Global Difference Constraint Propagation for Finite Domain Solvers

Thibaut Feydy
National ICT Australia, Victoria Laboratory
Department of Computer Science & Software Engineering
The University of Melbourne, Australia
tfeydy@csse.unimelb.edu.au

Andreas Schutt
National ICT Australia, Victoria Laboratory
Department of Computer Science & Software Engineering
The University of Melbourne, Australia
aschutt@csse.unimelb.edu.au

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

Abstract

Difference constraints of the form $x - y \leq d$ are well studied, with efficient algorithms for satisfaction and implication, because of their connection to shortest paths. Finite domain propagation algorithms however do not make use of these algorithms, and typically treat each difference constraint as a separate propagator. Propagation does guarantee completeness of solving but can be needlessly slow. In this paper we describe how to build a (bounds consistent) global propagator for difference constraints that treats them all simultaneously. SAT modulo theory solvers have included theory solvers for difference constraints for some time. While a theory solver for difference constraints gives the basis of a global difference constraint propagator, we show how the requirements on the propagator are quite different. We give experiments showing that treating difference constraints globally can substantially improve on the standard propagation approach.

Categories and Subject Descriptors D.3.2 [*Language Classifications*]: Constraint and logic languages; D.3.3 [*Language Constructs and Features*]: Constraints

General Terms Algorithms

Keywords Finite domain propagation, Separation logic, global constraints

1. Introduction

Finite domain propagation is a powerful approach to solving complex combinatorial satisfaction and optimization problems. The core of a finite domain propagation solver are the propagators which given a constraint c and a domain D representing the set of possible values of the variables in c, remove the possible values of the variables which cannot take part in any solution in c. The propagation solver interleaves the fixpoint computation of the propagators with user-controlled search in order to find a solution or optimal solution. In this paper we study how we should propagate constraints of the form $x - y \leq d$.

PPDP'08, July 15–17, 2008, Valencia, Spain.

Constraints of difference, that is $x - y \leq d$ where x, y are variables and d is an integer constant, are one of the simplest form of constraints. They are well-studied, with efficient algorithms known for their satisfaction and implication, because of their connection to shortest path algorithms.

But finite domain propagation solvers do not make use of these algorithms. Difference constraints are represented as individual propagators just like any other constraints. Unlike most constraints, propagation on difference constraints is complete. That is if C is a set of difference constraints then a propagation solver starting from domain D will fail if $C \wedge D$ is unsatisfiable.

But the order of the propagation of the constraints can make a significant difference. In a sense the propagation engine implements a version of Fords labelling and scanning algorithm (see (han 1990), p.558) that also checks for negative cycles. The potential bad behaviour of propagation on constraints of difference is well recognized.

EXAMPLE 1. Consider the constraints: $x - y \leq 0$, $y - x \leq -2$. Together these are unsatisfiable. If the initial domains of x and y are $D(x) = D(y) = [0 \,..\, n]$ then a propagation engine will take $\mathcal{O}(n)$ steps to determine the unsatisfiability of the constraints. This could be determined in constant time. □

Reified constraints of difference $b \Leftrightarrow x - y \leq d$, that is if $b = 1$ then the constraint $x - y \leq d$ holds, and if $b = 0$ then its negation $y - x \leq -d - 1$ holds, are very useful in expressing scheduling and other problems.

Finite domain propagation engines typically implement them as weak bounds propagators that only propagate to either set $b = 1$ when the bounds of x and y force the constraint to be true, or set $b = 0$ if the bounds of x and y force it to be false, or propagate as the difference constraint (or its negation) if b is set to 1 (resp. 0).

Finite domain propagation engines are not complete for checking implication of reified difference constraints by difference constraints.

EXAMPLE 2. Consider the constraints $c_1 \equiv y - x \leq -2$ $c_2 \equiv x - z \leq 3$ with initial domains $D(x) = D(y) = D(z) = [0 \,..\, 10]$. The resulting domains after propagation are $D'(x) = [0 \,..\, 8]$ and $D'(y) = D'(z) = [2 \,..\, 10]$. The reified constraint $b \Leftrightarrow y - z \leq 4$ does not force $b = 1$ although this is certainly a consequence of $c_1 \wedge c_2$ since $c_1 \wedge c_2 \models y - z \leq 1$ by simply summing the constraints. □

SAT Modulo Theories (SMT) solvers (Niewenhuis et al. 2005, 2006) treat difference constraints quite differently than finite domain propagators. Effectively, they treat reified difference con-

straints $b \Leftrightarrow x - y \leq d$ where b acts as the "name" of the constraint in the SAT solver. They use specialized shortest path algorithms (Cotton and Maler 2006) to determine the unsatisfiability of the difference constraints which are consequences of the current Boolean evaluation (e.g. if b is true then $x-y \leq d$, and if b is false then $y - x \leq -d - 1$), and to determine all the difference constraints that are entailed or disentailed by the difference constraints which are consequences of the current Boolean evaluation.

In this paper we investigate how we can build a global propagator for difference constraints for use in a finite domain propagation engine, that treats all difference constraints together, that can determine all consequences of reified constraints. As we shall see the questions that arise in a finite domain propagation context are different from those in SMT, and there are different trade offs for the implementation.

With this global propagator we can straightforwardly construct, for the first time, a domain propagation for the SEQUENCE constraint with best known complexity (the CD encoding of (Brand et al. 2007)).

The contributions of this paper are:

- the first implementation of a global difference logic propagator
- improved algorithms for bounds propagation using global difference logic
- the first implementation of the best known `sequence` constraint propagator
- the development of hybrid propagation approaches that combine the global propagator with individual difference constraint propagators
- experiments illustrating that treating difference logic constraint globally can be substantially more efficient than treating them as individual constraints.

The remainder of the paper is organized as follows. In Section 2 we introduce our notation for finite domain solving, and illustrate the usual propagators for difference constraints. In Section 3 we introduce notation for graphs and shortest paths, and give the fundamental theorem relating difference constraint satisfaction and implication to shortest paths. We then explain the state of the art incremental algorithms for difference constraints. In Section 4 we define the capabilities and default implementation of a global difference constraint propagator, before discussing how to improve it by handling bounds specially. Then in Section 5 we give experimental results, and finally in Section 6 we conclude.

2. Finite Domain Solvers

We now introduce our terminology for finite domain propagation based solving.

A *domain* D is a complete mapping from a fixed (finite) set of variables $\mathcal{V}$ to finite sets of integers. A *false domain* D is a domain with $D(x) = \emptyset$ for some $x \in \mathcal{V}$. We can understand a domain D as the logical formula $\wedge_{v \in \mathcal{V}} v \in D(v)$. A false domain is equivalent to false. A variable $x \in \mathcal{V}$ is *fixed* by a domain D, if $|D(x)| = 1$. The *intersection* of domains D_1 and D_2, denoted $D_1 \sqcap D_2$, is defined by the domain $D(x) = D_1(x) \cap D_2(x)$ for all $x \in \mathcal{V}$. A domain D_1 is *stronger* than a domain D_2, written $D_1 \sqsubseteq D_2$, if $D_1(x) \subseteq D_2(x)$ for all $x \in \mathcal{V}$.

