiable.

The complexity of LaMu is $\mathcal{O}(nm)$ time and $\mathcal{O}(n + m)$ space assuming the application of Bellman-Ford single source shortest path algorithm (Bellman, 1958; Ford and Fulkerson, 1962) for negative cycle detection.

Algorithm 1: LaMu (Lahiri and Musuvathi, 2005)

```
Input: $\phi$ a set of UTVPI constraints
Output: SAT if $\phi$ is satisfiable, UNSAT otherwise
1  Construct the constraint graph $G_\phi = (V, E)$ from $\phi$;
2  Run a negative cycle detection algorithm on $G_\phi$;
3  if $G_\phi$ contains a negative cycle then
4      return UNSAT
5  else
6      let $\pi$ be a valid potential function for $G_\phi$
7  $E' := \{(u, v) \mid (u, v, d) \in E, \pi(u) + d = \pi(v)\}$;
8  $H_\phi := (V, E')$;
9  Group the vertices in $H_\phi$ into strongly connected components (SCCs). Vertices $u$ and
   $v$ are in the same SCC if and only if there is a path from $u$ to $v$ and a path from $v$ to
   $u$ in $H_\phi$. $u$ and $v$ are in the same SCC exactly when there is a zero weight cycle in $G_\phi$
   containing $u$ and $v$;
10 for all $u \in V$ do
11     if $-u$ is in the same SCC as $u$ in $H_\phi$ and $\pi(-u) - \pi(u)$ is odd then
12         return UNSAT
13 return SAT
```

4. Incremental UTVPI Satisfaction

The incremental satisfaction problem is: Given a satisfiable set of UTVPI ϕ (with n variables and m constraints) and UTVPI constraint c, determine if $\phi \cup \{c\}$ is satisfiable. In this section we define an incremental satisfaction checker for UTVPI constraints that requires $\mathcal{O}(n \log n + m)$ time and $\mathcal{O}(n+m)$ space. It relies on simply making incremental the algorithm LaMu of Lahiri and Musuvathi. We examine the two major components of their algorithm: negative cycle detection, and the calculation of strongly connected components (SCCs).

The key is to incrementalize the negative cycle detection and the SCC computation. We make use of incremental negative cycle detection algorithms previously used for difference constraints. We also carefully consider how the SCCs can change under the addition of constraints, in order to minimize the work in recalculating SCCs.

For incremental negative cycle detection we use an algorithm due to Frigioni et al. (1998), using the simplified form (Algorithm 2: IncConDiff) of Cotton and Maler (2006) (since we are not interested in edge deletion). Given a graph $G = (V, E)$ and valid potential function π for G and edge $e = u \xrightarrow{d} v$, this algorithm returns $G' = (V, E \cup \{e\})$ and a valid potential function π' for G' or determines a negative cycle and returns UNSAT. The complexity is $\mathcal{O}(n \log n + m)$ time and $\mathcal{O}(n + m)$ space using Fibonacci heaps to implement argmin.

Algorithm 2: IncConDiff (Cotton and Maler, 2006)

```
Input: $G_\phi = (V, E)$ a graph, $\pi$ a valid potential function for $G_\phi$, edge $(u, v, d)$ a new
       constraint to add to $G_\phi$.
Output: UNSAT if $\phi \cup \{u - v \leq d\}$ is unsatisfiable, or $G_{\phi\cup\{u-v\leq d\}}$ and a valid
        potential function $\pi'$ for $G_{\phi\cup\{u-v\leq d\}}$.
1  $\pi' := \pi$;
2  $\gamma(v) := \pi(u) + d - \pi(v)$;
3  $\gamma(w) := 0$ for all $w \neq v$;
4  while $min(\gamma) < 0 \wedge \gamma(u) = 0$ do
5      $s :=$ argmin$(\gamma)$ ;
6      $\pi'(s) := \pi(s) + \gamma(s)$ ;
7      $\gamma(s) := 0$ ;
8      for all $s \xrightarrow{d'} t \in G$ do
9          if $\pi'(t) = \pi(t)$ then
10             $\gamma(t) := \min\{\gamma(t), \pi'(s) + d' - \pi'(t)\}$
11 if $\gamma(u) < 0$ then
12     return $UNSAT$
13 return $((V, E \cup \{(u, v, d)\}), \pi')$
```

IncConDiff works as follow: It takes a copy π' of the valid potential function π and the edge to be added (u, v, d) and repairs all π' values of nodes t violating $\pi'(s) + d' - \pi'(t) \leq 0$ for a edge $(s, t, d') \in G_{\phi\cup\{u-v\leq d\}}$. If the potential function is valid after the addition of (u, v, d), *i.e.* $\gamma(v) \geq 0$ (line 2) then the conditions (line 4 and 11) are not satisfied and algorithm terminates straightforwardly (line 13). Otherwise the π' value of v is fixed at first by adding $\gamma(v)$ to its previous value $\pi(v)$. This can lead to violation of the potential function condition on its outgoing edges $(v, t, .)$. For all these edges the node t is added to the priority queue with priority $\gamma(t)$ (line 10). The same procedure is applied for fixing the π' values of all nodes in the queue starting with the node with a lowest priority. The algorithm proves either satisfiability if it can empty the queue or unsatisfiability if $\gamma(u)$ becomes negative (line 10). Implementation details can be found in Cotton and Maler (2006).

To incrementalize the SCC computation of LaMu in the zero reduced-cost graph, we use the same SCC algorithm as before, but restrict its application. The *zero reduced-cost graph* is the subgraph of the reduced-cost graph with all zero weight edges. The key to incrementalizing the SCC calculation is the following lemma which shows that the addition of a new constraint will either make no change to the SCCs in a manner which is quick to detect, or can only collapse SCCs for the graph of the previous problem.

Lemma 4. *Let ϕ be a satisfiable set of UTVPI constraints and π_ϕ a valid potential function for G_ϕ. Let H_ϕ be the graph of zero weight reduced-cost edges in G_ϕ. Let $G_{\phi'}$ be G_ϕ with the addition of an edge (u, v, d) and define analogously $\pi_{\phi'}$ and $H_{\phi'}$. If $\pi_\phi(u) + d > \pi_\phi(v)$ the SCCs of H_ϕ and $H_{\phi'}$ are identical. If $\pi_\phi(u) + d \leq \pi_\phi(v)$ then the SCCs of $H_{\phi'}$ are either identical to those of H_ϕ or result from the union of SCCs in H_ϕ reachable from v in G_ϕ.*

Proof. We carry out the proof with respect to IncConDiff. If $\pi_\phi(u) + d > \pi_\phi(v)$ then $\gamma(v) > 0$ and no potential function values change so $H_{\phi'} = H_\phi$ and the result holds.

If $\pi_\phi(u) + d = \pi_\phi(v)$ then $\gamma(v) = 0$ and no potential function values change so $H_{\phi'} = H_\phi$ with the addition of the edge (u, v) if not already present. Clearly this can only union SCCs of H_ϕ reachable from v.

If $\pi_\phi(u) + d < \pi_\phi(v)$ then $\gamma(v) < 0$ and IncConDiff does create a new potential function $\pi_{\phi'}$. We show that each node s reachable from v in H_ϕ has its potential function value changed to $\pi_{\phi'}(s) = \pi_\phi(s) + \gamma(v)$ by induction on path length from v.

The base case clearly holds.

Suppose the result holds for s reachable from v in k steps. Each node t where $(s, t) \in H_\phi$ is such that there exists an edge (s, t, d') in G_ϕ where $\pi_\phi(s) + d' - \pi_\phi(t)$. When s is selected for updating in IncConDiff, then each t which has not been changed already, has $\gamma(t)$ set to $\gamma(v)$. When t is selected IncConDiff will define $\pi_{\phi'}(t) = \pi_\phi(t) + \gamma(v)$.