A range is a contiguous set of integers, we use *range* notation $[l \,..\, u]$ to denote the range $\{d \in \mathbb{Z} \mid l \leq d \leq u\}$ when l and u are integers. A *range domain* D is one where $D(x)$ is a range for all $x \in \mathcal{V}$. We shall only be interested in range domains for this paper.

An *integer valuation* θ is a mapping of variables to integer values, written $\{x_1 \mapsto d_1, \ldots, x_n \mapsto d_n\}$. We extend the valuation θ to map expressions and constraints involving the variables in the natural way.

Let vars be the function that returns the set of variables appearing in a valuation. We define a valuation θ to be an element of a domain D, written $\theta \in D$, if $\theta(x_i) \in D(x_i)$ for all $x_i \in \text{vars}(\theta)$.

A *constraint* c over variables $x_1, \ldots, x_n$ is a set of valuations θ such that $\text{vars}(\theta) = \{x_1, \ldots, x_n\}$. We also define $\text{vars}(c) = \{x_1, \ldots, x_n\}$.

We will *implement* a constraint c by a propagator $\text{prop}(c)$ that maps domains to domains. A *propagator* f is a monotonically decreasing function from domains to domains: $f(D) \sqsubseteq D$, and $f(D_1) \sqsubseteq f(D_2)$ whenever $D_1 \sqsubseteq D_2$. A propagator f is *correct* for a constraint c iff for all domains D $\{\theta \mid \theta \in D\} \cap c = \{\theta \mid \theta \in f(D)\} \cap c$. This is a very weak restriction, for example the identity propagator is correct for all constraints c. A *domain propagator* for c, $\text{dom}(c)$, is the strongest correct propagator for c. It is defined as $\text{dom}(c)(D)(v) = \{\theta(v) \mid \theta \in D \wedge \theta \in c\}$.

EXAMPLE 3. The domain propagator $f = \text{dom}(x - y \leq d)$ for the difference constraint $x - y \leq d$ can be implemented as:

$$f(D)(x) = \{e \in D(x) \mid e \leq d + \max D(y)\}$$
$$f(D)(y) = \{e \in D(y) \mid \min D(x) - d \leq e\}$$
$$f(D)(v) = D(v), v \notin \{x, y\} \ .$$

A domain propagator $f' = \text{dom}(y - x \leq -d - 1)$ can be implemented analogously. A domain propagator $g = \text{dom}(b \Leftrightarrow x - y \leq d)$ can be implemented as:

$$\begin{array}{ll} g(D)(v) = f(D)(v), & \text{if } D(b) = \{1\} \\ g(D)(v) = f'(D)(v), & \text{if } D(b) = \{0\} \\ g(D)(b) = D(b) \setminus \{0\}, & \text{if } \max D(x) - \min D(y) \leq d \\ g(D)(b) = D(b) \setminus \{1\}, & \text{if } \min D(x) - \max D(y) > d \\ g(D)(v) = D(v), & \text{otherwise} \end{array}$$

where $v \in \mathcal{V}$. □

A *propagation solver* $\text{solv}(F, D)$ for a set of propagators F and an initial domain D finds the greatest mutual fixpoint of all the propagators $f \in F$. In other words, $\text{solv}(F, D)$ returns a new domain defined by

$$\text{solv}(F, D) = \text{gfp}(\lambda d.\, \text{iter}(F, d))(D)$$

and

$$\text{iter}(F, D) = \bigsqcap_{f \in F} f(D)$$

where gfp denotes the greatest fixpoint w.r.t $\sqsubseteq$ lifted to functions.

3. Solving Difference Constraints

Difference constraints are highly connected to shortest paths, a statement we shall formalize shortly. In this section we give our graph notation, and then explain how the state of the art difference constraints algorithms work.

3.1 Graphs, Paths and Potential Functions

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 $e_i = (v_{i-1}, v_i, d_i) \in E$. 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 a *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 $\delta_x^{\leftarrow}, \delta_x^{\rightarrow} : V \rightarrow \mathbb{R}$ as

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

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.

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. The standard approach is the Bellman-Ford algorithm which is $\mathcal{O}(nm)$.

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 shortest paths in the reduced cost graph in time $\mathcal{O}(n \log n + m)$ instead of $\mathcal{O}(nm)$ assuming the use of a Fibonacci heap.

3.2 Difference Constraints

Difference constraints are a well studied class of constraints (see e.g. (Shostak 1981; Dechter et al. 1991)). Difference constraints are of the form $x - y \leq d$. Note that we can encode bounds constraints of the form $x \geq l$ and $x \leq u$ by selecting a *dummy* variable v_0 to represent the fixed value 0, and encoding them as $v_0 - x \leq -l$ and $x - v_0 \leq u$ respectively. We call these *encoded bounds constraints*. We can map difference constraints to a weighted directed graph. For the remainder of this paper let v_0 denote the dummy variable.

DEFINITION 1. Let C be a set of difference constraints and let $G_C = (V, E)$ be the graph comprised of one weighted edge $x \xrightarrow{d} y$ for every constraint $x - y \leq d$ in C. We call G_C 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_C its corresponding graph. C is satisfiable iff G_C has no negative weight cycles, and if C is satisfiable then $C \models x - y \leq d$ iff $wSP(x, y) \leq d$.* □

Ramalingam *et al* (Ramalingan et al. 1999) define efficient algorithms for satisfiability of difference constraints after incrementally adding or deleting a constraint. Cotton and Maler (Cotton and Maler 2006) define efficient incremental algorithms for difference constraints, the satisfiability algorithm for incremental addition is identical to that of (Ramalingan et al. 1999), while they also give an algorithm for checking implication of difference constraints on addition. For our purposes we are only interested in incremental addition algorithms so we will use the formulation of Cotton and Maler.

The incremental satisfaction algorithm for addition (Algorithm 1: INCSAT) of (Cotton and Maler 2006; Ramalingan et al. 1999) relies on maintaining a potential function π for the constraint graph G_C. In a sense it is an incremental Bellman-Ford algorithm. On addition of a new constraint $u - v \leq d$ the edge $u \xrightarrow{d} v$ is added to G_C and a new potential function π' is calculated or unsatisfiability is detected. The algorithm is $\mathcal{O}(n \log n + m)$ for m difference constraints on n variables (using a Fibonacci heap to implement argmin).

Algorithm 1: INCSAT

Input: $G_C = (V, E)$ a graph, π a valid potential function for G_C, edge (u, v, d) a new constraint to add to G_C.
Output: UNSAT if $C \cup \{u - v \leq d\}$ is unsatisfiable, or $G_{C \cup \{u-v \leq d\}}$ and a valid potential function π' for $G_{C \cup \{u-v \leq d\}}$.

```
γ(v) := π(u) + d − π(v);
γ(w) := 0 for all w ≠ v;
π'(v) := π(v) for all v ∈ V;
while min(γ) < 0 ∧ γ(u) = 0 do
    s := argmin(γ) ;
    π'(s) := π(s) + γ(s) ;
    γ(s) := 0 ;
    for all s --d'--> t ∈ G do
        if π'(t) = π(t) then
            γ(t) := min{γ(t), π'(s) + d' − π'(t)}
if γ(u) < 0 then
    return UNSAT
return ((V, E ∪ {(u, v, d)}), π')
```

The implication algorithm (Algorithm 2: INCIMP) of (Cotton and Maler 2006) simply checks for each difference constraint $x - y \leq d$ of interest whether the new edge has created a path from x to y of length $\leq d$. It makes use of the potential function previously calculated to compute shortest paths on the reduced cost graph using Dijkstra rather than using a more expensive algorithm that handles negative weight edges. The algorithm is $\mathcal{O}(n \log n + m + p)$ for m difference constraints on n variables and p constraints to check for implication.

Algorithm 2: INCIMP

Input: $G = (V, E)$ a *constraint graph* representing a set of difference constraints $C \cup \{u - v \leq d\}$, π a *valid potential function* on G, a set of difference constraints P where $C \not\models d, \forall d \in D$.
Output: The set $P' \subseteq P$ of constraints not implied by $C \cup \{u - v \leq d\}$.

```
P' := ∅ ;
compute δ_u^← and δ_v^→ by using rc(G) via π;
for all c' = (x − y ≤ d') ∈ P do
    if δ_u^←(x) + d + δ_v^→(y) > d' then P' := P' ∪ {c'};
return P'
```

EXAMPLE 4. Consider the system of constraints $C = \{x - y \leq -2, y - z \leq 3\}$ where $D(x) = D(y) = D(z) = [0 \,..\, 10]$. The corresponding graph G_C is shown in Figure 1(a). Given a valid potential function $\pi(v_0) = 0$, $\pi(x) = -3$, $\pi(y) = -8$, $\pi(z) = -7$, the reduced cost graph of G_C is shown in Figure 1(b).

Consider the addition of the constraint $y - x \leq 0$ using INCSAT. $\gamma(x) = -5$ and γ for the remaining variables is 0. x is the minimal γ value. We set $\pi'(x) = -8$ and set $\gamma(x) = 0$. We then adjust the γ values: $\gamma(y) = -2$. Now $\gamma(y) \neq 0$ and the loop terminates and returns UNSAT. The unsatisfiable loop has been found just examining the nodes x and y and their outgoing edges.

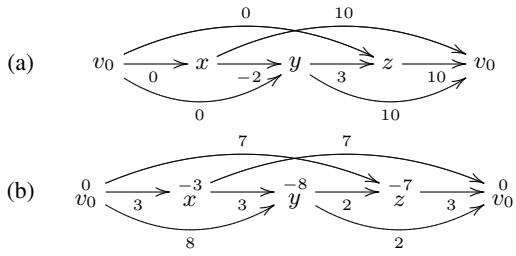

Figure 1. The corresponding constraint graph G_C for a system of difference constraints $C = \{x - y \leq -2, y - z \leq 3\}$ is shown in (a). The node for dummy variable v_0 is shown twice for clarity of presentation. The reduced cost graph is shown in (b) assuming the potential value shown above the nodes. Note how all edge lengths are non-negative.

Now consider the implication test for the constraint $y - z \leq 4$ using INCIMP assuming we have just added the constraint $x - y \leq -2$, and obtained the potential function illustrated in Figure 1(b). We compute $\delta_x^{\leftarrow}$ from the reduced cost graph in Figure 1(b). First we compute the weights of shortest paths to x $wSP(v_0, x) = 3$, $wSP(z, x) = 6$, $wSP(y, x) = 5$ and then calculate $\delta_x^{\leftarrow}$ using the potential values $\delta_x^{\leftarrow}(x) = 0$, $\delta_x^{\leftarrow}(v_0) = 0$, $\delta_x^{\leftarrow}(z) = 10$, $\delta_x^{\leftarrow}(y) = 10$. We similarly calculate $wSP(y, z) = 2$, $wSP(y, v_0) = 2$, $wSP(y, x) = 5$ in the reduced cost graph and hence $\delta_y^{\rightarrow}(y) = 0$, $\delta_y^{\rightarrow}(z) = 3$, $\delta_y^{\rightarrow}(v_0) = 10$, and $\delta_y^{\rightarrow}(x) = 10$. Examining $\delta_x^{\leftarrow}(x) + -2 + \delta_y^{\rightarrow}(z) = 0 + -2 + 3$ we find this is less that 4 and hence the constraint is implied. □

4. Difference Constraint Propagation

In this section we explain how we can create a global propagator for all difference constraints and reified difference constraints.

4.1 A Global Difference Constraint Propagator

We can use the incremental algorithms INCSAT and INCIMP to create a global propagator as follows:

The global propagator needs to support the following operations:

- Add a new difference constraints $x - y \leq d$
- Add a new bound $x \geq l$ or $x \leq u$
- Add a new reified difference constraint $b \Leftrightarrow u - v \leq d'$
- Mark a state and backtrack to a previous marked state

Adding a new difference constraint $x - y \leq d$ causes the propagator to run INCSAT, and fails if this determines unsatisfiability. If not it runs INCIMP to determine if any reified difference constraints $b \Leftrightarrow u - v \leq d'$ become entailed and disentailed, and sets the Booleans to true (1) or false (0) appropriately. This is the same as an SMT difference constraint solver.

This is not enough since we are also interested in any new bounds of variables z that result from the constraint addition. A corollary of Theorem 1 gives the key insight.

COROLLARY 1. *If C is a satisfiable set of difference and (encoded) bounds constraint then $C \models z \geq l$ iff $wSP(v_0, z) \leq -l$ and $C \models z \leq u$ iff $wSP(z, v_0) \leq u$.* □

To extract new bounds the propagator needs to determine for each variable z the weight of a shortest path from v_0 to z via the new edge (x, y, d), i.e. is $\delta_x^{\leftarrow}(v_0) + d + \delta_y^{\rightarrow}(z)$. The negation of this is a lower bound on z. It also determines the weight of a shortest path from z to v_0 via the new edge (x, y, d), $\delta_x^{\leftarrow}(z) + d + \delta_y^{\rightarrow}(v_0)$, which is an upper bound on z. The propagator updates the bounds of variables which have changed with respect to these weights.

EXAMPLE 5. Consider adding the new constraint $x - y \leq -2$ to obtain the system of constraints C of Example 4 illustrated in Figure 1(a). In order to extract new bounds we need to determine $\delta_x^{\leftarrow}(v_0)$ but this is just $-\min D(x)$, we have already determined $\delta_y^{\rightarrow}$ for INCIMP. The possibly new bounds are $0 + -2 + 0$ for y (or a lower bound of 2), $0 + -2 + 3$ for z (or a lower bound of -1, so not new). The upper bounds are calculated similarly as $0 + -2 + 10$ for x or 8. The resulting domain is $D(x) = [0 \,..\, 8]$, $D(y) = [2 \,..\, 10]$, and $D(z) = [0 \,..\, 10]$. □

Adding a new bound $x \leq u$ or $x \geq l$ is simply adding a new difference constraint $x - v_0 \leq u$ or $v_0 - x \leq -l$, which then is treated as described in the previous paragraphs.

Adding a new reified constraint $b \Leftrightarrow u - v \leq d'$ is a new feature not usually present in SMT solvers, since in this context all reified constraints are known from the beginning of solving. We need to check if the constraint is entailed or disentailed with the current set of difference constraints. To do so we calculate $wSP(u, v)$ and $wSP(v, u)$ by using Dijkstra's algorithm on the reduced cost graph. Note that reified bounds constraints $b \Leftrightarrow u \leq d'$ or $b \Leftrightarrow -v \leq d'$ can be handled by examining the domain D directly.