Clearly then each edge $(s, t) \in H_\phi$ also exists in $H_{\phi'}$.

Thus SCCs in $H_{\phi'}$ must be unions of SCCs in H_ϕ, and since the only potential values changing are those reachable from v in G_ϕ the result holds. □

The incremental UTVPI satisfaction algorithm Satis (shown in Algorithm 3) simply runs IncConDiff at line 3 of LaMu. If the newly added edge (u, v, d) has a reduced cost of zero it has to rebuild the zero reduced-cost graph H_ϕ graph to determine possible $\mathbb{Z}$ unsatisfiability, since the SCCs of the zero reduced-cost graph may have changed. It does so only for nodes reachable from v in G_ϕ. Therefore Satis computes the SCCs of the subgraph H (defined at line 8) just containing these nodes and their incident zero reduced-cost edges. Then it uses the same check as LaMu for $\mathbb{Z}$ unsatisfiability (line 10). The algorithm requires $\mathcal{O}(n + m)$ time and space for the SCC construction and checking, and hence the overall complexity is dominated by IncConDiff requiring $\mathcal{O}(n \log n + m)$ time and $\mathcal{O}(n + m)$ space.

```
Algorithm 3: SATIS
Input: $\phi$ a satisfiable set of UTVPI constraints, $G_\phi = (V, E)$ is the constraint graph
       of $\phi$, $\pi$ is a valid potential function for $G_\phi$, $c$ a UTVPI constraint
Output: SAT if $\phi \cup \{c\}$ is satisfiable, UNSAT otherwise
1  for all $(u, v, d) \in E(c)$ do
2      $r$ := INCCONDIFF($(V, E)$,$\pi$,$(u, v, d)$);
3      if $r = UNSAT$ then
4          return $UNSAT$
5      else
6          $((V, E), \pi) := r$
7      if $\pi(u) + d - \pi(v) = 0$ then
8          Determine the SCCs of the graph
           $H = (V, \{(s, t) \mid (s, t, d') \in E, \pi(x) + d' = \pi(y)\})$ reachable from the node $v$;
9          for all $s \in V$ reachable from $v$ in $H$ do
10             if $-s$ is in the same SCC as $s$ in $H$ and $\pi(-s) - \pi(s)$ is odd then
11                 return $UNSAT$
12 return $SAT$
```

Theorem 4. *Algorithm 3 (*SATIS*) is correct and runs in $\mathcal{O}(n \log n + m)$ time and $\mathcal{O}(n + m)$ space.*

Proof. The complexity follows straightforwardly, recall that constructing SCCs requires $\mathcal{O}(n + m)$ time and space. The correctness follows from the correctness of LAMU (Lemmas 2 and 3) as well as Lemma 4 which allows us to avoid generating the SCCs of the entire graph $G_{\phi \cup \{c\}}$. □

5. Incremental UTVPI Implication

The incremental implication problem is given by a set P of $p = |P|$ UTVPI constraints and a satisfiable set ϕ of m UTVPI constraints on n variables, where $\phi \not\models c', \forall c' \in P$, as well as a single new UTVPI constraint c. For each $c' \in P$ it is to check if $\phi \wedge c \models c'$.

Incremental implication is important if we wish to use UTVPI constraints in a Satisfiability Modulo Theories (SMT) solver (Niewenhuis et al., 2005), as well as for uses in abstract interpretation and spatial databases. Our approach to incremental implication is similar to the approach of Cotton and Maler (2006) for incremental implication for difference constraints. The fundamental new insight that our implication algorithm exploits is that building the tightened transitive closure TTC of a constraint set ϕ can be managed by first building the transitive closure $TC(\phi)$ and then applying the tightening on it.

In this section we first prove that $TTC(\phi) = TI(TC(\phi)$ using two lemmas. This means that we can determine most of the information about the tightened transitive closure by considering transitive closure from the constraint graph. We then show how using this insight we can reason about UTVPI implication simply using shortest paths, as well as a function to extract the upper and lower bounds of variables directly from the constraint graph.

To prove some of the following results we introduce the notion of a *proof* of a constraint being a member of $TTC(\phi)$ as follows:

Definition 3. A *proof* of a constraint $c \in TTC(\phi)$ (or analogously $TI(\phi)$ or $TC(\phi)$) is a tree of nodes labelled by constraints. The root of the tree is labelled c. If the constraint c is generated by transitive closure of c_1 and c_2 then a node labelled c has two child nodes labelled c_1 and c_2. If the constraint c is generated by tightening of c' then the node labelled c has a single child node labelled c'. If $c \in \phi$ then c is a leaf.

The next two lemmas show that $TTC(\phi)$ can be built up by the two closure steps $TI(TC(\phi))$. At first a constraint involving two variables in the tightened transitive closure is resulted from the transitive closure operator, *i.e.* in other words a tightening introduces constraints involving a single variable and any further transitive closure involving them can only create new constraints involving a single variable.

Lemma 5. *Let $ax + by \leq d \in TTC(\phi)$ where $\{a, b\} \subseteq \{-1, 1\}$ then $ax + by \leq d \in TC(\phi)$.*

Proof. Define a constraint $ax + by \leq d$ as a binary constraint if $\{a, b\} \subseteq \{-1, 1\}$.

The proof is by induction on proof size. If $c \in TTC(\phi)$ then it has a finite proof. If the proof size is 0, then c is a leaf and $c \in TC(\phi)$.

Suppose the result holds for all proofs of size less than k. Let $c \in TTC(\phi)$ be a binary constraint with proof size k. Now c is generated using the transitive closure rule, since the tightening rule cannot generate binary constraints. Examining the transitive closure rule, if the result is binary then the generating constraints $c_1 \equiv ax - y \leq d_1$ and $c_2 \equiv y + bz \leq d_2$ are also binary. Since the proof for the constraints c_1 and c_2 are less than k then by induction, $\{c_1, c_2\} \subseteq TC(\phi)$, hence $c \in TC(\phi)$. □

For $TTC(\phi) = TI(TC(\phi))$ we only have to show that any result of transitive closure on a new UTVPI constraint $by \leq d'$ introduced by tightening, can be mimicked using the constraint $by + by \leq \{2d', 2d' + 1\}$ that introduced it, and tightening the end result.

Lemma 6. *Let $ax \le d \in TTC(\phi)$ where $a \in \{-1, 1\}$ then $ax \le d \in TI(TC(\phi))$.*

Proof. We show that for each $ax \le d \in TTC(\phi)$ either $ax \le d \in TC(\phi)$ or $ax + ax \le d' \in TC(\phi)$ where $d' = 2d$ or $d' = 2d + 1$. Then clearly $ax \le d \in TI(TC(\phi))$.

The proof is by induction on proof size. Let $c \in TTC(\phi)$. If the proof size is 0 then $c \in \phi$ and $c \in TI(TC(\phi))$. Suppose the result holds for all proofs of size less than k. Let $ax \le d \in TTC(\phi)$ with proof size k.

If the root node is a transitive closure node with children labelled c_1 and c_2 then exactly one of c_1 and c_2 is binary. Assume c_1 is binary, the other case is similar. Now $c_1 \equiv ax - y \le d_1$ and $c_2 \equiv y \le d_2$ and $d = d_1 + d_2$. By Lemma 5 $c_1 \in TC(\phi)$. Since the proof of c_2 has size less than k by induction we have that $y \le d_2 \in TC(\phi)$ or $y + y \le d_2 + d_2 + e \in TC(\phi)$—if $y \le d_2 \notin TC(\phi)$—where $e \in \{0, 1\}$. In the first case, clearly $ax \le d \in TC(\phi)$. In the second case, we have $ax + ax \le d_1 + d_1 + d_2 + d_2 + e \in TC(\phi)$ by two applications of the transitive closure on c_1 and c_2, and then on c_1 again. Hence, the induction hypothesis holds.