For backtracking we need to restore the state of the difference constraint propagator to that at an earlier time. The only state of the solver is the set of difference constraints posted, and the set of posted reified difference constraints, as well as the valid potential function. The potential function π remains valid on backtracking since it simply ensures that $\pi(x) + d - \pi(y) \geq 0$ for all edges $(x, y, d) \in G$, and removing edges does not invalidate this. Hence the potential function need not be trailed, as been noted by Wang et al. (Wang et al. 2005). Backtracking must simply remove the representation of the constraints added since the marked state.

Under the assumptions that the domain D is a range domain and no Boolean variable appears twice in the set of difference constraints and bounds constraints and reified difference constraints C added, the global propagator defined in this subsection is a domain propagator for (the conjunction) C.

4.2 Handling Bounds Constraints Better

While in the SMT context bounds are simply another form of difference constraint, for a propagation engine bounds updates are much more frequent than difference constraint additions. Hence it is worth treating them separately. We will not use the dummy variable v_0 to encode bounds constraints but treat them directly. The basis of this treatment is the following theorem.

THEOREM 2. *Let C be a set of difference constraints and D a range domain, c a difference constraint, and $D' = \text{solv}(C, D)$ the domain after propagation.*

Then (a) D' is a false domain iff $D \wedge C$ is unsatisfiable. And (b) $C \wedge D \models c$ iff $C \models c$ or $D' \models c$. I.e. we can check implications by checking implication by difference constraints alone, and implication by bounds alone.

Proof: (a) ($\Leftarrow$) By the correctness of propagation $C \wedge D \models \text{solv}(C, D) = D'$ and hence if D' is a false domain $C \wedge D$ is unsatisfiable.

($\Rightarrow$) Let G be the graph encoding C and D (as encoded bounds constraints using dummy variable v_0), then by Theorem 1 we have that the corresponding graph has a negative weight cycle p. W.l.o.g., let p be from $v \rightsquigarrow v$ with weight $w < 0$, $D(v) = [l_v \,..\, u_v]$ be the range of v and k be chosen so that $kw + u_v - l_v < 0$.

Then consider the path

$$v_0 \xrightarrow{-l_v} \overbrace{v \rightsquigarrow v \cdots v \rightsquigarrow v}^{k \text{ times}} \xrightarrow{u_v} v_0$$

Now standard bounds propagation choosing the constraints in the order of this path eventually sets the lower bound of v to $l_v - kw > u_v$ and creates a false domain. Since any order of propagating constraints leads to the same result, D' is a false domain.

(b)($\Leftarrow$) Clearly $C \models c$ or $D' \models c$ imply that $C \wedge D \models c$ since $C \wedge D \models D'$.

($\Rightarrow$) By Theorem 1 we have that $c \equiv x - y \leq d$ is implied iff the graph G encoding C and D using encoded bounds constraints iff a shortest path from x to y is length less than or equal to d. Suppose the shortest path does not visit v_0. Then clearly $C \models c$ since G_C (without the encoded bound constraints) has the same shortest path. Otherwise $SP(x, y) = x \rightsquigarrow v_0 \rightsquigarrow y$ where $w(x \rightsquigarrow v_0) = w_1$ and $w(v_0 \rightsquigarrow y) = w_2$ and $w_1 + w_2 \leq d$. Bounds propagation on the path $v_0 \rightsquigarrow y$ sets the lower bound of y to at least $-w_2$. Bounds propagation on the path $x \rightsquigarrow v_0$ sets the upper bound of x to at most w_1. Hence $D' \models x \leq w_1 \wedge y \geq -w_2$ and hence $D' \models x - y \leq d$.

The above theorem implies we can check the satisfiability of bounds constraints using propagation on the difference constraints, and we can split implication into two checks: just using the difference constraints and just using the computed bounds.

At first let us consider bounds updating. We define an algorithm that simultaneously considers all bounds changes since the last time the global propagator was run, as opposed to adding them one by one which is required by the base approach.

Since the calculation of the new lower and upper bounds is symmetric, we describe only the algorithm INCLB (see Algorithm 3) for the lower bounds.

For the new lower bounds of a variable x we want to know if the negative of the weight of the shortest path to x from v_0, $-\delta^{\rightarrow}_{v_0}(x)$ is greater than the current lower bound of x, $\min D(x)$. Because we do not represent a dummy variable v_0 in the constraint graph, we compute a valid potential function value $\pi(v_0)$ for v_0 with respect to π in the first step and consider for this calculation only variables whose current lower bound is less than the value of the lower bound resulting from the last run of INCLB.

At the second step we run Dijkstra on the reduced cost graph with a starting priority queue of the variables which have changed lower bound since the last run of the propagator. The initial value for variable x is the reduced cost of the "imaginary" edge between v_0 and x. At last we create the bounds constraints for calculated new bounds.

Our Dijkstra's algorithm does not explore all variables in the graph, it visits only variables x for which $-\delta^{\rightarrow}_{v_0}(x) > \min D(x)$ holds, that is where a new lower bound has been found.

EXAMPLE 6. Consider the set of constraints $C = \{ x - y \leq -2, y - z \leq 3, z - u \leq -1, u - v \leq 2, x - t \leq 1, t - z \leq -1\}$. The constraint graph G_C is shown in Figure 2(a). A valid potential function for G_C is $\pi(x) = -3$, $\pi(y) = -8$, $\pi(z) = -7$, $\pi(u) = -9$, $\pi(v) = -7$, $\pi(t) = -4$. In fact since the graph does not include the dummy variable v_0 there are no cycles. The domain $D(x) = [0 .. 18]$, $D(y) = [2 .. 20]$, $D(z) = [6 .. 19]$, $D(u) = [8 .. 20]$, $D(v) = [11 .. 20]$, $D(t) = [0 .. 18]$ is a fixpoint for propagation on these constraints.

Suppose we update the lower bounds of variables t to 1 and x to 5. The algorithm works as follows: $\mathcal{V}_l = \{t, x\}$ and we compute

Algorithm 3: INCLB

Input: $G_C = (\mathcal{V}, E)$ a *constraint graph* representing set of difference constraints C, π a *valid potential function* on G_C, a range domain D_o giving the upper and lower bounds of variables the last time the propagator was run, and a range domain D giving the current bounds.

Output: A set of lower bounds constraints B giving new bounds for variables in $\mathcal{V}$.

```
1  V_l := {x ∈ V | min D(x) > min D_o(x)};
2  π(v_0) := max{min D(x) + π(x) | x ∈ V_l};
3  % Dijktras algorithm from v_0 on the reduced cost graph
   rc(G) augmented so that variable s is not considered if the
   new lower bound −δ→_{v_0}(s) is smaller than the existing
   bound min D(s);
4  γ(v) := π(v_0) − min D(v) + π(v) for all v ∈ V_l;
5  γ(v) := +∞ for all v ∈ V \ V_l;
6  wSP(v_0, v) := +∞ for all v ∈ V;
7  while min(γ) < +∞ do
8      s := argmin(γ) ;
9      wSP(v_0, s) := γ(s);
10     γ(s) := +∞ ;
11     δ→_{v_0}(s) := wSP(v_0, s) + π(s) − π(v_0);
12     if −δ→_{v_0}(s) > min D(s) then
13         for all s −d'→ t ∈ G do
14             if wSP(v_0, t) < +∞ then
15                 γ(t) := min{γ(t), π(s) + d' − π(t)}
16 % Convert new bounds into constraints;
17 for all v ∈ V do
18     if −δ→_{v_0}(v) > − min D(v) then
19         B := B ∪ {−v ≤ δ→_{v_0}(v)}
20 return B
```

$\pi(v_0) = 2$. Effectively we will be searching for shortest paths from v_0 from the reduced cost graph shown in Figure 2(b).

Dijktras algorithm determines $wSP(v_0, x) = 0$ or equivalently $\delta^{\rightarrow}_{v_0}(x) = 0 + -3 - 2 = -5$ and the lower bound of x is 5, a tighter bound, so the algorithm updates values for edges leaving x. Then $wSP(v_0, t) = 2$ or $\delta^{\rightarrow}_{v_0}(t) = 2 + -4 - 2 = -4$, again a new bound 4 so t's neighbours are enqueued. Then $wSP(v_0, y) = 3$ or $\delta^{\rightarrow}_{v_0}(y) = 3 + -8 - 2 = -7$, again a new lower bound. Then $wSP(v_0, z) = 4$ and $\delta^{\rightarrow}_{v_0}(z) = 4 + -7 - 2 = -5$, and a new bound. When we calculate $wSP(v_0, u) = 5$ and $\delta^{\rightarrow}_{v_0}(u) = 5 + -9 - 2 = -6$, the new bound 6 is not stronger than the existing bound, so no new propagation occurs. The algorithm never visits node v.

Compare this with the naive global difference approach of the previous section. Each new bound is a new constraint $v_0 - t \leq 1$ and $v_0 - x \leq -5$, and the graph shown in Figure 2(a) has 12 additional edges to and from the dummy node v_0. Adding the first constraint may determine a new potential function, and then shortest paths from v_0 are determined for each variable, and bounds updated. Then the second constraint is added and a possibly new potential function computed and once more the shortest paths from v_0 are determined and bounds updated. The improved method has a smaller graph, does not update potential function values, and visits each edge at most once regardless of the number of bounds changes since the last execution. □

Note that INCLB (and INCUB) requires $\mathcal{O}(n \log n + m)$ time and $\mathcal{O}(n + m)$ space for m difference constraints (not bounds constraints) on n variables. Incremental propagation of difference

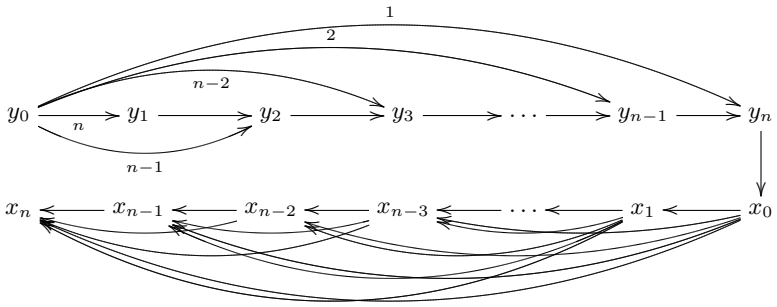

Figure 3. The corresponding constraint graph for a system of difference constraints of Example 7. Edges with 0 weight are unlabelled.

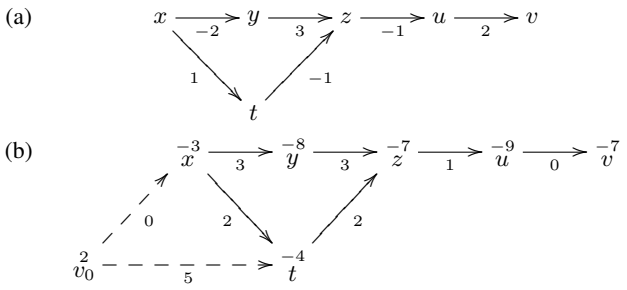

Figure 2. (a) The corresponding constraint graph G_C for a system of difference constraints of Example 6 and (b) the reduced cost graph with the dummy node v_0 and imaginary edges (dashed) added.

constraints using a FIFO queue of propagators[1] can require $\mathcal{O}(nm)$ time.

EXAMPLE 7. Consider the system of difference constraints C defined as $y_{i-1} - y_i \leq 0, 2 \leq i \leq n, y_0 - y_i \leq n - i + 1, 1 \leq i \leq n$ and $y_n - x_0 \leq 0$, $x_i - x_j \leq 0, 1 \leq i < j \leq n$ illustrated in Figure 3. Domain D is a fixpoint for C when $D(y_0) = [0 \,..\, (k-1)n]$, $D(y_i) = [n \,..\, kn], 1 \leq i \leq n$ and $D(x_i) = [n \,..\, kn], 0 \leq i \leq n$. None of the difference constraints is implied by the domain, which would mean it could be removed from the propagation engine.

Consider when the domain of y_0 becomes $[n \,..\, (k-1)n]$. All constraints involving y_0 are queued for propagation. If we are unlucky we will first bounds propagate on $y_0 - y_n \leq 1$, which will change $D(y_n)$ to $[n+1 \,..\, kn]$ and queue the $y_n - x_0 \leq 0$. We then propagate on $y_0 - y_{n-1} \leq 2$ which changes $D(Y_{n-1})$ to $[n+2 \,..\, kn]$ and queues $y_{n-1} - y_n \leq 0$. Continuing we modify each domain $D(y_i), 1 \leq i \leq n$ to $[n+n-i+1 \,..\, kn]$ and queue each $y_{i-1} - y_n \leq 0, 1 \leq i \leq n$ in reverse order. The next propagator considered is $y_n - x_0 \leq 0$ which changes $D(x_0)$ to $[n+1 \,..\, kn]$ and queues the constraints on x_0. Then we consider $y_{n-1} - y_n \leq 0$ and modify $D(y_n) = [n+2 \,..\, kn]$ and queue $y_n - x_0 \leq 0$ again. Continuing we modify each domain $D(y_i), 2 \leq i \leq n$ to $[n+n-i+2 \,..\, kn]$ and queue each $y_{i-1} - y_n \leq 0, 2 \leq i \leq n$ in reverse order. We then propagate the constraints on x_0 setting all $D(x_i)$ to $[n+1 \,..\, kn]$ and queuing all constraints on $x_i, 1 \leq i \leq n$. We then propagate on $y_n - x_n \leq 0$ again and modify $D(x_0) = [n+2 \,..\, kn]$. We queue the constraints on x_0 once more. To reach a fixpoint the domain of x_0 changes n times, and we queue each constraint on x n times, hence the process is $\mathcal{O}(n^3)$.

Compare this with the execution of INCLB. Since all the arcs are positive we can assume $\pi(v) = 0, \forall v \in V$ and the reduced cost graph is identical to the original graph. The algorithm will process each edge exactly once, visiting the nodes in order $y_0, y_1, \ldots, y_n$, x_0 followed by the remainder of the x nodes in any order. The process is $\mathcal{O}(n^2)$.

Finally note the whole process of increasing the lower bound of y_0 by n can be carried out $k-1$ times down a branch of the search tree! □

We can thus check satisfiability of addition of bounds constraints by running INCLB on the new lower bounds and INCUB on the new upper bounds and seeing if in the result any variable has an empty domain. Afterwards, we can check new implications of difference constraints caused by the addition of bounds constraints by simply checking for each $(x - y \leq d) \in P$ if it is implied by bounds i.e. $\max D(x) - \min D(y) \leq d$.