If the root node is a tightening node with child c' then c' is binary and hence by Lemma 5 $c' \in TC(\phi)$, and the induction hypothesis holds, too. □

The above two results show that $TC(\phi)$ is the crucial set of interest for UTVPI implication checking. The following result shows how the constraint graph can be used to reason about $TC(\phi)$. It also shows implicitly (in combination with Lemma 2) that the satisfaction in $\mathbb{Q}$ is decided by $TC(\phi)$ if a constraint $0 \le d \in TC(\phi)$ where d is negative.

Lemma 7. *$c \in TC(\phi)$ iff there is a cycle of length d, in the case of $c \equiv 0 \le d$, or a path $u \rightsquigarrow v$ of length d in G_ϕ where $(u, v, d) \in E(c)$.*

Proof. ($\Rightarrow$): The proof is by induction on proof size. Clearly if the proof size is 0, then $c \in TC(\phi)$ and $E(c)$ appear in the graph G_ϕ. Let $c \equiv ax + bz \le d_1 + d_2$ have proof size k. Then c is built using $c_1 \equiv ax - y \le d_1$ and $c_2 \equiv y + bz \le d_2$. Assume for simplicity $a = 1$, and $b = -1$ the remaining cases are similar.

By induction there exists a path from $x^+ \rightsquigarrow y^+$ of length d_1 in G_ϕ and exists a path $y^+ \rightsquigarrow z^+$ of length d_2 in G_ϕ. Hence there is a path of length $d_1 + d_2$ from x^+ to z^+. Now if $x = z$ this is a cycle.

($\Leftarrow$): The proof is by induction on the number of edges in path $u \rightsquigarrow v$. If the number of edges is 1 then $(u, v, d) \in G_\phi$ and hence $c \in \phi$.

Let $u \rightsquigarrow v$ be a path of length d involve k edges, then it has the form $u \rightsquigarrow w$ of length d_1 and $(w, v, d_2) \in G_\phi$ where $d = d_1 + d_2$. Now by induction there is $(u, w, d_1) \in E(c_1)$ for some $c_1 \in TC(\phi)$ and $(w, v, d_2) \in E(c_2)$ for some $c_2 \in \phi$. Assume c_1 is of the form $x - y \leq d_1$, and c_2 is of the form $y - z \leq d_2$, where $w = y^+$, $u = x^+$ and $v = z^+$. The other cases are similar. Then by transitive closure $c \equiv x - z \leq d_1 + d_2 \in TC(\phi)$ and $(u, v, d) \in E(c)$. □

Example 4. Consider ϕ of Example 1. Then for example $x + x \leq 1 \in TC(\phi)$ and there is a path $x^+ \rightsquigarrow x^-$ of length 1 in G_ϕ shown in Figure 1(c). Similarly $y - z \leq -5 \in TC(\phi)$ and there are paths $z^- \rightsquigarrow y^-$ and $y^+ \rightsquigarrow z^+$ of length -5 in G_ϕ.

The consequence of Lemma 7 is that we can use paths not only to reason about all constraints in $TC(\phi)$ but also to infer about the tightened constraints in $TTC(\phi)$ by looking for paths from a node u to its counternode $-u$ in the constraint graph G_ϕ. That means no tightened edges need to be added to G_ϕ just as in the satisfiability case. But still tightening has to be handled. For that we introduce a *bounds function* ρ which records the upper and lower bounds for each variable x, on the vertices x^+ and x^-. It is defined as:

$$\rho(u) = \left\lfloor \frac{wSP(u, -u)}{2} \right\rfloor .$$

We can show that $\rho(x^+)$ is the upper bound of x and $-\rho(x^-)$ is the lower bound of x. Using Lemmas 6 and 7 we have.

Lemma 8. *For UTVPI constraints* ϕ,

$$\begin{aligned} \rho(x^+) &= \min\{d \mid x \leq d \in TTC(\phi)\} \\ \rho(x^-) &= \min\{d \mid -x \leq d \in TTC(\phi)\} \end{aligned}$$

where we assume $\min \emptyset = +\infty$. □

Example 5. Consider the graph in Figure 1(c) for constraints ϕ of Example 1. Then $\rho(x^+) = 0$ since $wSP(x^+, x^-)$ equals 1 and $x \leq 0 \in TTC(\phi)$, while $\rho(z^-) = -4$ since $wSP(z^-, z^+) = -7$ and $-z \leq -4 \in TTC(\phi)$. Note e.g. $\rho(x^-) = +\infty$ and there is no constraint of the form $-x \leq d$ in $TTC(\phi)$.

The next two theorems state how the constraint graph G_ϕ and the bounds function ρ can be used to reason about satisfaction and implication. The key to incremental satisfaction is the following result.

Theorem 5. *If the constraint graph G_ϕ contains no negative weight cycle (i.e. ϕ is satisfiable in $\mathbb{Q}$) then ϕ is unsatisfiable in $\mathbb{Z}$ iff a vertex $v \in V$ exists with $\rho(v) + \rho(-v) < 0$.*

Proof. Let ϕ be a satisfiable set of UTVPI constraints in $\mathbb{Q}$. By Lemma 7 there is no constraint $0 < d \in TC(\phi)$ where $d < 0$. Therefore ϕ is unsatisfiable in $\mathbb{Z}$ iff such a constraint belongs to $TTC(\phi) \setminus TC(\phi)$ (Theorem 2), i.e. a possible unsatisfiability is caused by tightening.

Lemma 5 implies the equivalence for each constraint $c \in TTC(\phi) \setminus TC(\phi)$ to $ax \leq d$ where $a \in \{-1, 0, 1\}$. Hence, ϕ is unsatisfiable in $\mathbb{Z}$ iff two constraints $x \leq d_1$ and $-x \leq d_2$ with $d_1 + d_2 < 0$ exist in $TTC(\phi)$ iff (by Lemma 8) $\rho(x^+) + \rho(x^-) < 0$. □

Effectively failure can only be caused by tightening if the bounds of a single variable contradict.

Example 6. Consider the graph in Figure 1(a) for constraints ϕ' of Example 2. There is no negative weight cycle in $G_{\phi'}$ but $\rho(x^+) = 0$ and $\rho(x^-) = -1$ because of $x^- \xrightarrow{-4} z^+ \xrightarrow{3} x^+$. Hence the system is unsatisfiable.

Similarly the key to incremental implication is the following rephrasing of Theorem 3.

Theorem 6. *If ϕ is a satisfiable set of UTVPI constraints then $\phi \models c$ iff for at least one $(u, v, d) \in E(c)$ either $wSP(u, v) \leq d$ or $\rho(u) + \rho(-v) \leq d$.*

Proof. Let ϕ be a satisfiable set of UTVPI constraints. Because of Theorem 3 it holds $\phi \models c$ and $c \equiv ax + by \leq d$ iff $ax + by \leq d' \in TTC(\phi)$ and $d' \leq d$ or $\{ax \leq d_1, by \leq d_2\} \subseteq TTC(\phi)$ and $d_1 + d_2 \leq d$.

Now, the theorem holds straightforwardly due to Lemma 8 for the constraints with one variable, and Lemma 5 and 7 for the other constraints. □

Example 7. Consider the graph in Figure 1(c) for constraints ϕ of Example 1. $\phi \models -z \leq -3$ is shown since $wSP(z^-, z^+) = -7 \leq 2 \times -3$.

Algorithm 4: Impl shows the new algorithm. As input it takes the constraint graph G_ϕ, a valid potential function π, the bounds function ρ, a set P of UTVPI constraints to check for implication, as well as the UTVPI constraint c which should be added to ϕ.

In the first step (line 1) the constraint c is transformed to its corresponding edges $E(c)$ in a constraint graph. Then each edge in $E(c)$ is added consecutively to the constraint graph G_ϕ

Algorithm 4: IMPL – Incremental satisfaction and implication for UTVPI constraints.

```
Input: $G_\phi = (V, E)$ a *constraint graph* representing set of UTVPI constraints $\phi$, $\pi$ a
       *valid potential function* on $G_\phi$, $\rho$ the *bounds function* of $\phi$, $P$ a set of UTVPI
       constraints not implied by $\phi$, and a UTVPI constraint $c$ to be added.
Output: $G_{\phi\cup\{c\}}$, a *valid potential function* $\pi'$ and the *bounds function* $\rho'$ of $\phi \cup \{c\}$
        and the set $P' \subseteq P$ of constraints not implied by $\phi \cup \{c\}$, or UNSAT if
        $\phi \cup \{c\}$ is not satisfiable.
1  $G' := G_\phi$, $\pi' := \pi$, $\rho' = \rho$, compute $E(c)$;
2  for all $e \in E(c)$ do
3      $res :=$ IncConDiff$(G', \pi', e)$;
4      if $res = UNSAT$ then
5          return $UNSAT$
6      else
7          $(G', \pi') := res$
8  let $(u, v, d)$ be any edge in $E(c)$;
9  compute $\delta_u^{\leftarrow}$ and $\delta_v^{\rightarrow}$ by using the reduced-cost graph for $G'$ via $\pi'$;
10 for all $x \in V$ do
11     $sp := \delta_u^{\leftarrow}(x) + d + \delta_v^{\rightarrow}(-x)$;
12     $\rho'(x) := \min\{\rho(x), \lfloor \frac{sp}{2} \rfloor\}$;
13 for all $x \in V$ do
14     if $\rho'(x) + \rho'(-x) < 0$ then return $UNSAT$;
15 $P' := \emptyset$;
16 for all $c' \in P$ do
17     $(x, y, d') :=$ first element in $E(c')$;
18     if $\delta_u^{\leftarrow}(x) + d + \delta_v^{\rightarrow}(y) > d'$ and $\delta_u^{\leftarrow}(-y) + d + \delta_v^{\rightarrow}(-x) > d'$ and $\rho'(x) + \rho'(-y) > d'$
       then $P' := P' \cup \{c'\}$;
19 return $(G', \pi', \rho', P')$
```

by using the IncConDiff algorithm of Cotton and Maler (2006). After inserting all edges in G', the constraint graph equals $G_{\phi\cup\{c\}}$ and π' is a valid potential function for G'. Hence $\phi \cup \{c\}$ is satisfiable in $\mathbb{Q}$. The remainder of the algorithm maintains the bounds function ρ' (lines from 8 to 12) and it is used to test the satisfiability in $\mathbb{Z}$ (lines 13 and 14), and the implication of constraints in P (lines 15 to 18). For building an efficient implementation of shortest paths the reader is referred to Cotton and Maler (2006).

By Lemma 7 to maintain ρ we need to see if the shortest path from x to $-x$ has changed. We only need to scan for new shortest paths using the newly added edges. We can restrict attention to a single added edge (u, v, d) since if there is a path from x over the edge (u, v, d) to $-x$ ($x^+ \rightsquigarrow u \xrightarrow{d} v \rightsquigarrow x^-$) then because of Lemma 1 there is equal-weight path from x via the "counter-edge" $(-v, -u, d)$ to $-x$ ($x^+ \equiv -x^- \rightsquigarrow -v \xrightarrow{d} -u \rightsquigarrow -x^+ \equiv x^-$).

We calculate the shortest paths in $G_{\phi\cup\{c\}}$ from each vertex x to u ($\delta_u^{\leftarrow}(x)$) and from v to each vertex x ($\delta_v^{\rightarrow}(x)$) (line 9). The shortest path for $\delta_u^{\leftarrow}$ can be computed like $\delta_u^{\rightarrow}$ by simply reversing the edges in the graph.

We can then calculate the shortest path from x to $-x$ via the edge $u \xrightarrow{d} v$ using the path $x^+ \rightsquigarrow u \xrightarrow{d} v \rightsquigarrow x^-$ as $\delta_u^{\leftarrow}(x) + d + \delta_v^{\rightarrow}(-x)$. We update ρ' if required (line 12).

We can now check satisfiability of $\phi \cup \{c\}$ in $\mathbb{Z}$ using Theorem 5 (lines 13 and 14). Finally we check implications using Theorem 6.

Using the above results, it is not difficult to show that the algorithm is correct with the desired complexity bounds.

Theorem 7. *Algorithm 4 (*IMPL*) is correct and runs in $\mathcal{O}(n \log n + m + p)$ time and $\mathcal{O}(n + m + p)$ space.*

Proof. The algorithm is correct if it returns UNSAT in the case of unsatisfiability of $\phi \cup \{c\}$ or the constraint graph $G_{\phi\cup\{c\}}$, its valid potential function π', its bounds function ρ' and the set of constraints $P' \subseteq P$ not implied by $\phi \cup \{c\}$.

Lemma 2 and Algorithm INCCONDIFF (see Cotton and Maler (2006)) guarantee that after termination of INCCONDIFF $G' = G_{\phi\cup\{c\}}$ and π' is its valid potential function if $\phi \cup \{c\}$ is satisfiable in $\mathbb{Q}$; otherwise $\phi \cup \{c\}$ is unsatisfiable and the algorithm returns UNSAT.

After application of INCCONDIFF the algorithm maintains the bounds function (lines 8 to 12) by calculation of the shortest path $x \rightsquigarrow u \rightarrow v \rightsquigarrow -x$ via one added edge $(u, v, d) \in E(c)$ for each node x in $G_{\phi\cup\{c\}}$. Remark: we only have to considered the shortest paths via the added edges ρ give us the length of a shortest path without those added edges. Due to Theorem 5 the algorithm checks $\phi \cup \{c\}$ for unsatisfiability in $\mathbb{Z}$ in the next two lines. If it is unsatisfiable IMPL terminates and returns UNSAT.

The remainder of the algorithm computes the set of non-implied constraints $P' \subseteq P$ by testing for all constraints $c' \in P$ if the length of both paths $x \rightsquigarrow u \rightarrow v \rightsquigarrow y$, $-y \rightsquigarrow u \rightarrow v \rightsquigarrow -x$ are longer than d' and the sum of the upper bounds $\rho'(x) + \rho'(-y)$ is greater than d' where $(x, y, d') \in E(c')$. If all three cases hold then c' is not implied by $\phi \cup \{c\}$ thanks to Theorem 6.

The run-time is determined by the run-time of INCCONDIFF, the calculation of $\delta_u^{\leftarrow}$, $\delta_v^{\rightarrow}$ which are $\mathcal{O}(n \log n + m)$, and the implication check $\mathcal{O}(p)$. All the other computations can be done in constant or linear time with respect to n and m. So the overall run-time is $\mathcal{O}(n \log n + m + p)$.

The space is determined by the space to store the graph and implication constraints so it is $\mathcal{O}(n+m+p)$. The algorithm only needs to attach a constant amount of information to parts of the graph. □

6. Experimental Results

We present empirical comparisons of the algorithms discussed herein, first on satisfaction and then on implication questions.

For both experiments we generate 60 UTVPI instances ϕ in each problem class with the following specifications: the values d range uniformly in from -15 to 100, approximately 10% are negative, each variable appears in at least one UTVPI constraint, each constraint involves exactly two variables, and there is at most one constraint allowed between any two variables.