We check satisfiability on addition of a new difference constraints $u - v \leq d$ by running INCSAT on the graph G_C containing only the difference constraints (not bounds constraints), and then perform bounds propagation by determining the possibly new lower bound for v given by $-d + \min D(u)$ and doing bounds propagation with INCLB and the possibly new upper bound for u of $d + \max D(v)$ and doing bounds propagation with INCUB. We can then check implication by first checking if the bounds imply some difference constraint in P, and then running INCIMP on the graph $G_{C \cup \{u-v \leq d\}}$.

Adding a new reified difference constraint $b \Leftrightarrow u - v \leq d'$ requires us to check if it is entailed or disentailed by the domain D, and if not we calculate $wSP(u, v)$ and $wSP(v, u)$ using the reduced cost graph of G_C.

4.3 Variations

Theorem 2 also provides us with other ways of building difference constraint propagators by mixing the standard approach of a propagator per difference constraint and reified difference constraint with the global approach.

Standard propagators will perform bounds propagation and implication by bounds. Hence we can combine the global propagator which only considers addition of difference constraints $x - y \leq d$ and reified difference constraints $b \Leftrightarrow u - v \leq d'$ with the usual propagators for these constraints. By prioritizing the global propagator before the standard propagators we can avoid the worst case behaviour of Example 1 determine implication purely by difference constraints, and use the standard incremental queueing for bounds propagators/implication by bounds.

[1] A FIFO queue is the usual case, and for a LIFO queue it is easy to create even worse behaviour.

4.4 Optimizations

We can further improve upon the algorithms above by taking into account fixedness of variables, and reasoning better on which difference constraints in P can be implied by a constraint addition.

Any fixed variable x where $D(x) = \{d\}$ acts similarly to the dummy variable v_0 since any path $x \rightsquigarrow y$ implies a path $v_0 \xrightarrow{-d} x \rightsquigarrow y$, and similarly $y \rightsquigarrow x$ implies the path $y \rightsquigarrow x \xrightarrow{d} v_0$. Once a variable is fixed and we have updated the bounds caused by the fixing, then the variable plays no further role. We can ignore fixed variables x in INCSAT, INCIMP, INCLB and INCUB without compromising correctness using a result analogous to Theorem 1. Hence edges to and from fixed variables need not be considered in these algorithms.

After every addition of a constraint we need to check the constraints in P for implication. We can do better than checking every one if we keep track of what changes have been made by the latest constraints addition.

Cotton and Maler (Cotton and Maler 2006) show how we can restrict the shortest path calculations in INCIMP to those that may actually decrease a shortest path. There is a shorter path from x to y via new edge (u, v, d) iff there is a shorter path from x to v or u to y via this edge. If we calculate $\delta_v^{\leftarrow}$ (as opposed to $\delta_u^{\leftarrow}$) and $\delta_u^{\rightarrow}$ we can modify Dijkstra's algorithm to restrict attention to those nodes that give a shorter path using the edge (u, v, d) (see (Cotton and Maler 2006) for details). We can calculate the weight of a shortest path from x to y via (u, v, d) as $\delta_v^{\leftarrow}(x) - d + \delta_u^{\rightarrow}(y)$.

Note that $wSP(x, y) = w + \pi(y) - \pi(x)$ where w is the weight of a shortest path from x to y in the reduced cost graph. Clearly $w \geq 0$ and hence $wSP(x, y) \geq \pi(y) - \pi(x)$. Hence if $\pi(y) - \pi(x) > d$ we know that $x - y \leq d$ is not implied by the constraints. Note also that π only changes by *reducing* the values at particular nodes (See INCSAT). Hence if $\pi(y) - \pi(x) > d$ then at some future time $\pi'(y) - \pi'(x) > d$ unless the potential function value at y changed.

Given both the above observations we can improve the checking of implication of constraints in P by only checking constraints $x - y \leq d \in P$ if $\delta_u^{\rightarrow}(y)$ made use of the edge (u, v, d) in its calculation, and if $\pi(y) - \pi(x) > d$ in the past, then $\pi(y)$ has changed.

5. Experiments

We have built a global difference constraint propagator in the G12 finite domain solver engine (P.J. Stuckey et al. 2005). We will consider four implementations in the experiments: sp: only separate propagators for each constraint (the standard approach); sps: seperate propagators with a global satisfaction check INCSAT run first on addition of new difference constraints; spsi: separate propagators with IncSat and IncImp run on difference and reified difference constraints. gp: a global propagator using IncSat and IncImp on difference and reified difference constraints, and IncLB and IncUB to handle bounds. We do not consider the base implementation of Section 4.1 since it is an order of magnitude slower than gp when there are bounds constraints.

All the experiments were carried out on a 3GHz Intel Pentium D with 4GB of RAM running Debian GNU/Linux 3.1.

5.1 Worst case behaviour for separate propagators

Example 7 shows a problem where the use of single propagators could be terrible compared to the global propagator gp just for bounds propagation. In the first experiment, we validate this experimentally, comparing the separate propagators sp and the global propagator gp on this problem. Note that the sp implementation is the existing G12 engine implementation and is quite standard. For different number of variables n, we applied consecutive lower bounds updates of y_0 from $(k-1)n$ to kn, $k \in 1..10$, and measured the time needed to reach these 10 fixpoints. The results for the sp implementation are show in Figure 4. The gp times are all below 10 ms and are not reported. One can see from the results that very bad propagation behaviour can occur in practical solvers when using single propagators, while a global propagator only needs to run once to reach a fixpoint.

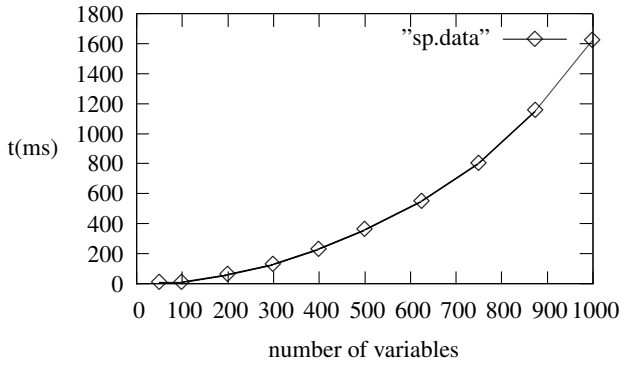

Figure 4. Times(ms) to reach fixpoint for Example 7 as a function of the number of variables

5.2 The generalized sequence constraint

The generalized sequence constraint, GEN-SEQUENCE, was introduced by van Hoeve *et al.* (van Hoeve et al. 2006) as a generalization of the SEQUENCE constraint. Given a sequence of Boolean variables $Y_1, \ldots, Y_n$ and a set of quadruples $(i, j, l, u) \in Q$ where $i \leq j$,

$$\text{GEN-SEQUENCE}([Y_1, \ldots, Y_n], Q) = \bigwedge_{(i,j,l,u)\in Q} l \leq Y_i + \cdots + Y_j \leq u$$

that is for each quadruple (i, j, l, u) the number of Booleans in the sub-sequence $Y_i, \ldots, Y_j$ that are true is between l and u (inclusive). The constraint $\text{SEQUENCE}([Y_1, \ldots, Y_n], k, l, u)$ is equivalent to $\text{GEN-SEQUENCE}([Y_1, \ldots, Y_n], R)$ where $R = \{(i, i+k-1, l, u) \mid 1 \leq i \leq n-k+1\})$.

The generalized sequence constraint can be encoded using difference constraints and reified difference constraints using a *cumulative sums* encoding using the sum variables $S_i = \sum_{j=1}^{k} Y_k$. The encoding is:

$$\begin{aligned}
S_i - S_{i-1} &\leq 1, && 1 \leq i \leq n \\
S_{i-1} - S_i &\leq 0, && 1 \leq i \leq n \\
Y_i \Leftrightarrow S_{i-1} - S_i &\leq -1, && 1 \leq i \leq n \\
S_i - S_j &\leq -l, && (i, j, l, u) \in Q \\
S_j - S_i &\leq u, && (i, j, l, u) \in Q.
\end{aligned}$$

This is a slight extension of the CD encoding of (Brand et al. 2007) which only considered SEQUENCE constraints. The result from (Brand et al. 2007) readily extends:

THEOREM 3. *The global difference constraint propagator on the encoding of the* GEN-SEQUENCE *constraint enforces domain consistency in $\mathcal{O}(n^2 \log n + n|Q|)$ time down each branch of a search tree.* □

To illustrate the performance of the difference constraint propagator we experimented using single SEQUENCE constraints.

For each combination of $n \in \{50, 200, 500\}$, $k \in \{7, 15, 25, 50\}$, $\Delta \in \{1, 5\}$, 20 SEQUENCE($[Y_i, \ldots, Y_n]$, k, l, $l + \Delta$) problems were randomly generated with l picked in $[0, \ldots, k - \Delta]$. These are similar experiments to those undertaken in (Brand et al. 2007). The average times and number of backtracks required to solve these problems are shown in Table 1 for different SEQUENCE implementations. All problems were solved with a random variables and values ordering over the Boolean variables Y_i. For these problems there are no bounds constraints so spsi is useless overhead compared to gp and is omitted

As the results in (Brand et al. 2007) show, there are not many cases where domain propagation of the sequence constraint improves upon weaker encodings. Although for our examples the domain propagator never needs to backtrack, the overhead of the global propagator is not repaid on these examples. We can however see that the difference constraint graph implementation of SEQUENCE has a better asymptotic behaviour for large n and k compared to other implementations in (Brand et al. 2007). We do see however that while adding a global consistency check of the difference constraints cannot improve search (as explained by Theorem 2) it does improves the running time, as it is a relatively cheap test compared to detecting failure through propagation.

5.3 Multiple sequences

The domain propagation of the sequence constraint is important for harder problems combining sequence constraints with other constraints. In order to illustrate this we constructed problems consisting of 3 sequence constraints on 3 sets of Booleans variables x, y, and z such that $z_i = x_i \oplus y_i$ ($\oplus$ is exclusive or). We generated satisfiable non-trivial instances for sequences of length 30 to 50, and solved them with a random static labeling order. Again we only compare sp, sps and gp since there are no bounds constraints so spsi is useless overhead compared to gp.

In Table 2, we show the number of instances solved by the different implementations with a time limit of 10 min. We also give the average times and failures for the instances that all implementations could solve. As we can see, for these problems the stronger propagation of gp significantly improves performance, allowing it to solve more problems, and to solve them on average 3 times faster.

5.4 Open Shop Scheduling Problems

We give experiments using open-shop scheduling problems from (CSP2SAT). Open-shop shedules are given a matrix of $m \times n$ tasks t with duration d_t constrained so that each task t can not overlap with tasks t' in the same row or column of the matrix. Non-overlap is encoded as $b_1 \Leftrightarrow s_t - s_{t'} \leq d_{t'}$, $b_2 \Leftrightarrow s_{t'} - s_t \leq d_t$, $b_1 + b_2 \geq 1$. While there are many specialized global propagators for scheduling problems (see e.g. (Laborie 2005)), and specialized approaches to open-shop scheduling (Tamura et al. 2006) with which a global difference constraint propagator cannot compete, they do provide a rich example to illustrate the advantages of global difference propagation.

The GPx-y are small hard instances of open shop scheduling problems, with x machines and x jobs (CSP2SAT). Table 3 shows the time and number of backtracks required to solve these instances for different combination of propagators. The number of backtracks are listed in the columns #bts for sp,sps, and #bts-i for spsi, gp. We show two search approachs: search by setting the earliest possible task to either start at its earliest possible time, or start later; and default labelling on the Boolean variables in the order of occurence. The second search is more effective, we did not run the larger examples using the first search

For the first search strategy the stronger implication does not reduce the search (since there are very few tasks overall). The addition of the global satisfaction check sps gives a significant improvement, and gp improves slightly on that, probably because of queueing savings. The second strategy is much better for these problems, and on the larger problems we see that the stronger implication check of spsi and gp reduces search. On the hardest example sps is slightly better than spsi and gp, while for remainder sp or gp are best.

6. Conclusion

Difference constraints appear widely in constraint programming models, and the default representation as individual propagators has some known bad behaviour. We show that by treating them globally we can improve upon propagator behaviour. We present 3 different approaches to using global information: sps simply adding a global satisfaction check to the usual individual propagators, which is always clearly beneficial; spsi adding a global satisfaction and implication check to individual propagators, which guarantees domain consistency; and gp replacing the individual propagators by a global propagator with special treatment of bounds propagation, which has better worst case behaviour than the previous methods.

We note that while SMT (Niewenhuis et al. 2005, 2006) solvers treat difference constraints globally, the naive importing of their methodology is impractical for finite domain propagation solving because of the relative importance of bounds propagation for FD. For problems with bounds constraints the approach outlined in Section 4.1 is an order of magnitude slower than gp.

Our current implementation of the global propagator is inefficient since we do not have advisors (Lagerkvist and Schulte 2007) to incrementally maintain which variables bounds have changed since the last time the global propagator was run. Hence we must scan all variables on each invocation. Thus it could be substantially improved with advisors. Finally it is straightforward to extend the global propagators to record the reasons for unsatisfiability or implication. If these are recorded as nogoods substantial search reduction can result (see e.g. (Tamura et al. 2006)).

References

Handbook of Theoretical Computer Science: Volume A: Algorithms and Complexity. Elsevier and MIT Press, 1990.

S. Brand, N. Narodytska, C-G. Quimper, P. J. Stuckey, and T. Walsh. Encodings of the sequence constraint. In C. Bessiere, editor, *Proceedings of the 13th International Conference on Principles and Practice of Constraint Programming*, volume 4741 of *LNCS*, pages 210–224. Springer-Verlag, 2007.

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

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

CSP2SAT. CSP2SAT. `http://bach.istc.kobe-u.ac.jp/csp2sat/`. [Dec07].

R. Dechter, I. Meiri, and J. Pearl. Temporal constraint networks. *Artificial Intelligence*, 49:61–95, 1991.

P. Laborie. Complete MCS-based search: Application to resource constrained project scheduling. In *Proceedings IJCAI 2005*, pages 181–186, 2005.