The experiments were run on a Sun Fire T2000 running SunOS 5.10 and a 1 GHz processor. The code was written in C and compiled with gcc 3.4.3 and option -O3. Each run was given a 2 minute time limit.

We run incremental satisfaction on a system of m constraints in n variables, adding the constraints one at a time. We compare: Satis the incrementalization of LaMu presented in Section 4, Impl the incremental implication checking algorithm of Section 5 where $p = 0$, mLaMu running LaMu m times for m satisfaction checks, and HaSt the algorithm of Harvey and Stuckey (1997). For the computation of the shortest paths we used a binary priority queue for argmin, and the early termination and caliber heuristics (Cotton and Maler, 2006).

The results are shown in left of Table 2, where d represents the density of a UTVPI instance, which is defined as $m/2n^2$ representing the percentage of the number of constraints m in the instance to the maximal number of non-quasi-syntactic redundant constraints for n variables. A constraint $ax + by \leq d$ is *quasi-redundant* with respect to ϕ iff $ax + by \leq d' \in \phi$ with $d' < d$ applies. We split the examples into cases that are satisfiable, $\mathbb{Z}$ unsatisfiable, and $\mathbb{Q}$ unsatisfiable. Moreover, the fourth row shows the number of examples for each case written (satisfiable, $\mathbb{Z}$ unsatisfiable, $\mathbb{Q}$ unsatisfiable) and the overall average run-time. For more dense satisfiable systems Impl is best, but overall Satis is the clear winner. Interestingly for very dense satisfiable systems (not shown here) HaSt beats the others.

For the implication benchmarks we chose 5 satisfiable, $\mathbb{Z}$ unsatisfiable, and $\mathbb{Q}$ unsatisfiable

Table 2: Average run-time in seconds of the satisfiability algorithms

Examples		Satis	Impl	mLaMu	HaSt
$n = 100$	feasible	0.03	0.13	1.02	1.56
$m = 1000$	Z-inf.	0.01	0.09	0.57	1.42
$d = 5\%$	Q-inf.	0.01	0.04	0.26	0.81
all (32, 8, 20)		0.02	0.10	0.70	1.29
$n = 100$	feasible	0.18	0.34	3.81	1.98
$m = 2000$	Z-inf.	<0.01	0.14	1.25	1.68
$d = 10\%$	Q-inf.	0.02	0.03	0.21	0.68
all (31, 9, 20)		0.10	0.20	2.23	1.50
$n = 100$	feasible	0.98	0.94	12.39	2.31
$m = 4000$	Z-inf.	0.02	0.15	1.17	1.82
$d = 20\%$	Q-inf.	0.01	0.04	0.25	0.80
all (28, 12, 20)		0.46	0.48	6.10	1.71
$n = 200$	feasible	0.73	1.28	16.84	16.72
$m = 4000$	Z-inf.	0.03	0.43	3.17	13.34
$d = 5\%$	Q-inf.	0.01	0.11	0.91	6.01
all (30, 11, 19)		0.37	0.76	9.29	12.71
$n = 200$	feasible	3.82	3.35	56.14	19.17
$m = 8000$	Z-inf.	0.03	0.52	4.03	14.86
$d = 10\%$	Q-inf.	0.01	0.09	0.79	4.84
all (29, 11, 20)		1.86	1.74	28.14	13.60
$n = 200$	feasible	17.76	11.01	>120.0	20.75
$m = 16000$	Z-inf.	0.04	0.65	5.59	18.13
$d = 20\%$	Q-inf.	0.01	0.10	0.84	5.19
all (28, 12, 20)		8.30	5.30	>57.4	15.04
$n = 800$	feasible	3.78	13.57	>120.0	>120.0
$m = 12800$	Z-inf.	0.12	6.80	63.98	>120.0
$d = 1\%$	Q-inf.	0.03	1.68	14.17	>120.0
all (27, 13, 20)		1.74	8.14	>72.59	>120.0

instances for each problem class. In addition, 5 implication sets P of size p were created for each n using the same restrictions as defined above. On average over all benchmarks, 65% of the constraints in P were implied by the corresponding ϕ.

The incremental implication experiments check satisfiability and the implications of constraints P incrementally as each of the m constraints were added one at a time. A run was terminated if there were no more constraints to add, or unsatisfiability was detected. We compare the two algorithms that can check implication: Impl versus HaSt. The right of Table 3 shows the results. Overall the checks for implication are cheap compared to the satisfaction check for HaSt, but not for Impl, since it must compute the shortest path after

Table 3: Average run-time in seconds of the implication algorithms

Examples		Impl	HaSt
$n = 100$	$p = 50$	0.31	1.28
$m = 1000$	$p = 100$	0.32	1.29
$d = 5\%$	$p = 200$	0.34	1.31
$n = 100$	$p = 50$	0.53	1.49
$m = 2000$	$p = 100$	0.54	1.50
$d = 10\%$	$p = 200$	0.56	1.52
$n = 100$	$p = 50$	1.10	1.66
$m = 4000$	$p = 100$	1.12	1.68
$d = 20\%$	$p = 200$	1.14	1.70
$n = 200$	$p = 100$	2.11	12.52
$m = 4000$	$p = 200$	2.16	12.56
$d = 5\%$	$p = 400$	2.26	12.66
$n = 200$	$p = 100$	3.84	12.19
$m = 8000$	$p = 200$	3.89	12.25
$d = 10\%$	$p = 400$	4.00	12.35
$n = 200$	$p = 100$	10.24	13.59
$m = 16000$	$p = 200$	10.31	13.66
$d = 20\%$	$p = 400$	10.46	13.82
$n = 800$	$p = 400$	34.74	>200.0
$m = 12800$	$p = 800$	35.50	>200.0
$d = 1\%$	$p = 1600$	37.08	>200.0

each constraint addition. Hence the times of HaSt are similar to the satisfaction case, and Impl needs about three times longer. While HaSt improves the more dense the system, Impl is the clear winner on sparse systems.

While UTVPI constraints are used in a number of practical applications, they are usually deeply embedded inside other systems, such as theorem provers, or program analysers, so there are no suites of stand-alone UTVPI problems we are aware of. Our experience of the kinds of UTVPI problems that arise from these applications are that they are very sparse. This indicates that our algorithms should be advantageous for real applications.

7. Non-Incremental Implication Checking and Generation

The incremental algorithm of the previous sections can be extended to create non-incremental implication algorithms which either check all constraints in a set P for implication or computes all (tightly) implied constraints. These algorithms are of particular importance for the

Algorithm 5: Non-incremental satisfaction and implication for UTVPI constraints.

```
   Input: $G_\phi = (V, E)$ a *constraint graph* representing set of UTVPI constraints $\phi$ and
          $P$ a set of UTVPI constraints not implied by $\phi$.
   Output: Either $P' \subseteq P$ a set of UTVPI constraint implied by $\phi$, or UNSAT, if $\phi$ is
           unsatisfiable.
1  if LaMu($\phi$) $= UNSAT$ then return $UNSAT$;
2  $P' := \emptyset$; let $\pi$ be the generated valid potential function of $G_\phi$;
3  for all $u \in V$ do
4      compute $\delta_u^{\rightarrow}$ by using $rc(G_\phi)$ via $\pi$;
5      for all $c \equiv u + by \leq d \in P$ do
6          if $y \neq 0 \wedge \delta_u^{\rightarrow}(y^{-b}) \leq d$ or $y = 0 \wedge \lfloor \delta_u^{\rightarrow}(-u)/2 \rfloor \leq d$ then $P' := P' \cup \{c\}$
7  return $P'$
```

use of UTVPI constraints in abstract interpretation (Miné, 2006) since they allow checking of implication between two sets of UTVPI constraints, and building a canonical form of a set of UTVPI constraints.

The non-incremental implication algorithm which checks constraints P for implication by ϕ is shown in Algorithm 5. At first it checks the satisfiability of ϕ by using LaMu. If ϕ is unsatisfiable then the algorithm terminates with *UNSAT*. Then the shortest path of all pairs of nodes is computed using Johnson's algorithm (see Johnson (1977)) which runs Dijkstra's algorithm from every node u in the reduced cost graph $rc(G_\phi)$ via the valid potential function π.

If the shortest path between u and v is d' then all constraints $u - v \leq d$ are implied with $d' \leq d$. To ensure that we check each constraint $ax + by \leq d \in P$ at most one time, we assume a map from $u = ax$ to all constraints of the form $ax + by \leq d \in P$.

The overall complexity is $\mathcal{O}(n^2 \log n + nm + |P|)$ time and $\mathcal{O}(n + m + |P|)$ space which is determined by Johnson's algorithm $\mathcal{O}(n^2 \log n + nm)$ time and the size of the implication set P.

To generate all tightly implied constraint of satisfiable system ϕ, that is for $\phi \models ax + by \leq d$ but $\phi \not\models ax + by \leq d'$ for $d' < d$, we use a variation of the same algorithm. Instead of checking the constraints in P for implication we use $\delta_u^{\rightarrow}$ to create new constraints. For each $u \in V$ and each v where $\delta_u^{\rightarrow}(v) < +\infty$ we create the constraint $u - v \leq \delta_u^{\rightarrow}(v)$ if $v \notin \{u, -u\}$, and the constraint $u \leq \lfloor \delta_u^{\rightarrow}(v)/2 \rfloor$ when $v = -u$.

Finally we generate $u + v \leq d_u + d_v$ for each $u \leq d_u$ and $v \leq d_v$ previously created where $v \notin \{u, -u\}$, and remove any quasi-syntactic redundant constraints, that is $ax + by \leq d$

where $ax + by \leq d'$ and $d' < d$, from the generated set.

This algorithm needs $\mathcal{O}(n^2 \log n + nm)$ time and $\mathcal{O}(n + m + |P|)$ space, since the number of implied constraints $|P|$ is bounded in $\mathcal{O}(n^2)$.

8. Generation of Minimal Unsatisfiable Subsets and Minimal Implicants

Given ϕ an unsatisfiable set of UTVPI constraints, then a minimal unsatisfiable subset of ϕ is a set $M \subseteq \phi$ such that M is unsatisfiable and each $M' \subset M$ is satisfiable. Suppose that $\phi \models c$, a minimal implicant M of c is a set $M \subseteq \phi$ such that $M \models c$ and for each $M' \subset M$ is $M' \not\models c$. These are highly related since $M \models c$ iff $M \wedge \neg c$ is unsatisfiable. Minimal unsatisfiable subsets (minimal implicants) are useful if we are using a UTVPI solver as a sub-solver in a SAT modulo theories (SMT) solver (Niewenhuis et al., 2005) which requires an explanation of unsatisfiability (and implication) to encode in Booleans the knowledge of the UTVPI solver.

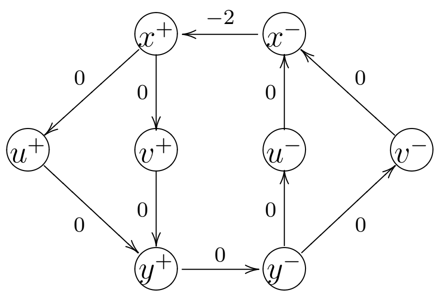
In the remainder of this section we explain the generation of a minimal subset $M \subseteq \phi$ in case of $\mathbb{Q}$-unsatisfiability of ϕ. A minimal subset generator for $\mathbb{Z}$-unsatisfiability and implication $\phi \models c$ can be adapted easily from the $\mathbb{Q}$-unsatisfiability case.

The underlying idea is to use the algorithm IncConDiff to recover a simple negative cycle as in Cotton and Maler (2006) and construct a minimal unsatisfiable set with respect to this cycle. The algorithm recovers a negative cycle by keeping track of the last edge (u, v, d) for every node v which refines $\gamma(v)$ in IncConDiff. Hence, each node in a negative cycle has one associated edge which will form a simple negative cycle. This cycle is extracted by backtracing from x to y where $(x, y, .)$ is the last added edge to the constraint graph.

The corresponding constraint set of the cycle defines a minimal unsatisfiable set in the difference constraint context, but not in all cases for UTVPI constraints, since a UTVPI constraint with two variables is represented by two edges in the constraint graph.

Example 8. Consider the satisfiable constraint set $\phi = \{x - u \leq 0, u - y \leq 0, x - v \leq 0, -v - y \leq 0, y \leq 0\}$ and the constraint $c \equiv -x \leq -1$. Here $\phi' = \phi \cup \{c\}$ is unsatisfiable in $\mathbb{Q}$.

The left-hand side in Figure 2 shows the constraint graph $G_{\phi'}$ and the right-hand side a table with a possible sequence of IncConDiff steps under the assumption that the algorithm was called with G_ϕ, π, and $(x^-, x^+, -2)$ where $\pi(v) = 0$ for all nodes v. The



No.	node	tracked edge	refines
1	x^+	$(x^-, x^+, -2)$	$\gamma(u^+) = -2$
			$\gamma(v^+) = -2$
2	u^+	$(x^+, u^+, 0)$	$\gamma(y^+) = -2$
3	v^+	$(x^+, v^+, 0)$	
4	y^+	$(u^+, y^+, 0)$	$\gamma(y^-) = -2$
5	y^-	$(y^+, y^-, 0)$	$\gamma(u^-) = -2,$
			$\gamma(v^-) = -2$
6	v^-	$(y^-, v^-, 0)$	$\gamma(x^-) = -2$
-	x^-	$(v^-, x^-, 0)$	

Figure 2: The outer cycle is one possible tracked cycle whose corresponding constraint set is not minimal. The steps of IncConDiff are shown on the right-hand side.

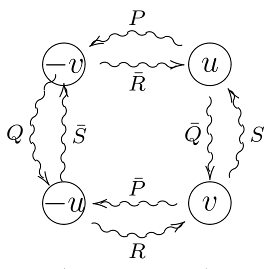
table shows: the number of the step, the dequeued node, its tracked edge, and the caused refinements of γ.

The constraint set of the tracked negative cycle (the outer cycle in the graph $G_{\phi'}$) is ϕ' which is not minimal, since $\phi' \setminus \{x - u \leq 0, u - y \leq 0\}$ and $\phi' \setminus \{x - v \leq 0, -v - y \leq 0\}$ are the only minimal unsatisfiable sets.

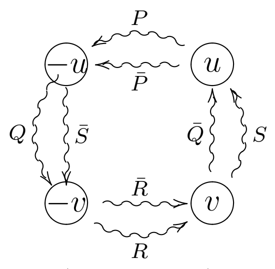
The reason for the tracked non-minimal set depends on the two equal-weight paths and theirs counterpaths from x^+ to y^+ and y^- to x^- respectively. Each path can refine the γ-value y^+ and x^- respectively, whose tracked edge decides which path we backtrace during recovery of the cycle. In our example we backtrace the path via v^- from x^- to y^-, but not its counterpath from y^+ to x^+, since the backtracked edge $(u^+, y^+, 0)$ of y^+ is part of the other path which leads to a non-minimal set.