M. Lagerkvist and C. Schulte. Advisors for incremental propagation. In C. Bessiere, editor, *Proceedings of the 13th International Conference on Principles and Practice of Constraint Programming*, volume 4741 of *LNCS*, pages 409–422. Springer-Verlag, 2007.

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

			gp		sps		sp	
			Backs.	Time(ms)	Backs.	Time(ms)	Backs.	Time(ms)
$n = 50$	$k = 7$	$\Delta = 1$	**0**	9	6	1	6	**0**
		$\Delta = 5$	**0**	6	0	2	0	**1**
	$k = 15$	$\Delta = 1$	**0**	6	4	**0**	4	1
		$\Delta = 5$	**0**	5	3	1	3	**0**
	$k = 25$	$\Delta = 1$	**0**	3	2	**0**	2	**0**
		$\Delta = 5$	**0**	4	4	1	4	**0**
	$k = 50$	$\Delta = 1$	**0**	2	**0**	1	**0**	**0**
		$\Delta = 5$	**0**	4	**0**	**1**	**0**	**1**
$n = 200$	$k = 7$	$\Delta = 1$	**0**	71	36	**11**	36	19
		$\Delta = 5$	**0**	81	3	**23**	3	30
	$k = 15$	$\Delta = 1$	**0**	80	22	**7**	22	8
		$\Delta = 5$	**0**	99	12	**15**	12	25
	$k = 25$	$\Delta = 1$	**0**	82	24	**8**	24	10
		$\Delta = 5$	**0**	97	22	**10**	22	22
	$k = 50$	$\Delta = 1$	**0**	74	11	**6**	11	8
		$\Delta = 5$	**0**	95	16	**8**	16	14
$n = 500$	$k = 7$	$\Delta = 1$	**0**	439	97	**54**	97	151
		$\Delta = 5$	**0**	464	9	**149**	9	227
	$k = 15$	$\Delta = 1$	**0**	459	65	**33**	65	60
		$\Delta = 5$	**0**	532	47	**81**	47	246
	$k = 25$	$\Delta = 1$	**0**	473	53	**27**	53	42
		$\Delta = 5$	**0**	545	63	**53**	63	191
	$k = 50$	$\Delta = 1$	**0**	494	43	**28**	43	39
		$\Delta = 5$	**0**	580	55	**38**	55	94

Table 1. Backtracks/Times(ms) of sequence constraint encodings, for sequence constraints over a length n seqeunce, where the sums of subsequences of length k are constrained to be is a range of size Δ.

	Solved	Timed Out	Backtracks	Time(ms)
sp	35/59	24/59	46636	11572
sps	35/59	24/59	46636	9200
gp	**40/59**	**19/59**	**10641**	**3114**

Table 2. Results for 59 hard sequence-based problems, combining three sequence constraints on sequences of variables $\bar{x}$, $\bar{y}$ and $\bar{z}$ where $z_i = x_i \oplus y_i$.

R. Niewenhuis, A. Oliveras, and C. Tinelli. Solving SAT and SAT Modulo Theories: from an abstract Davis-Putnam-Logemann-Loveland procedure to DPLL(T). *JACM*, 53(6):937–977, 2006.

P.J. Stuckey, M. Garcia de la Banda, M. Maher, K. Marriott, J. Slaney, Z. Somogyi, M. Wallace, and T. Walsh. The G12 project: Mapping solver independent models to efficient solutions. In P. Van Beek, editor, *Proceedings of the 11th International Conference on Principles and Practice of Constraint Programming*, number 3709 in LNCS, pages 13–16. Springer-Verlag, 2005.

G. Ramalingan, J. Song, L. Joskowicz, and R.E. Miller. Solving systems of difference constraints incrementally. *Algorithmica*, 23:261–275, 1999.

R. Shostak. Deciding linear inequalities by computing loop residues. *JACM*, 28(4):769–779, 1981.

N. Tamura, A. Taga, S. Kitagawa, and M. Banbara. Compiling finite linear CSP to SAT. In *Proceedings of CP-2006*, volume 4204 of *LNCS*, pages 590–603. Springer-Verlag, 2006.

Willem-Jan van Hoeve, Gilles Pesant, Louis-Martin Rousseau, and Ashish Sabharwal. Revisiting the sequence constraint. In *Proceedings of the 12th International Conference on Principles and Practice of Constraint Programming (CP '06)*, 2006.

Chao Wang, Franjo Ivančić, Malay Ganai, and Aarti Gupta. Deciding Separation Logic Formulae by SAT and Incremental Negative Cycle Elimination. In *Logic for Programming, Artificial Intelligence, and Reasoning*, volume 3835 of *LNCS*, pages 322–336. Springer-Verlag, 2005.

	Earliest						Booleans labelling					
	#bts	sp	sps	#bts-i	spsi	gp	#bts	sp	sps	#bts-i	spsi	gp
GP03-01	532512	58.90	31.97	532512	32.38	**28.16**	216	1.34	1.34	214	**1.33**	**1.33**
GP03-02	755164	81.73	42.75	755164	43.04	**38.64**	227	1.34	**1.32**	225	1.33	1.33
GP03-03	385593	43.86	23.81	385593	23.97	**21.60**	151	1.34	**1.33**	151	**1.33**	**1.33**
GP03-04	288415	33.27	18.81	288415	18.99	**16.50**	161	1.34	**1.33**	161	**1.33**	**1.33**
GP03-05	812349	86.31	44.86	812349	45.08	**40.54**	144	1.34	1.34	144	**1.33**	1.34
GP03-06	693235	75.35	40.12	693235	40.03	**35.45**	175	**1.33**	**1.33**	175	**1.33**	**1.33**
GP03-07	152158	19.07	11.50	152158	11.61	**10.06**	117	**1.33**	1.34	117	**1.33**	**1.33**
GP03-08	338889	38.61	21.39	338889	21.57	**19.04**	165	1.35	**1.33**	165	**1.33**	**1.33**
GP03-09	319610	36.31	20.03	319610	20.20	**18.15**	190	1.34	**1.33**	188	**1.33**	1.34
GP03-10	564040	61.38	32.87	564040	33.24	**29.44**	113	1.33	**1.32**	113	1.33	1.33
GP04-01							1082	**1.45**	1.80	1046	1.79	1.79
GP04-02							826	1.46	1.82	824	1.79	**1.36**
GP04-03							1663	**1.65**	1.87	1626	1.89	1.84
GP04-04							2655	2.11	1.87	2485	1.89	**1.86**
GP04-05							591721	85.93	19.12	557328	19.25	**18.12**
GP04-06							686	1.61	1.81	686	1.80	**1.37**
GP04-07							1679	**1.69**	1.87	1672	1.87	1.89
GP04-08							556	1.41	1.76	555	1.77	**1.35**
GP04-09							3488592	324.85	**110.08**	3430994	112.74	112.73
GP04-10							692	1.43	1.78	689	1.79	**1.36**
average	484196	53.479	28.811	484196	29.011	**25.758**	204591	21.8485	7.8545	199978	7.994	**7.8495**

Table 3. Comparison of propagator combination of Open Shop Scheduling. GPn-m is a problem with $n \times n$ tasks. We compare two different search strategies: Earliest setting a task with the earliest possible start time to this time or to start later; and Boolean labelling, where the Boolean variables in the model are labelled in order of appearance.