To generate a minimal unsatisfiable subset M of non-minimal unsatisfiable sets N we could apply a general algorithm for minimal unsatisfiability, for example that described in Junker (2004) which needs $\mathcal{O}(|N|)$ steps. In each step the algorithm checks the satisfiability of a subset of N, so the overall run-time is $\mathcal{O}(|N| \cdot \tau)$ where τ is the run-time complexity of one satisfiability check. In our case τ is $\mathcal{O}(nm)$ where n is the number of variables in N and $m = |N|$.

But we can do far better. Figure 3 shows the only two possible patterns of constraint graphs G_N which can occurs if the corresponding UTVPI constraint set N of a simple negative cycle is not minimal. A wiggly line represent a path between two nodes and a path labeled $\bar{P}$ is the counterpath of P.



(Pattern A)



(Pattern B)

Figure 3: Two possible patterns of constraint graph of a non-minimal unsatisfiable constraint set arising from a simple negative-weight cycle. In Pattern A $PQRS$ is a negative cycle but $P\bar{R}$ may represent a negative cycle derived from a strict subset of the constraints. In Pattern B $PQRS$ is a negative cycle but either $PQR\bar{Q}$ or $P\bar{S}RS$ may be negative cycles derived from a strict subset of the constraints.

Lemma 9. *Let C be a simple negative cycle in G_ϕ and N be its corresponding constraint set. If N is a non-minimal unsatisfiable constraint set then its constraint graph G_N fits into one of the patterns appearing in Figure 3.*

Proof. Let N be the corresponding constraint set of a simple negative cycle $C = (x_1, x_2, \dots, x_n)$ in G_ϕ, which is a non-minimal unsatisfiable constraint set. Since $0 \leq d \notin N$ with $d < 0$ and C is a simple cycle there exists a minimal unsatisfiable constraint set $M \subset N$ with $|M| > 1$. Due to Lemma 2 G_M contains a simple negative cycle C_M for which the following holds

$$C_M \cap C \neq \emptyset \qquad \text{and} \qquad C_M \setminus C \neq \emptyset \ .$$

Let $\bar{S} = (-u = -s_k \to -s_{k-1} \to \cdots \to -s_1 = -v)$ be a path in $C_M \setminus C$ such that nodes $\{-s_k, -s_1\}$ appear in C and nodes $-s_i, 1 < i < k$ do not appear in C. Moreover, let $C_{\bar{S}}$ be its corresponding constraint set. As $C_{\bar{S}} \subset C_M \subset C$ the counterpath $S = (v = s_1 \to s_2 \to \cdots \to s_k = u)$ must be a part of C.

W.l.o.g. let $x_i = s_{i-n+k}$ for all $n - k < i \leq n$, $x_i = -s_k$, and $x_j = -s_1$. Hence, $i \neq j$ and $i, j \leq n - k$.

Pattern A $(i > j)$**:** Therefore, C equals to $PQRS$, where $P = x_n \rightsquigarrow x_j$, $Q = x_j \rightsquigarrow x_i$, and $R = x_i \rightsquigarrow x_{n-k+1}$.

Pattern B $(i < j)$**:** For pattern B the paths on $C = PQRS$ are $P = x_n \rightsquigarrow x_i$, $Q = x_i \rightsquigarrow x_j$, and $R = x_j \rightsquigarrow x_{n-k+1}$.

Thus, G_C also includes theirs counterpath in both cases. Furthermore, because of the selection of $\bar{S}$ the paths Q, $\bar{Q}$, S, and $\bar{S}$ do not share any interior nodes. Note that this is not necessarily the case for the pairs or paths P, $\bar{R}$, and $\bar{P}$, R. This means that all paths from a node in Q, S to a node in P, or R must pass at least u, $-u$, v, or $-v$. □

Our approach to determining minimal unsatisfiable subsets extends Cotton and Maler (2006) by preventing the construction of negative cycles including pattern A or B. It does do by slightly changing of the order for dequeuing the nodes and extending the backtracing of a negative cycle, so that the generated unsatisfiable constraint set is minimal. First we introduce a new order $\prec$ on the pair $(\gamma, \#_s)$ over γ and the minimal number of edges of a shortest path from s to the nodes:

$$(\gamma(u), \#_s(u)) \prec (\gamma(v), \#_s(v)) \quad \Longleftrightarrow \quad \gamma(u) < \gamma(v) \text{ or } \gamma(u) = \gamma(v) \wedge \#_s(u) < \#_s(v) \ ,$$

where $\#_s(x) = \min\{|P| \mid P = s \rightsquigarrow x : w(P) = wSP(s,x)\}$.

Compared to the order used by IncConDiff this order changes the sequence of dequeuing of nodes—therefore also for the tracked edges, as well—but only for two nodes with the same γ-values. The behaviour of dequeuing on those nodes is like a breadth-first search. Thus, IncConDiff with $\prec$ will find a simple negative cycle with the shortest length and the minimal number of edges, so that the constraint graph of its corresponding constraints cannot match with pattern A.

Example 9. Consider the pattern A of Figure 3 and suppose that all paths only include edges with a weight of 0 except for P and $\bar{P}$ which only contain the edge $(u, -v, -2)$ and $(v, -u, -2)$ respectively. Moreover, let us assume that no path shares any of its interior nodes with any other path.

The graph with the paths Q, R, S, and their counterpaths has no negative cycle, but with the path P and $\bar{P}$ it does. Then a minimal unsatisfiable set can only result from the cycles $P\bar{R}$ or $\bar{P}R$. During a run of IncConDiff each γ-value can be refined exactly one time to -2. The new order $\prec$ avoids a refinement of $\gamma(u)$ through the last edge in the path S, which would lead to a non-minimal set, if IncConDiff starts at $-v$. In this example $\gamma(u)$ will always be refined by the last edge in $\bar{R}$, since the number of edges of the simple shortest path from $-v$ via $\bar{R}$ to u is smaller by $|QS|$ than the other simple path QRS.

To avoid a recovery of a cycle whose constraint graph fits in pattern B we change the tracked edge of some nodes during backtracing of a cycle as follows: If we backtrace over the edge (u, v, d) then we set the tracked edge of $-u$ to the counteredge $(-v, -u, d)$ iff

$0 = \pi(-v) + d - \pi(-u)$ and $\#_s(-u) = \#_s(-v) + 1$. That is, if the path from s via $-v$ to $-u$ is a simple shortest path from s to $-u$ and the number of edges of both paths are the same. (Only in this situation can IncConDiff make a "bad" decision concerning pattern B during the $\mathbb{Q}$-satisfiability check.)

Example 10. Let us consider the Example 8. The steps shown in the right of Figure 2 can still be performed by using the new order $\prec$, because the competing paths have the same number of edges.

During the recovery of the cycle—beginning at the node x^-—we switch the tracked edge $(u^+, y^+, 0)$ of y^+ to $(v^+, y^+, 0)$, since we backtrace over the edge $(y^-, v^-, 0)$ before reaching y^+. Thus, the recovered cycle contains the path via v^- and its counterpath instead of its competing path via u^+, so that the corresponding constraint set $\{-x \leq -1, x - v \leq 0, -v - y \leq 0, y \leq 0\}$ is a minimal unsatisfiable set.

The modified version of IncConDiff with $\prec$ and backtracing is presented in ModIncConDiff. In the $\mathbb{Q}$-unsatisfiability case the algorithm returns a minimal unsatisfiable set. Otherwise, it is the same as IncConDiff.

Theorem 8. *Let ϕ be a satisfiable set of UTVPI constraints and c be a UTVPI constraint. If $\phi \cup \{c\}$ is $\mathbb{Q}$-satisfiable then each run of* ModIncConDiff *which adds one edge of $E(c)$ more to G_ϕ is successful and the last run establishes a valid potential function on $G_{\phi\cup\{c\}}$; Otherwise one run of* ModIncConDiff *terminates with a minimal unsatisfiable set M of $\phi \cup \{c\}$.*

Proof (sketch): The correctness of the algorithm—whether it terminates with $G_{\phi\cup\{c\}}$ and a valid potential function π on that graph or it terminates with a minimal unsatisfiable set M—follows straightforwardly from the correctness of IncConDiff, since $\prec$ only changes the order of dequeuing the nodes where the "old" order chooses "randomly" the next to dequeued node.

To show that the algorithm generates a minimal unsatisfiable set M if $\phi \cup \{c\}$ is unsatisfiable leads to an extensive case study. For this reason we omit the details of this proof.

We need to show that because of the order $\prec$ the corresponding constraint graph of the recovered negative cycle by ModIncConDiff cannot match with the pattern A. Then we show that this corresponding graph also cannot fit in the pattern B due to the switching of the tracked edge during the backtracing of the cycle.

Due to Lemma 9 the tracked cycle must lead to a minimal unsatisfiable set. □

Algorithm 6: ModIncConDiff

```
Input: $G_\phi = (V, E)$ a graph, $\pi$ a valid potential function for $G_\phi$, $(u, v, d)$ a new edge
       to add to $G_\phi$.
Output: Minimal unsatisfiable set $M$ if $\phi \cup \{u - v \leq d\}$ is unsatisfiable, or
        $G_{\phi\cup\{u-v\leq d\}}$ and a valid potential function $\pi'$ for $G_{\phi\cup\{u-v\leq d\}}$.
1  $\gamma(v) := \pi(u) + d - \pi(v)$; $\#_v(v) := 0$;
2  $\gamma(w) := 0$ and $\#_v(w) := \infty$ for all $w \neq v$;
3  while $min(\gamma) < 0 \wedge \gamma(u) = 0$ do
4      $s :=$ argmin($\gamma$, $\#_v$) wrt. the order $\prec$ ;
5      $\pi'(s) := \pi(s) + \gamma(s)$ ;
6      $\gamma(s) := 0$ ;
7      for all $s \overset{d'}{\to} t \in G$ do
8          if $\pi'(t) = \pi(t)$ and $\gamma(t) > \pi'(s) + d' - \pi'(t)$ then
9              $\Delta := \pi'(s) + d' - \pi'(t)$;
10             if $\gamma(t) > \Delta$ or $(\gamma(t) = \Delta$ and $\#_v(t) > \#_v(s) + 1)$ then
11                 $\gamma(t) = \Delta$;
12                 $\#_v(t) = \#_v(s) + 1$;
13                 tracked_edge($t$) $:= (s, t, d')$;
14 if $\gamma(u) < 0$ then
15     $s := u$; $M := \{u - v \leq d\}$;
16     while $s \neq v$ do
17         $(w, t, d') :=$ tracked_edge($s$);                 /* $t$ is the current $s$ */
18         if $\pi(-t) + d' - \pi(-w) = 0$ and $\#_v(-w) = \#_v(-t) + 1 \neq \infty$ then
19             tracked_edge($-w$) $:= (-t, -w, d')$;
20         $M := M \cup \{w - t \leq d'\}$;
21         $s := w$;
22     return $M$
23 return $((V, E \cup \{(u, v, d)\}), \pi')$
```

The modified algorithm ModIncConDiff can also be used to generate a minimal unsatisfiable set in the $\mathbb{Z}$-unsatisfiable case, and a minimal implicant. For both cases we run the algorithm on ϕ with the edges in $E(c')$ of the constraint $c' \equiv ax + by \leq d - 2$, if $c \equiv ax + by \leq d$ causes the $\mathbb{Z}$-unsatisfiability of $\phi \cup \{c\}$ or $c \equiv -ax - by \leq -d$ is implied by ϕ, since $\phi \cup \{c'\}$ is $\mathbb{Q}$-unsatisfiable. The complexity of answering all these questions using ModIncConDiff is $\mathcal{O}(n \log n + m)$ time and $\mathcal{O}(n + m)$ space where $m = |N|$ and n is the number of variables occuring in N.

9. Conclusion

We have presented new incremental algorithms for UTVPI constraint satisfaction and implication checking which improve upon the previous asymptotic complexity, and perform better in practice for sparse constraint systems.

We have adapted the algorithms herein to provide non-incremental implication checking in $\mathcal{O}(n^2 \log n + nm + p)$ time and $\mathcal{O}(n + m + p)$ space, and generate all implied constraints in $\mathcal{O}(n^2 \log n + nm)$ time and $\mathcal{O}(n + m + p)$ space, where p is the number of implied constraints generated.

We also extended the algorithms to return a minimal unsatisfiable subset when unsatisfiability is detected, and a minimal implicant of an implied constraint.

Acknowledgements

We would like to thank the anonymous reviewers for their comments which improved the paper substantially. NICTA is funded by the Australian Government as represented by the Department of Broadband, Communications and the Digital Economy and the Australian Research Council.

References

Bellman, R. 1958. On a routing problem. *Quarterly of Applied Mathematics* **16** 87–90.

Cherkassky, B. V., A. V. Goldberg. 2006. Negative-cycle detection algorithms. *Proceedings of the European Symposium on Algorithms*. 349–363.

Cotton, S., O. Maler. 2006. Fast and Flexible Difference Constraint Propagation for DPLL(T). *Theory and Applications of Satisfiability Testing - SAT 2006*, *LNCS*, vol. 4121. Springer-Verlag, 170–183.

Ford, L. R., D. R. Fulkerson. 1962. *Flows in Networks*. Princeton University Press.

Frigioni, D., A. Marchetti-Spaccamela, U. Nanni. 1998. Fully dynamic shortest paths and negative cycle detection on digraphs with arbitrary edge weights. *European Symposium on Algorithms*. 320–331.

Harvey, W., P. J. Stuckey. 1997. A Unit Two Variable Per Inequality Integer Constraint Solver for Constraint Logic Programming. *The 20th Australasian Computer Science Conference (Australian Computer Science Communications)*. Sydney, Australia, 102–111.

Jaffar, J., M. J. Maher, P. J. Stuckey, R. H. C. Yap. 1994. Beyond finite domains. *Principles and Practice of Constraint Programming*, *LNCS*, vol. 874. Springer-Verlag, 86–94.

Johnson, D. B. 1977. Efficient algorithms for shortest paths in sparse networks. *J. ACM* **24** 1–13.

Junker, Ulrich. 2004. Quickxplain: Preferred explanations and relaxations for over-constrained problems. *Proceedings of the Nineteenth National Conference on Artificial Intelligence*. AAAI Press, 167–172.

Lahiri, S. K., M. Musuvathi. 2005. An Efficient Decision Procedure for UTVPI Constraints. *Frontiers of Combining Systems*, *LNCS*, vol. 3717. Springer-Verlag, 168–183.

Miné, A. 2006. The octagon abstract domain. *Higher-Order and Symbolic Computation* .

Niewenhuis, R., A. Oliveras, C. Tinelli. 2005. Abstract DPLL and Abstract DPLL Modulo Theories. *Logic for Programming, Artificial Intelligence, and Reasoning*, *LNAI*, vol. 3452. Springer-Verlag, 36–50.

Seshia, S., K. Subramani, R. Bryant. 2007. On solving Boolean combinations of UTVPI constraints. *Journal of Satisfiability, Boolean Modelling and Computation* **3** 67–90.

Sitzmann, I., P. J. Stuckey. 2000. O-trees: a constraint based index structure. M. Orlowska, ed., *Proceedings of the Eleventh Australasian Database Conference (ADC2000)*. IEEE Press, 127–135.